Preproceedings of

LION18

The 18th Learning and Intelligent Optimization Conference

June 9-13, 2024, Ischia, Italy

Paola Festa (Chair)
Maurizio Bruglieri
Ciriaco D'Ambrosio
Daniele Ferone
Giusy Macrina
Tommaso Pastore
Ornella Pisacane

This is the book of the proceedings accepted for presentation at the 18th Learning and Intelligent Optimization Conference, LION18, held in Ischia from June 9th to June 13th, 2024.

LION18 continues the successful series of LION events in Athens (LION14), in Milos Island (LION16) and in Nice (LION17) and it is organized with the support of the Dipartimento di Matematica e Applicazioni "Renato Caccioppoli", Università di Napoli Federico II and of Dipartimento di Ingegneria e Scienza dell'Informazione, Università di Trento.

LION18 explores the intersections between machine learning, artificial intelligence, mathematical programming, and algorithms for hard optimization problems. Therefore, experts from these areas meet each other to discuss new ideas and methods, challenges, developments, and opportunities in various application domains.

The book contains 40 contributions, divided into two categories: short contributions (i.e., extended abstracts of novel work) and long contributions (i.e., almost 15 pages in which original novel and unpublished work is described).

LION18 has five invited speakers from both academic and industrial world:

- **Mauricio G. C. Resende**, from University of Washington, Seattle, with the talk entitled "Random-Key Optimizers (Rko): Problem Independent Combinatorial Optimization";
- **Frank Hutter**, from University of Freiburg, Germany, with the talk entitled "AI That Builds and Improves Ai: Meta-Learning The Next Generation Of Learning Methods";
- **Ruth Misener**, from Imperial College London, United Kingdoms, with the talk entitled "Optimal Decision-Making Problems with Trained Surrogate Models Embedded";
- **Matthias Poloczek**, from Amazon, USA, with the talk entitled "Reliable High-Dimensional Bayesian Optimization for Combinatorial and Mixed Spaces";
- **Kevin Tierney**, from Bielefeld University, Germany, with the talk entitled "Deep Reinforcement Learning for Vehicle Routing Problems".

From June 10th, the LION18 contributions are organized into streams, three of which are special sessions, as in the following specified

- Special Session 1: Data-Driven Optimization with Business Process Mining, proposed by Om Prakash Vyas, from Indian Institute of Information Technology, Allahabad, India and Jerome Geyer-Klingeberg, from Celonis, Munich, Germany
- Special Session 2: Advances and Perspectives in Bayesian Optimization, organized by Antonio Candelieri, from University of Milano-Bicocca, Italy

– Special Session 3: Learning and Intelligent Optimization for Physical Systems, proposed by Konstantinos Chatzilygeroudis and Michael Vrahatis, from University of Patras, Greece.

The remaining part of the present book contains the contributions with information about the authors and their affiliations.

The LION18 Organizing Committee

Paola Festa (Chair)
Maurizio Bruglieri
Ciriaco D'Ambrosio
Daniele Ferone
Giusy Macrina
Tommaso Pastore
Ornella Pisacane

Acknowledgments A special thanks goes to the head of the Steering Committee, Prof. R. Battiti (Università di Trento), all the members of the Steering Committee and of the Technical Program Committee and all those who contributed to the realization of the conference. In particular, we sincerely thank the members of the Program Committee, the Dipartimento di Matematica e Applicazioni "Renato Caccioppoli", Università di Napoli Federico II and the Dipartimento di Ingegneria e Scienza dell'Informazione, Università di Trento.

Table of Contents

Randomized Greedy Sampling for JSSP

Henrik Abgaryan, Ararat Harutyunyan, and Tristan Cazenave

LAMSADE, Université Paris Dauphine - PSL, CNRS, Paris, France

Abstract. The job shop scheduling problem (JSSP) is a fundamental challenge in the field of operations research and manufacturing, representing the task of optimally assigning a set of jobs to a limited number of machines to optimize one or more objectives, such as minimizing the total processing time or reducing the delay of jobs. In recent years, AI-driven methods have introduced new approaches to solving the JSSP. Continuous exploration in deep reinforcement learning (DRL) is currently concentrated on refining strategies to address the JSSP. Established DRL techniques mostly focus on better modeling and training of the Policy networks for solving JSSP problems. This paper explores the utilization of Policy networks in search algorithms. We propose two novel algorithms, Random Second Greedy Choice (RSGC) and Greedy Sampling (GS). RSGC and GS employ a randomized approach to consider alternative paths, deviating from the primary heuristic, while adjusting the probability of selecting these paths dynamically during the search process. Through experimentation, we show the effectiveness of the proposed algorithms in comparison to the usual greedy first choice inference technique and the usual sampling method.

Keywords: Job Shop Scheduling, RSGC, GS, GNN, reinforcement learning, limited discrepancy search, sampling, search algorithms

1 Introduction

The job shop scheduling problem (JSSP) stands as a pivotal challenge in operations research and manufacturing [9]. This problem entails a set of jobs, each bound by specific processing rules (like the sequential use of assigned machines), across a variety of machines. The aim here is to optimize certain parameters such as the overall completion time, workflow duration, or delay minimization.

It is proved that for JSSP instances having more than 2 machines is NP-hard [10]. Therefore, deriving precise solutions for JSSP is generally unfeasible, thereby making heuristic and approximate strategies more common for practical efficiency [6]. Traditional approaches to this problem have predominantly relied on search and inference techniques developed by the constraint programming community [1]. These techniques effectively utilize constraints to define the relationships and limitations between jobs and resources, thus enabling efficient exploration of feasible solution spaces and the identification of optimal or near-optimal schedules [13]. Another common technique is the Priority dispatching rule (PDR). It is a heuristic method that is widely used in real-world scheduling systems [17]. However, designing an effective PDR is very time-consuming, it requires solid domain knowledge for complex JSSP problems. Another

approach for finding suboptimal solutions is through the help of Deep Learning and Neural Networks [3], [18]. The method based on learning can generally be categorized into two paradigms: supervised learning and reinforcement learning (RL). Ongoing research in deep reinforcement learning (DRL) is actively focused on developing new and improved methods to tackle the JSSP. Existing DRL methods represent Job Shop Scheduling Problem (JSSP) as a Markov decision process (MDP) and then learn the Policy network based on DRL. The prevailing trend leans towards refining policy networks or MDP formulation to generate better solutions. However, the exploration of effective search methods on top of the policy networks decisions and the integration of alternative strategies such as tree search, have received less attention. This indicates a potential area for further research.

In this paper, we present two search algorithms RSGC, GS, provided by pretrained Policy network inspired by [11] that utilize the categorical distribution of the pretrained Policy network to find a solution to JSSP problem. In order to check the effectiveness of the proposed search methods, we have experimented on 2 public test datasets: TA dataset [15], and DMU dataset [7]. Then we compare the results of RSGC and GS to the usual sampling algorithm and the current state-of-the-art results [18].

2 Related Work

The success of Deep Reinforcement Learning (DRL) represents a significant advancement in the field of artificial intelligence. Generally, DRL combines the principles of deep learning and reinforcement learning, enabling computers to learn complex behaviors by interacting with an environment and optimizing actions based on feedback.

In [12] the authors utilized deep Q-network (DQN) to solve a JSSP in semiconductor manufacturing plant. In [18], a new method using Deep Reinforcement Learning (DRL) is introduced to develop effective Priority Dispatching Rules (PDRs) for the Job Shop Scheduling Problem (JSSP). The approach involves formulating an MDP (Markov Decision Process) for PDR-oriented scheduling. This method utilizes a disjunctive graph representation of JSSP to capture the states, efficiently integrating operation dependencies and machine statuses for informed scheduling decisions. Additionally, the paper uses a Graph Isomorphism Network (GIN) strategy that efficiently encodes disjunctive graph nodes into fixed-dimensional embeddings. On top of the these embeddings the authors utilized Proximal Policy Optimization (PPO) algorithm [14] to train Policy network. [16] introduces a new approach for solving JSSP with DRL. It addresses the deficiencies in state representation, adding more features. This approach also models JSSP as a Markov decision process, employing a new state representation based on bidirectional scheduling. This representation allows the agent to capture more effective state information and avoid the issue of multiple optimal action selections. They also utilize the technique of Invalid Action Masking (IAM), which narrows the search space, steering the agent away from sub-optimal solutions.

Current methods are mostly based on modeling state representation and the architecture of the models. There are very few papers that use sampling or search algorithms on top of policy network to solve JSSP. A widely recognized strategy is to facilitate the step-by-step development of solutions to JSSP problems, guided by a single-shot

(greedy) policy derived from a neural network, as documented in sources like [18] and [16]. These methods primarily concentrate on developing robust policy network models, aiming to elevate the quality of solutions generated in a single iteration as close to the optimal as possible. However, there is a noticeable scarcity of studies dedicated to finding effective inference methods for neural JSSP. In addition to designing and training high-quality policy networks, devising an effective inference strategy is equally crucial to maximize the quality of solutions within a specified time budget. In contrast, neural construction methods focus on generating solutions by sampling from the neural network's output probability distributions, a technique highlighted in studies such as [2]. An alternative approach to sampling is the use of Monte-Carlo Tree Search (MCTS) [4], and its variants [5]. These methods create partial solutions within a search tree using rollouts. MCTS, requires a lot of resources and long running time. Beam search represents another method used in combinatorial optimization problems [8]. Nevertheless, beam search operates on a greedy algorithm, which unconditionally adheres to the neural network's predictions, regardless of their accuracy. Like MCTS, Beam search also requires a lot of resources and long running time. Another approach called Limited Discrepancy Search (LDS) incorporates the idea of deviating from the main heuristic and sometimes selecting less promising choices [11]. It is based on the idea that sometimes, choosing solutions that don't initially seem promising might actually lead to better results. This approach acknowledges that the most obvious choice is not always the best one.

In developing our approach, we focus on a key strategy: introducing variations to the primary heuristic, specifically diverging from the initial greedy choice recommended by the policy network. This strategy lays the groundwork for our newly designed probabilistic sampling algorithms. The essence of these algorithms lies in the smart utilization of the Policy network's probability distribution also incorporating a measure of randomness to enrich decision-making processes. By integrating randomness at the beginning of our algorithm, it diverges from conventional deterministic methods, offering a nuanced way to explore the search space. It harnesses the robustness of greedy strategies while mitigating their inherent limitations through calculated randomness. This hybrid approach aims to strike a balance between the exploration and exploitation, optimizing the search process.

3 Preliminary

The Job-Shop Scheduling Problem (JSSP) is formally defined as a problem involving a set of jobs J and a set of machines M. The size of the JSSP problem instance is described as $N\_J \times N\_M$, where N_J represents the number of jobs and N_M the number of machines. For each job $J_i \in J$, it must be processed through n_i machines in a specified order $O_{i1} \rightarrow \ldots \rightarrow O_{in_i}$, where each O_{ij} (for $1 \leq j \leq n_i$) represents an operation of J_i with a processing time $p_{ij} \in \mathbb{N}$. This sequence also includes a precedence constraint. Each machine can process only one job at a time, and switching jobs mid-operation is not allowed. The objective of solving a JSSP is to determine a schedule, that is, a start time S_{ij} for each operation O_{ij}, to minimize the makespan

$C_{\max} = \max_{i,j}\{C_{ij} = S_{ij} + p_{ij}\}$ while meeting all constraints. The complexity of a JSSP instance is given by $N\_J \times N\_M$.

Furthermore, a JSSP instance can be represented through a disjunctive graph, a concept well-established in the literature [18]. Let $O = \{O_{ij}|\forall i, j\} \cup \{S, T\}$ represent the set of all operations, including two dummy operations S and T that denote the starting and ending points with zero processing time. A disjunctive graph $G = (O, C, D)$ is thus a mixed graph (a graph consisting of both directed edges (arcs) and undirected edges) with O as its vertex set. Specifically, C comprises of directed arcs (conjunctions) that represent the precedence constraints between operations within the same job, and D includes undirected arcs (disjunctions) connecting pairs of operations that require the same machine. Solving a JSSP is equivalent to determining the direction of each disjunctive arc such that the resulting graph becomes a Directed Acyclic Graph (DAG) [18]. Markov Decision Process Formulation (MDP), state representation, action space, state transition follow the methods described in [18]. An action $a_t \in A_t$ is an eligible operation at decision step t. The Policy is a neural network $\pi(a_t|s_t)$ that outputs a distribution over the actions in A_t. We use the same 2D raw features as in [18], namely a binary indicator $I(O, s_t)$ which equals to 1 only if O is scheduled in s_t, and an integer $CLB(O, s_t)$ which is the lower bound of the estimated time of completion (ETC) of O in s_t. The training has been done using Proximal Policy algortihm, which is an actor-critic algorithm [14]. "Actor" denotes the Policy network, whereas the "critic" is a distinct network that evaluates the outcomes of the decisions made by the Policy network. Both the Actor (Policy) network and the Critic network have multilayer perceptron architecture. We use the same Graph Isomorphism Network (GIN) as feature extractor, the Actor and the Critic networks as presented in [18]. GIN extracts feature embeddings of each node in an iterative and non-linear fashion. Then the Policy network uses these fixed-size embeddings for outputting a distribution over the actions. The training has been done on 20x20 training instances for 10000 steps. The training is as described in [18]. Because the Policy network is based on the fixed-sized embeddings outputted by the GIN network, it is size-agnostic which enables generalization to instances of different sizes without requiring any additional training. During the training we sample the actions according to the probability distribution outputted by the Policy network. During the inference time we utilize our search algorithms over the probability distribution outputted by the Policy network.

4 Search Algorithm

Finding the right action a_t at each state s_t is a search problem, where the Policy network $\pi(a_t|s_t)$ is used as a heuristic. When the $\pi(a_t|s_t)$ is trained and is ready for the inference, the usual method is to pick the first greedy choice action a_t, where a_t is the action with the highest probability at the state s_t, according to the categorical probability distribution of $\pi(a_t|s_t)$.

Our approach is slightly different; we formulate the problem of selecting the "correct" action a_t at each state s_t as a search problem. Let us denote by T the binary search tree. At each node n_i of the search tree T there are two options: selecting the action with the highest probability (the first greedy choice), or another action according to the

probability distribution recommended by $\pi(a_t|s_t)$. Then the height of the search tree T becomes $h = N\_J \times N\_M$. At each node n_i the approach of [18] is to always choose the first greedy choice, action a_t with the highest probability at the state s_t. However as described in [11], always following the heuristic, or in our case only the first greedy choice recommended by $\pi(a_t|s_t)$, might not lead to the best possible solution. We argue that doing "discrepancies" or deviating from the first greedy choice of $\pi(a_t|s_t)$ can sometimes be better. In the context of the search trees, a "discrepancy" refers to a deviation from the most preferred or recommended path, typically represented as taking a right turn in a binary search tree that is organized based on heuristic evaluations (where the left path is usually considered the default or the more promising direction based on some criteria). There is a systematic approach of considering all of the paths in a binary search tree, as described in [11]. This systematic approach allows the search to first consider paths that are closely aligned with heuristic recommendations, and only later to explore less recommended paths. But this method can quickly become unfeasible when the search tree is large, in our case when the set of jobs J and a set of machines M is large. Our Algorithm (RSGC) Random Second Greedy Choice with Decreasing Probability gets inspiration from [11], but it also utilizes randomness. The pseudo code of our algorithm is presented in Algorithm 1. Our algorithm has two hyperparameters D_{min} and D_{max}, which are the least possible and the most possible second greedy choices we can make during the search. At each node n_i of the search tree T we compute the probability (based on the hyperparameters) of selecting the second greedy choice action. At the beginning, the probability of selecting the second greedy choice action is high. The probability then decreases as we make more and more greedy second choices. The $greedy\_2nd\_action\_count$ is the number of second greedy choices done so far. Initially, $total\_num\_greedy\_2nd\_action\_count$ is set to D_{min}. The full search is done using this fixed hyperparameter D_{min} obtaining the first makespan. Then we adjust the value of $total\_num\_greedy\_2nd\_action\_count$ by increasing it by an amount of 5% of the height h, obtaining the next makespan. We repeat this process until $total\_num\_greedy\_2nd\_action\_count$ reaches D_{max} . We then choose the best makespan.

$$probability\_of\_2nd\_action = 1 - \frac{greedy\_2nd\_action\_count}{total\_num\_greedy\_2nd\_action\_count}$$

The idea is that it is harder for the policy network to select correct actions at the beginning of the search, than at the end. Through extensive experimentation, we found that it is best to start to do around $0.25h$ second greedy choice actions. We increment the value of $0.25h$ by $0.01h$ for each subsequent iteration until we reach $0.75h$ (i.e., $D_{min} = 0.25h$ and $D_{max} = 0.75h$). This methodical approach facilitates a more effective exploration of options, thereby aiding in the identification of the optimal path characterized by the minimal completion time (*makespan*).

We also have a second similar algorithm called Greedy Sampling (GS). In GS instead of randomly selecting second probable action in line 14 of the Algorithm 1 (RSGC) we randomly sample an action according to the probability distribution recommended by the policy network $\pi(a_t|s_t)$. Note that this is not the usual sampling algorithm, as we decide whether we sample or just take the first greedy choice with

probability $probability\_of\_2nd\_action$. In the Experimental section we compare the results of these two algorithms with the usual sampling algorithm.

Algorithm 1 (RSGC) Random Second Greedy Choice with Decreasing Probability

Require: $dataset$
Ensure: Results list
1: $h \leftarrow N\_J \times N\_M$
2: $D_{min} \leftarrow integer(0.25 \times h)$
3: $D_{max} \leftarrow integer(0.75 \times h)$
4: Initialize results list
5: **for** each $data$ in $dataset$ **do**
6: Initialize best makespan tracking list
7: **for** $total\_num\_greedy\_2nd\_action\_count$ in D_{min} to D_{max} **do**
8: Reset environment with $data$
9: Initialize episode reward and $greedy\_2nd\_action\_count$
10: **while** not Terminal **do**
11: $categorical\_policy\_distribution \leftarrow$ inference with policy network $\pi(a_t|s_t)$
12: $probability\_of\_2nd\_action = 1 - \frac{greedy\_2nd\_action\_count}{total\_num\_greedy\_2nd\_action\_count}$
$r \sim$ Uniform$(0, 1)$
13: **if** r $< probability\_of\_2nd\_action$ **then**
14: Select the second probable action from $categorical\_policy\_distribution$
15: Increment $greedy\_2nd\_action\_count$
16: **else**
17: Greedily select the most probable action from $categorical\_policy\_distribution$
18: **end if**
19: Execute the action
20: **end while**
21: Update best makespan if current makespan is better
22: Record makespan
23: **end for**
24: Save results for current data instance
25: **end for**
26: **return** Results list

5 Experimental Results

In order to show the effectiveness of our searching algorithm, we have conducted experiments on well known TA dataset [15], and DMU dataset [7]. The Policy network is trained on instances of size $N\_J = 20$ and $N\_M = 20$. We also compare the results with those of [18]. Each run of our search algorithm makes $(D_{max} - D_{min}) \times h$ calls to the Policy network, where $h = N\_J \times N\_M$ is the height of the search tree. During the experiments we have repeated the search algorithms 10 times and noted the overall minimum and the average makespans on the corresponding tables. We compare the makespans of (RSGC) Algorithm 1 with Greedy Sampling (Ours-GS). The two algorithms just differ on line 14 of the Algorithm 1, where instead of the Greedy second choice we randomly sample an action from the probability distribution provided by the

pre-trained Policy network. For the Taillard dataset instances 50×15, 50×20 and 100×20 and also for the DMU dataset 40×15, 40×20, 50×15, 50×20 we compared the best result of [18], which was inferred with Policy network trained on 30×20 instances. We have also tried to run beam search algorithm, however we had to keep many copies of the environment, and due to memory constraints it was unfeasible.

5.1 Result on Taillard's Benchmark Dataset

Table 1: Results on Taillard's Benchmark (Part I). Ours - RSGC is the result of the Algorithm 1. In Ours-GS instead of randomly selecting second probable action in line 14 of the Algorithm 1 (RSGC) we randomly sample an action according to the probability distribution recommended by the policy network $\pi(a_t|s_t)$. We repeat each experiment 10 times due to the algorithm's randomness and record the minimum and the average makespan across the 10 experiment. The "Samp (min)" and the "Samp (avg)" columns are the results of the usual sampling method. The "UB" column represents the best-known solutions from literature, with "*" indicating optimal solutions.

Instance	L2D	Ours-RS (min)	Ours-RS (avg)	Ours-RSGC (min)	Ours-RSGC (avg)	Samp (min)	Samp (avg)	UB
15x15								
Ta01	1443 (17.22%)	1401 (13.81%)	1413.4 (14.79%)	**1387 (12.66%)**	1403.9 (14.03%)	1444(17.30%)	1448.3(17.65%)	1231*
Ta02	1544 (24.12%)	1404 (12.86%)	1404.0 (12.86%)	**1361 (9.40%)**	1394 (12.06%)	1447(16.32%)	1462.0(17.52%)	1244*
Ta03	1440 (18.23%)	1420 (16.59%)	1420.6 (16.64%)	**1408 (15.60%)**	1430.1 (17.40%)	1447(18.80%)	1452.6(19.26%)	1218*
Ta04	1637 (39.32%)	**1412 (20.17%)**	1412.7 (20.23%)	1423 (21.11%)	1433.8 (22.04%)	1460(24.26%)	1484.0(26.30%)	1175*
Ta05	1619 (32.27%)	**1394 (13.89%)**	1410.0 (15.20%)	1431 (16.91%)	1435.7 (17.29%)	1437(17.40%)	1448.0(18.30%)	1224*
Ta06	1601 (29.32%)	**1392 (12.44%)**	1401.8 (13.23%)	1413 (14.12%)	1418.3 (14.55%)	1444(16.64%)	1444.6(16.69%)	1238*
Ta07	1568 (27.79%)	1402 (14.26%)	1411.5 (15.03%)	**1384 (12.78%)**	1392.3 (13.45%)	1451(18.26%)	1469.0(19.72%)	1227*
Ta08	1468 (20.62%)	**1372 (12.73%)**	1380.1 (13.39%)	1404 (15.36%)	1406.4 (15.53%)	1425(17.09%)	1427.3(17.28%)	1217*
Ta09	1627 (27.70%)	1483 (16.41%)	1491.4 (17.05%)	**1467 (15.15%)**	1469.4 (15.34%)	1544(21.19%)	1546.0(21.35%)	1274*
Ta10	1527 (23.04%)	**1401 (12.89%)**	1419.1 (14.34%)	1437 (15.79%)	1444.4 (16.39%)	1444(16.36%)	1466.6(18.18%)	1241*
20x15								
Ta11	1794 (32.19%)	**1583 (16.65%)**	1622.4 (19.57%)	1617 (19.16%)	1631.1 (20.18%)	1668(22.92%)	1684.1(24.10%)	1357*
Ta12	1805 (32.01%)	1590 (16.31%)	1595.1 (16.68%)	**1568 (14.70%)**	1573.6 (15.11%)	1658(21.29%)	1665.0(21.80%)	1367*
Ta13	1932 (43.85%)	1628 (21.22%)	1634.5 (21.69%)	**1619 (20.55%)**	1619.0 (20.55%)	1693(26.06%)	1697.0(26.36%)	1343*
Ta14	1664 (23.72%)	**1596 (18.66%)**	1600.3 (18.97%)	1604 (19.26%)	1624.7 (20.78%)	1640(21.93%)	1651.1(22.76%)	1345*
Ta15	1730 (29.20%)	**1625 (21.36%)**	1629.1 (21.65%)	1633 (21.96%)	1640.1 (22.47%)	1690(26.21%)	1694.9(26.58%)	1339*
Ta16	1710 (25.74%)	1613 (18.60%)	1641.6 (20.71%)	**1598 (17.50%)**	1610.1 (18.39%)	1693(24.49%)	1696.3(24.73%)	1360*
Ta17	1897 (29.75%)	**1728 (18.19%)**	1728.0 (18.19%)	1735 (18.66%)	1736.5 (18.76%)	1797(22.91%)	1798.0 (22.98%)	1462*
Ta18	1794 (28.51%)	**1664 (19.20%)**	1667.9 (19.49%)	1675 (20.00%)	1679.0 (20.34%)	1712(22.64%)	1712.3(22.66%)	1396
Ta19	1682 (26.28%)	**1591 (19.44%)**	1602.2 (20.29%)	1592 (19.52%)	1603.5 (20.39%)	1650(23.87%)	1653.8(24.16%)	1332*
Ta20	1739 (28.97%)	1635 (21.29%)	1635.0 (21.29%)	**1628 (20.77%)**	1657.7 (22.98%)	1666 (23.59%)	1666.3 (23.61%)	1348*
20x20								
Ta21	2252 (37.18%)	1964 (19.61%)	1973.5 (20.18%)	**1905 (16.02%)**	1914.6 (16.60%)	2021(23.08%),	2044.0(24.48%)	1642*
Ta22	2102 (31.38%)	1887 (17.94%)	1895.7 (18.48%)	**1853 (15.81%)**	1854.6 (15.91%)	1950(21.88%)	1953.7(22.11%)	1600
Ta23	2085 (33.91%)	1838 (18.05%)	1841.0 (18.24%)	**1818 (16.76%)**	1822.9 (17.05%)	1902(22.16%)	1918.8(23.24%)	1557
Ta24	2200 (33.82%)	1930 (17.40%)	1931.2 (17.48%)	**1907 (16.00%)**	1916.2 (16.54%)	1986(20.80%)	2035.3(23.80%)	1644*
Ta25	2201 (38.00%)	1918 (20.25%)	1924.4 (20.65%)	**1903 (19.31%)**	1904.6 (19.42%)	1997(25.20%)	2002.2(25.53%)	1595
Ta26	2176 (32.44%)	1961 (19.36%)	1970.4 (19.91%)	**1918 (16.73%)**	1928.9 (17.38%)	2014(22.58%)	2042.2(24.30%)	1643
Ta27	2132 (26.90%)	**2019 (20.18%)**	2028.8 (20.76%)	2026 (20.60%)	2026.0 (20.60%)	2092(24.52%)	2099.0(24.94%)	1680
Ta28	2146 (33.94%)	1883 (17.47%)	1886.6 (17.68%)	**1833 (14.35%)**	1841.5 (14.87%)	1947(21.46%)	1954.8(21.95%)	1603*
Ta29	1952 (20.12%)	1885 (16.00%)	1905.8 (17.28%)	**1877 (15.51%)**	1897.4 (16.76%)	1937(19.20%)	1984.4(22.12%)	1625
Ta30	2035 (28.47%)	**1868 (17.93%)**	1868.8 (17.98%)	1876 (18.43%)	1892.2 (19.44%)	1927(21.65%)	1934.6(22.13%)	1584
30x15								
Ta31	2565 (45.37%)	**2115 (19.91%)**	2121.0 (20.24%)	2169 (22.94%)	2183.5 (23.81%)	2170(23.02%)	2183.1(23.76%)	1764*
Ta32	2388 (33.87%)	**2212 (23.99%)**	2212.0 (23.99%)	2230 (24.94%)	2242.1 (25.65%)	2303(29.09%)	2318.0(29.93%)	1784
Ta33	2324 (29.80%)	**2204 (23.09%)**	2219.0 (23.92%)	2292 (28.03%)	2299.5 (28.45%)	2301(28.48%)	2307.8(28.86%)	1791
Ta34	2332 (27.60%)	**2207 (20.75%)**	2226.3 (21.81%)	2225 (21.74%)	2225.7 (21.79%)	2250(23.09%)	2262.7(23.78%)	1828*
Ta35	2505 (24.82%)	**2250 (12.08%)**	2260.5 (12.63%)	2256 (12.39%)	2256.0 (12.39%)	2293(14.25%)	2303.0(14.75%)	2007*
Ta36	2497 (37.31%)	**2243 (23.31%)**	2252.1 (23.83%)	2253 (23.87%)	2259.1 (24.22%)	2318(27.43%)	2320.6(27.58%)	1819*
Ta37	2325 (31.28%)	**2146 (21.20%)**	2157.1 (21.79%)	2159 (21.93%)	2168.0 (22.44%)	2181(23.15%)	2213.4(24.98%)	1771*
Ta38	2302 (37.61%)	2046 (22.30%)	2048.6 (22.45%)	**2014 (20.38%)**	2017.8 (20.63%)	2079(24.27%)	2114.9(26.41%)	1673*
Ta39	2410 (34.26%)	**2152 (19.94%)**	2152.8 (19.99%)	2160 (20.33%)	2166.8 (20.77%)	2181(21.50%)	2199.7(22.55%)	1795*
Ta40	2140 (28.21%)	2002 (19.95%)	2018.5 (20.93%)	**1989 (19.17%)**	1989.0 (19.17%)	2067(23.85%)	2083.4(24.83%)	1669

Continued on next page

Table 1 continued from previous page

Instance	L2D	Ours-GS (min)	Ours-GS (avg)	Ours-RSGC (min)	Ours-RSGC (avg)	Samp (min)	Samp (avg)	UB
30x20								
Ta41	2667 (33.02%)	**2489 (24.14%)**	2507.5 (25.06%)	2490 (24.19%)	2496.4 (24.51%)	2540(26.68%)	2568.7(28.11%)	2005
Ta42	2664 (37.53%)	2383 (23.03%)	2383.0 (23.03%)	**2358 (21.73%)**	2373.1 (22.51%)	2479(27.98%)	2490.3(28.56%)	1937
Ta43	2431 (31.69%)	2347 (27.14%)	2364.8 (28.10%)	**2332 (26.33%)**	2332.0 (26.33%)	2437(32.02%)	2457.6(33.13%)	1846
Ta44	2714 (37.14%)	2512 (26.93%)	2515.2 (27.09%)	**2471 (24.86%)**	2482.2 (25.43%)	2590(30.87%)	2595.8(31.17%)	1979
Ta45	2637 (31.85%)	2435 (21.75%)	2440.7 (22.03%)	**2380 (19.00%)**	2410.8 (20.54%)	2518(25.90%)	2530.5(26.53%)	2000
Ta46	2776 (38.38%)	2523 (25.77%)	2531.7 (26.21%)	**2435 (21.39%)**	2449.6 (22.11%)	2590(29.11%)	2595.5(29.39%)	2006
Ta47	2476 (31.07%)	2364 (25.15%)	2381.0 (26.05%)	**2344 (24.09%)**	2354.6 (24.65%)	2418(28.00%)	2429.0(28.59%)	1889
Ta48	2490 (28.55%)	2397 (23.75%)	2401.4 (24.00%)	**2339 (20.75%)**	2354.3 (21.54%)	2458(26.90%)	2462.9(27.15%)	1937
Ta49	2556 (30.34%)	2362 (20.45%)	2363.8 (20.54%)	**2349 (19.79%)**	2365.3 (20.62%)	2448(24.83%)	2459.0(25.40%)	1961
Ta50	2628 (36.66%)	**2395 (24.54%)**	2395.0 (24.54%)	2411 (25.38%)	2417.4 (25.71%)	2462(28.03%)	2463.7(28.12%)	1923
50x15								
Ta01	3599 (30.40%)	**3361 (21.78%)**	3368.2 (22.04%)	3417 (23.80%)	3417.0 (23.80%)	3404(23.33%)	3409.2(23.52%)	2760*
Ta02	3341 (21.23%)	**3212 (16.55%)**	3212.0 (16.55%)	3249 (17.89%)	3270.0 (18.65%)	3304(19.88%)	3313.8(20.24%)	2756*
Ta03	3186 (17.26%)	**3000 (10.42%)**	3004.1 (10.57%)	3045 (12.07%)	3050.1 (12.26%)	3084(13.51%)	3091.6(13.79%)	2717*
Ta04	3266 (15.04%)	**3120 (9.90%)**	3132.6 (10.34%)	3134 (10.39%)	3134.0 (10.39%)	3183(12.12%)	3201.0(12.75%)	2839*
Ta05	3232 (20.64%)	**3132 (16.91%)**	3140.1 (17.21%)	3111 (16.13%)	3124.8 (16.64%)	3196(19.30%)	3226.3(20.43%)	2679*
Ta06	3378 (21.47%)	**3134 (12.69%)**	3146.7 (13.15%)	3169 (13.95%)	3175.4 (14.18%)	3197(14.96%)	3206.4(15.30%)	2781*
Ta07	3471 (17.94%)	**3310 (12.47%)**	3316.6 (12.69%)	3349 (13.80%)	3353.6 (13.95%)	3340(13.49%)	3357.1(14.07%)	2943*
Ta08	3732 (29.36%)	3289 (14.00%)	3296.6 (14.27%)	**3276 (13.55%)**	3300.3 (14.40%)	3350(16.12%)	3352.9(16.22%)	2885*
Ta09	3381 (27.34%)	3131 (17.93%)	3136.8 (18.15%)	**3129 (17.85%)**	3129.0 (17.85%)	3151(18.68%)	3174.4(19.56%)	2655*
Ta10	3352 (23.10%)	**3035 (11.46%)**	3045.5 (11.84%)	3065 (12.56%)	3067.7 (12.66%)	3076(12.96%)	3078.7(13.06%)	2723*
50x20								
Ta11	3654 (27.40%)	**3447 (20.18%)**	3464.6 (20.78%)	3472 (21.04%)	3474.5 (21.11%)	3534(23.22%)	3551.2(23.82%)	2868*
Ta12	3722 (29.73%)	3419 (19.16%)	3461.7 (20.65%)	**3408 (18.78%)**	3411.7 (18.92%)	3544(23.53%)	3555.3(23.92%)	2869*
Ta13	3536 (28.32%)	**3143 (14.08%)**	3172.7 (15.17%)	3147 (14.23%)	3165.5 (14.88%)	3253(18.08%)	3257.2(18.23%)	2755*
Ta14	3631 (34.39%)	3143 (16.32%)	3149.1 (16.54%)	**3132 (15.91%)**	3135.2 (16.03%)	3239(19.87%)	3239.8(19.90%)	2702*
Ta15	3359 (23.28%)	3233 (18.67%)	3242.0 (18.98%)	**3186 (16.92%)**	3193.3 (17.19%)	3325(22.02%)	3338.4(22.51%)	2725*
Ta16	3555 (24.96%)	3296 (15.85%)	3304.1 (16.14%)	**3270 (14.94%)**	3275.5 (15.13%)	3374(18.59%)	3393.2(19.27%)	2845*
Ta17	3567 (26.27%)	3423 (21.17%)	3428.3 (21.36%)	**3384 (19.79%)**	3403.6 (20.48%)	3463(22.58%)	3490.4(23.55%)	2825*
Ta18	3680 (32.18%)	**3168 (13.79%)**	3168.0 (13.79%)	3178 (14.15%)	3196.9 (14.83%)	3300(18.53%)	3304.7(18.70%)	2784*
Ta19	3592 (16.97%)	3482 (13.38%)	3494.7 (13.80%)	**3480 (13.32%)**	3505.8 (14.16%)	3559(15.89%)	3560.5(15.94%)	3071*
Ta20	3643 (21.64%)	3551(18.56%)	3601 (20.23%)	**3482 (16.26%)**	3490.8 (16.55%)	3596(20.07%)	3599.7(20.19%)	2995*
100x20								
Ta21	6452 (18.08%)	6049 (10.71%)	6057.0 (10.85%)	**6023 (10.23%)**	6036.0 (10.47%)	6064(10.98%)	6084.2(11.35%)	5464*
Ta22	5695 (9.92%)	5609 (8.26%)	5615.0 (8.38%)	**5540 (6.93%)**	5540.6 (6.94%)	5667(9.38%)	5698.1(9.98%)	5181*
Ta23	6462 (16.06%)	6148 (10.42%)	6168.4 (10.78%)	**6109 (9.72%)**	6109.0 (9.72%)	6139(10.26%)	6139.0(10.26%)	5568*
Ta24	5885 (10.23%)	**5717 (7.08%)**	5731.0 (7.34%)	5749 (7.68%)	5749.0 (7.68%)	5775(8.17%)	5780.0(8.26%)	5339*
Ta25	6355 (17.86%)	6167 (14.37%)	6174.4 (14.51%)	6247 (15.86%)	6276.8 (16.41%)	**6137(13.82%)**	6177.1(14.56%)	5392*
Ta26	6135 (14.84%)	**5890 (10.26%)**	5890.0 (10.26%)	5905 (10.54%)	5932.0 (11.04%)	5909(10.61%)	6019.2(12.68%)	5342*
Ta27	6056 (11.41%)	**5768 (6.11%)**	5791.2 (6.53%)	5777 (6.27%)	5799.2 (6.68%)	5824(7.14%)	5879.0(8.15%)	5436*
Ta28	6101 (13.11%)	5924 (9.83%)	5926.0 (9.86%)	5984 (10.94%)	5985.4 (10.96%)	**5920(9.75%)**	5958.0(10.46%)	5394*
Ta29	5943 (10.92%)	5782 (7.91%)	5782.0 (7.91%)	**5738 (7.09%)**	5747.4 (7.27%)	5839(8.98%)	5850.0(9.18%)	5358*
Ta30	5892 (13.68%)	**5692 (9.82%)**	5705.2 (10.08%)	5695 (9.88%)	5705.0 (10.07%)	5707(10.11%)	5717.0(10.30%)	5183*

Table 2: Results on DMU's Benchmark (Part I). Ours - RSGC is the result of the Algorithm 1. In Ours-GS instead of randomly selecting second probable action in line 14 of the Algorithm 1 (RSGC) we randomly sample an action according to the probability distribution recommended by the policy network $\pi(a_t|s_t)$. We repeat each experiment 10 times due to the algorithm's randomness and record the minimum and the average makespan across the 10 experiment. The "Samp (min)" and the "Samp (avg)" columns are the results of the usual sampling method. The "UB" column represents the best-known solutions from literature, with "*" indicating optimal solutions.

Instance	L2D	Ours-GS (min)	Ours-GS (avg)	Ours-RSGC (min)	Ours-RSGC (avg)	Samp (min)	Samp (avg)	UB
20x15								
Dmu01	3323 (29.65%)	**3132 (22.20%)**	3132.0 (22.20%)	3198 (24.78%)	3204.7 (25.04%)	3218(25.56%)	3270.4(27.60%)	2563
Dmu02	3630 (34.15%)	3269 (20.81%)	3302.5 (22.04%)	**3266 (20.69%)**	3270.8 (20.87%)	3360(24.17%)	3363.8(24.31%)	2706
Dmu03	3660 (34.02%)	**3287 (20.36%)**	3287.0 (20.36%)	3375 (23.58%)	3384.4 (23.93%)	3392(24.20%)	3429.9(25.59%)	2731*
Dmu04	3816 (42.97%)	**3171 (18.81%)**	3192.9 (19.63%)	3261 (22.18%)	3267.4 (22.42%)	3229(20.98%)	3286.7(23.14%)	2669
Dmu05	3897 (41.76%)	**3377 (22.84%)**	3392.0 (23.39%)	3394 (23.46%)	3394.0 (23.46%)	3465(26.05%)	3491.5(27.01%)	2749*
Dmu41	4316 (32.88%)	4070 (25.31%)	4085.2 (25.78%)	**4050 (24.69%)**	4070.0 (25.31%)	4250(30.85%)	4256.8(31.06%)	3248
Dmu42	4858 (43.30%)	**4493 (32.54%)**	4493.0 (32.54%)	4526 (33.51%)	4541.3 (33.96%)	4650(37.17%)	4654.6(37.30%)	3390
Dmu43	4887 (42.02%)	**4373 (27.09%)**	4373.0 (27.09%)	4445 (29.18%)	4455.1 (29.47%)	4531(31.68%)	4558.4(32.47%)	3441

Continued on next page

Table 2 continued from previous page

Instance	L2D	Ours-GS (min)	Ours-GS (avg)	Ours-RSGC (min)	Ours-RSGC (avg)	Samp (min)	Samp (avg)	UB
Dmu44	5151 (47.68%)	**4592 (31.65%)**	4604.9 (32.02%)	4637 (32.94%)	4637.0 (32.94%)	4692(34.52%)	4807.5(37.83%)	3488
Dmu45	4615 (41.05%)	**4377 (33.77%)**	4387.4 (34.09%)	4387 (34.08%)	4399.6 (34.46%)	4454(36.12%)	4472.4(36.69%)	3272
20x20								
Dmu06	4358 (34.34%)	3880 (19.61%)	3890.2 (19.92%)	**3844 (18.50%)**	3876.0 (19.48%)	3977(22.60%)	4011.5(23.66%)	3244
Dmu07	3671 (20.51%)	3621 (18.88%)	3640.3 (19.51%)	**3621 (18.88%)**	3637.8 (19.43%)	3733(22.55%)	3750.2(23.12%)	3046
Dmu08	4048 (26.98%)	**3726 (16.88%)**	3760.2 (17.95%)	3730 (17.00%)	3755.1 (17.79%)	3882(21.77%)	3946.3(23.79%)	3188
Dmu09	4482 (44.95%)	3790 (22.57%)	3817.8 (23.47%)	**3695 (19.50%)**	3695.0 (19.50%)	3893(25.91%)	3925.0(26.94%)	3092
Dmu10	4021 (34.75%)	**3494 (17.09%)**	3551.9 (19.03%)	3519 (17.93%)	3550.2 (18.97%)	3578(19.91%)	3632.5(21.73%)	2984
Dmu46	5876 (45.63%)	5066 (25.55%)	5066.0 (25.55%)	**5063 (25.48%)**	5085.5 (26.03%)	5310(31.60%)	5314.3(31.71%)	4035
Dmu47	5771 (46.51%)	4930 (25.16%)	4965.7 (26.06%)	**4872 (23.69%)**	4962.5 (25.98%)	5177(31.43%)	5227.6(32.71%)	3939
Dmu48	5034 (33.78%)	4725 (25.56%)	4741.0 (25.99%)	**4640 (23.31%)**	4667.0 (24.02%)	4891(29.98%)	4906.5(30.39%)	3763
Dmu49	5470 (47.44%)	4671 (25.90%)	4683.3 (26.23%)	**4731 (27.52%)**	4738.8 (27.73%)	4952(33.48%)	4984.1(34.34%)	3710
Dmu50	5314 (42.50%)	4862 (30.38%)	4880.1 (30.87%)	**4837 (29.71%)**	4845.5 (29.94%)	5052(35.48%)	5088.6(36.46%)	3729
30x15								
Dmu11	4435 (29.30%)	**4183 (21.95%)**	4194.5 (22.29%)	4232 (23.38%)	4236.5 (23.51%)	4282(24.84%)	4285.3(24.94%)	3430
Dmu12	4864 (39.17%)	4247 (21.52%)	4260.3 (21.90%)	**4247 (21.52%)**	4254.5 (21.73%)	4414(26.29%)	4431.4(26.79%)	3495
Dmu13	4918 (33.60%)	4486 (21.87%)	4515.2 (22.66%)	**4457 (21.08%)**	4478.5 (21.67%)	4613(25.32%)	4641.5(26.09%)	3681*
Dmu14	4130 (21.69%)	3969 (16.94%)	3977.7 (17.20%)	**3930 (15.79%)**	3933.5 (15.90%)	4085(20.36%)	4097.5(20.73%)	3394*
Dmu15	4392 (31.38%)	**4054 (21.27%)**	4064.0 (21.57%)	4072 (21.81%)	4076.4 (21.94%)	4114(23.06%)	4138.1(23.78%)	3343*
Dmu51	6241 (49.77%)	**5919 (42.04%)**	5951.7 (42.83%)	6090 (46.15%)	6095.2 (46.27%)	6093(46.22%)	6128.1(47.06%)	4167
Dmu52	6714 (55.74%)	**6032 (39.92%)**	6093.2 (41.34%)	6081 (41.06%)	6172.6 (43.18%)	6235(44.63%)	6266.0(45.35%)	4311
Dmu53	6724 (53.03%)	**6010 (36.78%)**	6096.0 (38.73%)	6163 (40.26%)	6187.8 (40.82%)	6287(43.08%)	6308.5(43.57%)	4394
Dmu54	6522 (49.52%)	**6034 (38.33%)**	6050.8 (38.72%)	6072 (39.20%)	6072.0 (39.20%)	6201(42.16%)	6234.9(42.94%)	4362
Dmu55	6639 (55.44%)	**5917 (38.54%)**	5917.0 (38.54%)	5931 (38.87%)	5945.2 (39.20%)	6031(41.21%)	6042.4(41.48%)	4271
30x20								
Dmu16	4593 (32.04%)	**4462 (18.95%)**	4656.7 (24.15%)	4696 (25.19%)	4713.5 (25.66%)	4773(27.24%)	4801.6(28.00%)	3751
Dmu17	5379 (41.03%)	4747 (24.46%)	4751.0 (24.57%)	**4710 (23.49%)**	4736.9 (24.20%)	4907(28.65%)	4930.0(29.26%)	3814
Dmu18	5100 (32.67%)	4639 (20.68%)	4651.8 (21.01%)	**4546 (18.26%)**	4564.4 (18.74%)	4804(24.97%)	4834.9(25.77%)	3844*
Dmu19	4889 (29.75%)	4659 (23.65%)	4668.5 (23.90%)	**4651 (23.43%)**	4678.5 (24.16%)	4780(26.85%)	4792.8(27.19%)	3768
Dmu20	4859 (30.97%)	4442 (19.73%)	4489.7 (21.02%)	**4388 (18.27%)**	4441.5 (19.72%)	4554(22.75%)	4628.2(24.75%)	3710
Dmu56	7328 (48.31%)	**6784 (37.30%)**	6784.0 (37.30%)	6861 (38.86%)	6861.7 (38.87%)	6924(40.13%)	6998.7(41.65%)	4941
Dmu57	6704 (44.02%)	**6421 (37.94%)**	6435.0 (38.24%)	6428 (38.09%)	6509.0 (39.83%)	6625(42.32%)	6638.0(42.60%)	4655
Dmu58	6721 (42.76%)	**6322 (34.28%)**	6353.7 (34.96%)	6398 (35.90%)	6427.9 (36.53%)	6564(39.42%)	6584.0(39.85%)	4708
Dmu59	7109 (53.74%)	**6388 (38.15%)**	6420.2 (38.85%)	6486 (40.27%)	6492.4 (40.41%)	6398(38.37%)	6579.3(42.29%)	4624
Dmu60	6632 (39.47%)	**6400 (34.60%)**	6471.4 (36.10%)	6459 (35.84%)	6465.4 (35.97%)	6631(39.45%)	6665.5(40.18%)	4755
40x15								
Dmu21	5317 (21.42%)	**5034 (14.91%)**	5069.7 (15.75%)	5072 (15.82%)	5077.8 (15.95%)	5162(17.85%)	5235.3(19.53%)	4380*
Dmu22	5534 (17.12%)	**5293 (12.03%)**	5360.3 (13.45%)	5316 (12.51%)	5331.9 (12.85%)	5432(14.96%)	5475.5(15.88%)	4725*
Dmu23	5620 (20.41%)	**5171 (10.78%)**	5195.8 (11.30%)	5178 (10.92%)	5179.4 (10.94%)	5291(13.35%)	5303.3(13.61%)	4668*
Dmu24	5753 (23.77%)	**5138 (10.54%)**	5138.0 (10.54%)	5224 (12.41%)	5236.5 (12.68%)	5324(14.54%)	5353.5(15.18%)	4648*
Dmu25	4775 (14.66%)	4659 (11.89%)	4662.2 (11.97%)	**4618 (10.90%)**	4618.0 (10.90%)	4755(14.19%)	4756.4(14.23%)	4164*
Dmu61	8203 (58.62%)	**7634 (47.58%)**	7685.5 (48.54%)	7818 (51.14%)	7829.2 (51.37%)	7802(50.85%)	7829.1(51.37%)	5172
Dmu62	8091 (53.66%)	**7528 (42.96%)**	7561.4 (43.59%)	7810 (48.32%)	7810.0 (48.32%)	7768(47.54%)	7799.9(48.15%)	5265
Dmu63	8031 (50.76%)	**7543 (41.66%)**	7597.2 (42.69%)	7610 (42.92%)	7621.2 (43.15%)	7636(43.37%)	7744.8(45.41%)	5326
Dmu64	7738 (47.39%)	**7628 (45.20%)**	7670.0 (46.10%)	7757 (47.66%)	7773.5 (47.97%)	7717(46.99%)	7797.6(48.53%)	5250
Dmu65	7577 (46.07%)	**7345 (41.58%)**	7370.5 (42.00%)	7648 (47.40%)	7648.0 (47.40%)	7583(46.11%)	7584.0(46.13%)	5190
40x20								
Dmu26	5946 (27.94%)	5646 (21.51%)	5662.9 (21.86%)	**5583 (20.14%)**	5592.2 (20.34%)	5769(24.14%)	5823.2(25.31%)	4647*
Dmu27	6418 (32.38%)	5874 (21.14%)	5902.2 (21.74%)	**5849 (20.63%)**	5851.7 (20.68%)	6014(24.05%)	6046.4(24.72%)	4848*
Dmu28	5986 (27.57%)	5610 (19.56%)	5622.8 (19.81%)	**5604 (19.43%)**	5609.8 (19.54%)	5831(24.28%)	5848.5(24.65%)	4692*
Dmu29	6051 (29.00%)	5776 (23.09%)	5776.0 (23.09%)	**5685 (21.14%)**	5685.0 (21.14%)	5928(26.37%)	5942.8(26.69%)	4691*
Dmu30	5988 (26.58%)	**5584 (18.01%)**	5676.0 (19.95%)	5718 (20.85%)	5718.0 (20.85%)	5791(22.38%)	5847.2(23.57%)	4732*
Dmu66	8475 (48.24%)	**8069 (41.14%)**	8106.8 (41.80%)	8099 (41.67%)	8129.0 (42.19%)	8224(43.85%)	8265.6(44.57%)	5717
Dmu67	8832 (51.94%)	**8227 (41.53%)**	8259.6 (42.09%)	8252 (41.96%)	8318.0 (43.09%)	8361(43.83%)	8377.3(44.11%)	5813
Dmu68	8693 (50.58%)	**8198 (42.01%)**	8259.6 (43.07%)	8364 (44.88%)	8432.7 (46.07%)	8348(44.60%)	8348.0(44.60%)	5773
Dmu69	8634 (51.23%)	**8107 (42.00%)**	8159.4 (42.92%)	8202 (43.67%)	8228.3 (44.13%)	8209(43.79%)	8240.7(44.34%)	5709
Dmu70	8735 (48.33%)	8341 (41.64%)	8375.2 (42.22%)	**8230 (39.75%)**	8244.0 (39.99%)	8416(42.91%)	8531.2(44.86%)	5889
50x15								
Dmu31	7156 (26.88%)	**6400 (13.48%)**	6423.2 (13.89%)	6512 (15.48%)	6524.3 (15.70%)	6552(16.17%)	6603.6(17.09%)	5640*
Dmu32	6506 (9.76%)	**6025 (1.65%)**	6032.9 (1.78%)	6168 (4.06%)	6175.2 (4.18%)	6133(3.48%)	6137.6(3.55%)	5927*
Dmu33	6192 (8.08%)	6001 (4.75%)	6016.9 (4.97%)	**5898 (2.96%)**	5929.1 (3.51%)	6015(5.01%)	6117.5(6.80%)	5728*
Dmu34	6257 (16.18%)	5948 (10.47%)	5954.1 (10.57%)	**5868 (8.95%)**	5892.4 (9.41%)	6133(13.89%)	6169.5(14.57%)	5385*
Dmu35	6302 (11.83%)	6012 (6.70%)	6021.3 (6.85%)	**6012 (6.70%)**	6015.2 (6.74%)	6084(7.97%)	6129.8(8.78%)	5635*
Dmu71	9797 (57.24%)	**9440 (51.47%)**	9461.6 (51.81%)	9521 (52.78%)	9526.9 (52.88%)	9571(53.55%)	9585.6(53.79%)	6233
Dmu72	9926 (53.11%)	**9681 (49.33%)**	9681.0 (49.33%)	9650 (48.85%)	9670.9 (49.17%)	9688(49.44%)	9710.2(49.78%)	6483

Continued on next page

Table 2 continued from previous page

Instance	L2D	Ours-GS (min)	Ours-GS (avg)	Ours-RSGC (min)	Ours-RSGC (avg)	Samp (min)	Samp (avg)	UB
Dmu73	9933 (61.17%)	**9298 (50.87%)**	9365.8 (51.97%)	9545 (54.88%)	9545.0 (54.88%)	9447(53.29%)	9469.5(53.65%)	6163
Dmu74	9833 (58.09%)	**9440 (51.77%)**	9461.0 (52.11%)	9607 (54.45%)	9618.0 (54.63%)	9560(53.70%)	9581.0(54.04%)	6220
Dmu75	9892 (59.63%)	**9287 (49.86%)**	9300.9 (50.09%)	9299 (50.06%)	9307.0 (50.19%)	9397(51.64%)	9448.0(52.46%)	6197
50x20								
Dmu36	7470 (32.89%)	**6665 (18.57%)**	6745.3 (20.00%)	6787 (20.74%)	6790.5 (20.81%)	6825(21.42%)	6840.6(21.70%)	5621*
Dmu37	7296 (24.70%)	**6843 (16.95%)**	6873.0 (17.47%)	6866 (17.35%)	6897.9 (17.89%)	6895(17.84%)	6927.4(18.40%)	5851*
Dmu38	7410 (29.70%)	**6968 (21.97%)**	7012.0 (22.74%)	7136 (24.91%)	7149.7 (25.15%)	7075(23.84%)	7079.8(23.92%)	5713*
Dmu39	6827 (18.79%)	6580 (14.49%)	6601.6 (14.87%)	**6434 (11.95%)**	6443.8 (12.12%)	6654(15.78%)	6706.7(16.70%)	5747*
Dmu40	7325 (31.34%)	**6718 (20.46%)**	6771.7 (21.42%)	6773 (21.45%)	6776.1 (21.50%)	6835(22.56%)	6841.6(22.68%)	5577*
Dmu76	**9698 (42.35%)**	9823 (44.18%)	9851.4 (44.60%)	9991 (46.65%)	9997.3 (46.74%)	9922(45.63%)	9963.5(46.24%)	6813
Dmu77	10693 (56.74%)	**9930 (45.56%)**	9940.8 (45.72%)	9948 (45.82%)	9974.4 (46.21%)	10070(47.61%)	10070.0(47.61%)	6822
Dmu78	9986 (47.50%)	**9940 (46.82%)**	10013.6 (47.91%)	9990 (47.56%)	10029.2 (48.14%)	10126(49.57%)	10151.5(49.95%)	6770
Dmu79	10936 (56.90%)	10303 (47.82%)	10310.6 (47.93%)	**10181 (46.07%)**	10181.0 (46.07%)	10457(50.03%)	10486.5(50.45%)	6970
Dmu80	9875 (47.70%)	**9674 (44.69%)**	9742.9 (45.73%)	9859 (47.46%)	9874.8 (47.69%)	9732(45.56%)	9803.5(46.3%)	6686

6 Conclusion

In this study, we present RSGC and GS, algorithms designed to optimize JSSP through Policy network-guided search. By incorporating second greedy choices and policy network sampling methods with decreasing probabilities RSGC offers a balance between exploration and exploitation in the search space, thereby mitigating the risk of finding solutions having larger makespan. Our systematic approach to adjusting the probability of selecting alternative paths enables efficient exploration of the search tree, leading to improved makespan. Through extensive experimentation, we establish optimal parameters for RSGC and GS, enhancing its performance across two benchmark datasets. We also compare the minimum makespans and the average makespans using the two algorithms. The experiments showed that the GS algorithm results were slightly better on most instances, the reason perhaps being that it explored more. Overall, RSGC presents a promising avenue for enhancing scheduling algorithms, showcasing the potential of Policy networks in addressing complex optimization challenges.

References

1. J. Christopher Beck, T. K. Feng, and Jean-Paul Watson. Combining constraint programming and local search for job-shop scheduling. *INFORMS Journal on Computing*, 23(1):1–14, 2010.
2. Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning, 2017.
3. Giovanni Bonetta, Davide Zago, Rossella Cancelliere, and Andrea Grosso. Job shop scheduling via deep reinforcement learning: a sequence to sequence approach. *Not Specified*, Aug 2023.
4. Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.
5. Tristan Cazenave. Nested Monte-Carlo search. In *Proceedings of the IJCAI International Joint Conference on Artificial Intelligence*, pages 456–461, 2009.
6. Ceren Cebi, Enes Atac, and Ozgur Koray Sahingoz. Job shop scheduling problem and solution algorithms: A review. In *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pages 1–7, 2020.

7. Ebru Demirkol, Sanjay Mehta, and Reha Uzsoy. Benchmarks for shop scheduling problems. *European Journal of Operational Research*, 109(1):137–141, 1998.
8. Rupert Ettrich, Marc Huber, and Günther Raidl. *A Policy-Based Learning Beam Search for Combinatorial Optimization*, pages 130–145. Springer, 03 2023.
9. Kaizhou Gao, Zhiguang Cao, Le Zhang, Zhenghua Chen, Yuyan Han, and Quanke Pan. A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems. *IEEE/CAA Journal of Automatica Sinica*, 6(4):904, 2019.
10. Michael R Garey, David S Johnson, and Ravi Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129, 1976.
11. William D Harvey and Matthew L Ginsberg. Limited discrepancy search. In *IJCAI*, pages 607–615, 1995.
12. Chun-Cheng Lin, Der-Jiunn Deng, Yen-Ling Chih, and Hsin-Ting Chiu. Smart manufacturing scheduling with edge computing using multiclass deep q network. *IEEE Transactions on Industrial Informatics*, 15(7):4276–4284, 2019.
13. Eugeniusz Nowicki and Czeslaw Smutnicki. An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling*, 8(2):145–159, 2005.
14. John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.
15. Eric Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.
16. Erdong Yuan, Shuli Cheng, Liejun Wang, Shiji Song, and Fang Wu. Solving job shop scheduling problems via deep reinforcement learning. *Applied Soft Computing*, 143:110436, 2023.
17. Mohamed Habib Zahmani, Baghdad Atmani, Abdelghani Bekrar, and Nassima Aissani. Multiple priority dispatching rules for the job shop scheduling problem. In *3rd International Conference on Control, Engineering Information Technology (CEIT'2015)*, Tlemcen, Algeria, 2015.
18. Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Chi Xu. Learning to dispatch for job shop scheduling via deep reinforcement learning. In *34th Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

Binarized Monte Carlo Search for Selection Problems

Matthieu Ardon[1], Yann Briheche[1], and Tristan Cazenave[2]

[1] Thales Research & Technology - Palaiseau, France
[2] LAMSADE, Université Paris Dauphine, PSL, CNRS - Paris, France

Abstract. In this paper, we consider adaptations of Monte Carlo Search methods on binary decision trees where actions are simulated using heuristics and where choices are made deterministically or stochastically. We explain how these adaptations are fitted for combinatorial problems such as element selection problems in order to compete with other approximate resolution methods such as metaheuristics. We present results on a theoretical problem (Set Covering) and on an applied problem (Pulse Repetition Frequency Selection) with different simulation heuristics. We then discuss the usefulness of these new methods based on the characteristics of the problems and on the quality of the simulation heuristics used to construct the decision tree.

Keywords: Tree Search · Monte Carlo Search · Set Cover Problem · Radar · Decision Trees · Binary Choices · Heuristic · Simulation

After a general presentation of the selection problems, we detail the two problems considered. We then introduce Binarized Monte Carlo Search, based on Monte Carlo Search on particular decision trees. Adaptations of two different methods are developed. Finally, we present the experimental results obtained with the new methods on our selection problems and discuss their interests.

1 Selection Problems

Selection problems arise as problems in which a subset of elements must be selected from a given set, usually under given constraints, in order to optimize a given objective function.

Due to the size of the initial set of candidates and the large number of possible combinations among them, these problems often manifest as combinatorial optimization problems. Exact solution methods are therefore not always suitable for these problems. Approximate resolution methods can be an efficient alternative, finding close-to-optimal solutions rather than exact solutions to complex problems.

A greedy algorithm can quickly and iteratively produce an approximate solution to a selection problem. Its basic idea is to make a locally optimal choice

at each step, without considering the consequences of this choice on future steps, with the hope of finding a globally optimal solution. As long as the solution is not fully formed, the algorithm assigns a *greedy score* to all valid candidates based on a computationally inexpensive function. The validity of those candidates depends on constraints preventing specific combinations of candidates in some selection problems. The candidate with the highest score is then added to the solution and removed from the set of available candidates if a solution cannot be formed by the same candidate more than once. Greedy algorithms are easy to implement but can provide less than optimal solutions on certain instances.

Another alternative is to use other heuristics, metaheuristics or tree search procedures. Combinations of these different resolution methods are also increasingly being considered, this is the case in this paper.

1.1 Set Cover Problem

The Set Cover Problem (SCP) is a classic optimization problem in which the goal is to select, from a collection of subsets whose union equals the universe, the smallest number of subsets such that their union is also equal to the universe. This problem is known to be NP-hard [13], meaning that there is no known efficient algorithm to solve it optimally in polynomial time.

One variant of this problem is the Weighted Set Cover Problem, in which every set is assigned a positive weight, indicating its cost, and the objective is to identify a set cover which minimizes the sum of the costs of the chosen sets.

Formulation. The problem can be formally defined as follows:

$$\text{minimize} \sum_{j \in J} c_j x_j \tag{1}$$

$$\text{under the constraints} \sum_{j \in J} a_{ij} x_j \geq 1, \forall i \in I \tag{2}$$

$$\text{where } x_j = \begin{cases} 1 & \text{if subset } j \text{ is selected} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{and } a_{ij} = \begin{cases} 1 & \text{if item } i \text{ is covered by subset } j \\ 0 & \text{otherwise} \end{cases}$$

Simply put, each candidate $j \in J$ is a binary column vector possessing a weight c_j, and the set to cover is initially a zero vector formed of as many items/rows/cells as each candidate column vector ($|I|$).

Chvatal's Greedy Procedure. The classic greedy algorithm follows the ideas of Chvatal's algorithm [9], where the next candidate is selected based on its *effective cost*. This is the score function used to determine which candidate to add incrementally.

With I_j representing the set of still uncovered rows that column j covers and J_i representing the set of candidates which cover row i, the candidate is chosen at each step from the following formula:

$$\arg\min_{j \in J} \frac{c_j}{|I_j|} \tag{3}$$

Surprisal-Based Greedy Heuristic. The previous procedure is simple and intuitive but can be improved to assign a more relevant score to candidates based on the characteristics of all others. The Surprisal-Based Greedy Heuristic is derived from Information Theory and can yield better results when applied to the SCP than the Chvatal's procedure [1]. Using the same notation, the candidate is chosen according to:

$$\arg\min_{j \in J} \frac{c_j}{|I_j|} \prod_{i \in I_j} \frac{|J_i| - 1}{|J_i|} \tag{4}$$

Thus, a candidate column is always chosen if it is the only one covering a given row, since one of the terms in the product is zero. However, if alternative candidates exist for all the rows covered by a candidate column, the score of this column will be higher and tend toward the Chvatal's *greedy score*.

1.2 Pulse Repetition Frequency Selection Problem

A major topic in radar engineering is target detection. Pulse Doppler radars can determine the range and radial velocity of a target. They use multiples series of pulses, called bursts, to achieve detection, each requiring the selection of its Pulse Repetition Frequency (PRF, representing the number of pulses transmitted by the radar each second).

The periodicity of pulse Doppler radars cause several problems:

- *range blind zones*: the radar cannot receive a pulse when another is emitted.
- *range ambiguities*: the radar use detection delays to determine the target range, but has no way to identify the original pulse, and thus only determine range within a modulo.

A burst with a given PRF is characterized by blind zones and ambiguities on the range axis. Sending a waveform made of an ordered sequence of several bursts associated with different PRFs can solve the aforementioned problems:

- blind zones of different bursts can compensate each other if they do not overlap.

– different bursts measure range with different modulos. According to the Chinese Remainder Theorem, if the Pulse Repetition Intervals (PRIs, the inverse of PRFs) have a high enough lowest common multiple, then detection of the same target between both bursts will overlap on a unique range below the maximum radar detection range, yielding the actual range of the target. Ambiguity removal, also called *decodability*, thus requires the prohibition of some PRFs combinations [12].

In this paper, we will consider that all bursts have a constant and identical duration regardless of the PRF of each burst. This means that the number of pulses in each burst is different and deduced from its PRF and that constant duration.

Methods for resolving this problem from the literature have been tested with evolutionary algorithms [12] for the selection of PRFs for bursts with constant numbers of pulses, as well as with metaheuristics such as simulated annealing [2] for the selection of PRFs for bursts with constant emission durations.

We use an M of N scheme: we select a set of N bursts and target detection is achieve if M bursts can detect a given target. In this paper, all the tests have been performed with the same scheme. We use a recursive approach to compute the overall probability when adding a new burst [4]:

$$P(m,n) := p_n \cdot P(m-1,n-1) + (1-p_n) \cdot P(m,n-1) \tag{5}$$

with p_n the detection probability of the n-th burst, $P(m,n)$ the overall detection probability of m among n bursts, with the following initial conditions:

$$P(0,0) = 1 \text{ and } \forall m,n \geq 1, P(m,0) = P(-1,n) = 0 \tag{6}$$

2 Binarized Monte Carlo Search

In the field of operations research, tree search methods can explore the search space systematically in a deterministic or stochastic manner. Different traversal strategies can be considered, exhaustive or not, depending on the complexity and the restrictions of the problem. The Limited Discrepancy Search [11] (LDS) lends itself well to problems with a clear heuristic on the possible actions that can occur. Indeed, it finds a solution deterministically by following a heuristic while allowing a limited number of deviations from this heuristic to encourage the exploration of alternative choices. Monte Carlo Tree Search uses random sampling for part of its exploration and statistical analysis which allows it to deepen the research in the most promising parts of the tree. Originally developed for game-playing programs, especially those with vast search spaces like Go [10], Monte Carlo Tree Search has since found applications in various fields such as optimization.

2.1 Nested Monte Carlo Search

Recursive approaches on Monte Carlo Search methods seems promising, especially for single-player problems [8], whose objectives can be similar to selection problems. The addition of nested search procedures allows in some cases the intensification of the search in parts of the decision tree (Nested Monte Carlo Search), and in other cases prevent a premature convergence caused by a learning that slows down the exploration too quickly (Nested Rollout Policy Adaptation). A rollout, or playout, is a path that descends the tree with a randomized decision at each depth until reaching a leaf [14].

Nested Monte Carlo Search. The Nested Monte Carlo Search (NMCS) is a method that performs recursive improvements of playouts using nested levels [5]. At level zero, the playouts are random. At higher levels, they are guided by results obtained at lower levels.

Nested Rollout Policy Adaptation. The Nested Rollout Policy Adaptation (NRPA) is a method that learns a policy which presents itself as a weight vector influencing the probabilities of choosing each candidate [14]. NRPA has set new world records in Morpion Solitaire and Crosswords Puzzles. The nested levels are now associated to the best sequence found at this level. During the learning (called the adaptation) of the policy, the weights of candidates present in the best sequence are incremented, and the others are reduced. The learning rate α is an hyperparameter controlling the update rate of these weights. The probability to choose a child node *child* created from a parent node *parent* with the selection of a given action *action* is:

$$p_{child} = \frac{e^{policy(child_{action})}}{\sum_{brother \in \{children(parent)\}} e^{policy(brother_{action})}} \tag{7}$$

Calls to lower nested levels are limited to a given number of iterations. Nested Rollout Policy Adaptation with Limited Repetitions [7] (NRPALR) ends the playouts of a level when the best performance remains the same for a given number of times. After a lower level call, the best sequence is returned to the upper level and the upper level policy is updated then passed again to a lower level call. Calls at level 0 use stochastic playouts.

Generalized Nested Rollout Policy Adaptation [6] (GNRPA) is another way to integrate a heuristic or problem feature to guide the search in the tree. In this method, computation of probabilities for choosing a node accounts for features of the chosen node state. However this requires computing for each node its state and features.

NRPA, GNRPA or GNRPALR perform a search that is partially random and partially directed. They allow the integration of multiple heuristics during initialization phase, and during search phase. In the search phase, they guide exploration at a chosen speed (learning rate) with regular updates of the probabilities of choosing children node (adaptation of the initial policy).

2.2 Adapted Representations of Decision Trees

Decision trees can be well-suited representations for solving selection problems with the following assumptions. A node represents a solution formed by the set of candidates used in the sequence leading to that node. A branch represents the choice of a specific candidate from the available set of candidates. Each candidate is an action mapping the state of the parent node to the state of the child node, through the branch connecting both. The root of the tree is the neutral state. Regardless of the chosen representation, the tree is pruned at the end of every sequence when no candidate can comply with the problem constraints: for the SCP the tree is pruned when the state of a node is entirely covered, and for the PRF Selection Problem when either the desired number of bursts is reached or when there is no remaining burst which can comply with *decodability*.

Classic exhaustive tree search methods and some less parameterizable Monte Carlo Search methods do not always require the computation of the state of each simulated node to traverse the tree and retain the best sequences. Sometimes, only evaluating the leaves is sufficient. However, in the case of constrained problems, states must be computed each time. For example, to solve the Set Cover Problem (SCP), it is necessary to check at each state whether the covering condition has been satisfied, in order to stop the search, return the result corresponding to the traversed sequence, and prune the tree.

Thus, reducing the number of explored states tends to reduce the overall computational complexity. A tree with a high branching factor requires the computation of many states at each step. Long sequences will also increase the number of computed states. While the length of the sequences cannot be reduced, we can try to use more efficient representations for solving selection problems. These representations should have a low branching factor and should not introduce biases that favor certain candidates over others based solely on their initial ordering or on an arbitrary position in the tree.

Binary Decision Trees. All branches leading to a parent node's children correspond to a specific action and its associated candidate. The parent node *node* has two children: a first node ($child(node, 0)$ in the Algorithms of this paper) where the candidate is rejected (with the same state as the parent node), and a second ($child(node, 1)$) where the candidate is added to the solution (the state is thus the parent node state with the addition of the candidate).

Simulation Heuristics. The simulation of the next nodes takes into account the construction of the current sequence, and greatly impact the quality of research in binary trees.

Nodes can be simulated according to a heuristic which will determine an interesting action to choose among all available actions. Random simulations/moves can quickly produce solutions. Greedy simulations are likely to yield better solutions.

In a binary decision tree, a traversal ignoring each action would be possible to favor the exploration of actions that are locally less interesting, but may be globally interesting. Simulations can also be based on domain-specific rule-based heuristics in order to intensify the quality of returned candidates. Some heuristics can perform better than another depending on the problem.

In the simulation function, the heuristic assigns a score to each possible action, in the same ways as the previously cited *greedy score*. The chosen action is the one with the best score. If multiple actions achieve the same best score, one can selected either through a predefined order or either randomly. As we will see, the latter can introduce a stochastic aspect to a deterministic method.

The binary tree can be constructed with an initial ordering of the actions. However, this will significantly decrease the effectiveness of the heuristic and thus the quality of the search. Despite the need for greater computing resources, we therefore favored a call to the heuristic each time a node is traversed for the first time.

2.3 Dynamic Binarized Nested Monte Carlo Search (DBNMCS)

We adapted NMCS to perform the search partly on the binary decision tree, as presented above, and partly through an iterative heuristic used to finalize the solutions, see Algorithm 5.

The search occurs in two steps, each based on a heuristic which finds the best action to consider in the rest of the current sequence, see Algorithm 4. In our experiments, we used the same heuristic for both steps, but different heuristics could be used for each step.

In the first step, we search through a nested level on a binary tree (starting from the root), where each node, both simulated by the first heuristic, represents a choice to add a candidate to the solution or not. For each child node of a given parent node, the second step will perform a deterministic playout by completing the solution using the second step heuristic. We select the node with the best solution among the two children. The first step starts again from that node.

Recursively, the two steps are chained. Once we reach a zero nested level, we perform two deterministic playouts. We store the node with the best solution among the two as a starting point for an in-depth search (binary search then playout), to try to find a better solution from that starting point.

Since each of the 2 branches is always traversed in the first step, and solutions are obtained using in the second step using an deterministic heuristic, there is no randomness in the algorithm, which is therefore totally deterministic.
Furthermore, after reaching the zero nested level, the playout are performed until a leaf and then, the first step on a binary tree takes over from the node corresponding to the best playout. The tree is therefore dynamic.

To summarize, the Dynamic Binarized NMCS involves a two-step process. In the first step, a heuristic-based search is performed on the binary tree, exploring

different sequences and forming partial solutions. In the second step, in the form of local searches, playouts are conducted at the lowest nested levels of the tree to finalize these partial solutions. The performances of the resulting solutions are compared. The process repeats recursively until a leaf node is reached, potentially refining the solution by considering variations of the initial choices. The entire process is deterministic and dynamic. Ultimately, DBNMCS allows for a non-exhaustive search of the decision tree guided by the most interesting choices to focus on the parts of the tree that appear most promising. It does not need a lot of hyperparameter tuning or precise information about the problem to initially guide the search with complex heuristic.

2.4 Binarized Nested Rollout Policy Adaptation (BNRPA)

Another interesting possibility for exploring binary decision trees with heuristic-based simulations are NRPA methods.

Unlike NMCS methods, playouts here always start at the root node, see Algorithm 2. Each node in the binary tree representing whether a candidate is added to the solution or not. We also use heuristic-based simulations to select the available action with the best score, see Algorithm 4.

A policy vector is initialized before the start of the search, as is done in classic NRPA. However, since the representation is now a binary tree where half of the branches represent the choice to ignore a candidate, the encoding of the policy needs to be modified. Thus, each candidate is now linked to a binary policy. One component represents the choice to select it and the other represents the choice not to select it. Before the start of the search, if the policy is initialized uniformly, its values no longer depend on the number of candidates: for each candidate, it is set to $1/2$ for the component representing the choice to select it and similarly for the one representing the choice not to select it.

Like in classic NRPA, we copy the policy from higher levels to lower levels, and we adapt the policy by updating its values after each playout. This improves the probabilities of the choices which might be proposed again in subsequent playouts on the same nested level. In a binary tree, the best solution found so far is made of choices of some actions and refusal of other actions. Depending on the addition or refusal of each action *action* in the best found solution among all playouts of a given nested level, we will increment its associated policy for selection ($policy[(action, 1)]$) or its associated policy for refusal ($policy[(action, 0)]$) and decrement both policies, see Algorithm 3, to then keep a sum of probabilities equal to 1 in the playouts. The learning rate α controls the speed and intensity of the update of the policy. Note that at the end of any nested level, the policy is not kept, and does not become the initial policy of the higher level. Otherwise, it would prevent exploration due to adaptation occurring in the same way to favor the same choices. Instead, only the best solution is returned to the higher nested level, and the policy of the higher level is adapted based on the best solution found in the lower level.

The number of calls to the lower nested level per level is set in our first implementation of Binarized NRPA (BNRPA). Another approach is to implement a stop criteria on this level based on the number of times the exact best performance is returned, hinting that further improvement is unlikely. This is done in BNRPA with Limited Repetitions (BNRPALR), see Algorithm 1.

We can construct the tree statistically or dynamically. In the first case, the root is always the same. The tree is built gradually as the playouts progress. Playouts increasingly pass through existing branches and nodes as the probabilities of traversing those branches increase. These branches become mandatory paths at the top of the tree since all available paths have already been simulated. In the second case, the root is reset with each new playout. The child nodes must be simulated again at each depth until the playout stops at a leaf. Thereby, the sequences are dynamic and can differ between playouts when multiple actions obtain the same heuristic score, since we do not always choose the same action. This dynamic construction can favor exploration and diversification. However, reaching the repetition limit on the best performance can become harder, since finding the same solution twice becomes unlikely and some problems will rarely have the same performance for two different solutions.

Empirical evidences indicates that BNRPALR performs better than BNRPA on static binary trees, and that BNRPALR performs better on static trees rather than dynamic ones, for the same computation time. The variance observed on different runs with BNRPALR is lower than the variance of BNRPA.

BNRPA with heuristic-based simulations uses a heuristic to choose the actions of the following nodes. Classic NRPA allows for the integration of another heuristic or a prior in policy initialization. This is more difficult to do with BNRPA due to the binary policy. GNRPA also allows the integration of additional information to guide the search by modifying the node choice probabilities based on a weighted prior. With BGNRPA, it might be interesting to construct the prior based on the average of a statistic across all remaining candidates to be considered for the node that does not add a candidate. For the other node, it would be the statistic specific to that candidate. Yet, with BGNRPA, actions are not simulated in the same order in each sequence and biases then cannot be precomputed. Nevertheless, this step is often the most time-consuming with GNRPA. However, this method still represents a promising approach for improving BNRPA because it allows for the integration of additional information to guide the search.

3 Experimental Results

We compare new Binarized Monte Carlo Search methods (DBNMCS and BNRPALR) with Limited Discrepancy Search (LDS) using the same simulations heuristics and a discrepancy parameter set to 7 (SCP) or 8 (PRF Selection Problem).

3.1 Weighted Set Cover Problem

OR-library provides datasets of problem instances for benchmarking combinatorial optimization, including the Weighted Set Cover Problem [3].

Each problem instance is characterized by the number of columns candidates (or variables) $|J|$, the number of cells to cover (or constraints) $|I|$, and the density of the covered cells by the candidates. A weight has been assigned to each candidate, and the objective is to minimize the sum of the weights of a subset of chosen candidates.

We perform tests on OR-Library instance sets "*4*" to "*NRH*", with the number of instances in each set indicated in Table 1 under the column |Set|.

We used DBNMCS and BNRPALR with both Chvatal's Greedy Procedure (CGP) and Suprisal-Based Greedy Heuristic (SBH) as simulation heuristics. Each had 25 runs, under the same computation time limit. For BNRPALR, we set the learning rate at 0.75, the nested level at 9 and the repetition limit at 5, with a uniform initial policy. For DBNMCS, we set the nested level at 4. All the results are displayed and compared to the optimal or best known score (BS) on Table 3. The stars indicates that at least one run has reached the best performance (BS) with the corresponding method and simulation heuristic. The GAP compares the average of the obtained performances from the 25 runs (AVG) to the BS, such as:

$$GAP = 100 \times \frac{AVG - BS}{BS} \tag{8}$$

The smaller the average gap across a set of instances, the better the method. Table 1 displays the average gap of each method over the sets of instances, and the characteristics (number of variables and constraints, density) of each set.

Table 1. Comparison of Binarized Monte Carlo Search methods with CGP heuristic on the sets of instances of the SCP from OR-library

Set	**|Set|**	**|I|**	**|J|**	**Density**	**GAP averages**	
					DBNMCS	BNRPALR
4	10	200	1000	2	4.36	**0.87**
5	10		2000		5.43	**1.25**
6	5		1000	5	4.04	**1.18**
A	5	300	3000	2	6.62	**2.14**
B	5			5	3.19	**1.19**
C	5	400	4000	2	8.56	**5.31**
D	5			5	3.45	**1.39**
E	5	50	500	20	**0**	**0**
NRE	5	500	5000	10	1.88	**1.05**
NRF	5			20	2.97	**2.05**
NRG	5	1000	10000	2	**9.91**	22.5
NRH	5			5	**7.17**	9.99

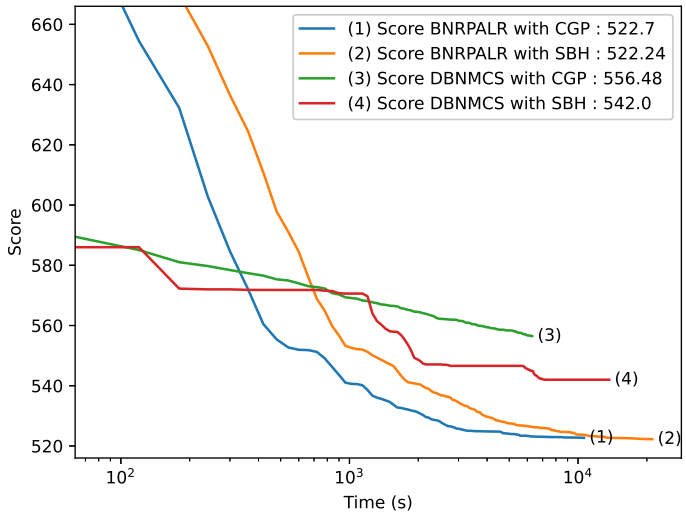


Fig. 1. Evolution of the best performance found over time for instance *scp43* from OR-Library (BS = 516)

Figure 1 highlights the performances of DBNMCS and BNRPALR methods. Each one was tested with the Chvatal's Greedy Procedure and the Surprisal-Based Heuristic as simulation heuristics, with a number of calls set at 3,000,000. 100 runs were performed for each configuration on the *scp43* instance. It is a moderately sized and low-density instance which proved to be quite discriminative in the various tests.

3.2 PRF Selection Problem

We performed tests on two different PRF Selection Problem instances, for which we aim to maximize the detection probability at a given target range. We used the same radar parameters (mean power, signal duration, carrier bandwidth...) for both instances, and only changed the detection range, with the first problem (*Instance 1*) at low range (thus being easier to optimize) and the second (*Instance 2*) at long range (thus being harder to optimize). These instances are made up of 140 candidates and the search is stopped as soon as we have added N candidates to our solution in accordance with the chosen M of N scheme.

We use both DBNMCS and BNRPALR with two different simulation heuristics: one based on a simple greedy heuristic (GH) and the other on based an initial random rating (RR). For BNRPALR, we set the learning rate at 0.75, the nested level at 9 and the repetition limit at 5, with a uniform initial policy. For

DBNMCS, we set the nested level at 8 (more important than the SCP since the sequences are made up of only 6 candidates and therefore much shorter).

Greedy Resolution. The greedy resolution of the SCP relies on additional information unrelated to candidate costs (which is the optimization criterion) such as the number of additional rows that could be covered by a candidate. We can still use a basic greedy process even when such information is not available: at each step, we compute the solutions with each possible candidate (in other words, each PRF compatible with the PRFs already chosen in the solution), and select the one which maximize our optimization criterion, the detection probability.

On *Instance 1*, the basic iterative greedy resolution of the problem gives poor results. On *Instance 2*, they are even worse.

Randomized Rating. Here, the problem lacks additional information that can be used to compute specific priorities for each candidate (such as CGP or SBH). But there are other ways to determine the interest of some candidates over others. Before the optimization, we can assign an interest score to each candidate based on the average performance of randomly sampled solutions that contained that candidate. Those interest scores will help favor certain candidates over others when they achieved the same score for a simulation-based heuristic. It is particularly suitable here because the candidates are not extremely numerous (140). The initial random scoring RR of each candidate $c \in C$ was computed over 3,000 random simulations, by normalizing the sum of the scores they obtained during the runs in which they were selected in the waveform, by this number of runs, as:

$$RR_{c\_norm} = \frac{\frac{\sum score(r)}{\frac{r \in \{run|c \in solution\}}{|\{run|c \in solution\}|}} - \min_{c \in C} RR_c}{\max_{c \in C} RR_c - \min_{c \in C} RR_c} \tag{9}$$

This prior can be combined with the score of the greedy simulation heuristic to weight it according to the initial interest of the candidates (GH & RR).

The results are compared in Table 2 with those of Simulated Annealing (SA), obtained using *scipy.optimize.dual_annealing*, and the Limited Discrepancy Search (LDS).

Table 2. PRF Selection Problem results

		SA	**LDS**			**BNRPALR**			**DBNMCS**		
			GH	RR	GH & RR	GH	RR	GH & RR	GH	RR	GH & RR
Instance 1	AVG	0.913	0.961	0.943	0.960	**0.966**	0.944	0.959	0.964	0.887	0.955
	MAX	0.965	**0.972**	0.969	0.969	**0.972**	0.969	0.971	**0.972**	0.950	0.969
Instance 2	AVG	0.579	0.569	0.592	0.604	0.585	0.588	**0.605**	0.586	0.576	0.594
	MAX	0.608	0.602	**0.614**	**0.614**	0.602	**0.614**	**0.614**	0.607	0.608	**0.614**

4 Discussion

Weighted Set Cover Problem. BNRPALR seems to be more efficient than DBNMCS on most instances for the same calculation time, see Table 1.

A possible explanation is the search getting lost in the resolution of instances from the NRG and NRH sets which are very large and not very dense. Since each candidate does not cover a lot of rows, the sequences are longer before reaching the coverage condition. Escpecially for BNRPALR, whose entire search is carried out on a binary tree with an initial non-negligible probability of ignoring actions. In contrast, DBNMCS is better because playouts at nested level zero are shorter and allow testing of many small variations when constructing solutions.

Moreover, regardless of the instance, BNRPALR takes more time to find a better solution than a basic iterative greedy algorithm, whereas DBNMCS finds quickly a close solution from its first traversed sequence, see Figure 1.

However, for all more moderate instances, BNRPALR give much better results, see Table 3. It often comes close to, and sometimes even achieves, the best-known results for several instances. Generally the most efficient policy learning and the most frequently optimal results are obtained on the densest instances (sets 4, 5, 6, B, D, E, NRE, NRF).

Standard deviations in Table 3 are much larger with BNRPALR due to its stochastic nature (each branch choice depends on a probability linked to the simulated action policy). Non-zero standard deviations for DBNMCS, despite it being deterministic, are due to random choice when several candidates have the same score in the simulations. Additionally, these standard deviations are bigger with CGP heuristic than with SBH, which has a more complex formula. Thus two different solutions are less likely to have the same score.

Despite similar computational complexities, SBH takes a bit more time than CGP. Under the same computation time limit, CGP is often more efficient at it allows simulation of more nodes, and thus exploration of more solutions, see Table 3. Within the same number of calls to the simulation heurisitc, BNRPALR-SBH outperforms BNRPALR-CGP, see Figure 1.

For DBNMCS, SBH is significantly better than CGP. Indeed, the actions will be chosen more scrupulously with SBH and therefore the numerous variations around solution constructions lead to better performance. The evolution of the DBNMCS performance with CGP is slow but regular, see Figure 1. The evolution of the performance of the same method with SBH is sometimes very quickly improved thanks to the more efficient simulation heuristic.

Pulse Repetition Frequency Selection Problem. On its own, greedy iterative resolution performs poorly on the PRF Selection Problem, especially on *Instance 2*. Binarized Monte Carlo Search methods can greatly improves the results.

According to the SCP results, BNRPALR is more effective when an efficient simulation heuristic is available. This is not the case for this problem, and this explains the small difference between the performances of DBNMCS and BNRPALR, see Table 2. According to the conclusion of the original paper [5], NMCS

can be used even without a good heuristic to guide the search, which is the case with the PRF Selection Problem.

The search can also be improved with an initial randomized rating as the simulation heuristic. This approach can be suitable even without an effective greedy heuristic.

LDS also performs better because the size of the instances lends itself better to this type of search. LDS beats Simulated Annealing but is not overall better than Binarized Monte Carlo Search.

In summary, Dynamic Binarized NMCS requires little hyperparameter tuning and is therefore suitable when the knowledge of the problem is limited or when there is no existence of a good simulation heuristic that is not too computationally intensive. Binarized NRPA(LR) is suitable when a good result needs to be achieved over a slightly longer period of time, or when additional information from the data can be used to help improve this result continually. Possible improvements could come from GNRPA [6] and should primarily improve its robustness (variance).

DBNMCS and BNRPA(LR) significantly outperform the basic iterative greedy algorithm since they use it as a simulation heuristic for choosing actions while being more extensive on the exploration. DBNMCS will directly surpass this result, while BNRPA(LR) may take some time before exceeding it.

A good simulation heuristic applied to these methods significantly improves the final result. However, there is a tradeoff to find between the efficiency of the heuristic and its computational cost. Here, for an equivalent computation time, the simulation with the least effective heuristic (and consequently the fastest) will be preferred with BNRPA(LR).

The characteristics of the instances can also influence the choice of the resolution method. On instances of reasonable sizes ($<\sim$ 5000 candidates) and balanced (consistent density for the SCP), BNRPALR is very effective. On more aberrant instances, DBNMCS is more appropriate. Furthermore, LDS is worth considering on small instances.

A deterministic method with a more discriminative simulation heuristic will improve the robustness (variance) of the optimization.

5 Conclusion

In this paper, we have presented adaptations of two Monte Carlo Search methods, Dynamic Binarized Nested Monte Carlo Search (DBNMCS) and Binarized Nested Rollout Policy Adaptation (BNRPA) including its variants BNRPALR and BGNRPA. With the use of binary decision trees and heuristic-based simulations, these adaptations are suitable for solving selection problems and potentially other optimization problems . We have shown that it is the case for the Weighted Set Cover Problem, where the most relevant method managed to find the best score on certain suitable instances, and for the Pulse Repetition

Frequency Selection Problem, where we beat the results of other tested methods. In addition , we have also highlighted and explained the interest of each method depending on the characteristics and knowledge of the problem and instances to be solved and the quality of the heuristic on which the simulation is based. Further work may focus on accelerating the playouts of these methods and improving DBNMCS's performance and BNRPA's variance.

References

1. Adamo, T., Ghiani, G., Guerriero, E., Pareo, D.: A surprisal-based greedy heuristic for the set covering problem. Algorithms **16**(7) (2023). https://doi.org/10.3390/a16070321, https://www.mdpi.com/1999-4893/16/7/321
2. Ahn, S., Lee, H., Jung, B.: Medium prf set selection for pulsed doppler radars using simulated annealing. In: 2011 IEEE RadarCon (RADAR). pp. 090–094 (2011). https://doi.org/10.1109/RADAR.2011.5960505
3. Beasley, J.: An algorithm for set covering problem. European Journal of Operational Research **31**(1), 85–93 (1987). https://doi.org/https://doi.org/10.1016/0377-2217(87)90141-X, https://www.sciencedirect.com/science/article/pii/037722178790141X
4. Brunner, J.: A recursive method for calculating binary integration detection probabilities. IEEE Transactions on Aerospace and Electronic Systems **26**(6), 1034–1035 (1990). https://doi.org/10.1109/7.62256
5. Cazenave, T.: Nested monte-carlo search. In: IJCAI International Joint Conference on Artificial Intelligence. pp. 456–461 (2009)
6. Cazenave, T.: Generalized nested rollout policy adaptation. In: Monte Search at IJCAI (2020)
7. Cazenave, T.: Generalized nested rollout policy adaptation with limited repetitions. arXiv preprint arXiv:2401.10420 (2024)
8. Cazenave, T., Teytaud, F.: Application of the nested rollout policy adaptation algorithm to the traveling salesman problem with time windows. In: Learning and Intelligent Optimization - 6th International Conference, LION 6. pp. 42–54 (2012)
9. Chvatal, V.: A greedy heuristic for the set-covering problem. Mathematics of Operations Research **4**(3), 233–235 (1979), http://www.jstor.org/stable/3689577
10. Gelly, S., Silver, D.: Monte-carlo tree search and rapid action value estimation in computer go. Artif. Intell. **175**(11), 1856–1875 (2011). https://doi.org/10.1016/j.artint.2011.03.007, https://doi.org/10.1016/j.artint.2011.03.007
11. Harvey, W.D., Ginsberg, M.L.: Limited discrepancy search. In: IJCAI. pp. 607–615. Morgan Kaufmann (1995)
12. Hughes, E., Alabaster, C.: Medium prf radar prf optimisation using evolutionary algorithms. In: Proceedings of the 2003 IEEE Radar Conference (Cat. No. 03CH37474). pp. 192–197 (2003). https://doi.org/10.1109/NRC.2003.1203401
13. Karp, R.M.: Reducibility among Combinatorial Problems, pp. 85–103. Springer US, Boston, MA (1972). https://doi.org/10.1007/978-1-4684-2001-2_9, https://doi.org/10.1007/978-1-4684-2001-2_9
14. Rosin, C.D.: Nested rollout policy adaptation for Monte Carlo Tree Search. In: IJCAI. pp. 649–654 (2011)

Algorithm 1: BNRPALR(level, policy)

```
if level = 0 then
  return BNRPALR_Playout(policy)
else
  bestScore ← −∞
  repetitions ← 0
  while repetitions ≤ R do
    (score, sequenceActions) ← BNRPALR(level − 1, policy)
    if score = bestScore then
      repetitions ← repetitions + 1
    else if score > bestScore then
      repetitions ← 0
      bestScore ← score
      bestSequenceActions ← sequenceActions
    end if
    policy ← BNRPALR_Adapt(policy, bestSequenceActions)
  end while
  return (bestScore, bestSequenceActions)
end if
```

Algorithm 2: BNRPALR_Playout(policy)

```
node ← root
sequenceActions ← {}
while true do
  if state(node) is a complete solution then
    return (score(state(node)), sequenceActions)
  else
    if node has not yet been traversed then
      action ← Simulate(node)
      state(child(node, 0)) ← state(node)
      state(child(node, 1)) ← state(node) ∪ action
    else
      action ← action(children(node))
    end if
  end if
  z ← exp(policy[(action, 0)]) + exp(policy[(action, 1)])
  child ← choose child i with probability proportional to
          exp(policy[(action, i)]) / z
  sequenceActions ← sequenceActions + [(action, i)]
  node ← child
end while
```

Algorithm 3: BNRPALR_Adapt(policy, sequenceActions)

```
policy´ ← policy
for num ← 1 to length(sequenceActions) do
  (action, i) ← sequenceActions[num]
  policy´ [(action, i)] += α
  z ← exp(policy[(action, 0)]) + exp(policy[(action, 1)])
  policy´ [(action, 0)] -= α × exp(policy[(action, 0)]) / z
  policy´ [(action, 1)] -= α × exp(policy[(action, 1)]) / z
end for
return policy´
```

Algorithm 4: Simulate(node)

```
greedyScores ← {}
for each available action do
  greedyScores ← greedyScores + [greedy score of action on state(node)
  according to the simulation heuristic]
return action corresponding to argmax(greedyScores)
```

Algorithm 5: DBNMCS(level, node)

```
if level = 0 then
  while state(node) is not a complete solution do
    action ← Simulate(node)
    state(child(node)) ← state(node) ∪ action
    node ← child(node)
  end while
  return score(state(node))
else
  bestScore ← −∞
  while node is not a leaf do
    action ← Simulate(node)
    state(child(node, 0)) ← state(node)
    state(child(node, 1)) ← state(node) ∪ action
    for children i of node do
      temp ← child(node, i)
      score ← DBNMCS(level − 1, temp)
      if score > bestScore then
        bestScore ← score
        bestChild ← temp
      end if
    end for
    node ← bestChild
  end while
  return bestScore
end if
```

Table 3. Set Cover Problem results

		LDS				DBNMCS						BNRPALR					
Instance	**BS**	CGP		SBH		CGP			SBH			CGP			SBH		
		AVG	GAP	AVG	GAP	AVG	STD	GAP	AVG	STD	GAP	AVG	STD	GAP	AVG	STD	GAP
41	429	465.9	8.6	465	8.4	443	0.4	3.3	442	0	3	433	0.9	0.9	437.2	1.9	1.9
42	512	567.1	10.8	560	9.4	543.9	0.8	6.2	534.2	0.1	4.3	514*	0.8	0.4	515.2*	1.1	0.6
43	516	572.4	10.9	560	8.5	550.4	1.2	6.7	542	0	5	522.6	1.2	1.3	523	1.3	1.4
44	494	533.2	7.9	534	8.1	522.3	0.5	5.7	521	0	5.5	498.5	0.3	0.9	499.6	0.4	1.1
45	512	570.1	11.4	568	10.9	534.8	0.6	4.5	532	0	3.9	515	0.2	0.6	516.3	0.8	0.8
46	560	596.1	6.4	599	7	575	0.4	2.7	574	0	2.5	564.4	0.4	0.8	565.1	0.5	0.9
47	430	469.4	9.2	476	10.7	445.2	0.5	3.5	441	0.5	2.6	433.5	0.1	0.8	433.4	0.1	0.8
48	492	525.9	6.9	524	6.5	505.8	0.5	2.8	502	0	2	499.2	0.5	1.5	499.2	0.6	1.5
49	641	715.9	11.7	700	9.2	674.7	0.5	5.3	672	0	4.8	647.1	0.7	0.9	649.1	1.4	1.3
410	514	544.3	5.9	543	5.6	529.6	0.4	3	529	0	2.9	517	0.2	0.6	517.2	0.3	0.6
51	253	283.8	12.2	282	11.5	269.6	0.9	6.6	265	0.1	4.8	259	0.7	2.4	259.7	1	2.7
52	302	337.5	11.8	335	10.9	322.7	0.7	6.8	315	0	4.3	308.4	0.9	2.1	310.2	1.3	2.7
53	226	243.9	7.9	241	6.6	236	0.4	4.4	233	0	3.1	228.8	0.3	1.2	229.4	0.7	1.5
54	242	257.2	6.3	259	7	253	0.4	4.6	254.2	2.1	5	244.5	0.8	1	247.5	1.6	2.3
55	211	234.2	11	229.3	8.7	222.4	0.9	5.4	220	0	4.3	213.4	0.6	1.1	215.2	1.4	2
56	213	242.1	13.7	243	14.1	228.4	0.9	7.2	221.9	0.1	4.2	214.5	0.4	0.7	215	0.6	0.9
57	293	314.1	7.2	319	8.9	303.8	0.7	3.7	304	0	3.8	296.1	0.5	1.1	297.1	0.6	1.4
58	288	315.3	9.5	318	10.4	303.4	0.5	5.4	300	0	4.2	290.7	0.3	0.9	291.8	0.8	1.3
59	279	297.4	6.6	295	5.7	293	0.6	5	287	0	2.9	280.4*	0.6	0.5	281.8*	0.8	1
510	265	284.7	7.4	284	7.2	278.7	0.3	5.2	277	0	4.5	268.7	0.5	1.4	268.6	0.5	1.4
61	138	148.9	7.9	148	7.2	142.4	0.4	3.2	149.6	0.2	8.4	139*	0.8	0.7	139.3*	1	1
62	146	163.3	11.8	163	11.6	150.5	0.6	3.1	154	0	5.5	147.1*	0.6	0.7	147*	0.6	0.7
63	145	157.9	8.9	158	9	152	0	4.8	153	0	5.5	149.8	0.9	3.3	149.9	1	3.4
64	131	140.5	7.3	140	6.9	134.2	0.7	2.4	136	0	3.8	132	0	0.8	132.2	0.3	0.9
65	161	183.3	13.8	182	13	171.8	0.7	6.7	175	0	8.7	161.6*	0.3	0.4	161.2*	0.2	0.1
A1	253	281.6	11.3	280	10.7	267.2	0.6	5.6	261.5	0.4	3.4	258	0.5	2	260.3	0.8	2.9
A2	252	282.9	12.2	279	10.7	265.4	0.5	5.3	264	0	4.8	257.1	0.9	2	260.1	1.1	3.2
A3	232	260	12.1	256	10.3	249.2	1.9	7.4	244	0	5.2	238.3*	1.4	2.7	240.6	1.3	3.7
A4	234	269.9	15.4	266	13.7	256.1	1	9.5	253	0	8.1	239.3	0.7	2.3	242.9	1.2	3.8
A5	236	256.7	8.8	254.4	7.8	248.5	0.4	5.3	244	0	3.4	240.1	0.5	1.7	242.1	0.8	2.6
B1	69	73.9	7.1	75	8.7	71.9	0.4	4.2	73	0	5.8	70.2	0.6	1.8	70.3	0.5	1.9
B2	76	83	9.2	82	7.9	77.8	0.6	2.4	78	0	2.6	76.2*	0.3	0.3	76.8*	0.8	1.1
B3	80	85.4	6.8	86	7.5	82.4	0.3	3.1	86	0	7.5	81.7	0.4	2.2	81.9	0.4	2.4
B4	79	86.5	9.5	87	10.1	82	0.5	3.8	85	0	7.6	79.5*	0.3	0.6	79.6*	0.4	0.7
B5	72	78	8.3	78	8.3	73.8	0.2	2.4	74	0	2.8	72.8*	0.8	1.1	73*	0.7	1.4
C1	227	256.4	13	252	11	244.6	0.5	7.8	243	0	7	239.5	0.9	5.5	241.8	0.9	6.5
C2	219	251.5	14.8	246	12.3	238.5	0.7	8.9	234.1	0.7	6.9	231.4	1.2	5.6	234.2	2	7
C3	243	267.8	10.2	266	9.5	263.2	1.3	8.3	265.6	2	9.3	257.1	1.5	5.8	261.4	2.1	7.6
C4	219	256.3	17	252	15.1	241.7	1.4	10.4	241	0	10	229.3	1.5	4.7	233.1	1.9	6.4
C5	215	234.9	9.2	235	9.3	231	0.8	7.4	233	0	8.4	225.6	1.4	4.9	230.8	1.5	7.3
D1	60	68.1	13.4	68	13.3	63.3	0.5	5.5	64	0	6.7	61.7	0.5	2.8	62.4	0.6	3.9
D2	66	70.5	6.8	70	6.1	67.6	0.3	2.4	68	0	3	66.3*	0.3	0.5	66.7*	0.4	1
D3	72	79.1	9.9	81	12.5	74.9	0.3	4	75	0	4.2	73.3*	0.5	1.8	73.5*	0.7	2.1
D4	62	66.1	6.5	65.5	5.7	62.8*	0.5	1.3	63.4	0.3	2.2	62.7*	0.5	1.1	63.4*	0.7	2.3
D5	61	66.9	9.7	69	13.1	63.5	0.4	4.1	64	0	4.9	61.5*	0.4	0.8	62.1*	0.7	1.8
E12345	5	5*	0	5*	0	5*	0	0	5*	0	0	5*	0	0	5*	0	0
NRE1	29	30.3	4.4	30	3.4	29.1*	0.3	0.3	30	0	3.4	29.1*	0.3	0.3	29.2*	0.3	0.6
NRE2	30	32.8	9.3	33	10	31.1*	0.5	3.6	32	0	6.7	30.5*	0.5	1.6	31.3*	0.5	4.3
NRE3	27	28.3	4.7	28	3.7	27.9*	0.3	3.3	28	0	3.7	27.6*	0.5	2.1	27.2*	0.4	0.7
NRE4	28	30.8	10	31	10.7	28.6*	0.4	2.3	29	0	3.6	28.4*	0.5	1.3	28.6*	0.5	2.1
NRE5	28	31	10.7	31	10.7	28*	0	0	28*	0	0	28*	0	0	28*	0.2	0.1
NRF1	14	14*	0	14*	0	14*	0	0	14*	0	0	14*	0	0	14*	0	0
NRF2	15	15*	0	15*	0	15*	0	0	15*	0	0	15*	0	0	15*	0	0
NRF3	14	15	7.1	15	7.1	15	0	7.1	15	0	7.1	14.9*	0.4	6.3	14.8*	0.6	5.4
NRF4	14	14*	0	14*	0	14*	0	0	14*	0	0	14*	0	0	14*	0	0
NRF5	13	13.5*	3.7	14	7.7	14	0	7.7	14	0	7.7	13.5*	0.7	4	13.8*	0.5	6.2
NRG1	176	202.1	14.8	197	11.9	194	0.6	10.2	193	0	9.7	217.4	1.9	23.5	232.9	2.3	32.3
NRG2	154	176.3	14.4	171	11	169.5	0.6	10.1	169	0	9.7	188.2	2.6	22.2	203.4	3.2	32.1
NRG3	166	189.1	13.9	187	12.7	182.7	0.6	10	179.2	0.2	8	203.7	3.1	22.7	220.7	2.8	33
NRG4	168	188.6	12.3	192	14.3	184	0.5	9.5	184.4	0.2	9.8	204.9	2.3	22	221.8	2.7	32
NRG5	168	189.9	13	194	15.5	184.3	0.5	9.7	182	0	8.3	205.2	2.7	22.2	218.8	2.4	30.3
NRH1	63	72.7	15.4	71	12.7	68.1	0.5	8.1	68	0	7.9	69.6	1	10.5	70.9	1	12.6
NRH2	63	71.9	14.1	71	12.7	68.2	0.5	8.3	69	0	9.5	69.9	0.6	10.9	70.9	1	12.6
NRH3	59	66.3	12.4	67	13.6	63.4	0.4	7.5	64	0	8.5	65.6	0.8	11.1	66.8	1	13.2
NRH4	58	64.6	11.3	65	12.1	61.8	0.5	6.6	63	0	8.6	63.6	0.8	9.7	65.2	1.1	12.4
NRH5	55	61.6	12	61	10.9	57.9	0.4	5.2	56	0	1.8	59.2	1	7.6	61.5	1.4	11.8

Effective Kinodynamic Planning and Exploration through Quality Diversity and Trajectory Optimization*

Konstantinos A. Asimakopoulos[1,2], Aristeidis A. Androutsopoulos[3], Michael N. Vrahatis[2], and Konstantinos I. Chatzilygeroudis[1,2]

[1] Laboratory of Automation & Robotics (LAR), Department of Electrical & Computer Engineering, University of Patras, GR-26504 Patras, Greece, *Emails:* konassimako@gmail.com, costashatz@upatras.gr
[2] Computational Intelligence Laboratory (CILab), Department of Mathematics, University of Patras, GR-26110 Patras, Greece, *Email:* vrahatis@math.upatras.gr
[3] Computer Engineering and Informatics Department (CEID), University of Patras, GR-26504 Patras, Greece, *Email:* a.a.androutsopoulos@gmail.com

Abstract. Efficient and rapid kinodynamic planning is crucial for numerous real-world robotics applications. Various methods have been proposed to address this challenge, primarily falling into two categories: (a) randomized planners and (b) trajectory optimization utilizing simplified models and numerical optimization. Randomized planners such as RRT and PRM excel in exploring the state space, while trajectory optimization methods, like direct collocation, are adept at discovering optimal trajectories within well-defined spaces. We aim to achieve effective and efficient kinodynamic planning and exploration by integrating evolutionary algorithms (Quality-Diversity) with trajectory optimization. Our preliminary experiments showcase that using the proposed methodology we get the best from both worlds on two simulated experiments.

1 Introduction and Related Work

Effective exploration of the state space is crucial in real-world robotic applications. However in hard exploration problems a large amount of iterations is usually required to reach areas of high-reward. This necessitates the use of an efficient exploration strategy. Randomized planners like Rapidly exploring Random Trees (RRT) [5] and Probabilistic Roadmaps (PRM) [4] navigate large state spaces efficiently, with RRT* even attempting to find the optimal path [3]. Trajectory optimization excels in well-defined spaces but has limitations for extensive state-space coverage. Go-Explore [2] is a Quality-Diversity (QD) optimization algorithm [1, 7] and is able to provide numerous diverse high-performing solutions despite having a simple random exploration strategy. We propose a more structured approach by substituting this strategy with trajectory optimization

* This work was supported by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the "3rd Call for H.F.R.I. Research Projects to support Post-Doctoral Researchers" (Project Acronym: NOSALRO, Project Number: 7541).

for a synergistic solution. Our improvements, outlined in Section 2, enhance the Go-Explore framework with the robustness of trajectory optimization, while also improving its exploration abilities in difficult scenarios.

2 Proposed Approach

Go-Explore framework: Go-Explore [2] addresses detachment and derailment challenges by (1) maintaining a global archive of all uniquely interesting states and (2) dividing exploration into deterministic returns to states followed by stochastic exploration. To enhance robustness, the algorithm operates in two phases: (a) task-solving exploration and (b) solution robustification, involving processes like imitation learning on the discovered trajectories[4]. The initial phase of Go-Explore involves building an archive of interesting states, each associated with a *history* of *actions* and a *score* indicating exploration interest. This state/history/score group is called a *cell*. The archive, starting with the initial state, is denoted as $\mathcal{A}$. At each iteration, the algorithm follows the steps in Alg. 1. During cell insertion, scores are updated and a binary similarity metric evaluates the state in comparison to existing states. If distinct or higher performing, it is added and it replaces the similar cells in the archive.

Algorithm 1 Go-Explore Framework

procedure GO-EXPLORE($\boldsymbol{s}_0$, n_{batch}, n_{iter})
  $\mathcal{A} \leftarrow$ new_cell($\boldsymbol{s}_0, \emptyset$) ▷ Initialize archive, $\mathcal{A}$, with initial state $\boldsymbol{s}_0$
  for $n = 1$ **to** n_{iter} **do**
    cells $\leftarrow$ SELECTCELLS($\mathcal{A}$, n_{batch}) ▷ Select n_{batch} exploration cells
    new_cells $\leftarrow$ EXPLORE(**cells**) ▷ Run stochastic exploration
    ADDTOARCHIVE($\mathcal{A}$, **new_cells**) ▷ Add explored cells to the archive
    UPDATESCORES($\mathcal{A}$) ▷ Update the scores of cells
  end for
  return $\mathcal{A}$ ▷ The outcome of the algorithm is the archive
end procedure

Go-Explore with trajectory optimization: In the original implementation of Go-Explore [2] (Vanilla Go-Explore) the exploration strategy used is essentially a sequence of random actions. We propose to leverage trajectory optimization instead of random exploration. At the start of every exploration phase, *batch_size* cells are chosen from the archive to explore from. For every chosen *cell* we sample a random point within a small distance. We then use trajectory optimization to find the path between the cells and their respective random points. In this manner, we leverage the robust solutions provided by optimization on a smaller scale, all the while retaining the exploration benefits inherent in Go-Explore.

Bidirectional Go-Explore: To achieve a faster and more effective exploration, we propose a bidirectional Go-Explore algorithm with two agents—forward and backward—initialized at different state space points. Each agent runs a slightly

[4] We only care about the first phase in this work.

different version of our trajectory optimization enhanced Go-Explore, featuring separate archives. The forward version reflects our proposed model, while the backward version addresses the inverse optimization problem. During exploration for the backward version, the randomly sampled point is considered the starting point, aiming to return to the selected cell. The bidirectional exploration involves choosing a batch of cells, returning to their states, sampling a nearby random state, using trajectory optimization for the forward and backward version and updating their respective archives with the trajectory optimization results. We use this "reverse" design for the backwards version to facilitate exchange of useful discovered information between the two agents.

Trajectory merging: In order to be able to merge information from the forward and backward archives, we use a merging mechanism. After each exploration phase, we check if cells from the two archives are close based on a *merging distance*. If so, trajectory optimization concatenates their paths into one merged trajectory that is then added to the forward archive. The step-by-step merging process involves checking proximity between backward and forward archive cells, using trajectory optimization for optimal paths, concatenating histories, and adding the merged cell to the archive.

3 Experimental Results

In order to showcase the effectiveness of our proposed method, we designed and performed a number of simulated experiments. We tested our proposed method in two distinct environments and compared its performance to that of the Vanilla Go-Explore and RRT algorithms in space coverage and average trajectory length; we compare to RRT only in the Maze scenario.

Maze environment: For this experiment we use the 2D car-like kinematic vehicle model from [6] which does not allow skidding. We use this model to explore a 2D maze environment (see Fig. 1).

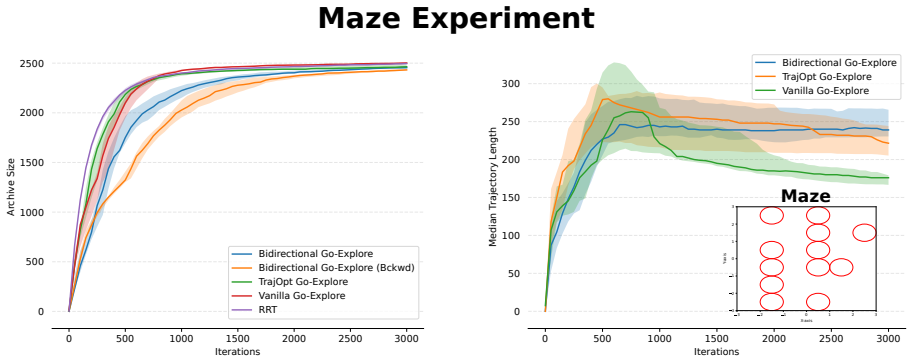


Fig. 1: 2D Maze Experiments. Median and 5th/95th percentiles over 5 replicates.

Planar Quadrotor: We use the Planar Quadrotor environment (Fig. 2) because a) it is higher dimensional, and b) the control commands matter (a.k.a., the quadrotor will crash if given bad commands).

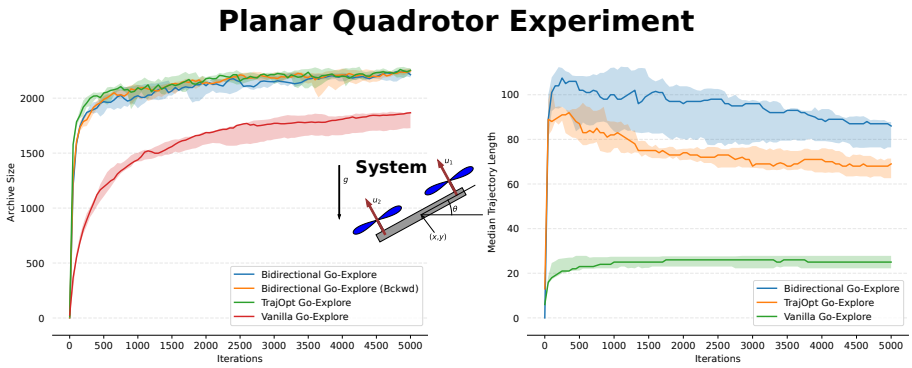


Fig. 2: Planar Quadrotor. Median and 5th/95th percentiles over 5 replicates.

Results: We run 5 replicates of each scenario for each algorithm and report the space coverage and average trajectory length. Trajectory length is important (especially in the quadrotor) as it showcases that the algorithm has found effective control policies. The results showcase that our proposed method on simpler environments like the 2D car maze is on par with state of the art methods (Fig. 1) in both state space coverage and control efficacy, while clearly outperforming other methods in the more complicated and delicate quadrotor environment (Fig. 2).

4 Concluding Remarks

We have introduced a variant of the Go-Explore where we utilize trajectory optimization in a strategic manner in order to boost its performance while keeping its exploration capabilities. We also provided preliminary results on a bidirectional Go-Explore scheme. Overall, our methods are able to efficiently explore the state space, while also discovering effective controllers.

References

1. Chatzilygeroudis, K., Cully, A., Vassiliades, V., Mouret, J.B.: Quality-diversity optimization: a novel branch of stochastic optimization. In: Black Box Optimization, Machine Learning, and No-Free Lunch Theorems, pp. 109–135. Springer (2021)
2. Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K.O., Clune, J.: First return, then explore. Nature **590**(7847), 580–586 (2021)
3. Karaman, S., Frazzoli, E.: Sampling-based algorithms for optimal motion planning. IJRR **30**(7), 846–894 (2011)
4. Kavraki, L.E., Svestka, P., Latombe, J.C., Overmars, M.H.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE transactions on Robotics and Automation **12**(4), 566–580 (1996)
5. LaValle, S.M., Kuffner Jr, J.J.: Randomized kinodynamic planning. In: IEEE International Conference on Robotics and Automation. vol. 1, pp. 473–479 (1999)
6. Pepy, R., Lambert, A., Mounier, H.: Path planning using a dynamic vehicle model. In: 2006 2nd International Conference on Information & Communication Technologies. vol. 1, pp. 781–786. IEEE (2006)
7. Pugh, J.K., Soros, L.B., Stanley, K.O.: Quality diversity: A new frontier for evolutionary computation. Frontiers in Robotics and AI **3**, 40 (2016)

Learning to accelerate a modular QP solver: Challenges and preliminary results

Jeremy Bertoncini[0009−0003−6093−7532], Alberto De Marchi[0000−0002−3545−6898], and Simon Gottschalk[0000−0003−4305−5290]

University of the Bundeswehr Munich, Department of Aerospace Engineering, Institute of Applied Mathematics and Scientific Computing, 85577 Neubiberg, Germany
{jeremy.bertoncini,alberto.demarchi,simon.gottschalk}@unibw.de

Keywords. Reinforcement Learning, Quadratic Programming, Amortized Optimization, Acceleration Strategies

Quadratic programming is a workhorse of modern nonlinear optimization, control, and data science. Although regularized methods offer convergence guarantees under minimal assumptions on the problem settings [4], they can exhibit the slow tail-convergence typical of first-order schemes [2]. Thus, they tend to require many iterations to achieve high-accuracy solutions. Moreover, the solver performance is sensitive to the hyperparameter tuning. To address these issues, we explore how data-driven approaches can accelerate the solution process. Indeed, learning techniques may serve as a remedy for slow convergence by proposing ideal hyperparameters sequences (cf. [5]). We will show that Reinforcement Learning (RL) [10] can make a significant contribution to speeding up the optimization process.

We present in this preliminary research a modular implementation of a proximal augmented Lagrangian interior point method and investigate the influence of penalty parameters' dynamics. Exploiting proximal regularization schemes to guarantee robustness and numerical stability, we focus on two proximally stabilized interior point methods [3,7] (and recently implemented in [9]) whose dynamics depend on the problem settings and some (sequences of) hyperparameters. Using an object-oriented implementation the solver is constituted of multiple modules offering each different features. Each ready-to-use optimization method, linesearch, regularisation techniques, and termination criteria can be easily combined to build a plethora of different optimization architectures. We investigate and explore how to accelerate such robust yet slowed-down algorithms using data-driven and learning-based techniques. In particular, we leverage the minimal requirements posed by proximal methods to dynamically construct the sequences of hyperparameters, possibly adapting the solver's behaviour to the problem instance at hand. A common approach is to decrease proximal and penalty parameters, monitoring some early termination criteria, in order to tackle the problem faster. Preliminary evidence suggests that for some decrease strategies the solver exhibits a much better convergence speed, although staying very problem-dependent. Our framework's preliminary results are based on convex random problems and the Maros-Mészáros benchmark test-set [6].

This research is funded by dtec.bw – Digitalization and Technology Research Center of the Bundeswehr. dtec.bw is funded by the European Union – NextGenerationEU.

Learning-based acceleration methods (e.g. *amortized* optimization [1]) were already successfully applied in multiple academic problems for parameter tuning and warm-start strategies [5,8]. However, it is not yet clear how beneficial such strategies can be and at what training-cost. In [1] multiple application cases of amortized optimization are presented and attempt to predict the solutions to problems in these settings, exploiting the shared structure between similar problem instances. Using RL a policy is learned that outputs the next hyperparameters depending on the current training progress. During the training, the policy interacts with steps of the solution approach, which takes over the role of the system to be controlled. Furthermore, we apply a model-based approach in order to take advantage of the knowledge about the model, instead of treating it as a black-box. The final outcome of the training is a fast automation of the parameter decision for the proximal interior point method, which outperforms the usual heuristic choice.

References

1. Brandon Amos. Tutorial on amortized optimization. *arXiv:2202.00665*, 2023.
2. David Applegate, Mateo Diaz, Oliver Hinder, Haihao Lu, Miles Lubin, Brendan O' Donoghue, and Warren Schudy. Practical large-scale linear programming using primal-dual hybrid gradient. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 20243–20257. Curran Associates, Inc., 2021.
3. Stefano Cipolla and Jacek Gondzio. Proximal stabilized interior point methods and low-frequency-update preconditioning techniques. *Journal of Optimization Theory and Applications*, 197(3):1061–1103, 2023.
4. Alberto De Marchi. On a primal-dual Newton proximal method for convex quadratic programs. *Computational Optimization and Applications*, 81(2):369–395, 3 2022.
5. Jeffrey Ichnowski, Paras Jain, Bartolomeo Stellato, Goran Banjac, Michael Luo, Francesco Borrelli, Joseph E. Gonzalez, Ion Stoica, and Ken Goldberg. Accelerating quadratic optimization with reinforcement learning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 21043–21055. Curran Associates, Inc., 2021.
6. István Maros and Csaba Mészáros. A repository of convex quadratic programming problems. *Optimization Methods and Software*, 11(1–4):671–681, 1999.
7. Spyridon Pougkakiotis and Jacek Gondzio. An interior point-proximal method of multipliers for convex quadratic programming. *Computational Optimization and Applications*, 78(2):307–351, 2021.
8. Rajiv Sambharya, Georgina Hall, Brandon Amos, and Bartolomeo Stellato. End-to-end learning to warm-start for real-time quadratic optimization. In N. Matni, M. Morari, and G. J. Pappas, editors, *Proceedings of the 5th Annual Learning for Dynamics and Control Conference*, volume 211 of *Proceedings of Machine Learning Research*, pages 220–234. PMLR, 6 2023.
9. Roland Schwan, Yuning Jiang, Daniel Kuhn, and Colin N. Jones. PIQP: A proximal interior-point quadratic programming solver. In *2023 62nd IEEE Conference on Decision and Control (CDC)*, pages 1088–1093, 2023.
10. Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

Decoupled Design of Experiments for Expensive Multi-objective Problems

Mickaël Binois[1][0000−0002−7225−1680],
Jürgen Branke[2][0000−0001−5880−1925],
Jonathan Fieldsend[3][0000−0002−0683−2583], and
Robin C. Purshouse[4][0000−0001−5880−1925]

[1] Inria Centre at Université Côte d'Azur mickael.binois@inria.fr
[2] University of Warwick juergen.branke@wbs.ac.uk
[3] University of Exeter j.e.fieldsend@exeter.ac.uk
[4] University of Sheffield r.purshouse@sheffield.ac.uk

Abstract. In this paper we look at the experimental design for multi-objective problems, where the objectives can be evaluated independently (decoupled) and thus it may make sense to evaluate different solutions for each objective if the objectives have different evaluation costs and/or different landscape characteristics. We propose to iteratively add design points in a way that minimises the total integrated mean squared prediction error assuming a Gaussian process response surface model, and show that allowing decoupled evaluations can lead to significantly better Pareto front estimations than a coupled design of experiments if the evaluation costs of the objectives are different. We also find that our approach of minimising mean squared prediction error yields significantly better results than standard Latin Hypercube designs even if the evaluation costs and landscape characteristics of the objectives are the same.

Keywords: Expensive optimisation · Varying costs · Multi-objective experimental design.

1 Introduction

Fundamental to the performance of surrogate-based optimisation frameworks is the need to construct an initial model based on a carefully selected set of initial designs and any prior system knowledge. This is both in the case of Bayesian optimisation (BO), which uses and iteratively updates model(s) mapping decision vectors to predicted performance criteria values, and for evolutionary computation approaches which involve surrogates. The selection and construction of initial designs, which are often treated separately to the decision vectors queried during the subsequent optimisation process, are usually referred to as the *design of experiments* (or DoE for short). This is because these decision vectors are selected to—in some fashion—be maximally informative on the global underlying process, rather than being biased towards particular regions.

Without any prior information regarding the properties of the objective function(s) such DoE for model fitting are commonly based around *space filling* sequences such as Latin hypercube sampling (LHS) [15] or Sobol sequences [16], as purely random sampling tends to naturally result in clusters, which do not serve model fitting well, particularly when the budget for sampling is tight.

Where there are multiple criteria being modelled, this leads to an interesting and under-explored question: *should one evaluate all initial designs fully, or instead selectively evaluate a subset of objectives per design, allowing a greater number of locations to be partially evaluated when building the model(s)?* A few works have looked at *decoupling* objective evaluations during the search process—particularly where there are different costs associated with each objective, but this can also be advantageous where there is a difference in the complexity of the functions being modelled (e.g. one being smooth and slowly changing, the other being rugged and fast changing). As such, this appears to be a promising direction for further investigation and research, as even small improvements in such areas can effectively lead to large savings for expensive optimisation problems. A possible drawback, on the other hand, is that the Pareto dominance cannot be determined for sure on decoupled designs (only with some confidence depending on the accuracy of the surrogate model prediction).

The remainder of the paper is set out as follows. In Section 2 we introduce existing work and methods relating to decoupled and cost-aware multi-objective optimisation, and highlight how our work relates to these. Section 3 presents results of the proposed approach with different problem configurations, and highlights the circumstances where there appears to be a significant benefit to decoupling the DoE locations. In Section 4 we discuss the results, and highlight future research directions.

2 Related Work

In single objective optimisation, there are various papers taking into account the cost of evaluating a solution where this cost depends on the solution evaluated. The de facto standard is to divide the acquisition function value by the corresponding cost value (e.g., [17]). [13] demonstrates that this is not always a good choice, and proposes an alternative mechanism. In particular, they propose an initial space-filling design that takes cost into account, by iteratively and greedily adding points that are inexpensive to evaluate but have a large distance from points already chosen. During optimisation, their algorithm reduces the emphasis on cost, starting with the standard division by cost, then slowly changing into a standard acquisition function optimisation without considering cost. In [12], the authors propose a non-myopic approach to BO with cost considerations.

A small number of existing works have considered decoupled and/or cost-aware multi-objective optimisation—some of which have considered these factors during the initial DoE phase. Below we discuss the most relevant approaches. A wider survey on the topic of objectives with different costs can be found in [2].

In [1], a user can define a cost ranking of the decision variables, e.g. in the case that decision variables represent the amount of an ingredient, and the ingredients have different costs. The acquisition function then favours solutions with small values in particular for the expensive variables, and this preference is reduced over the course of the run, eventually removing cost considerations.

Hernández-Lobato and colleagues proposed the *Predictive Entropy Search for Multi-Objective Bayesian Optimization* (PESMO) method [10]. PESMO uses predictive entropy search as the acquisition function. This function represents each objective using an additive component, which enables a decoupled evaluation approach to be adopted. The approach was subsequently extended to also consider constraints (again where decoupling is possible) [9].

Suzuki et al. developed the *Pareto-frontier entropy search* (PFES) approach [18]. PFES is also an entropy approach but considers the entropy in objective-space rather than decision-space, which is computationally simpler. This method also includes cost in evaluating the objectives by including cost in the denominator of the acquisition function. Like PESMO, the approach is easily extended to consider decoupled evaluations.

The Joint Entropy Search (JES) proposed in [19] is also able to take into account different costs and decoupled evaluations, although the authors did not actually experiment with it because they expected little benefit from decoupled evaluations.

Iqbal and colleagues proposed the *Flexible Multi-Objective Bayesian Optimization* (FlexiBo) algorithm [11]. The approach uses a decoupled evaluation in the Bayesian optimisation run but a coupled initial DoE procedure. It additionally learns the solution-dependent cost function for each objective. FlexiBo estimates for each individual optimistic and pessimistic objective values, which are identical if the objective has been evaluated. From that, it computes an optimistic and pessimistic Pareto front as the boundaries of the "Pareto region", and uses an acquisition function that estimates the expected reduction in the volume of this Pareto region, divided by the respective cost.

Buckingham et al. extended the multi-attribute Knowledge Gradient [3] to the case where objectives can be evaluated independently [6]. The authors demonstrate the benefit of independent evaluation not only when the computational costs for objectives differ, but also when the lengthscales of the modelled landscapes (which determine the smoothness of the landscape) differ. Independently, [8] propose to adapt a hypervolume-based Knowledge Gradient approach to allow for decoupled evaluation of the objectives.

A slightly different problem is considered in [14,5], where one objective is much cheaper (essentially free) to evaluate than the other. They directly incorporate evaluation of the cheap objectives into a pair of hypervolume-based acquisition functions for BO. Consequently, the cheap objectives are evaluated many times while the acquisition function is optimised.

A summary of the different approaches is shown in Table 1.

Table 1: Existing methods for decoupled cost-aware multi-objective optimisation

	Design of experiments		Optimisation		
Approach	Decoupled?	Cost-aware?	Decoupled?	Cost-aware?	Acquisition function
PESMO [10]	✓	✗	✓	✗	predictive entropy search
PFES [18]	✗	✗	✓	✓	cost-weighted Pareto frontier entropy
FlexiBO [11]	✗	✗	✓	✓	cost-weighted objective space entropy
C-MOKG [6]	✗	✗	✓	✓	cost-weighted multi-objective knowledge gradient
CA-MOBO [1]	✗	✓	✗	✓	cost-weighted Tchebycheff scalarised UCB
HV-KG [8]	✗	✗	✓	✓	cost-weighted hypervolume knowlege gradient
JES [19]	✗	✗	✓	✓	joint entropy search
This paper	✓	✓	✗	✗	N/A

3 Empirical work

In this section we consider a range of different properties/configurations of a problem which may influence the effectiveness of a decoupled DoE, and investigate these empirically. LHS designs are generated using the `R` package `lhs` [7] with the maximin option.

3.1 Initial DoE when evaluations are decoupled

We begin with an illustration of a greatly simplified case, where the costs of querying each of two objectives are the same. The two objective functions are generated by Gaussian process models (GPs)—so we are assured that emulation by a trained GP will fit the modelling assumptions, and we also directly utilise the hyperparameters of the objective function GPs, removing the effect of having to infer these, so there is no model mismatch (i.e. our model is perfectly capable of modelling the generating process).

Our goal is to study the effect of coupled versus decoupled designs of experiments (DoE) on the uncertainty on the Pareto front in this very controlled problem configuration, before moving towards a more realistic scenario. We generate samples from a GP model for each objective and use it as the ground truth for fitting the GP approximation models. An example of the generating models and respective mapping to the objective space is given in Figure 1.

In Figure 2 an example is shown where the DoE for the first objective is the same while the second objective is either coupled (left panel) or decoupled (right

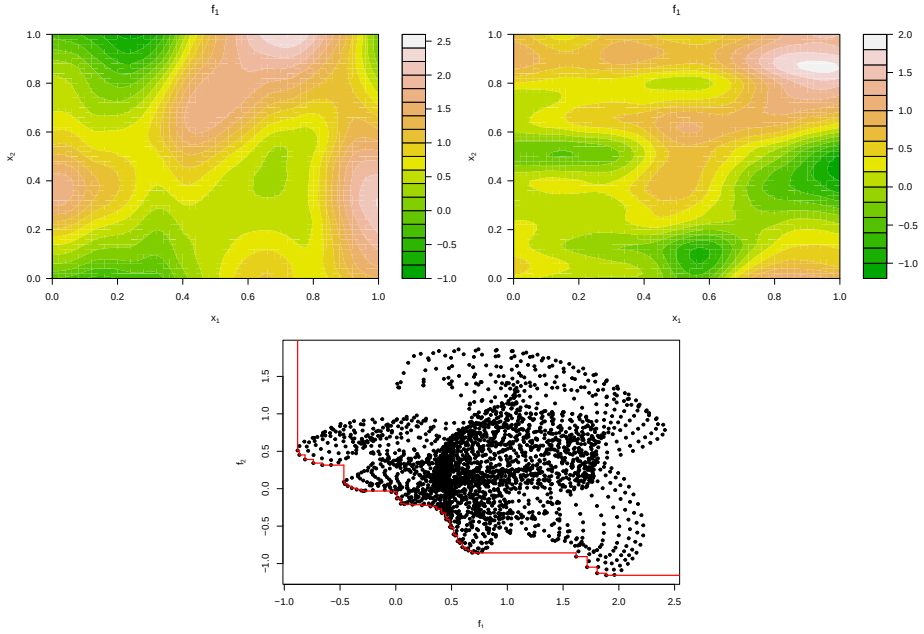


Fig. 1: Top: two realisations of Gaussian process priors, with Matérn 5/2 covariance kernel, with lengthscale hyperparameters $(0.3, 0.4)$ (resp. $(0.4, 0.2)$) for f_1 (resp. f_2), and unit variance. Bottom: corresponding image in the objective space (grid sampled), with the estimated Pareto front highlighted in red.

panel). The decoupled DoE of the second objective is obtained by augmenting the first objective DoE while maintaining the LHS structure. Attainment functions are obtained by taking a joint sample on a 51×51 grid from the GP posterior for each objective conditioned on the observations, then determining the non-dominated observations. The q-Attainment front is then representing the area that is dominated by a fraction q of all the estimated Pareto fronts generated. One visible effect is that when both objectives are jointly evaluated, the area that is dominated (attainment value $= 1$) is larger. This is probably because in the decoupled case, solutions are never surely dominated (even though the domination probability is extremely low), as no location has been queried under *both* objective functions (this can be further seen with the left panel having triangles denoting locations with a pair of known objective values, and the right panel having no triangles).

To help measure the uncertainty on the Pareto front associated with the fitted GPs, we use below the so called Vorob'ev deviation (VD), a set based variance metric that measures the variability of the q-Attainment fronts relative to the true frontier—see, e.g., [4] for further details on its properties. Algorithm 1 summarises the testing procedure.

Algorithm 1: Pseudo-code for the testing procedure

1: Generate the first design of experiments X_1 for objective 1.
2: **if** *Coupled case* **then**
| $X_2 = X_1$, the DoE of the second objective is the same.
end
else
| Generate X_2 the second DoE. (Decoupled case)
end
3: Build GP models.
4: Generate s conditional samples on some designs X_s from all GPs.
5: Compute the s sets of non-dominated points on couples of samples from the different GPs.
6: Compute the corresponding Vorob'ev deviation.

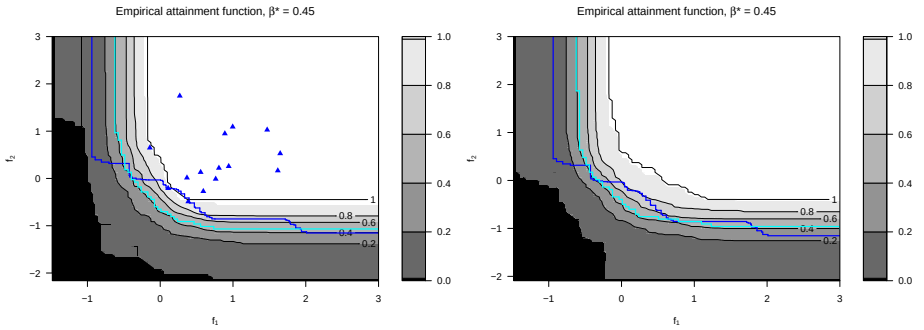


Fig. 2: Attainment function representation in the coupled (left) and decoupled (right) cases. The blue triangles mark observations in the coupled case, where both objectives are evaluated. The cyan line represents the estimated Pareto front of the GP while the reference Pareto front is in blue.

Figure 3 shows the Vorob'ev deviation of the coupled and decoupled designs for two cases. In the left panel, the design for each objective is uniformly random, while in the right panel, a LHS design is used for the first objective, and then an augmented LHS is used for the second objective. The panels show the results of 11 independent runs, with 10 replications for the design of the second objective in case of decoupled design (visualised as boxplots).

As expected, LHS designs (right panel) lead to slightly lower Vorob'ev deviations than random uniform designs (left panel), in particular for the coupled case. The larger benefit in case of the coupled design is probably due to the fact that a cluster or gap in the sample space of the first objective is unlikely to be duplicated for the second objective in the decoupled design. When LHS is used, the coupled design (red dots) seems to yield a lower Vorob'ev deviation than the decoupled designs, possibly due to the effect mentioned above on the size of the known dominated region. This difficulty in precisely estimating the Pareto front may also pose challenges for the optimisation procedure, as a reference Pareto

front is generally required by acquisition functions. Note, however, that in these experiments we assume equal cost of sampling the two objectives, and equal lengthscales of the two objectives. As we see later, in other cases decoupling may be beneficial.

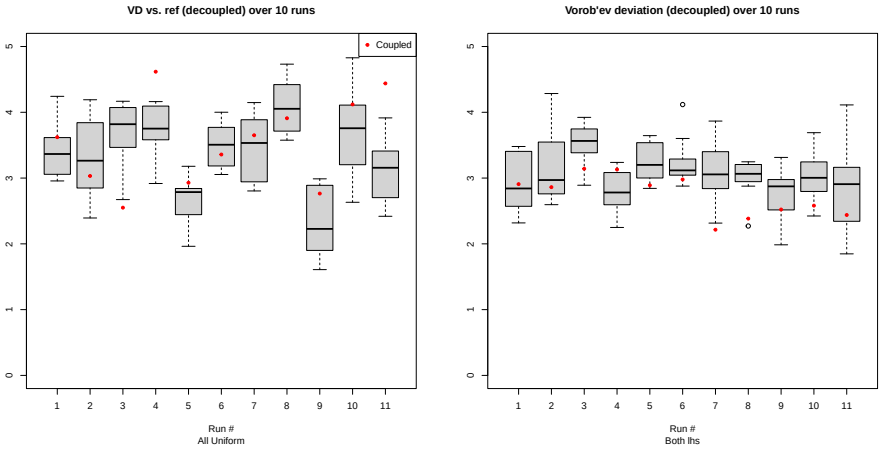


Fig. 3: Vorob'ev deviation against the true Pareto front. Boxplots are for decoupled designs, over 11 different runs and 10 replications per run for the second stage design in the decoupled case; red dots are the coupled designs. In the left figure, the designs are uniformly sampled, while in the right figure, an augmented LHS is used to complement the first LHS design. The value of the coupled design is in red.

3.2 Initial DoE when evaluations have different costs

Now let us assume the cost is different between different objectives f_1 and f_2. The first tasks are to define the total time budget for experiments and get relative costs of f_1 and f_2. We will then consider four alternative approaches to DoE, including a coupled LHS baseline.

1. (Fixed LHS) Rather than sampling incrementally, we use a fixed coupled LHS design of the required size.
2. (Coupled) Both objective functions are evaluated together.
3. (Decoupled naïve) Both objective functions are evaluated the same number of times, but at differing locations (generated by Augmented LHS using the `optAugmentLHS` function from the `R` package `lhs` [7]).
4. (Decoupled) The allocation of total budget to the two functions depends on lengthscales and relative costs, according to Eq. 1. Objectives with smaller lengthscales and smaller cost are sampled more often.

Considering how to split the computational budget, let us consider the simplest case of optimising a (weighted) sum of two objectives. In such a case, if we want to minimise integrated mean squared prediction error (IMSPE) assuming Gaussian process surrogate models with identical lengthscales, then it is not possible to improve beyond coupled sampling, as the variances of the two functions just add up, and the optimal design for each function would be the same. However, if the costs or lengthscales are different, then we could use IMSPE to determine an appropriate allocation of the budget to the two functions by choosing the number of samples n_1 allocated to objective 1 such that the following is minimised:

$$\min \frac{\text{IMSPE}_1(n_1)}{c_1 \times n_1} + \frac{\text{IMSPE}_2(N - n_1)}{c_2 \times (N - n_1)}, \tag{1}$$

where N is the total budget, IMSPE_1 (IMSPE_2) and $c_1(c_2)$ are the IMSPE and cost of evaluating objective $f_1(f_2)$, respectively.

In practice, if the lengthscales are not known, they may be estimated from initial data and Eq. 1 may be optimised sequentially rather than all at once. That is, in the coupled case we iteratively sample the solution that, if both its objectives are evaluated, reduces the IMSPE the most. For the decoupled case, we sample the solution and objective which maximally reduces the IMSPE as calculated in Eq. 1.

As in the previous section, we rely on GP samples in a two-dimensional decision variable space to define a ground truth. We start with the same four coupled initial designs for each objective in the various cases, based on LHS, then add additional samples in a way that minimises IMSPE. For the coupled option, a discrete search over a thousand uniformly sampled candidates is performed at each iteration. As for the decoupled version, a local optimisation is conducted from the best out of one hundred uniformly sampled candidates.

Figure 4 shows the results for the case that the lengthscales for both objectives are equal and known, here, (0.3, 0.4) for the Matérn 5/2 covariance kernel. In the left column, the evaluation cost for both objectives is the same, in the middle column the second objective is five times more expensive, and in the right column, the second objective is 10 times more expensive.

The top row depicts the IMSPE separately for each objective. In the case of equal cost to evaluate both objectives, also the decoupled designs reduce IMSPE equally for both objectives. However, if the evaluation cost for the objectives differ, the decoupled design samples the cheaper f_1 (black +) more often, reducing its IMSPE much more than the IMSPE of the expensive f_2 (red +).

The following rows 2-4 show aggregated performance metrics, namely the average IMSPE, the average root mean squared error (RMSE), and the Vorob'ev deviation. The results are consistent across all metrics: if the costs of the different objectives are equal, the IMSPE-minimising coupled and decoupled approaches perform very similarly (and best), not only with respect to IMSPE but also RMSE and VD. Next best is the fixed LHS and then, significantly worse, the naïve design. This is interesting as it suggests that iteratively choosing points to minimise IMSPE (decoupled as well as coupled) yields not only a lower IMSPE

but also a lower Vorob'ev deviation than a standard LHS of the same size. Decoupling naïvely by two augmenting LHS is clearly worst.

If we look at cases of different costs (f_2 five times as expensive (middle column) or 10 times as expensive (right column)), the differences become more pronounced, and the decoupled design clearly beats the coupled design, as it can sample the cheaper objective more often and make better use of the available budget.

Similar results are obtained if the lengthscales are estimated and updated every iteration, see Figure 5. Note, however, that in these experiments we took 20 initial samples based on LHS, as more data is needed to estimate lengthscales reliably.

Additional experiments (see Appendix) with different lengthscales for the two objectives ($(0.3, 0.4)$ for the first, $(0.4, 0.2)$ for the second objective) do not seem to show a significant differences, but this may simply be because the chosen lengthscales were still quite similar.

Finally, we treat the case when the costs are varying depending on both x and the objectives, and we know the cost function. The cost functions correspond as well to samples from GPs, this time with lengthscales (0.5, 0.8) and (0.6, 0.7) for the respective objectives, while for the objective values we again use $(0.3, 0.4)$ for the first, $(0.4, 0.2)$ for the second objective. The results are given in Figure 6. The decoupled strategy is again more efficient to reduce the IMSPE the fastest, but there is no noticeable difference in terms of Vorob'ev deviation. The naïve decoupled design does not make use of the cost information and is thus clearly worse. Note that the fixed LHS strategy is not sensible here, as it is necessary to learn about the evaluation cost and use this information in an incremental design.

4 Discussion and future research ideas

In this paper, we have examined the possibility of improving the quality of the surrogate models obtained through a DoE in case of multi-objective optimisation where the evaluation of the different objectives can be decoupled. We found that for the case of equal costs and lengthscales for the two objectives, decoupling the evaluations (i.e. evaluating different solutions on different objectives) did tend to worsen the quality of the Pareto front estimate as measured by Vorob'ev deviation. However, when objectives had different costs, decoupling could improve results substantially in terms of total IMPSE, RMSE, and Vorob'ev deviation. Interestingly, we found that even in the case of equal costs and lengthscales, allocating samples iteratively by minimising IMSPE yielded better IMSPE, RMSE and Vorob'ev deviation than using an equally sized LHS design.

While in this paper we have only considered the case of two objectives, we see no reason why our conclusions should be any different also for more than two objectives. Indeed, one might expect that the greater flexibility in terms of which objectives to evaluate for a solution could lead to even larger benefits of decoupled experimental designs. However, we leave the experimental confirma-

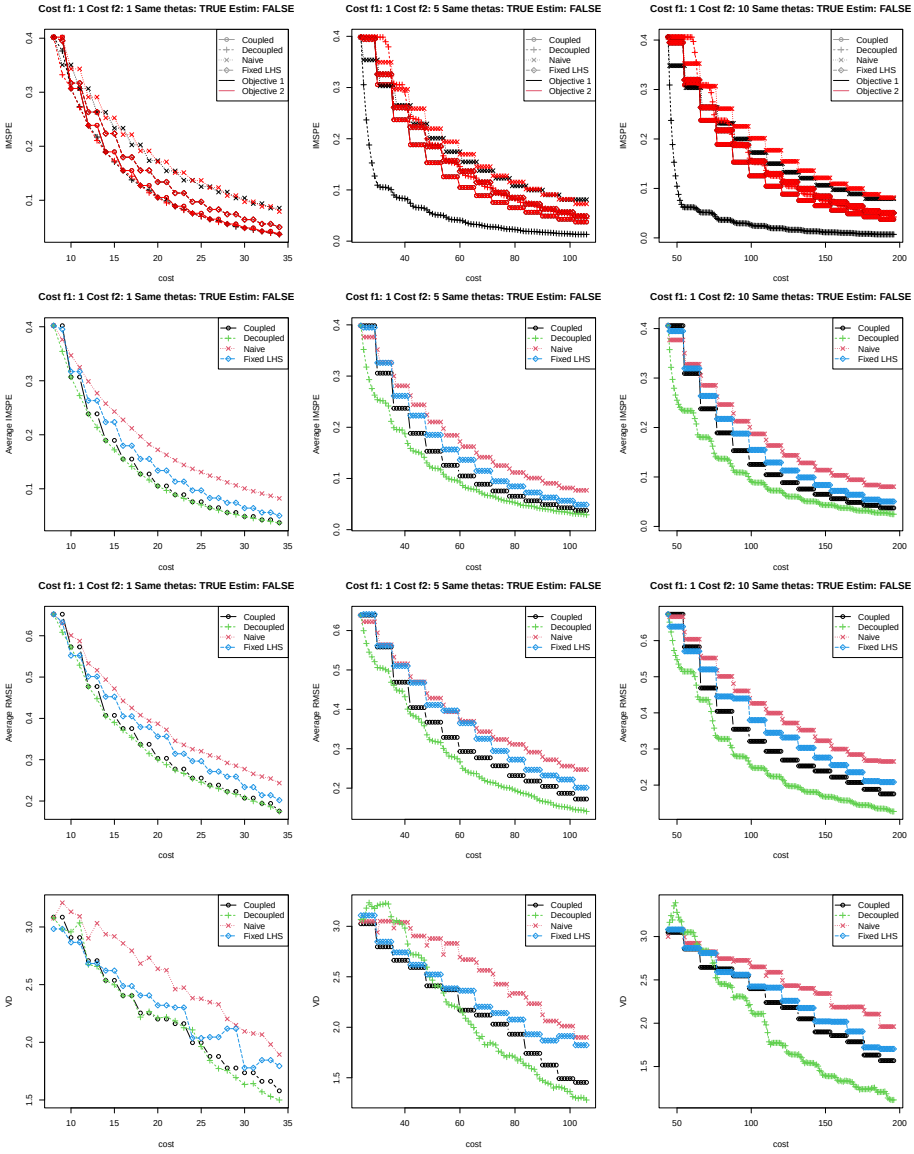


Fig. 4: Results of different metrics depending on the cost incurred. In the left column, both objectives have equal cost, in the middle column the cost for f_2 is 5 times as high, and in the right column, the cost for f_2 is 10 times as high. In this figure, lengthscales are equal and assumed known.

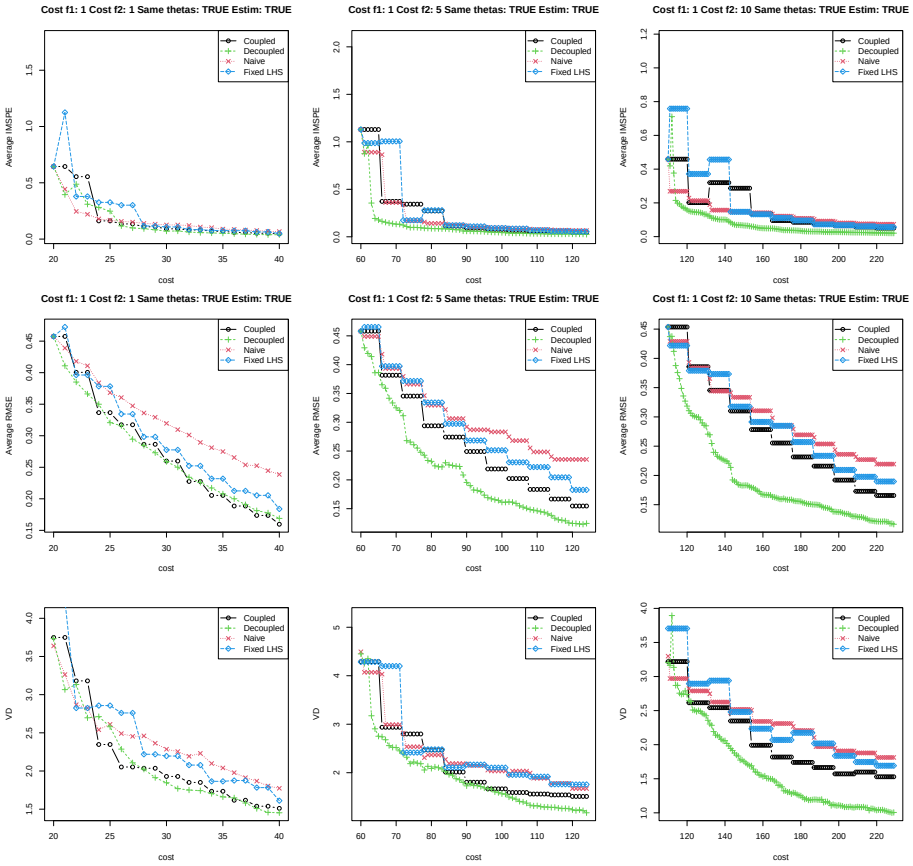


Fig. 5: Results of different metrics depending on the cost allocated. In the left column, both objectives have equal cost, in the middle column the cost for f_2 is 5 times as high, and in the right column, the cost for f_2 is 10 times as high. In this figure, lengthscales are equal but unknown (learned and in every iteration).

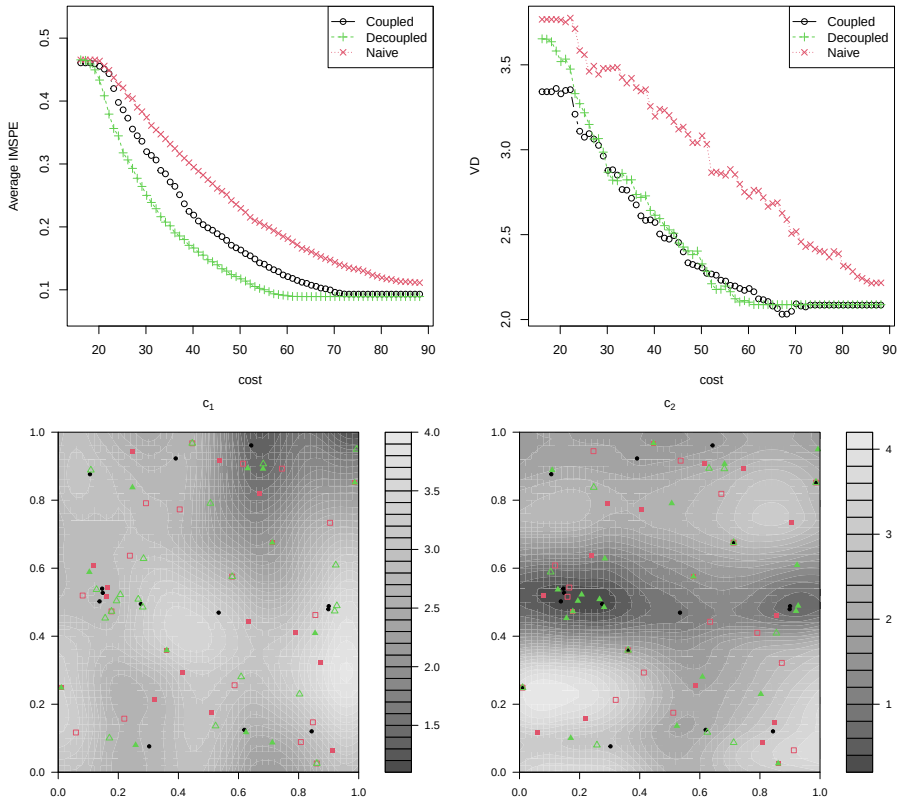


Fig. 6: Variable cost for each objective. Top left (resp. right): IMSPE (resp. VD) vs. cost for the various strategies. Bottom: cost surface and evaluated points for each strategy: black dots for coupled, red squares for naïve and green triangles for decoupled. Empty squares and triangles are evaluated on the other objective. naïve doesn't take into account the cost.

tion of this hypothesis to future work. Our results use GP generated functions to avoid the issue of model mismatch. However, it would be good to confirm results also on other types of functions. Finally, in the future we plan to investigate other sampling strategies such as taking into account the posterior of the first objective when deciding where to evaluate the second objective, or to learn the cost landscape (if the cost depends on the solution evaluated) on the fly.

The code for reproducing the results is available at `https://github.com/mbinois/DecoupledDoe`.

5 Acknowledgements

This report benefited from wider discussions within the "Surrogates" working group of the Dagsthul Seminar 23361 Multiobjective Optimization on a Budget. This group's members included Thomas Bäck, Mickaël Binois, Jürgen Branke, Jonathan Fieldsend, Ekhine Irurozki, Pascal Kerschke, Boris Naujoks, Robin Purshouse, Tea Tusar, Vanessa Volz, Hao Wang and Kaifeng Yang. For the purpose of open access, the authors have applied a Creative Commons Attribution (CC BY) licence to any Author Accepted Manuscript version arising from this submission. RCP's contribution was supported by SIPHER (MR/S037578/1), a UK Prevention Research Partnership funded by the UK Research and Innovation Councils, the Department of Health and Social Care (England) and the UK devolved administrations, and leading health research charities https://ukprp.org/.

References

1. Abdolshah, M., Shilton, A., Rana, S., Gupta, S., Venkatesh, S.: Cost-aware multi-objective Bayesian optimisation. arXiv preprint arXiv:1909.03600 (2019)
2. Allmendinger, R., Knowles, J.: Heterogeneous objectives: State-of-the-art and future research. arXiv arXiv:2103.15546 (2 2021)
3. Astudillo, R., Frazier, P.: Multi-attribute Bayesian optimization under utility uncertainty. In: Proceedings of the NIPS Workshop on Bayesian Optimization. vol. 172 (2017)
4. Binois, M., Ginsbourger, D., Roustant, O.: Quantifying uncertainty on Pareto fronts with Gaussian process conditional simulations. European Journal of Operational Research **243**(2), 386–394 (2015)
5. Binois, M., Picheny, V.: GPareto: An R package for Gaussian-process-based multi-objective optimization and analysis. Journal of Statistical Software **89**(1), 1–30 (2019)
6. Buckingham, J.M., Gonzalez, S.R., Branke, J.: Bayesian optimization of multiple objectives with different latencies. arXiv preprint arXiv:2302.01310 (2023)
7. Carnell, R.: lhs: Latin Hypercube Samples (2022), `https://CRAN.R-project.org/package=lhs`, r package version 1.1.6
8. Daulton, S., Balandat, M., Bakshy, E.: Hypervolume knowledge gradient: a lookahead approach for multi-objective Bayesian optimization with partial information. In: International Conference on Machine Learning. pp. 7167–7204. PMLR (2023)
9. Garrido-Merchán, E.C., Hernández-Lobato, D.: Predictive entropy search for multi-objective Bayesian optimization with constraints. Neurocomputing **361**, 50–68 (2019)
10. Hernández-Lobato, D., Hernandez-Lobato, J., Shah, A., Adams, R.: Predictive entropy search for multi-objective Bayesian optimization. In: International Conference on Machine Learning. pp. 1492–1501. PMLR (2016)
11. Iqbal, M.S., Su, J., Kotthoff, L., Jamshidi, P.: FlexiBO: A decoupled cost-aware multi-objective optimization approach for deep neural networks. J. Artif. Intell. Res. **77**, 645–682 (2023)
12. Lee, E.H., Eriksson, D., Perrone, V., Seeger, M.: A nonmyopic approach to cost-constrained Bayesian optimization. In: Uncertainty in Artificial Intelligence. pp. 568–577. PMLR (2021)

13. Lee, E.H., Perrone, V., Archambeau, C., Seeger, M.: Cost-aware Bayesian optimization. arXiv preprint arXiv:2003.10870 (2020)
14. Loka, N., Couckuyt, I., Garbuglia, F., Spina, D., Nieuwenhuyse, I.V., Dhaene, T.: Bi-objective Bayesian optimization of engineering problems with cheap and expensive cost functions. Engineering with Computers (1 2022)
15. M. D. Mckay, R.J.B., Conover, W.J.: A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. Technometrics **42**(1), 55–61 (2000)
16. Niederreiter, H.: Random number generation and quasi-Monte Carlo methods. SIAM (1992)
17. Snoek, J., Larochelle, H., Adams, R.P.: Practical Bayesian optimization of machine learning algorithms. Advances in neural information processing systems **25** (2012)
18. Suzuki, S., Takeno, S., Tamura, T., Shitara, K., Karasuyama, M.: Multi-objective Bayesian optimization using Pareto-frontier entropy. In: International Conference on Machine Learning. pp. 9279–9288. PMLR (2020)
19. Tu, B., Gandy, A., Kantas, N., Shafei, B.: Joint entropy search for multi-objective Bayesian optimization. Advances in Neural Information Processing Systems **35**, 9922–9938 (2022)

A Results if objectives have different lengthscales

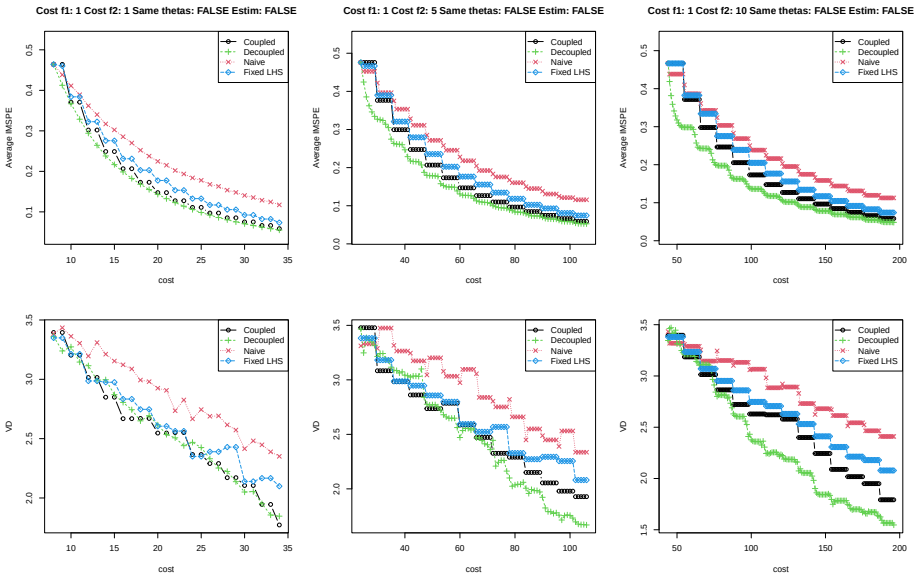


Fig. 7: Results of different metrics depending on the cost incurred. In the left column, both objectives have equal cost, in the middle column the cost for f_2 is 5 times as high, and in the right column, the cost for f_2 is 10 times as high. In this figure, lengthscales are equal and assumed known.

[6] observe that allowing decoupled evaluation of objectives is beneficial if the different objective functions have different lengthscales, i.e., if one objective is smooth and varying slowly, while the other is highly multimodal. Intuitively, one would like to allocate more samples to the more difficult objective. To test this, we have also run experiments where objectives have different lengthscales, in particular we used $(0.3, 0.4)$ for the first, $(0.4, 0.2)$ for the second objective. Results are summarised in Figure 7 for know lengthscales and Figure 8 for learned lengthscales. The results are very similar to the results with equal lengthscales reported above, which may be due to the fact that the lengthscales chosen were too similar to observe a significant difference.

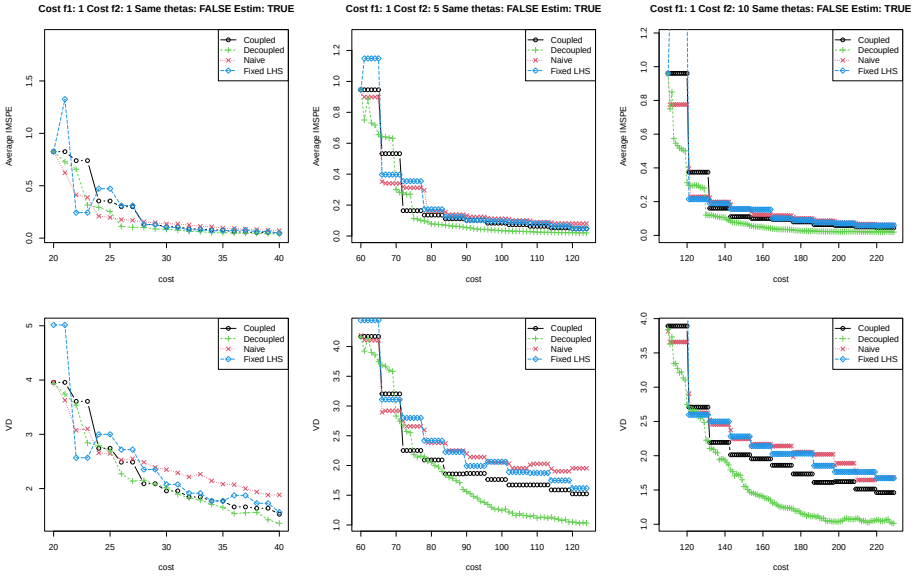


Fig. 8: Results of different metrics depending on the cost allocated. In the left column, both objectives have equal cost, in the middle column the cost for f_2 is 5 times as high, and in the right column, the cost for f_2 is 10 times as high. In this figure, lengthscales are equal but unknown (learned and in every iteration).

C2VRPTW: Assigning capacity to vehicles and nodes in a Vehicle Routing Problem for real-world delivery application

Cosimo Birtolo[1[0000-0002-7057-3416]] and Francesca Torre[1[0009-0000-4051-6536]]

[1] Poste Italiane, Digital, Technology and Operations, Rome, Italy
{birtoloc,francesca1.torre}@posteitaliane.it

Abstract. Minimizing the number of vehicles and the total time travelled in order to visit each delivery points, is a challenging task studied within the Vehicle Routing Problems and applied to several domains. In this paper, we introduce a multi-vehicle and multi-depot pickup and delivery problem with time constraints considering the capacities of both vehicles and logistic nodes. Our solution, structured on three-layer architecture, integrates information about sorting nodes and delivery nodes, travel distances, and demands (items to be delivered daily) allowing us to simulate arrival times at each destination points. We apply our model to a real-world scenario in the postal and logistics domain and we provide computational experiments to demonstrate the value of our approach. The model allows logistic designers to assess the impact of volumes and capacity changes on overall delivery times.

Keywords: Vehicle Routing Problem, Multi Depot, Google OR-Tools, Capacity and Time Windows, Digital twin, Logistic Network, MDCVRPTW.

1 Introduction

Nowadays companies in the postal and logistics domain offer different services to the citizens and the same-delivery is becoming an added-value services for e-Commerce and retailer's market. The evolution of the needs is leading to different scenarios by increasing the alternative delivery endpoints as lockers, post offices, retailers' networks as supermarkets and shopping malls or delivery points at jointly owned buildings. These alternatives delivery points added to the traditional customers' addresses are in place for reducing the delivery attempts, impacting couriers' costs and citizen satisfactions, and maximizing the delivery performance in terms of delivery time and delivery success at the first attempt, i.e., the First Time Delivery Success (FTDS) Index. In a previous work [1], we analyzed the fleet, and we clustered the vehicles according to their costs influenced by working conditions, followed routes, traffic found in urban cycle (stop-start traffic), and the type of vehicle (e.g., Fiat Panda, Fiat Punto, Fiat Doblò). The e-Commerce boom in post-COVID era leads new challenges and increased performances requested for the delivery steps as same-day delivery [2], as Amazon Prime, DHL SameDay, FedEx SameDay Delivery, and Poste Delivery Business

Express offer same-day delivery for eligible items in selected areas. Therefore, postal and logistic industries can benefit from novel decision support systems that can solve the problem described by considering the logistic infrastructure required for picking and delivery operations and ensuring working time slots and customers' delivery time windows. The literature discusses this class of decision problems as integrated order picking and vehicle routing problems. Our aim is to model this problem in three-layer delivery architecture from sorting to middle-mile delivery and to propose a tool able to simulate the delivery time considering volumes, logistics nodes and related capacity being a digital twin of the entire networks. In Section 2 we describe Vehicle Routing Problem (VRP) and related problems, in Section 3 we discuss the foundation of our model and in Section 4 the proposed approach for a real-world application. Next, we describe in Section 5 experimentation and computational results. Finally, we present our conclusions and some future directions.

2 Literature review

2.1 Heuristic and meta-heuristic approaches for solving VRP and its evolution

VRP is a type of transportation problem, which has many applications in real life such as logistics and transportation. This problem is an NP-hard that needs to find an optimal set of routes for serving a set of customers by a fleet of vehicles. It generalizes the Travelling Salesman Problem (TSP) by including different vehicles from the departure point and aims at minimizing the total cost for all the vehicle tours with a solution that consider the shortest path ensuring that each customer is visited exactly once by a vehicle. Christofides' algorithm [3], proposed in 1976, tackle the TSP, but its insights and principles have influenced approaches to solving the VRP when it is reduced to a series of TSP instances. The algorithm consists of four steps: (i) Find a minimum spanning tree of the graph, (ii) find a minimum weight perfect matching to form a multigraph, i.e., a graph where multiple edges between the same pair of vertices, are allowed, and (iv) build a Hamiltonian circuit (a closed tour that visit each vertex once) from the multigraph by skipping repeated vertices.

The goal of Capacity VRP (CVRP) is to determine a set of routes, each starting and ending at the warehouse, while adhering to a limited capacity constraints on each vehicle. In addition, delivery points may include Time Windows (TW) requests: meaning each customer should be visited by only one vehicle during a specified time interval, i.e., CVRPTW problem.

Time Windows constraints have been studied by Solomon and Desrosiers [4] that summarized the routing problems with time windows. On the other hand, for this complex VRP variants such as the VRP with Time Windows in cases where the time windows are relatively loose, adaptations of Christofides' algorithm can be particularly useful in constructing initial feasible solutions that are then improved upon using other optimization techniques as experimented by Ufuk Dereci and Muhammed Erkan Karabekmez [7] for a case study in Turkey.

Heuristics and meta-heuristics as Genetic Algorithms [5] and Tabu Search [6] have been proposed in the last decades to solve this class of NP-hard problem.
Moreover, the VRP can be Single Depot (SD) or Multi Depot (MD) [8] in case of the departure is a single fixed node or different points.
The focus of this research would be on Multi Depot Capacity Vehicle Routing Problem with Time Windows (MDCVRPTW). For the sake of simplicity, we call it Capacity Vehicle Routing Problem with Time Windows (CVRPTW) with MD. The key objective of MDVRPTW is to minimize the total cost, which can include various factors such as: (i) Total distance or travel time, (ii) Number of vehicles, (iii) Penalties for time window violations, and (iv) combination of costs including operational costs.
Recent research has focused on the MDVRP and optimization of vehicles and routes among multiple depots. Among solution studied, Rapanaki et al. [9] addressed the problem by means of the Artificial Bee Colony (ABC) algorithm, a meta-heuristic approach inspired by the behavior of the real honeybee colony. They compared this approach with other meta-heuristic such as Particle Swarm Optimization and Genetic Algorithms to prove the feasibility. More recently, in December 2023, P. Stodola and J. Nohel [10] explored the adaptation of Ant Colony Optimization (ACO) with Node Clustering. The algorithm was inspired by the behavior of ants in nature when searching for food and it is adopted in conjunction with node clustering for solving the minimization of total costs in MDVRP. Node clustering enhances the algorithm's performance because it is used in the phase of creating a solution when the algorithm searches the next node to be inserted into one of the routes of vehicles (ants).

2.2 Contribution to the state-of-the-art

The contribution of this paper is two-fold. First, we propose a model that assigns capacity to vehicles and nodes. The variability of the time spent in the logistic nodes has been modelled to make the algorithm more applicable in real-life where logistic nodes in the first-mile of delivery process are crucial hub of the network for the sorting activities. Moreover, this paper provides an overall architecture for first-mile and middle-mile delivery tasks for the postal sector. The proposed approach defines a three-layer architecture and solve the routing problem per layer keeping into account cost of the solution with the allocated fleet and time constraints.

3 C2VRPTW with Multi Depots

3.1 Formulation of CVRPTW problem

Every day the items to be delivered are collected at the first-mile depots and addressed to the first-mile destinations. The connections between nodes indicate the time needed from depot to destination, and the minimization of vehicle is classified as a VRP with multi-depots. The problem falls into the domain of CVRPTW as it requires that node has a fixed time slot to be served and each vehicle has a fixed capacity. Many formulations have been proposed for the CVRPTW. In particular, we refer to the review published by Solomon et al. [4] and the research works of Ursani ed al. [5] and Baldacci et

al. [11] which described the problem and its solution, focusing on the state-of-the-art of exact algorithms and meta-heuristics, respectively.

Formally, the CVRPTW with Multi Depots (MD) can be formulated as follows: Let $G = (\mathcal{N} \cup \mathcal{H}, \mathcal{A})$ be a directed graph whose node set is the union of a set $\mathcal{N}$ of customers and a set $\mathcal{H}$ of depots.

Non-negative costs $d_{i,j}$ are associated with each $arc(i,j) \in \mathcal{A}$, representing the travel cost, and $t_{i,j}$ is the travel time to reach $j \in \mathcal{N}$, starting from $i \in \mathcal{N}$, i.e., the difference between the arrival and the starting time of the route.

Each customer $i \in \mathcal{N}$ has a delivery demand q_i, a delivery time window $[a_i, b_i]$ and a vehicle must arrive at the customer before b_i. If it arrives before the time window opens, it has to wait until a_i to service the customer. Moreover, each customer i has a service time s_i^k that expresses the arrival time in the node i, with k $\in$ $\mathcal{K}$ that represents a set of vehicles. Each vehicle has a known capacity $Q_k \, \forall k \in \mathcal{K}$. In this version of the problem, we assume that all vehicles can be freely associated with any depot: this corresponds to the situation in which the location of the vehicles at the depots is a decision. The vehicles do not necessarily leave their depots at time 0: due to customer time windows, it may delay the departure time to arrive at customer locations within their time windows. The goal of this problem is the minimization of the total distance travelled by vehicles as described in Eq. (1), where x_{ij}^k expresses whether the vehicle *k* travelled from node *i* to node *j* and it takes the value 0 or 1.

$$\min \sum_{k \in K} \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}, j \neq i} d_{ij} \cdot x_{ij}^k \tag{1}$$

Subject to:

$$\sum_{k \in \mathcal{K}} \sum_{i \in \mathcal{N}, i \neq j} x_{ij}^k = 1 \quad \forall j \in \mathcal{N} \tag{2}$$

$$\sum_{k \in \mathcal{K}} \sum_{j \in \mathcal{N}, j \neq i} x_{ij}^k = 1 \quad \forall i \in \mathcal{N} \tag{3}$$

$$\sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}} q_i \cdot x_{ij}^k \leq Q_k \quad \forall k \in \mathcal{K} \tag{4}$$

$$x_{ij}^k \cdot (s_i^k + t_{ij} - s_j^k) \leq 0 \tag{5}$$

$$a_i \leq s_i^k \leq b_i \qquad \forall \mathrm{i} \in \mathcal{N}, \qquad \forall \mathrm{k} \in \mathcal{K} \tag{6}$$

Eqs. (2) and (3) constrain that each customer can be visited by only one vehicle. Eq. (4) ensures that the vehicle capacity is not exceeded. The Eq. (5) establishes the relationship between the vehicle departure time from a customer and its immediate successor. Moreover, constraints in Eq. (6) affirm that the time windows are observed. We assumed that the transportation cost of each vehicle depends on the travelled distance.

The transportation network is considered asymmetrical, that is the time spent from node *i* to reach node *j* can be different from the time spent from node *j* to node *i*.

3.2 Introducing node capacity in the problem

As described in the previous sections in the context of CVRPTW, capacity is typically assigned to vehicles rather than nodes. The fundamental reason for this is that capacity constraints are designed to limit the amount of goods or services a vehicle can carry or perform on a single route, ensuring that the solution is feasible in terms of real-world logistics capabilities. In this paper we formulate a real-world problem where the capacity needs to be assigned to the nodes too. The main goal is to model the first-mile sorting scenario, where items and goods are collected by the national postal operator, addressed to the national sorting centers that have the key task of routing it to destination sorting center. In this scenario the sorting centers act both as depots and customers, according to the goods flow: (i) the sorting center collects the items of its region and routes the items to the national sorting centers according to the destination of the items collected, (ii) the same sorting center receives the items from the other sorting centers for items to be delivered in its region.
In details, Fig. 1 depicts the solution of a traditional CVRPTW with 4 nodes (depot at node 1 and customers at node 2, 3 and 4). The minimization of the total cost allocating the depot in each node is the problem we consider as CVRPTW with MD.

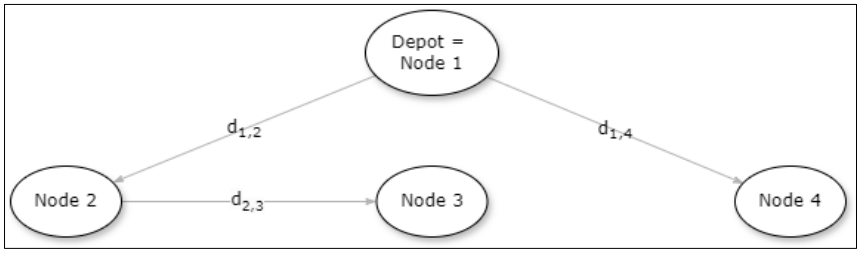


Fig. 1. CVRP with 4 nodes when Node1 is the depot, and the remaining 3 nodes are customers.

However, in the sorting scenario, each node has a dedicated sorting capacity that affects the total time spent. Therefore, this paper extends the CVRPTW problem by introducing capacity also in the nodes, defining it as C2VRPTW (i.e., VRP with Time Windows and Capacity constraints for both nodes and vehicles). For the solution, we model the problem by replacing every node *n* with three nodes n_{input_i} and n_{output_h}, where n_{input_i} are the nodes that represent the ingestion flows of a real-world logistic node, while n_{output_h} are the output of the logistic node in the fixed working time windows. These new nodes have their own time-windows, while the arc connecting n_{input_i} and n_{output_h} is the processing capacity of the logistic node. The vehicle cannot travel among n_{input_i}nodes or among n_{output_h}ones.
Fig. 2 depicts our approach for solving the C2VRPTW with 4 nodes as a CVRPTW with 12 nodes and having the distance $d_{i,j}$ as the distance between node i and node j and

representing the time travelled for reaching node *j* from node *i*, while d_{i_in1,i_out1} the time travelled within the node for processing the demand of node *i*. The output node within node *i* has been duplicated to take into account different time windows and available resource within the node *i*.

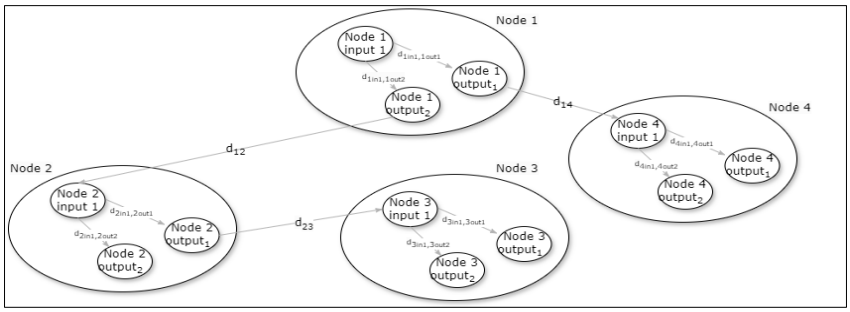


Fig. 2. C2VRP with 4 nodes when Node1 is the depot, and the remaining 3 nodes are customers.

In the real-word scenario the capacity of logistic node, that has been modelled by means of d_{i_in1,i_out1} , is evaluated according to available resources (employees and sorting machines) and depends to the processed demand of the node *i*. The distance matrix used by C2VRPTW is defined as follows:

$$D = \begin{cases} \text{elapsed working processing time in node i} & \text{if } d_{i_{in},i_{out}} \quad \forall i \in \mathcal{N} \\ \text{travel time from node i to node j} & \text{if } d_{i_{out},j_{in}} \quad \forall i,j \in \mathcal{N} \\ +\infty & \text{if } d_{i_{in},j_{out}} \quad \forall i \neq j \ \forall i,j \in \mathcal{N} \\ +\infty & \text{if } d_{i_{out},j_{out}} \quad \forall i \neq j \ \forall i,j \in \mathcal{N} \\ +\infty & \text{if } d_{i_{in},j_{in}} \quad \forall i \neq j \ \forall i,j \in \mathcal{N} \end{cases} \quad (7)$$

This definition of distance matrix allows the resolution of C2VRPTW with n nodes as a CVRPTW problem with $n \times 2$ nodes and distance *D*, considering the split of each node in input node and output node. For instance, let assume the depot node *A* (sorting center), that sends items to the node *B*. In our model, we split *A* in three different nodes: A_{input_i}, A_{output_1}, A_{output_2} , where A_{output_1}, A_{output_2} differ in terms of time windows. The arc connecting A_{input_i} and A_{output_1} represents the elapsed working time within the sorting center *A*, while the arc connecting A_{output_1} and B_{input_i} is the travel time from *A* to *B*. Then the CVRPTW model is applied.

4 Application of C2VRPTW to real-world logistic delivery problem

4.1 Three Layer Architecture Design

The main depots are the first level of architecture and send and receive items for their networks, in other words they work as pick-up and delivery points. Transshipment

nodes are intermediate depots for Delivery Centers, closer to the delivery area than the main depots. And finally, the third level is the delivery centers which are the collection points for customers.
This architecture can be applied at international level, where the first level is made by office of exchanges defined by each country for sending and collection items worldwide. The second level is national sorting centers which receive the delivery items from the arrival office of exchanges, while the third level are the delivery centers in charge of last-mile delivery. Similarly, in a national level the architecture has been applied as shown in Table 1, where the first-level is the connection among the sorting centers which collect regional items and goods to be delivered and address it to the target sorting center, the second level is the city or the facility acting as a transshipment node for its logistic area to serve, and finally for the third level, each transshipment node address the middle-mile segment to send the item to the target delivery center, the closest to the final addressee.

Table 1. Three level architecture for multi-depot VRP where each first-mile sorting centers collects items to send in Italy.

Routing level	Node	Example
Depot	Sorting Center at Origin	Bari
First-mile	Sorting Center at Destination	Florence
Intermediate Logistic Node	Transshipment Node	Lucca
Middle-mile end point	Delivery Center	Viareggio

Fig. 3 depicts the three layers, i.e., blue circle represents the Origin and Destination of the first level (i.e., multi-depot model), and it is called first-mile of the logistic network.

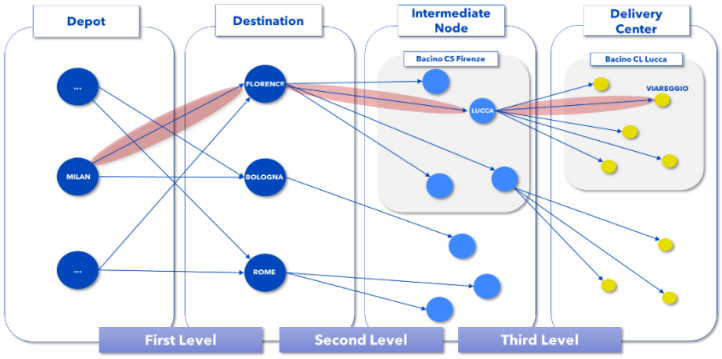


Fig. 3. Three-layer architecture highlighting the direct connections between layers for depot in Milan and destination in Viareggio with a transshipment in Lucca.

This level communicates with the second level of architecture (light blue nodes) in charge of regional distribution of item addressed to the target region. Each intermediate

node serves its third level targets that are the delivery centers allocated to each of transshipment node. In the figure, each level relates to direct connection that is not visible in terms of number of vehicles needed for the delivery tasks, that leads to the design of nested VRP problems. In this formulation, we do not consider the time that each vehicles requires to return to the depot because the fleet is managed to avoid the movement of empty vehicles, therefore in our model, each vehicle starts from a depot and there is no need to return to the same depot.

4.2 Addressing the problem for the different layers

The three-layer architecture describes the routing problem in a flexible way in order to address delivery challenges as the same-day delivery or cost optimization problem. Each layer can be modelled by means of a dedicated VRP formulation and differs for type of operations and vehicle adopted. In a national scenario, the first-mile is served by trucks and the sorting machines installed in the node must route the goods and items to the target region and city, for this reason we classify this first level by means of our proposed C2VRPTW with MD. Additional constraints in this level is that each node acting as a depot and at the same time as a customer node, must look for a trade-off between capacity allocated to sorting activities and capacity allocated to delivering received items. In other words, the total capacity of each node is allocated: (i) in sorting the items, when the node acts as a depot, and (ii) in routing the received items from the other nodes, when the node acts as a customer of the first level. These constraints have been introduced due to the machine installed in the node that cannot perform the two kinds of operations at the same time.

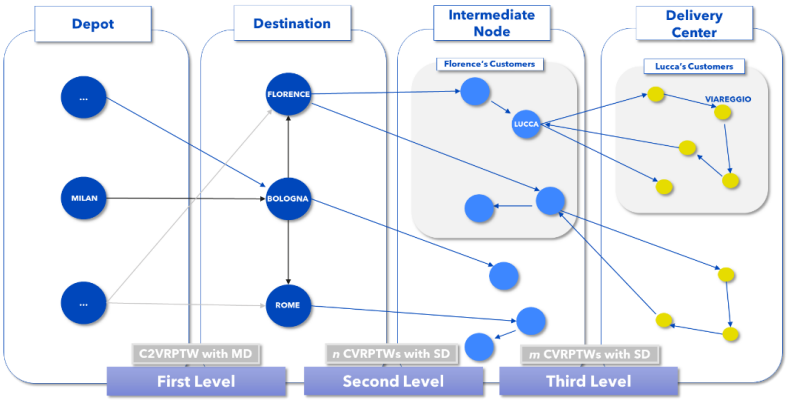


Fig. 4. Mapping of the three-layer architecture with three instances of VRPs.

The second level oversees connection of sorting center, acting as a depot, to the intermediate nodes which are customers without the processing or sorting needs. For this reason, we use for this level a simplified CVRPTW with SD solved by an exact

algorithm for the limited number of nodes involved. The last level considers the intermediate nodes as depot and the delivery centers as customers to be served.
The delivery centers have their own time windows, without routing needs and so for this level we can use a CVRPTW with SD. It is a single node because we consider for each intermediate nodes a set of customers to be served with their own vehicle (small tracks or vans). Therefore, the third level is solved by *m* problems where *m* is the number of transshipment nodes.

5 Experimental Results

5.1 Materials and methods

For our experimentation we consider 3 layers as described in Section 4.2 made by 10 sorting centers, 149 intermediate nodes and 715 delivery centers. The sorting center have their own capacity expressed in parcels per hour in sorting activities that is the set of activities performed for routing the items toward other regions and process the item received from the network for the area served. The first layer of architecture is modelled by the proposed C2VRPTW with MD in order to collect in each node the demand and route it to the target nodes. The second layer of architecture is depicted in Fig. 5 where each sorting center is connected to the transshipment nodes.
Finally, the last layer of the architecture is modelled by several VRPTWs with SD, one solution per transshipment node that serve a set of customers (i.e., the delivery centers of the logistic network).

Fig. 5. Sorting centers in yellow and the transshipment node in blue.

5.2 Software

The solution is implemented in Databricks using Python language. We adopt a cluster made by 8 cores and active memory of 28 GB with Databricks Runtime Version 14.3

LTS that includes Apache Spark 3.5.0, Scala 2.12. Moreover, we integrate Python libraries for the evaluation of distances between nodes and for adapting the VRP solvers, respectively TomTom library and Google OR-Tools [14]. OR-Tools allows executing heuristics and meta-heuristics to solve the VRP and has been developed by Google. CVRPTW is solved by heuristics and meta-heuristics algorithms, respectively. Coordinates, demands (items to be delivered), distance matrix, number and capacities of vehicles, and the sorting centers are inputs of our model. The algorithm implements a solution strategy, i.e., local search option, in order to generate a valid solution. Christofides' algorithm [3] is selected as initial solution strategy and ensures the generation of a feasible solution. Then, Guided Local Search (GLS) option tries to improve initial solution generated by Christofides' algorithm. GLS [15] is a general optimization technique suitable for a wide range of combinatorial optimization problems. In our implementation the local search is limited by a fixed number of iteration (avoiding the time limit due to possible reproduction of this experiments in a compute node with different performances) that refers to the maximum number of attempts for algorithm can search for each node.

5.3 Experiments

In this section we focus on the results, the data used in C2VRPTW model and the relative constraints. We collected the information related to: (i) the incoming volume i.e., the number of items picked up and delivered from each sorting center (items from origin to destination), (ii) the working time slot per node, (iii) the number and the types of machines installed in each sorting center in order to estimate the related capacity.
To address the real-world logistic capabilities of each center, we estimated the total time requested to process the incoming volume based on machines and human resources. The estimated total time is then directly proportional to the number of items to be delivered (or picked up), to the machine's type used to process them and the number of human resources in the specific sorting center as well.
Once we defined the distance matrix (see Eq. (7)), we developed the first layer using the C2VRPTW model applied to 10 fully connected sorting center. Additionally, we assumed that post offices and local senders collect the items by the 5 pm daily and these items are processed by each sorting center from 6 pm daily. The solution provides the routes and the arrival time in the network for each depot. In details, Fig. 6 shows the solution found for Milano PB sorting center, where the number of vehicles is 7, estimating the departure of the vehicle at 08:20 pm, due to the elapsed working time needed for sorting the items collected. Each vehicle has its own assigned route and for instance, the vehicle 0 depart from output node of Milano PB and serves two customer nodes: Firenze and Padova. The total elapsed time (i.e., from 6:00 pm of the first day to 9:26 am of the second day) includes the travelled time from Milano to Firenze and from Firenze to Padova and the time spent in the two crossed nodes too. Similarly, the solution provides the routes for the remaining 9 nodes acting as depots.

```
Vehicle 0:
MILANO PB_input1 H06:00pm -> MILANO PB_output1 H08:20pm -> FIRENZE_input1 H03:20am -> FIRENZE_output1 H03:37am ->
-> PADOVA_input1 H08:52am ->PADOVA_output1 H09:26am

Vehicle 1:
MILANO PB_input1 H06:00pm -> MILANO PB_output1 H08:20pm -> BOLOGNA_input1 H02:05am -> BOLOGNA_output1 H02:26am

Vehicle 2:
MILANO PB_input1 H06:00pm -> MILANO PB_output1 H08:20pm -> MILANO_input1 H04:35am -> MILANO_output1 H05:13am

Vehicle 3:
MILANO PB_input1 H06:00pm -> MILANO PB_output1 H08:20pm -> ROMA_input1 8:35 ->ROMA_output1 H08:48am ->
-> CATANIA_input2 H03:48am ->CATANIA_output2 H04:06am

Vehicle 4:
MILANO PB_input1 H06:00pm -> MILANO PB_output1 H08:20pm -> BARI_input1 H10:50am -> BARI_output1 H11:00am

Vehicle 5:
MILANO PB_input1 H06:00pm -> MILANO PB_output1 H08:20pm -> NAPOLI_input1 H09:35am > NAPOLI_output1 H09:49am

Vehicle 6:
MILANO PB_input1 H06:00pm -> MILANO PB_output1 H08:20pm -> TORINO_input1 H06:20am -> TORINO_output1 H06:47am
```

Fig. 6. Vehicles' route for first layer architecture from Milano PB sorting center

To deal with logistic constraints related to the non-parallelization of processing activities within the sorting centers, mainly due to a limited number of machines handling sorting jobs and the available human resources, we did not allow processing job into sorting centers when they are engaged in processing items coming from sorting center within the network. In other words, the time slot allocated to collecting the items and send them to the entire networks differs from the working time allocated for processing the incoming volume.

Analyzing the overall results, we monitor the behavior of each node and the related processing time. In

Fig. **7**, the dark gray area represents the elapsed working time for Firenze, during which it sorts items collected within its own area. The light gray area indicates the elapsed working time for Firenze when it processes the incoming volume from other sorting centers. This distinction highlights the advantages of the routing plan, which facilitates the serialization of processing procedures.

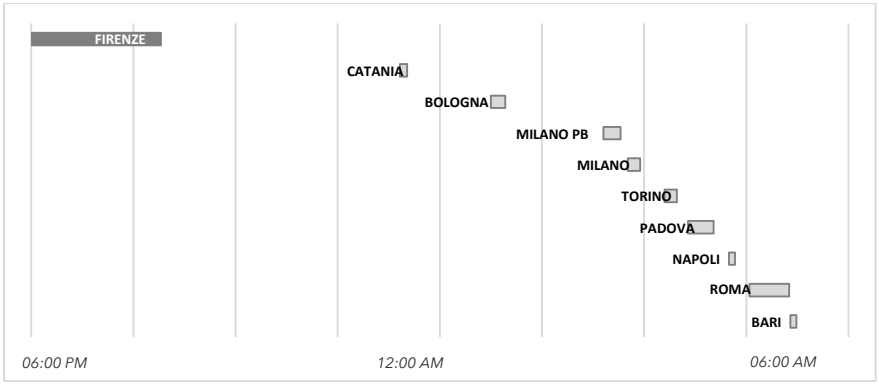


Fig. 7. Daily elapsed working time of sorting center in Firenze

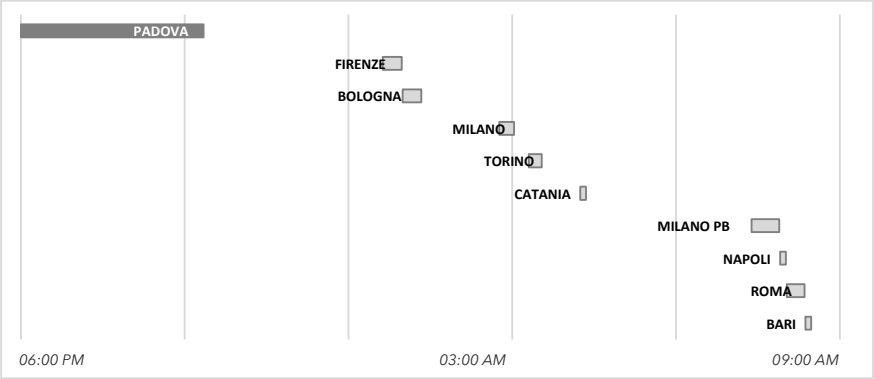


Fig. 8. Daily elapsed working time of sorting center in Padova

Similarly, Fig. 8 shows the elapsed working time processed by Padova daily. The processing steps are entirely sequential, allowing the incoming and outgoing items management at distinct time slot within a day.

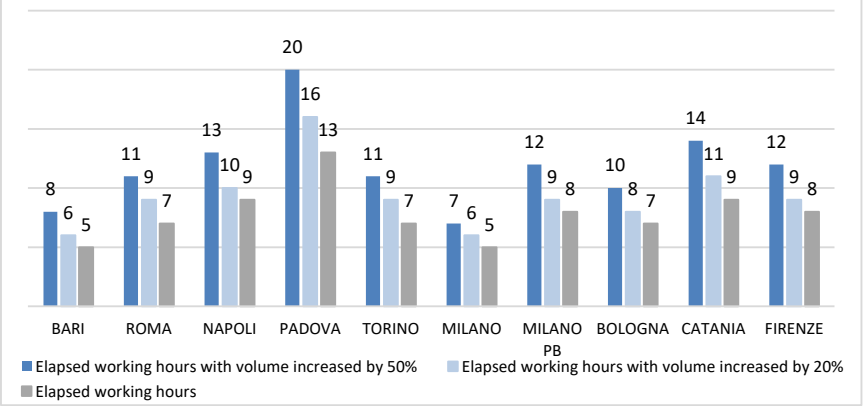


Fig. 9. Sorting centers' working time in three different scenarios

The model also allows the monitoring of sorting centers' working time, aiming at identifying centers where a volume increase may lead to delivery time increases. For this purpose, we run the model in three different scenarios: (i) a standard load, (ii) volumes with 20% of increase per each segment, and (iii) volumes increased of 50% compared to the annual average daily load. If we consider that a sorting center can perform sorting operations maximum 19 hours per day (leaving 5 hours for maintenance or administrative tasks), Fig. 9 shows that if the volume increase by 50%, Padova is not able to process them in a single day, resulting in an increase of the delivery times.

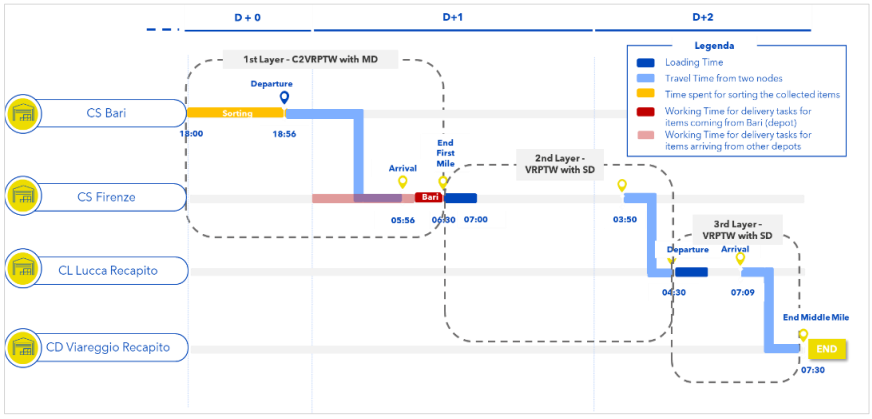


Fig. 10. Analysis of mail flow from Bari (depot) to Viareggio (end customer)

Our experimentation is extended to the second and third level of the proposed architecture as described in Section 4.2. As example of application (see Fig. 10), we run the VRPTW instance with SD, because the set of customers is fixed per each sorting center at destination (e.g., Firenze) and each transshipment node (e.g., CL Lucca Recapito). In the VRP settings of the third layer, we compared the pro and cons in terms of operational costs and elapsed time at varying the number of vehicles.

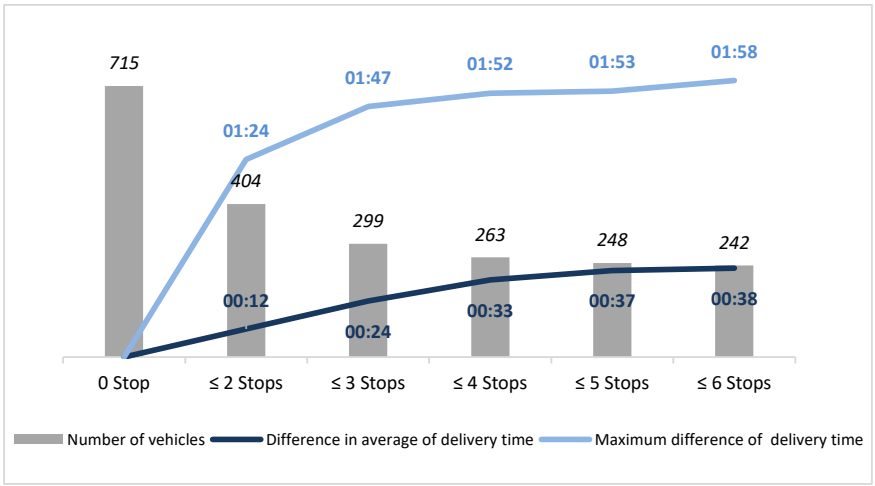


Fig. 11. Comparing the number of vehicles in the third level or architecture

Our aim is to measure the time spent on the different configurations, providing a tool able to suggest the optimal number of vehicles in order to address the middle mile delivery process. Running VRPTW (unlimited capacity per vehicle), we studied the

elapsed total time at varying the number of allowed stops in the routes and Fig. 11 depicts the number of vehicles needed for a direct connection within the third level of our architecture (715 vehicles, one per customer nodes), up to the minimum number of vehicles allowing some stops in the route and an increase of delivery tasks lesser than 2 hours (242 vehicles). Considering the average increase of the delivery times, the more the average elapsed times increases (up to 38 minutes), the fewer vehicles are requested (from 715 vehicles to 242 ones).

6 Conclusions and future works

In the vehicle routing problem with capacity and time window (CVRPTW), the objective is to minimize the number of vehicles with a fixed capacity and then minimize the total time travelled. In this paper we considered time constraints and the limited capacity of logistic nodes by introducing the C2VRPTW problem which extends the CVRPTW by means of the capacity constraints of each node, thus affecting the total time travelled.
This problem leads to more real-world models for certain applications, such as waste collection (where bins have a maximum capacity), delivery to warehouses with limited receiving docks, or services that involve equipment or personnel with limited availability at the customer site. In the logistic domain, the application is the modelling of the sorting centers and delivery centers.
To sum up, while classical CVRP formulations focus on vehicle capacities, we extended the problem to include node capacities and we provided a solution, generating a digital twin that incorporates information about sorting nodes, delivery nodes, travel distances, and daily demand. This allows the simulation of arrival times at each destination point. Computational experiments are presented to demonstrate the effectiveness of the proposed approach. The model allows logistic designers to evaluate the impact of volume and capacity changes on the final delivery times at the expected delivery points for the design of same-day delivery services. As further works, we propose an optimization engine for the network design by means of Genetic Algorithm approach [12][13] which provide promising results in optimization problems and in the VRP domain [5].

Acknowledgments. The authors would like to thank Poste Italiane's logistic experts and Lorenzo Gangemi, Silvia Ruffini and Erica Occhionero whose support made this work possible.

Disclosure of Interests. The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Birtolo C., Ronca D., Capasso G., Sorrentino G.: Clustering of Vehicle Usage Behavior by Means of Artificial Bee Colony. In: Rutkowski, L., Korytkowski, M., Scherer, R.,

Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds) Artificial Intelligence and Soft Computing. ICAISC 2014, LNCS, vol. 8467 pp 335-346. Springer, Cham (2014)
2. Schubert D., Kuhn H., Holzapfel A.: Same-day deliveries in omnichannel retail: Integrated order picking and vehicle routing with vehicle-site dependencies. Naval Research Logistics, **68** (6), 721–744 (2021)
3. Christofides. N.: Worst-case analysis of a new heuristic for the travelling salesman problem. Carnegie Mellon University (1976)
4. Solomon M. M., Desrosiers J.: Survey Paper—Time Window Constrained Routing and Scheduling Problems. Transportation Science **22** (1), 1-13 (1988)
5. Ursani Z., Essam D., Cornforth D., Stocker R.: Localized genetic algorithm for vehicle routing problem with time windows. Appl Soft Comput, **11** (8), 5375–5390 (2011)
6. Hedar A., Bakr A: Three Strategies Tabu Search for Vehicle Routing Problem with Time Windows. Computer Science and Information Technology **2** (2), 108–119 (2014)
7. Dereci U., Erkan Karabekmez M., The applications of multiple route optimization heuristics and meta-heuristic algorithms to solid waste transportation: A case study in Turkey, Decision Analytics Journal, **4**, (2022)
8. Bettinelli A., Ceselli A., Righini G.: A branch-and-cut-and-price algorithm for the multi-depot heterogeneous vehicle routing problem with time windows. Transportation Research Part C: Emerging Technologies, **19** (5), 723–740 (2011)
9. Rapanaki E., Psychas I., Marinaki M., Marinakis Y.: An Artificial Bee Colony Algorithm for the Multiobjective Energy Reduction Multi-Depot Vehicle Routing Problem. In: Matsatsinis, N., Marinakis, Y., Pardalos, P. (eds) Learning and Intelligent Optimization. LION 2019, LNCS, vol. 11968. Springer, Cham. (2020)
10. Stodola P., Nohel J.: Adaptive Ant Colony Optimization with Node Clustering for the Multidepot Vehicle Routing Problem. IEEE Transactions on Evolutionary Computation, **27** (6), 1866-1880 (2023)
11. Baldacci R., Toth P., Vigo D.: Exact algorithms for routing problems under vehicle capacity constraints. Annals of Operations Research, **175** (1) 213-245 (2010)
12. Troiano L., Birtolo C.: Genetic Algorithms supporting generative design of User Interfaces: Examples. Information Science **259**:433–451 (2014)
13. Troiano L., Birtolo C., Cirillo G., "Interactive Genetic Algorithm for choosing suitable colors in User Interface" In: 2009 Proceedings of Learning and Intelligent OptimizatioN Conference, LION3, Trento, Italy, January 14-18 (2009)
14. Furnon V., Perron L.: OR-Tools Routing Library v9.8. Google. https://developers.google.com/optimization/routing/ (2023)
15. Kilby P., Prosser P., Shaw P.: Guided local search for the vehicle routing problem, in: Proceedings of the Second International Conference on Metaheuristics (1997).

A constrained-JKO scheme for effective and efficient Wasserstein Gradient Flows

Antonio Candelieri[1][0000−0003−1431−576X], Andrea Ponti[1,2][0000−0003−4187−4209], and Francesco Archetti[1][0000−0003−1131−3830]

[1] University of Milano-Bicocca, Milan, Italy
{antonio.candelieri,francesco.archetti}@unimib.it
[2] OAKS srl, Milan, Italy
a.ponti5@campus.unimib.it

Abstract. We present an innovative Wasserstein Gradient Flow algorithm for solving optimization problems over the space of probability densities. Differently form other state of the art methods, the proposed approach does not involve any computationally expensive training of neural networks. The main contribution is a novel parametrization of the transport map that allows to recast the widely adopted JKO schema from a scalarized bi-objective to a constrained optimization problem. This provides three relevant advantages: a better control of the desired quality of approximation of the optimal transport map; convergence to the optimum without affecting the quality of the transport map; and a significantly lower computational cost (i.e., runtime-per-iteration). Experimental results on a set of well diversified test cases provide the empirical evidence of the advantages offered by the proposed approach.

Keywords: Wasserstein Gradient Flow · JKO · Optimal Transport

1 Introduction

1.1 Reference problem

The reference problem is the optimization of a function $\mathcal{F}(P)$ over the space of probability densities, $\mathcal{P}$, with support $\mathcal{X} \subseteq \mathbb{R}^d$. Formally, we search for:

$$P^* \in \arg\min_{P \sim \mathcal{P}} \mathcal{F}(P) \tag{1}$$

Generally speaking, optimizing means generating a sequence of solutions converging to the optimizer. In our setting this translates into searching for a sequence of probability densities starting from an initial random guess P_0 and converging to P^*, that is $P_0 \to P_1 \to ... \to P_k \to ... \to P^*$. A common choice is to set P_0 as the d-dimensional normal distribution.

Intriguingly, generating such a sequence is analogous to modelling diffusion processes across time, that is solving Partial or Stochastic Differential Equations (PDEs and SDEs). The most fundamental result was given in [5] by Jordan, Kinderlehrer, and Otto, who showed that solving the Fokker-Planck PDE

is equivalent to minimize an entropy functional over the space of probability measures, equipped with the so-called Wasserstein distance.

Wasserstein distance comes from the Optimal Transport (OT) theory [13, 14, 16]. In informal terms, it is the minimum cost for transporting probability density mass to transform a *source* probability density P into a *target* probability density P^*, where the cost is computed in terms of a *ground metric*, that is a distance between points of the support $\mathcal{X}$.

In this paper we consider the case that the ground metric is the Euclidean distance, leading to the so-called 2-Wasserstein distance:

$$\mathcal{W}_2^2(P, P^*) = \min_{T\natural P = P^*} \int_{\mathcal{X}} \|T(x) - x\|_2^2 \, dP(x) \tag{2}$$

where $T\natural P = P^*$ means that applying $T : \mathcal{X} \to \mathcal{X}$ to P will return P^* as result. Finally, the symbol $T_\natural$ denotes the so-called *push-forward* operator: while $T(x)$ is interpreted as a function moving a single data point over $\mathcal{X}$, $T_\natural$ represents its extension to an entire probability measure.

Importantly, under its assumptions, the Brenier Theorem [13] guarantees that the problem (2) has a unique optimal solution:

$$T^* = \underset{T:T\natural P = P^*}{\arg\min} \, \mathcal{W}_2^2(P, P^*) = \underset{T:T\natural P = P^*}{\arg\min} \int_{\mathcal{X}} \|T(x) - x\|_2^2 dP(x) \tag{3}$$

with T^* named *optimal transport map*. Moreover, the Brenier theorem also states that $T^*(x)$ is equal to the gradient of the convex function $\varphi(x) = \frac{\|x\|^2}{2} - f(x)$, with $f(x)$ the Kantorovich's potential from the dual of (3), therefore

$$T^*(x) = \nabla\varphi(x) = x - \nabla f(x) \tag{4}$$

By introducing a coefficient $\tau \in [0, 1]$, a *continuous-time* representation of the optimal transport map T^* is obtained:

$$T_\tau^*(x) = x - \tau\nabla f(x), \text{with } \tau \in [0, 1] \tag{5}$$

It is important to remark that the initial aim was to search for a sequence $P_0 \to P_1 \to ... \to P_k \to ... \to P^*$, therefore it is more convenient to consider the following *discrete-time* representation of T^*:

$$T_k^*(x) = x - \frac{k}{K}\nabla f(x), \text{with } k = 0, ..., K \tag{6}$$

where k/K is a discretization of τ from the continuous-time representation.

As a result, the sequence $P_0 = T_0^*\natural P_0 \to ... \to T_k^*\natural P_0 \to ... \to T_K^*\natural P_0 = P^*$ solves the reference problem (1), with T^* the solution of (3).

Another important theoretical result from OT is that the 2-Wasserstein distance is convex on $\mathcal{P}$ – this is why T^* is unique – meaning that P_0 is *transformed* into P^* by following the so-called Wasserstein Gradient Flow (WGF).

Consequently, searching for T^* by solving (3) translates into *convexifying* the reference problem (1), over the space $\mathcal{P}$ of probability densities equipped with the 2-Wasserstein distance.

Since P^* is not known a priori, T^* cannot be directly computed. However, it is still possible to *approximately* follow the WGF, as summarized in the following.

1.2 Wasserstein Gradient Flow via JKO

The JKO (Jordan–Kinderlehrer–Otto) scheme [5] is the standard algorithm to solve (1) by approximately following the WGF. It is a *proximal point method* with respect to the 2-Wasserstein distance, and can also be considered as a backward Euler discretization. At a generic iteration k, JKO transports the current probability density, P_k, into the next P_{k+1} according to:

$$P_{k+1} = \underset{P \sim \mathcal{P}}{\arg\min} \left\{ \frac{1}{2h} \mathcal{W}_2^2(P, P_k) + \mathcal{F}(P) \right\} \tag{7}$$

with h the JKO's step size.

Solving (7) means searching for the next probability density P_{k+1} which minimizes $\mathcal{F}$ while remaining close – in 2-Wasserstein terms – to the current P_k.

Iteratively solving (7) generates a sequence $P_0 \to ... \to P_k \to ... \to P_K \simeq P^*$ whose rate of convergence to P^* and *closeness* to the WGF both depend on h.

Indeed, with $h \to \infty$ the first term in (7) goes to zero, meaning that we are minimizing $\mathcal{F}$ completely ignoring the WFG. As a result, the obtained sequence will not comply with T^*. On the contrary, if $h \to 0$ then the first term becomes more relevant, so that the generated sequence will comply with T^* but it will converge significantly slowly to P^*.

Finally, (7) is formulated in terms of $P \in \mathcal{P}$, so it is necessary to find a way to deal with probability densities as decision variables. State-of-the-art approaches recast (7) as an optimization problem in terms of a *parametrized* transport map from P_k to P. Some parametrizations are described as follows.

1.3 Related works

The most widely adopted WGF approaches propose to parametrize $T(x)$ by approximating the convex function $\varphi(x)$ through an **Input Convex Neural Network** (ICNN) [7, 8, 12], a specific type of Deep Neural Networks (DNNs).

DNNs are also used in [9, 10] to parametrize $T(x)$ and the Kantorovitch potential $f(x)$ through two different networks. The first DNN provides an estimate of $f(x)$ which is then used as an input of the second DNN – along with x – to estimate $T(x)$. Since $T(x)$ is estimated by the second DNN, the gradient computation is avoided, contrary to the ICNN-based approaches.

More recently, [3] proposed to parametrize $T(x)$ through **a Residual Neural Network (ResNet)** and a variational formulation of the objective function formulated as maximization over a parametric class of functions. This allows for reducing computational burden, with respect to the previous methods, by using

small data samples and scaling well with the dimensionality d of the support $\mathcal{X}$. The small data regime setting has been also recently investigated in [1], who proposed to combine OT solvers and Gaussian Process regression to efficiently learn transportation map.

The most relevant issue, for all the neural approaches, is that at least one neural network must be trained, leading to relevant computational costs at each JKO steps. Since compliance to WGF requires $h \to 0$, this issue becomes even more critical. Moreover, there are not specific guidelines on how to choose the most suitable network: it is not only a matter of type (e.g., ICNN, DNN, ResNet), but also *architectural* (number and type of layers, number of units per layer, etc.).

Moreover, [6] has recently analyzed the expressive power of a class of normalizing flow models based on neural networks. These models can only generate *planar* flows, which are proved to be highly expressive only in the case of univariate probability densities (i.e., having support $\mathcal{X}$ with dimensionality $d = 1$). Otherwise, they may have poor approximation power. Another relevant study about the convergence of WGF is given in [2].

As far as the compliance to the optimal transport map T^* is concerned, the topic has been recently investigated in [11] – defined as *self-consistency* in the case of PDE solutions – and in [15], who introduced the so-called Monge Gap as a measure of how much a transport T deviates from the ideal properties expected by the optimal T^*. Specifically, they propose to use the Monge Gap as a regularization term for learning all the transportation maps.

1.4 Contributions

We summarize here the main contributions of this paper.

- A simple algebraic parametrization of T leading to a more effective, efficient, and explainable formalization of JKO: from a bi-objective to a constrained optimization problem.
- As belonging to the family of *radial* flows, the proposed parametrization overcomes most of the limitations of the *planar* flows induced by DNNs [6].
- A set of empirical results showing the benefits of the proposed approach, on a well-diversified set of experiments on bi-variate probability densities.
- A sketch of the extension of the approach to the case of multi-variate probability densities with $d > 2$.

2 The proposed approach

2.1 A simple algebraic parametrization of T

For the sake of simplicity we present our approach for $d = 2$ (i.e., bivariate probability densities) and then extend it to $d > 2$. Our parametrization is:

$$\widetilde{T}(x) = x - \lambda(x)\mathbf{v}\mathbf{R}_{\alpha(x)} \tag{8}$$

with $\lambda : \mathbb{R}^d \to \mathbb{R}_0^+$, $\alpha : \mathbb{R}^d \to [0, 2\pi]$, and $\mathbf{R}_\alpha$ a rotation matrix, such as:

$$\mathbf{R}_{\alpha(x)} = \begin{bmatrix} \cos(\alpha(x)) & -\sin(\alpha(x)) \\ \sin(\alpha(x)) & \cos(\alpha(x)) \end{bmatrix} \tag{9}$$

and where both clockwise and counter-clockwise rotation matrix can be used indistinctly. Finally, $\mathbf{v}$ is a reference versor: for simplicity we consider $\mathbf{v} = \left(\frac{1}{\sqrt{d}}\right) \mathbf{1}^d$, where $\mathbf{1}^d$ denotes the all-ones vector and, therefore, $\|\mathbf{v}\| = 1$.

From the Brenier theorem we know that $T(x) = x - \nabla f(x)$, so we are posing $\nabla f(x) = \lambda(x)\mathbf{v}\mathbf{R}_{\alpha(x)}$. The parameters to be learned in our parametrization are the two functions $\lambda(x)$ and $\alpha(x)$. This is in line with the other parametrization approaches which also approximate functions (i.e., $\nabla\varphi(x), f(x), T(x)$). Contrary to the ICNN based parametrizations, but analogously to [10, 9, 3], our approach does not require any assumptions on these two functions.

It is important to remark that the proposed parametrization is significantly close to the definition of radial flows reported in [6] and analysed along with planar flows induced by DNNs. Importantly, planar flows are proved to offer poor normalizing flows when $d > 1$.

Finally, according to the proposed parametrization, the JKO schema can be recast into:

$$\lambda_k(x), \alpha_k(x) = \arg\min \left\{ \frac{1}{2h} W_2^2(P_{k+1}, P_k) + \mathcal{F}(P_{k+1}) \right\} \tag{10}$$

with $P_{k+1} = \widetilde{T}_k \natural P_k$. In the next section we describe how our parametrization leads to a new – and more *effective, efficient*, and *explainable* – way to solve the reference problem (1).

2.2 From JKO to Constrained JKO

Exactly as the other DNN based approaches, we learn our parameters – that are the two functions $\lambda(x)$ and $\alpha(x)$ – by accessing to point clouds (i.e., empirical distributions) sampled from probability densities $P_0, ..., P_k$.

Analogously to [12], denote with $\mathbf{X}_k \sim P_k$ the point cloud at iteration k and, obtained as:

$$\mathbf{X}_k = \widetilde{T}_{k-1} \natural \left(\widetilde{T}_{k-2} \natural \left(... \natural \left(\widetilde{T}_0 \natural X_0 = \mathbf{X}_0 \right) \right) \right) \tag{11}$$

with $\mathbf{X}_0 \sim P_0$ and $|\mathbf{X}_0| = N$. According to our parametrized transport map $\widetilde{T}(x)$, every point of the current cloud $\mathbf{X}_k$ is transported as follows:

$$x_{k+1}^{(i)} = x_k^{(i)} - \lambda_k^{(i)} \, \mathbf{v} \, \mathbf{R}_{\alpha_k^{(i)}} \, , \; \forall i \in \{1, ..., N\} \tag{12}$$

with $\lambda_k^{(i)}$ and $\alpha_k^{(i)}$ shorthand for $\lambda(x_k^{(i)})$ and $\alpha(x_k^{(i)})$.

Since we are working with point clouds, and according to our parametrization, we can rewrite the 2-Wasserstein distance as follows:

$$\begin{aligned}\mathcal{W}_2^2(\mathbf{X}_{k+1}, \mathbf{X}_k) &= \frac{1}{N}\sum_{i=1}^{N} \|x_k^{(i)} - \lambda_k^{(i)} \mathbf{v}\mathbf{R}_{\alpha_k^{(i)}} - x_k^{(i)}\|_2^2 = \\ &= \frac{1}{N}\sum_{i=1}^{N} \sqrt{|\lambda_k^{(i)}|}\, \|\mathbf{v}\mathbf{R}_{\alpha_k^{(i)}}\|_2^2\end{aligned} \tag{13}$$

Since rotation does not modify the module of the rotated $\mathbf{v}$, and according to our choice for $\mathbf{v}$, we have $\|\mathbf{v}\mathbf{R}_{\alpha_k^{(i)}}\|_2^2 = 1$. Finally, we can write:

$$\mathcal{W}_2^2(\mathbf{X}_{k+1}, \mathbf{X}_k) = \frac{1}{N}\sum_{i=1}^{N} \sqrt{\lambda_k^{(i)}} \tag{14}$$

with $\lambda_k^{(i)} > 0$.

This result is crucial because $\mathcal{W}_2^2(\mathbf{X}_{k+1}, \mathbf{X}_k)$ is the first term of the objective function in the JKO scheme (7) – when point clouds are used as samples from probability densities in $\mathcal{P}$. Thus, we can directly set a threshold on it, instead of weighting its relevance with respect to $\mathcal{F}(P_{k+1})$ through the JKO's step h. More important, according to (14), we can now easily compute $\mathcal{W}_2^2(\mathbf{X}_{k+1}, \mathbf{X}_k)$, without the need to use expensive OT solvers (e.g., based on dual methods) or approximated methods.

The value of the threshold ε explicitly quantifies how much we want to comply with the WGF.

Thanks to our parametrizantion, and according to the previous considerations, we can now formalize our **constrained-JKO (cJKO)** scheme, that is:

$$\begin{aligned}\boldsymbol{\lambda}_k, \boldsymbol{\alpha}_k = \arg\min_{\substack{\boldsymbol{\lambda}\in\mathbb{R}_0^{+N}, \\ \boldsymbol{\alpha}\in[0,2\pi]^N}} & \mathcal{F}(P_{k+1}) \\ s.t. \quad & \frac{1}{N}\sum_{i=1}^{n} \sqrt{\lambda_k^{(i)}} \leq \varepsilon\end{aligned} \tag{15}$$

with ε a small positive quantity, $\boldsymbol{\lambda}_k = \left(\lambda_k^{(1)}, ..., \lambda_k^{(N)}\right)$, $\boldsymbol{\alpha}_k = \left(\alpha_k^{(1)}, ..., \alpha_k^{(N)}\right)$, and $\mathbf{X}_{k+1} = \left\{x_{k+1}^{(i)} = x_k^{(i)} - \lambda_k^{(i)}\mathbf{v}\mathbf{R}_{\alpha_k^{(i)}}\right\}_{i=:N}$.

In simpler terms, we are searching for $\mathbf{X}_{k+1} \sim P_{k+1}$ which minimizes $\mathcal{F}(P_{k+1})$ within a 2-Wasserstein neighbourhood of $\mathbf{X}_k$.

Analogies and (important) differences with JKO. The most important analogy is about ε, whose role is analogous to that of h. Indeed, $\varepsilon \to 0$ in cJKO should have the same effect of $h \to 0$ in JKO: the compliance to the WGF becomes more relevant than optimizing $\mathcal{F}(P)$, requiring a longer sequence to converge to P^* (i.e., $K \to \infty$). However, from a quantitative perspective, their impacts are completely different.

Simply, h is just a scalarization weight into a scalarized bi-objective problem. Clearly, it is difficult to establish a suitable value for h without any prior knowledge about $\mathcal{F}(P)$. Moreover, using a static value for h – that is the common choice – leads to a significantly different relevance of the compliance to the WGF along the iterative optimization scheme: it becomes larger with $\mathcal{F}(P)$ approaching to zero[3]. However, a poor compliance to WGF during the first JKO steps will definitely compromise the overall compliance.

The proposed cJKO scheme and its parameter ε allow to overcome these limitations. Indeed, the minimum level of compliance to the WGF (i.e., ε) can be chosen in advance and kept fixed along the overall iterative optimization process, independently on the values of $\mathcal{F}(P)$.

Figure 1 shows an example of the trajectories from $\mathbf{X}_0$ to $\mathbf{X}_K$ depending on three different values of ε. The three presented cases converges to the same value of $\mathcal{F}(X_K)$ (and same final point cloud) but within a different number of cJKO steps: the smaller the value of ε, the larger the number of iterations K, and the better the compliance to the WFG (i.e., trajectories are closer to the actual T^*, because they are more straight and do not overlap).

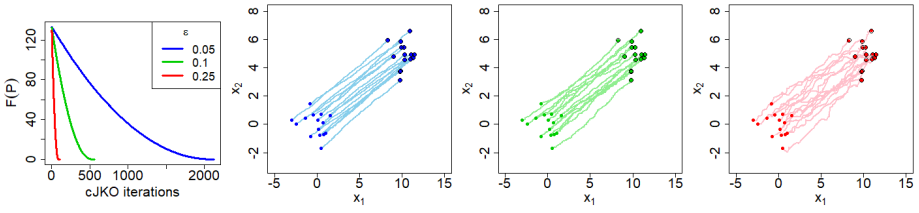


Fig. 1. An example of results from cJKO, for three different values of ε. In all the cases the minimum of $\mathcal{F}(P)$ is reached (left), within a different number of steps and a different compliance to the WGF (and to the actual T^*, consequently).

3 Experiments

3.1 Experimental setting

The experiments presented in this paper are devoted to demonstrate the higher effectiveness and efficiency of the proposed cJKO with respect to the standard JKO. For both the algorithms, the proposed parametrization of $T(x)$ is used.

[3] assuming $\mathcal{F}(P) \geq 0 \; \forall \; P \in \mathcal{P}$

Objective function $\mathcal{F}(P)$. We have considered two objective functions, separately. The first is exactly the 2-Wasserstein distance, that is $\mathcal{F}(P) = \mathcal{W}_2^2(P, P^*)$, while the second objective function is the symmetrised Kullback-Liebler divergence, that is $\mathcal{F}(P) = \text{sym}\mathcal{D}_{KL}(P||P^*)$, with

$$\text{sym}\mathcal{D}_{KL}(P||P^*) = \frac{\mathcal{D}_{KL}(P||P^*) + \mathcal{D}_{KL}(P^*||P)}{2} \tag{16}$$

and $\mathcal{D}_{KL}(A||B)$ the Kullback-Liebler divergence between A and B.

The aim is to investigate the capability of JKO and cJKO to comply with the WFG, when $\mathcal{F}(P)$ is exactly, and is not, the 2-Wasserstein distance.

Source and target distributions. Three different settings are analysed, separately for the two objective functions previously mentioned:

- from a Multivariate Normal (MVN) distribution to a different one,
- from a MVN distribution to a Gaussian Mixture, and
- from a Gaussian Mixture to a MVN distribution.

A graphical representation of the three case studies is reported in Figure 2.

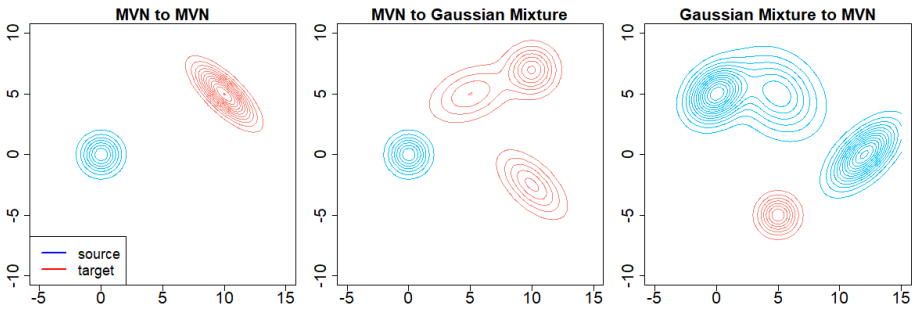


Fig. 2. The three different settings considered in the paper.

Configurations of the algorithms. Here, we summarize the setups of the two algorithms. They work with a point cloud sampled form the source distribution and consisting of $N = 15$ data points. Different values for the JKO's and cJKO's parameter are tested, specifically $h, \varepsilon \in \{0.5, 0.25, 0.1, 0.05, 0.01\}$. Termination criteria are: no improvement for 10 consecutive iterations (aka patience or early-stopping) or a maximum number of iterations reached, specifically 1000 for JKO and 5000 for cJKO (the difference will be motivated in the results section).

Finally, due to their different nature, two methods are separately used to solve the JKO and cJKO steps. Specifically, L-BFGS-B is used for JKO while COBYLA (Constrained Optimization BY Linear Approximation) is used for cJKO (from the NLOPT library).

The code has been developed in R and is freely available at the following repository: https://github.com/acandelieri/LION_2024_cJKO.git

3.2 Performance metrics

Quality of $\widetilde{T}(x)$ with respect to $\mathcal{F}(P^*)$. The most obvious way to measure the quality of a solution $\widetilde{T}$ is the associated value of the objective function.

Quality of $\widetilde{T}(x)$ with respect to P^*. It is aimed at measuring the quality of $\widetilde{T}$ in terms of closeness to P^*, and we have used the Frechét Inception Distance (FID) [4], that is the standard metric for assessing the quality of images created by a generative model. It compares the distribution of generated images with the distribution of a set of real images (i.e., ground truth). Interestingly, FID is nothing more than the 2-Wasserstein distance between data generated from the source distribution, through a given $\widetilde{T}$, and the actual target distribution:

$$FID(\widetilde{T}) := \mathcal{W}_2(\widetilde{T}\natural P_0, P^*) \tag{17}$$

With respect to our reference problem, FID quantifies the mismatch between our final solution, $\widetilde{T}\natural P_0$, and the actual optimizer P^*.

Monge Gap has been recently introduced to measure how far a given T deviates from the optimal one [15]. Thus, we decided to use it as a measure of compliance of every $\widetilde{T}$ obtained via JKO and cJKO. It is defined as:

$$\mathcal{M}_P(\widetilde{T}) = \int_{\mathcal{X}} \|\widetilde{T}(x) - x\| \, dP(x) \; - \; \mathcal{W}_2^2(P, \widetilde{T}\natural P) \tag{18}$$

Specifically, the first term is the transportation cost from P to $\widetilde{T}\natural P$ through $\widetilde{T}$, while the second is the optimal (i.e., minimum) transportation cost between P and $\widetilde{T}\natural P$. Denote with $\widetilde{T}^*$ the optimal transport from P to $\widetilde{T}\natural P$: the higher the value of $\mathcal{M}_P(\widetilde{T})$, the lower the compliance between $\widetilde{T}$ and $\widetilde{T}^*$.

Runtime is finally used to measure the efficiency of JKO and cJKO. Both the algorithms were run on the same system: Intel(R) Core(TM) i7-7700HQ, 2.80GHz, with 16.0 GB RAM, Microsoft Windows 10. No GPU has been used.

4 Results

4.1 Results for $\mathcal{F}(P) = \mathcal{W}_2^2(P, P^*)$

We start with the easiest case, with $\mathcal{F}(P)$ exactly the 2-Wasserstein distance from P^*. This means that both the algorithms are "forced" to follow the WFG; however, they show completely different behaviours.

Results are organized into three tables, one for each case study: from a MVN to another MVN (Table 1), from a MVN to a Gaussian Mixture (Table 2), and from a Gaussian Mixture to a MVN (Table 3).

Result #1. As expected, the Monge Gap $\mathcal{M}_{P_0}(\widetilde{T})$, for both the algorithms, decreases with the parameters – h and ε – decreasing, meaning that the compliance of $\widetilde{T}$ to the WGF increases.

Result #2. The most interesting result is related to the quality of the $\widetilde{T}$ provided by the two algorithms, with the quality measured both in terms of objective value and distance from the optimizer. It is important to remark that, in this case $\mathcal{F}(P) = \mathcal{W}_2^2(P, P^*)$, thus $FID(\widetilde{T}) = [\mathcal{F}(P)]^{\frac{1}{2}}$.

Specifically, the quality of the $\widetilde{T}$ provided by JKO gets worst with h decreasing, contrary to cJKO. The underlying motivation is the bi-objective nature of JKO, which searches for a trade-off between compliance to the WGF and minimization of $\mathcal{F}(P)$. On the contrary, cJKO minimizes $\mathcal{F}(P)$ while guaranteeing a minimum level of compliance to the WFG, at each step.

However, if the number of cJKO are not sufficient to converge, the quality of $\widetilde{T}$ will result dramatically poor (i.e., $\varepsilon = 0.001$ in all the experiments). This leads to the following result.

Result #3. cJKO requires more steps than JKO to converge to an optimal solution. Moreover, in the case that the maximum number of steps is prematurely reached, then the quality of $\widetilde{T}$ is significantly poor, due to the fact that the final P_K is somewhere in the middle of the trajectory defined by the optimal transport map. On the other hand, the computational efficiency of cJKO is significantly higher than JKO, as detailed in the next result.

Result #4. The average *runtime-per-iteration* is significantly different between the two algorithms, with 4.393 seconds for JKO and 0.178 seconds for cJKO. This means that for each JKO step cJKO performs, approximately, 23 steps. Thus, it is important to remark that 5000 maximum iterations assigned to cJKO are anyway few when compared to 1000 assigned to JKO. However, cJKO has always provided the best solution in this first session of experiments.

Table 1. Case study: **from MVN to another MVN**, $\mathcal{F}(P) = \mathcal{W}_2^2(P, P^*)$. The best solution in bold (i.e., lowest $\mathcal{F}(P)$, $FID(\widetilde{T})$, and $\mathcal{M}_{P_0}(\widetilde{T})$). The symbol * denotes that the maximum number of iterations was reached.

Method	Parameter	$\mathcal{M}_{P_0}(\widetilde{T})$	$\mathcal{F}(P_K)$	$FID(\widetilde{T})$	iterations	Runtime [secs]
JKO	$h = 0.5$	0.0752	0.0000	0.0018	59	201.602
	$h = 0.25$	0.0413	0.0000	0.0031	75	267.452
	$h = 0.1$	0.0369	0.0000	0.0055	168	563.958
	$h = 0.05$	0.0338	0.0001	0.0107	284	1023.167
	$h = 0.01$	0.0022	0.0008	0.0289	1000*	4664.409
cJKO	$\varepsilon = 0.5$	0.1050	0.0292	0.1709	52	8.777
	$\varepsilon = 0.25$	0.0172	0.0005	0.0227	118	21.167
	$\varepsilon = 0.1$	0.0066	0.0000	0.0044	559	105.415
	$\varepsilon = \mathbf{0.05}$	**0.0000**	**0.0000**	**0.0009**	**2138**	**370.080**
	$\varepsilon = 0.01$	0.0000	112.6710	10.6147	5000*	895.828

Table 2. Case study: **from MVN to a Gaussian Mixture**, $\mathcal{F}(P) = \mathcal{W}_2^2(P, P^*)$.

Method	Parameter	$\mathcal{M}_{P_0}(\widetilde{T})$	$\mathcal{F}(P_K)$	$FID(\widetilde{T})$	iterations	Runtime [secs]
	$h = 0.5$	0.1981	0.0000	0.0017	76	273.837
	$h = 0.25$	0.0650	0.0000	0.0028	83	341.697
JKO	$h = 0.1$	0.0972	0.0000	0.0049	185	652.702
	$h = 0.05$	0.0326	0.0001	0.0082	310	1300.729
	$h = 0.01$	0.0027	0.0006	0.0252	1000*	5119.579
	$\varepsilon = 0.5$	0.1491	0.0197	0.1404	52	10.778
	$\varepsilon = 0.25$	0.0738	0.0012	0.0351	149	28.969
cJKO	$\varepsilon = 0.1$	0.0036	0.0000	0.0031	628	127.741
	$\varepsilon = \mathbf{0.05}$	**0.0000**	**0.0000**	**0.0010**	**2471**	**540.053**
	$\varepsilon = 0.01$	0.0000	139.5640	11.8137	5000*	1090.61

Table 3. Case study: **from MVN to a Gaussian Mixture**, $\mathcal{F}(P) = \mathcal{W}_2^2(P, P^*)$.

Method	Parameter	$\mathcal{F}(P_K)$	$FID(\widetilde{T})$	$\mathcal{M}_{P_0}(\widetilde{T})$	iterations	Runtime [secs]
	$h = 0.5$	0.1387	0.0000	0.0018	64	238.925
	$h = 0.25$	0.0989	0.0000	0.0031	82	273.268
JKO	$h = 0.1$	0.0780	0.0000	0.0060	163	770.369
	$h = 0.05$	0.0751	0.0001	0.0099	381	1550.395
	$h = 0.01$	0.0006	0.0011	0.0338	1000*	4421.626
	$\varepsilon = 0.5$	0.2029	0.0503	0.2242	32	5.281
	$\varepsilon = 0.25$	0.0754	0.0012	0.0344	88	14.459
cJKO	$\varepsilon = 0.1$	0.0074	0.0001	0.0091	418	68.844
	$\varepsilon = \mathbf{0.05}$	**0.0011**	**0.0000**	**0.0011**	**1557**	**261.011**
	$\varepsilon = 0.01$	0.0000	70.5449	8.3991	5000*	837.707

4.2 Results for $\mathcal{F}(P) = \text{sym}\mathcal{D}_{KL}(P, P^*)$

Now we move to the more complicated experiment, in which we have considered $\mathcal{F}(P) = \text{sym}\mathcal{D}_{KL}(P, P^*)$. In this case the algorithms are not forced to follow the WGF because the objective function is not the 2-Wasserstein distance from P^*. Instead, they are now minimizing the symmetrised version of a divergence measure, that is not a distance over the space of probability density. Moreover, it is not convex on $\mathcal{P}$ – contrary to the 2-Wasserstein distance – and, consequently, it could have many local optima.

As in the previous section, results have been organized in three different tables, one for each case study: MVN to another MVN (Table 4), MVN to Gaussian Mixture (Table 5), and Gaussian Mixture to MVN (Table 6).

All the previous results (#1, #2, #3 and #4) still hold also in this setting. Furthermore, we can make other relevant considerations.

Result #5. Now, more frequently JKO terminates because the maximum number of steps is reached. This occurs when the source is a MVN, and especially if the target is a Gaussian Mixture.

Result #6. Runtime increased for both the algorithms, due to the computation of $\text{sym}\mathcal{D}_{KL}(P)$. The ratio between the runtime-per-iteration is unchanged: it is $7.151/0.291 = 25$ (against 23 in the previous experiment).

Table 4. Case study: **from MVN to another MVN**, $\mathcal{F}(P) = \text{sym}\mathcal{D}_{KL}(P, P^*)$. The best solution in bold (i.e., lowest $\mathcal{F}(P)$, $FID(\widetilde{T})$, and $\mathcal{M}_{P_0}(\widetilde{T})$). The symbol * denotes that the maximum number of iterations was reached.

Method	Parameter	$\mathcal{M}_{P_0}(\widetilde{T})$	$\mathcal{F}(P_K)$	$FID(\widetilde{T})$	iterations	Runtime [secs]
	$h = 0.5$	0.0217	0.0000	1.0779	103	652.321
	$h = 0.25$	0.0130	0.0000	1.1237	193	1227.840
JKO	$h = 0.1$	0.0145	0.0000	1.0660	484	2593.077
	$h = 0.05$	0.0029	0.0001	1.0769	1000*	5067.950
	$h = 0.01$	0.0000	0.1061	1.4073	1000*	6316.788
	$\varepsilon = 0.5$	0.1534	0.0019	1.2496	60	14.114
	$\varepsilon = 0.25$	0.0438	0.0000	1.2384	119	27.349
cJKO	$\varepsilon = 0.1$	0.0044	0.0000	1.0200	648	194.489
	$\varepsilon = \mathbf{0.05}$	**0.0029**	**0.0000**	**1.1358**	**2522**	**732.500**
	$\varepsilon = 0.01$	0.0000	124.1790	11.2725	5000*	1228.102

Table 5. Case study: **from MVN to a Gaussian Mixture**, $\mathcal{F}(P) = \text{sym}\mathcal{D}_{KL}(P, P^*)$.

Method	Parameter	$\mathcal{M}_{P_0}(\widetilde{T})$	$\mathcal{F}(P_K)$	$FID(\widetilde{T})$	iterations	Runtime [secs]
	$h = 0.5$	0.0498	2.0537	3.6999	792	3744.600
	$h = 0.25$	0.0341	2.0540	3.9952	1000*	5585.741
JKO	$h = 0.1$	0.0248	2.0561	4.0512	1000*	7514.572
	$h = 0.05$	0.0074	2.0869	3.8257	1000*	10331.930
	$h = 0.01$	0.0000	5.5040	7.5413	1000*	8185.894
	$\varepsilon = 0.5$	0.1811	2.0544	4.0787	59	21.892
	$\varepsilon = 0.25$	0.0422	2.0537	4.4647	183	60.722
cJKO	$\varepsilon = 0.1$	0.0080	2.0537	3.9348	1026	359.578
	$\varepsilon = \mathbf{0.05}$	**0.0010**	**2.0537**	**3.8310**	**3991**	1414.969
	$\varepsilon = 0.01$	0.0000	43.2750	12.6719	5000*	1718.191

Table 6. Case study: **from Gaussian Mixture to MVN**, $\mathcal{F}(P) = \text{sym}\mathcal{D}_{KL}(P, P^*)$.

Method	Parameter	$\mathcal{M}_{P_0}(\widetilde{T})$	$\mathcal{F}(P_K)$	$FID(\widetilde{T})$	iterations	Runtime [secs]
	$h = 0.5$	0.0867	0.0000	0.9249	56	357.089
	$h = 0.25$	0.0813	0.0000	0.8735	83	584.501
JKO	$h = 0.1$	0.0363	0.0000	0.9800	187	1405.229
	$h = 0.05$	0.0194	0.0000	1.0349	394	2844.871
	$h = 0.01$	0.0000	0.0015	1.1259	1000*	9204.589
	$\varepsilon = 0.5$	0.6064	0.0025	0.9240	35	8.667
	$\varepsilon = 0.25$	0.0925	0.0000	0.9608	90	22.147
cJKO	$\varepsilon = 0.1$	0.0117	0.0000	1.0054	405	89.379
	$\varepsilon = \mathbf{0.05}$	**0.0000**	**0.0000**	**0.9457**	**1594**	**428.945**
	$\varepsilon = 0.01$	0.0000	40.6964	8.5314	5000*	1316.771

4.3 Further results

To investigate the impact of the size N of the point cloud, we have repeated the experiments of the previous two subsections with $N = 30$ instead of $N = 15$. Due to page size limitations, we cannot report these results extensively, so we summarize the most relevant outcomes. First, the previous results are all confirmed. Second, a large number of steps should be used to achieve the same quality levels of the solutions. Third, the runtime-per-iteration resulted more than doubled for JKO, while the increase is smaller for cJKO.

Finally, we considered the DNN based WGF recently proposed in [12] – for which the code is freely available. We just addressed the case "from MVN to a Gaussian Mixutre" with $\mathcal{F}(P) = \text{sym}\mathcal{D}_{KL}$. We used the value for the JKO's step h suggested by the authors, while the maximum number of steps K was set to 40, 60, and 80. As expected, the objective value decreased with K increasing – even if it was significantly larger than cJKO (i.e., 4 to 6 times larger). On the other hand, the Monge Gap was always 0.000. Runtime was comparable to cJKO, specifically: 791.765, 1525.722, and 2558.914 seconds, for $K = 40, 60, 80$, respectively. However, the *runtime-per-iteration* is significantly larger than cJKO and increases with K: 1.994, 25.429, and 31.986 seconds-per-iteration. This is due to the need for repeatedly training a DNN on an dataset whose size increases with K. On the contrary, the runtime-per-iteration of cJKO does not depend on K, making our approach more computationally efficient. s

5 Extending to $d > 2$ dimensions.

We provide in this section the extension of our approach – and specifically our parametrization – to multivariate probability densities, with $d > 2$. At the time of writing, the code is not yet entirely developed for $d > 2$, and this is the main reason for the lack of experiments in higher dimensions.

Recalling the proposed parametrization (8), the only modification regards the rotation matrix $\mathbf{R}_{\alpha(x)}$. Indeed, the versor $\mathbf{v}$ was already defined for any $d > 0$.

In a space with $d > 2$ dimensions, rotation must be considered for each pair of axes, so $\alpha(x)$ is now a vector-valued function $\mathcal{A} : \mathbb{R}^d \to [0, 2\pi]^{(d-1)}$.

Consequently, the entries of the $d \times d$ rotation matrix $\mathbf{R}_{\mathcal{A}(x)}$ are now defined as explained in the following. Assume that $\mathcal{A}_p(x)$ is the angle between axes ℓ and κ, then we construct a rotation matrix for that angle, whose entries are:

$$R_{ij}^{[p]} = \begin{cases} \cos(\mathcal{A}_p(x)) & \text{if } (i,j) = (\ell,\ell) \\ -\sin(\mathcal{A}_p(x)) & \text{if } (i,j) = (\ell,\kappa) \\ \sin(\mathcal{A}_p(x)) & \text{if } (i,j) = (\kappa,\ell) \\ \cos(\mathcal{A}_p(x)) & \text{if } (i,j) = (\kappa,\kappa) \\ 1 & \text{if } i = j \ \wedge i \neq \ell,\kappa \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

Therefore, we have a matrix $R^{[p]}$ for each angle (i.e., $d-1$ matrices overall), and the final rotation is the concatenation of the single rotations, that is:

$$\mathbf{R}_{\mathcal{A}(x)} = \prod_{p=1}^{(d-1)} R^{[p]} \tag{20}$$

Interestingly, there is no need to modify $\lambda(x)$ and, consequently, there is not any impact on the computation of the constraint in (15).

On the other hand, there is a criticality related to the number of decision variables in (15): their number increases linearly with d, specifically it is given by $N + (N \times (d-1))$. However, it is important to remark that, when one works with small data samples, this number is way lower than the parameters to be optimized for training a DNN.

6 Conclusions

We have presented a novel transport map parametrization enabling a more effective, efficient, and explainable formulation of the JKO scheme for WFG. Specifically, we have presented the constrained-JKO (cJKO) scheme and evaluated it on a set of diversified case studies, all related to the optimization over a space of probability densities. The practical advantages are clearly quantified, in terms of quality of the solution as well as optimality of the transport map (i.e. Monge Gap). Moreover, number of iterations for converging to an optimal solution, run-time, and run-time-per iteration, have been used as efficiency measures, resulting significantly lower for cJKO with respect to JKO (with the same transportation map parametrization).

It is important to remark that the new parametrization, and the consequent cJKO formalization, represent a relevant leverage for all DNN methods, because DNN – as well as other Machine Learning algorithms – could be used to learn and model the two functions underlying the proposed prametrization (i.e., $\lambda(x)$ and $\alpha(x)$). Indeed, cJKO allows to significantly reduce the computational burden required to solve the standard JKO step.

Finally, future works will address experiments on $d > 2$ case studies.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Candelieri, A., Ponti, A., Archetti, F.: Generative models via optimal transport and gaussian processes. In: International Conference on Learning and Intelligent Optimization. pp. 135–149. Springer (2023)
2. Cheng, X., Lu, J., Tan, Y., Xie, Y.: Convergence of flow-based generative models via proximal gradient descent in wasserstein space. arXiv preprint arXiv:2310.17582 (2023)

3. Fan, J., Zhang, Q., Taghvaei, A., Chen, Y.: Variational wasserstein gradient flow. In: International Conference on Machine Learning. pp. 6185–6215. PMLR (2022)
4. Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S.: Gans trained by a two time-scale update rule converge to a local nash equilibrium. Advances in neural information processing systems **30** (2017)
5. Jordan, R., Kinderlehrer, D., Otto, F.: The variational formulation of the fokker–planck equation. SIAM journal on mathematical analysis **29**(1), 1–17 (1998)
6. Kong, Z., Chaudhuri, K.: The expressive power of a class of normalizing flow models. In: International conference on artificial intelligence and statistics. pp. 3599–3609. PMLR (2020)
7. Korotin, A., Egiazarian, V., Asadulaev, A., Safin, A., Burnaev, E.: Wasserstein-2 generative networks. arXiv preprint arXiv:1909.13082 (2019)
8. Korotin, A., Li, L., Solomon, J., Burnaev, E.: Continuous wasserstein-2 barycenter estimation without minimax optimization. In: International Conference on Learning Representations (2021)
9. Korotin, A., Selikhanovych, D., Burnaev, E.: Kernel neural optimal transport. arXiv preprint arXiv:2205.15269 (2022)
10. Korotin, A., Selikhanovych, D., Burnaev, E.: Neural optimal transport. arXiv preprint arXiv:2201.12220 (2022)
11. Li, L., Hurault, S., Solomon, J.M.: Self-consistent velocity matching of probability flows. Advances in Neural Information Processing Systems **36** (2024)
12. Mokrov, P., Korotin, A., Li, L., Genevay, A., Solomon, J.M., Burnaev, E.: Large-scale wasserstein gradient flows. Advances in Neural Information Processing Systems **34**, 15243–15256 (2021)
13. Peyré, G., Cuturi, M., et al.: Computational optimal transport. Center for Research in Economics and Statistics Working Papers (2017-86) (2017)
14. Peyré, G., Cuturi, M., et al.: Computational optimal transport: With applications to data science. Foundations and Trends® in Machine Learning **11**(5-6), 355–607 (2019)
15. Uscidda, T., Cuturi, M.: The monge gap: A regularizer to learn all transport maps. arXiv preprint arXiv:2302.04953 (2023)
16. Villani, C.: Topics in optimal transportation, vol. 58. American Mathematical Soc. (2021)

MLE-free Gaussian Process based Bayesian Optimization

Antonio Candelieri[1][0000−0003−1431−576X] and Elena Signori[1]

University Milano-Bicocca, 20126, Milan, Italy
antonio.candelieri@unimib.it
https://en.unimib.it/antonio-candelieri
elena.signori@unimib.it

Abstract. Gaussian Process based Bayesian Optimization, also known as *kernelized* bandits, is the standard method for sequentially optimizing black-box and expensive to evaluate functions. Its sample efficiency is the reason behind its success. However, due to the need for fitting a Gaussian Process model at each iteration, its own computational cost cubically increases with the number of function evaluations, while instability possibly arises especially in the noise-free setting. Moreover, selecting the most suitable trade-off mechanism between exploration and exploitation is not trivial, and theoretical convergence proofs for well-known acquisition functions only hold under the (impractical) assumption of knowing in advance the specific Reproducing Kernel Hilbert Space which the objective function belongs to. We propose a novel method for fitting the Gaussian Process at a constant time, independently on the number of function evaluations, while also dealing with instability and exploitation-exploration trade-off. This leads to a novel Bayesian optimization framework in which modelling of the objective function and acquisition are simultaneously addressed by fitting the Gaussian Process. Empirical results on a preliminary set of well differentiated test problems show that the proposed algorithm provides a higher effectiveness and efficiency than the most well-known no regret algorithm (i.e., GP-LCB).

Keywords: Bayesian Optimization · Gaussian Process · no-regret.

1 Introduction

Bayesian Optimization (BO) [1, 17] is used to solve real-life problems having black-box and expensive to evaluate objective functions. Relevant examples arise in Automated Machine Learning [13, 19, 21], robotic [4, 28], drug discovery [14].

The basic BO algorithm consists of two nested loops. The *outer loop* is devoted to query the objective function, collect the associated observation(s), and consequently fit a *probabilistic surrogate model* of the objective function. The *inner loop* is devoted to identify the next promising query by optimizing an *acquisition function* which balances exploration and exploitation, given the current

surrogate. Intuitively, it is easy to associate the outer and the inner loop respectively to *learning* (an approximation of the objective function) and *optimizing* (the choice of the next query to do). Not surprisingly, BO resulted really close to humans' strategies when the assigned task is to search for the optimum [10].

Gaussian Process (GP) regression [26, 18] is the most common choice for the surrogate model, leading to GP-based BO (also known as *kernelized bandits*). Obviously, BO is convenient when the computational cost of each iteration of the outer loop is lower than querying the objective function. However, the cost of GP-based BO cubically increases with the number of queries, due to the GP learning step.

From a *"human perspective"* this is quite counter-intuitive: *the more I know about the problem the more expensive is to learn further... how is it possible?.*

There is a "computational reason": as a kernel method, the GP's core operation is the inversion of a kernel matrix, with complexity $\mathcal{O}(n^3)$, where n is the number of observations. Moreover, most of the GP learning methods require many matrix inversions to tune the kernel's hyperparameters depending on data.

Importantly, it was not a critical issue for the success of BO. Indeed, BO is *sample-efficient*, meaning that it can find a better solution than other methods given a small number of queries, typically 15-20 times the number of dimensions of the problem [1, 17]. Although this claim holds up to 20 dimensions[1], it was the leverage for the GP-based BO's success: if few queries are allowed, the cost for learning the GP is negligible w.r.t. that for querying the objective function.

Unfortunately, computational cost/complexity is not the only issue: the *condition number* of the kernel matrix increases with n, leading to instability and then the impossibility to invert it. The most common workaround consists into adding *artificial* noise (also known as *nugget*) to avoid instability at the expense of a poorer approximation. As discussed later, noise is crucial for convergence proofs, which indeed require to assume $n \to \infty$ and sub-Gaussian noise (Lemma 1 in [5]).

Related works. In the BO literature, the idea of setting the GP kernel's hyperparameters via strategies alternative to common methods like Maximum Likelihood Estimation (MLE) or Maximum A Posteriori (MAP), is not new. However, they started from a different motivation, that is convergence analysis.

The most widely GP-based BO algorithm is [22], widely quoted for being *no regret.*[2] However, its theoretical convergence proof holds only in the standard setting [7], in which one assumes a *realizable* (i.e., well-specified) scenario, meaning that the objective function is a member of a specific Reproducing Kernel Hilbert Space (RKHS) with bounded norm, known a-priori. In other terms, convergence to the optimum is proved under the assumption that the GP kernel's hyperpa-

[1] Problems in more than 20 dimensions are addressed through High Dimensional Bayesian Optimization (HDBO) approaches, recently surveyed in [6].

[2] Denote with $R_N = \sum_{i=1}^{N} \left(f(x^{(i)}) - f(x^*) \right)$ the *regret* accumulated by an algorithm generating N solutions. The algorithm has no-regret if $\lim_{N\to\infty} \frac{R_N}{N} \to 0$.

rameters are known in advance, that is not the case in practice. Otherwise GP kernel's hyperparameters must be tuned but, how demonstrated in [8], this leads no regret algorithms to get stuck into local optima. Heuristic methods [24, 25, 23, 5] have been proposed over time, basically using the Lemma 4 of [8], stating that decreasing the *length-scale* hyperparameter of the kernel leads to consider a larger RKHS, increasing exploration at the expense of a higher regret [17].

As far as instability of the kernel matrix is concerned, [27] proposed a BO method selecting the next query driven by both the exploration-exploitation trade-off and the minimization of the kernel matrix's condition number.

Contributions. We propose a new method for learning a GP model at an approximately constant time, that is $\mathcal{O}(1)$, instead of $\mathcal{O}(n^3)$, also avoiding the ill-conditioning of the kernel matrix. Specific innovations and contributions are: (*i*) a more efficient outer loop of BO; (*ii*) avoiding ill-conditioning, so that there is no need to inject artificial noise when the objective function is noise-free; and (*iii*) a balance between exploration-exploitation directly through the GP's predictions (i.e.,avoiding other acquisition functions). Experimental results on a set of ten diversified test problems empirically show the benefits of our method.

2 Background

2.1 GP-based BO

Without loss of generality, we refer to the following global optimization problem:

$$\mathbf{x}^* \in \underset{\mathbf{x}\in\mathcal{X}\subset\mathbb{R}^d}{\arg\min} f(\mathbf{x}) \tag{1}$$

with $f(\mathbf{x})$ black-box and expensive to evaluate, and $\mathcal{X}$ a box-bounded search space. In this paper we consider $\mathcal{X} \triangleq [0,1]^d$, which any box-bounded domain can be scaled to.

BO generates a sequence of queries $\mathbf{X}_{1:n} = \left\{\mathbf{x}^{(i)}\right\}_{i=1:n}$ and collects the associated observations $\mathbf{y}_{1:n} = \left\{y^{(i)}\right\}_{i=1:n}$, with $y^{(i)} = f\left(\mathbf{x}^{(i)}\right) + \varepsilon^{(i)}$ in the case that $f(\mathbf{x})$ is *noisy*. Usually, it is assumed $\varepsilon^{(i)} \sim \mathcal{N}(\mathbf{0}, \lambda^2)$.

The next query $\mathbf{x}^{(n+1)}$ is obtained by optimizing an acquisition function, balancing between exploration and exploitation. We consider the well-known and widely used Lower Confidence Bound (LCB) [22], leading to:

$$\mathbf{x}^{(n+1)} \in \underset{\mathbf{x}\in\mathcal{X}}{\arg\min}\ \mu(\mathbf{x}) - \beta^{\frac{1}{2}}\sigma(\mathbf{x}) \tag{2}$$

where $\mu(\mathbf{x})$ is the prediction for $f(\mathbf{x})$ provided by the GP trained on $(\mathbf{X}_{1:n}, \mathbf{y}_{1:n})$, the term $\sigma(\mathbf{x})$ is the uncertainty associated to the prediction, and β regulates the *uncertainty bonus*: higher values incentive exploration, small values increase exploitation. We specifically discuss about the scheduling of β in the next.

The equations for the GP's predictive (also known as posterior) mean $\mu(\mathbf{x})$ and uncertainty $\sigma(\mathbf{x})$ are:

$$\mu(\mathbf{x}) = m(\mathbf{x}) + \mathbf{k}(\mathbf{x}, \mathbf{X}) \left[\mathbf{K} + \lambda^2 \mathbf{I}\right]^{-1} (\mathbf{y} - m(\mathbf{X})) \tag{3}$$

$$\sigma^2(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}) - \mathbf{k}(\mathbf{x}, \mathbf{X}) \left[\mathbf{K} + \lambda^2 \mathbf{I}\right]^{-1} \mathbf{k}(\mathbf{X}, \mathbf{x}) \tag{4}$$

where $m(\mathbf{x})$ is the prior mean (usually, set to zero), $\mathbf{k}(\mathbf{x}, \mathbf{X}) = \mathbf{k}(\mathbf{X}, \mathbf{x})^\top$ is a vector of kernel-based similarities between $\mathbf{x}$ and every other $\mathbf{x}^{(i)} \in \mathbf{X}$, and $\mathbf{K}$ is an $n \times n$ kernel matrix with entries $K_{ij} = k\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right)$. Finally, λ^2 is the variance of a zero-mean Gaussian noise if $f(\mathbf{x})$ is noisy and/or additional noise is injected because $\mathbf{K}$ cannot be inverted.

For our analysis, we consider the well-known Squared Exponential (SE) kernel, defined as $k(\mathbf{x}, \mathbf{x}') = e^{-\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2\ell^2}}$, with ℓ its *length-scale* hyperparameter.

2.2 Gaussian Process learning

GP learning consists into tuning the kernel's hyperparameters depending on a dataset, that in BO is the set of queries $(\mathbf{X}_{1:n}, \mathbf{y}_{1:n})$. This is typically done by maximizing MLE with respect to the kernel's hyperparameters, where:

$$MLE \triangleq -\frac{1}{2}\mathbf{y}^T \left[\mathbf{K} + \lambda^2 \mathbf{I}\right]^{-1} \mathbf{y} - \frac{1}{2}\log|\mathbf{K} + \lambda^2 \mathbf{I}| - \frac{n}{2}\log 2\pi \tag{5}$$

and $|\cdot|$ denotes the determinant. Clearly, only $\mathbf{K}$ depends on the kernel's hyperparameters (i.e., ℓ in our case), whose value affects both the first and the second term of (5). They are, respectively, a matrix inversion and a determinant computation. The three terms have different roles: only the first term involves the observed function values (i.e., it is associated to the quality of fit), the second can be interpreted as a complexity penalty depending on the kernel (both kernel type and hyperpramaters' values), and the third is a normalization constant [26].

Alternatively, MAP can be considered, also known as *penalized* MLE. The only difference is that the likelihood is weighted with respect to some prior.

Maximization of MLE as well as MAP can be addressed through gradient-based or derivative free algorithms. In any case, they require to evaluate many values for the kernel hyperparameters and, consequently, to perform many matrix inversions and determinant computations, both with complexity $\mathcal{O}(n^3)$. Moreover, MLE in GP is known to be ill-posed, as also recently analysed in [20].

2.3 Kernel's hyperparameters and exploitation-exploration trade-off

In GP-based BO with LCB as acquisition function – compactly, GP-LCB – the scheduling for β is given by (Theorem 2 from [22]):

$$\beta = 2\log(n^2 2\pi^2/(3\delta)) + 2d\log(n^2 dbr\sqrt{\log(4da/\delta)}) \tag{6}$$

which provides theoretical guarantee of convergence to the optimum only in the standard setting (also known as realizable or well-specified scenario). Otherwise,

tuning the kernel's hyperperameters is needed, leading GP-LCB to get stuck into local optima [8, 5], despite the (logarithmic) increasing of the uncertainty bonus (6).

In [5], experiments show that, contrary to GP-LCB, the proposed approach always converges to the optimum, even if it provides a higher, but eventually sub-linear, cumulative regret. Indeed, the main idea consists into adapting the hyperparameters – instead of tuning them via MLE or MAP – to increase exploration at the expense of a larger cumulative regret. This is also the underlying idea of previous methods, such as the schedules proposed in [23–25] aimed at iteratively decreasing ℓ. This is in line with Lemma 4 from [8], stating that when decreasing the length-scale, the resulting function space contains the previous one, that is we are considering progressively larger RKHSs [5].

In other terms, all these methods aim at increasing the exploration with respect to that provided by (6) whenever a miss-specified scenario is targeted. The miss-specification is so common that, reasonably, is the reason why other recent works on new acquisition functions report that GP-LCB is almost always outperformed by new methods, such as [2, 3, 16].

Finally, the need for switching to exploration has been also reported in [11, 12] about human search strategies: when searching for the optimum, subjects move towards pure random search whenever they perceive that there is a negligible chance to improve the current best solution. This is in contrast with cumulative regret, which should be more suitable for tasks aimed at "accumulating scores over time", instead of searching for the best score overall. Indeed, it is more related to multi-armed bandit than global optimization. On the other hand, cumulative regret is the basic concept which convergence proofs of BO algorithms are currently based on [17].

In [22] it is proved that the cumulative regret of GP-LCB is $\mathcal{O}(\sqrt{n\beta^{(n)}\gamma^{(n)}})$, where the quantity $\gamma^{(n)}$, also named *information capacity* [5], is:

$$\gamma^{(n)} = \max_{\mathbf{X}_{1:n} \in \mathcal{X}} \mathcal{I}(\mathbf{y}_{1:n}; f) \tag{7}$$

that is the largest amount of *mutual information* that can be obtained by any algorithm from at most n observations. Intriguingly, for GPs the mutual information only depends on queries $\mathbf{X}_{1:n}$ and not the associated observations $\mathbf{y}_{1:n}$:

$$\mathcal{I}(\mathbf{y}_{1:n}; f) \triangleq \frac{1}{2} \log |\mathbf{I} + \lambda^{-2}\mathbf{K}| \tag{8}$$

In the standard setting (also known as realizable or well-specified scenario) equations (7) and (8) lead to a finite bound for the the regret, given the scheduling of β in (6). However, in miss-specified scenarios this bound does not hold any longer.

Moreover, by rewriting the second term of MLE by using (8) one easily obtains:

$$\begin{aligned} MLE \triangleq &= -\frac{1}{2}\mathbf{y}^T \left[\mathbf{K} + \lambda^2\mathbf{I}\right]^{-1}\mathbf{y} - \frac{1}{2}\log\left|\lambda^2(\lambda^{-2}\mathbf{K} + \mathbf{I})\right| - \frac{n}{2}\log 2\pi = \\ &= -\frac{1}{2}\mathbf{y}^T \left[\mathbf{K} + \lambda^2\mathbf{I}\right]^{-1}\mathbf{y} - \frac{1}{2}\log\left(\lambda^2\left|\lambda^{-2}\mathbf{K} + \mathbf{I}\right|\right) - \frac{n}{2}\log 2\pi = \\ &= -\frac{1}{2}\mathbf{y}^T \left[\mathbf{K} + \lambda^2\mathbf{I}\right]^{-1}\mathbf{y} - \frac{1}{2}\log(\lambda^2) - \frac{1}{2}\log\left|\lambda^{-2}\mathbf{K} - \mathbf{I}\right| - \frac{n}{2}\log 2\pi = \\ &= -\frac{1}{2}\mathbf{y}^T \left[\mathbf{K} + \lambda^2\mathbf{I}\right]^{-1}\mathbf{y} - \frac{1}{2}\log(\lambda^2) - \mathcal{I}(\mathbf{y}_{1:n}; f) - \frac{n}{2}\log 2\pi \end{aligned} \tag{9}$$

Thus, tuning the kernel hyperparameters by maximizing MLE implies the minimization of the mutual information, leading a GP possibly biased towards exploitation. Indeed, [15] has proposed a GP-based BO algorithm based with maximization of mutual information as an acquisition function.

When in the noise-free setting artificial noise has to be added to avoid ill-conditioning of the kernel matrix $\mathbf{K}$, λ is tuned along with the kernel's hyperparameters by maximizing (9). As a result, switching from $\lambda = 0$ to $\lambda > 0$ could drastically result into a more exploitative GP, along the BO iterations.

3 The proposed approach

Starting from the previous considerations, we propose a new GP-based BO algorithm **for the noise-free setting** that learns a GP at a constant time and also avoids ill-conditioning, without the need for injecting noise (i.e., $\lambda = 0$, always).

3.1 Looking at a GP depending on the length-scale

First, we report an illustrative one-dimensional example to briefly recall the role of ℓ on the predictive mean $\mu(\mathbf{x})$ and uncertainty $\sigma(\mathbf{x})$ of a GP with a SE kernel (Figure 1). When the length-scale is too small (i.e., $\ell = 0.001$, in the example), $\mu(\mathbf{x}) = m(\mathbf{x})$ everywhere but at the observations ($m(\mathbf{x}) = \mathbf{0}$, in the example). Then, $\mu(\mathbf{x})$ becomes smoother with ℓ increasing (i.e., $\ell = 0.01$ and $\ell = 0.1$), while $\sigma(\mathbf{x})$ decreases. Predictive uncertainty almost completely disappears for larger length-scale ($\ell = 1$) up to the rising of instabilities (with $\ell = 1.1$ $\mu(\mathbf{x})$ becomes *jiggling*). Finally, higher values ($\ell = 2$) bring to ill-conditioning of $\mathbf{K}$, requiring to inject artificial noise. However, as a result this leads, in many cases, to a poor-approximation of the true function (just like in the example).

Our idea is to estimate an ℓ^* close to instability, but avoiding jiggling, so that the resulting $\mu(\mathbf{x})$, alone, can suitably balance exploitation and exploration.

Intriguingly, we are moving in the opposite way with respect to Lemma 4 from [8]: we are not interested in considering larger RHKSs at each iteration, but the smallest possible one for that specific iteration. Only in the case that it could be impossible to estimate ℓ^*, then we automatically select the largest possible RKHS (i.e., the smallest positive value for ℓ^*). In the next, we explain how ℓ^* can be estimated at constant cost, contrary to the $\mathcal{O}(n^3)$ of MLE.

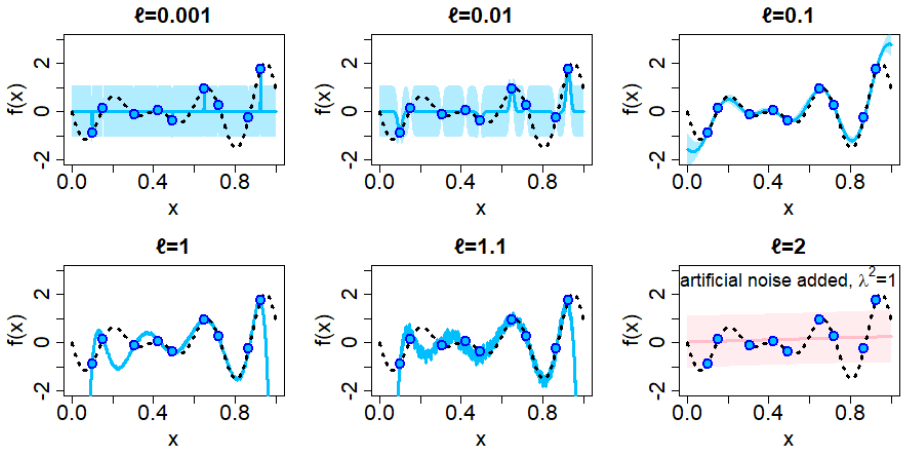


Fig. 1. Effect of the SE kernel length-scale on the GP model.

3.2 Estimating the condition number to fit a GP at a constant time

Denote with $\mathbb{k}$ the condition number of $\mathbf{K}$, with instability arising when $\mathbb{k}$ becomes too large. According to our previous considerations, we want to find:

$$\begin{aligned} \ell^* = & \max_{\ell \in (0, \sqrt{d}]} \mathbb{k} \\ & s.t. \quad \mathbb{k}^{-1} > \tau \end{aligned} \tag{10}$$

with τ the numeric precision of the specific computational system. Unfortunately, the cost for calculating $\mathbb{k}$ is still $\mathcal{O}(n^3)$, due to the need for computing the eigenvalues of $\mathbf{K}$. Our idea is to create a model predicting the inverse of $\mathbb{k}$, specifically $\log_{10}(\mathbb{k}^{-1})$, for different values of ℓ and depending on $\mathbf{D}$, that is the $n \times n$ matrix with entries $D_{ij} = \|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|$. Obviously, $D_{ij} = D_{ji}$ and $D_{ii} = 0$; moreover, $0 \leq D_{ij} \leq \sqrt{d}$, since we are considering $\mathcal{X} = [0,1]^d$.

To make clear the relation among ℓ, $\mathbf{D}$, and $\mathbb{k}$, remind that by denoting with ρ the quantity $\|\mathbf{x} - \mathbf{x}'\|$ one can rewrite the SE kernel as $k(\rho) = e^{-\frac{\rho^2}{2\ell^2}}$, then $\mathbf{K} = k(\mathbf{D})$, with the kernel function applied element-wise to $\mathbf{D}$. However, at each BO iteration the size of $\mathbf{D}$ increases, and it is not practical to have a predictive model for every possible n. Instead, we propose to collapse $\mathbf{D}$ into a vector $\mathbf{w} \in [0,1]^H$, with $\sum_{h=1}^{H} w_h = 1$ (i.e., $\mathbf{w}$ is in the unitary simplex), and:

$$w_h = \begin{cases} |\{D_{i>j} : 0 \leq D_{ij} \leq h\sqrt{d}/H\}|, & \text{if } h = 1 \geq 1 \\ |\{D_{i>j} : (h-1)\sqrt{d}/H < D_{ij} \leq h\sqrt{d}/H\}|, & \text{otherwise} \end{cases}$$

where $H = N(N-1)/2$, and N is the maximum number of BO iterations.

With $n \in \{n_0, ..., N\}$, n points are sampled in $\mathcal{X}$, and the associated matrices $\mathbf{D}^{(n)}$ and vectors $\mathbf{w}^{(n)}$ are computed, along with $\mathbb{k}^{(n)}$. Then, a nonlinear support

vector regression model $\mathcal{M}_d$ is trained such that $\mathcal{M}_d(\mathbf{w}, n, \ell) \simeq \log_{10}(\mathbb{k}^{-1})$.

Importantly, $\mathcal{M}_d$ is generated once but used for any problem in $\mathcal{X}$. Indeed, $\mathcal{M}_d$ only depends on data points, not on any associated function values, because instability is due to too close points in $\mathcal{X}$.

At a generic BO iteration, $\mathbf{w}^{(n)}$ is computed from $\mathbf{X}_{1:n}$, and ℓ^* is obtained by rewriting (10) as:

$$\begin{aligned} \ell^* = & \min_{\ell \in (0, \sqrt{d}]} \mathcal{M}_d(\mathbf{w}^{(n)}, n, \ell) \\ & s.t. \quad \mathcal{M}_d(\mathbf{w}^{(n)}, n, \ell) > \log_{10}(\tau) \end{aligned} \tag{11}$$

If the problem (11) is infeasible, at a certain BO iteration, than ℓ^* is set to the smallest positive value, that is $\ell^* = \tau$. Figure 2 shows some examples.

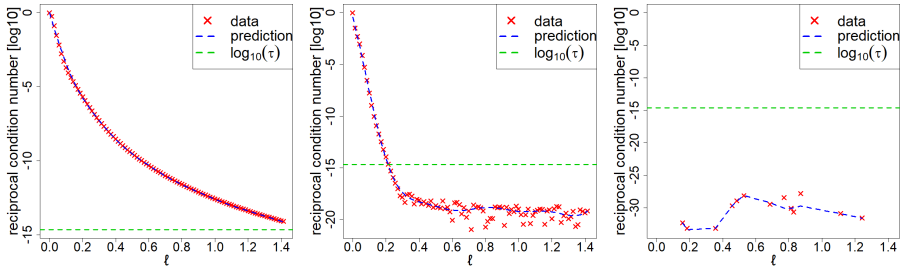


Fig. 2. $\log_{10}(\mathbb{k}^{-1})$ predicted through $\mathcal{M}_d$ (with $d = 2$), for different ℓ and for fixed $\mathbf{w}$ and n. Three different situations: $\ell^* = \sqrt{d}$, $\ell^* < \sqrt{d}$, and $\ell^* = \tau$ (i.e., (11) is infeasible).

4 Experiments

4.1 Test problems

We have selected a set of test problems to easily control and validate the behaviour of the proposed method against GP-LCB. Specifically, we have selected 5 one-dimensional and 5 two-dimensional well-known test problems from the *Infinity77* repository[3] and the *Simon Fraser University's Virtual Library of Simulation Experiments: Test Functions and Datasets*[4].

The aim was to define a well diversified set of test problems (depicted in Figure 3 and Figure 4), in terms of smoothness, location and number of optimizer(s). The search spaces of all the test problems have been preliminary scaled to $[0, 1]^d$.

[3] https://infinity77.net/global_optimization/test_functions.html
[4] https://www.sfu.ca/ ssurjano/

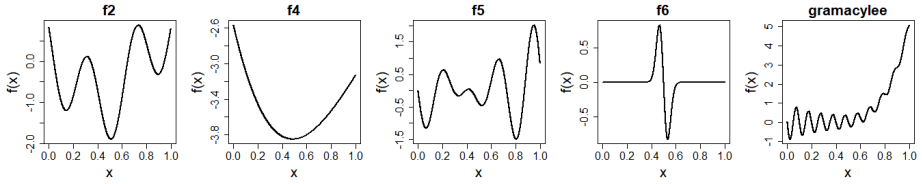


Fig. 3. One-dimensional test problems: f2, f4, f5, and f6 are from the Infinity77 repository, while gramacylee is from the Simon Fraser University's repository. The search spaces of all the problems have been preliminary scaled to $[0, 1]$.

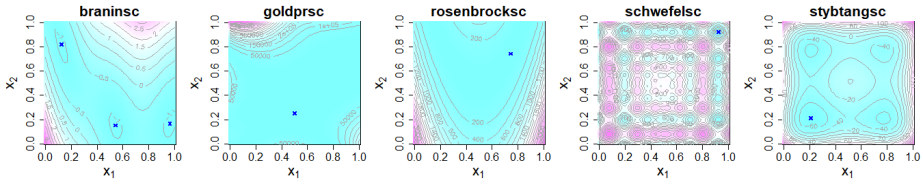


Fig. 4. Two-dimensional test problems, scaled to $[0, 1]^2$. Optimizers are marked in blue.

4.2 Performance metrics

Here, we briefly describe the performance metrics adopted in our study.

Gap metric is a measure of the effectiveness and efficiency of any sequential global optimization algorithm. It is defined as:

$$G^{(n)} = \frac{y^{+(1)} - y^{+(n)}}{y^{+(1)} - f(\mathbf{x}^*)} \tag{12}$$

with $y^{+(n)}$ the best value after n queries (*best seen*), that is $y^{+(n)} = \min_{i=1:n} \{y^{(i)}\}$.

Interestingly, $G^{(n)}$ ranges from 0 to 1 and can be therefore used to fairly compare different algorithms, also on different problems. Charting $G^{(n)}$ with respect to $n \in \{1, ..., N\}$ leads to the Gap metric curve: the closer and the faster the Gap goes to 1, the more effective and efficient is the algorithm.

Area Under the Gap metric Curve (AUGC) has been recently used [9] to quantify, into a unique value, the information of a Gap curve:

$$AUGC = \sum_{n=1}^{N} G^{(n)} \tag{13}$$

Discrepancy from Uniform distribution is used as a measure of exploration. It quantifies how far a data sample (i.e., the queries $\mathbf{X}_{1:n}$) deviates from a perfectly uniform distribution. Although many discrepancy measures are available,

we have decided to use the L2-discrepancy because it has an analytical form:

$$D_{L2}(\mathbf{X}_{1:n}) = \left[\int_{[0,1]^{2d}} \left(\frac{A(\mathbf{X}_{1:n}; J_{ab})}{n} - Vol(J_{ab}) \right)^2 da\, db \right]^{1/2} \tag{14}$$

where the term $A(\mathbf{X}_{1:n}, J_{ab})$ is the number of queries falling into the subset $J_{ab} \in [0,1]^d \triangleq [a_1, b_1) \times ... \times [a_d, b_d)$, with $a, b \in [0,1]^d$.

Time saving is simply computed as the ratio between the wall-clock time of the proposed MLE-free GP-BO algorithm and GP-LCB. Time saving for GP learning, optimization of the acquisition function, and total will be reported.

4.3 Experimental settings and results

Experiments are aimed at comparing the proposed MLE-free GP-BO against GP-LCB. They share the same setting: isotropic SE kernel (with max amplitude $\sigma_f^2 = 1$), $n_0 = 5d$ initial queries randomly sampled via LHS (but ensuring that $\|\mathbf{x}^{(i)} - \mathbf{x}^*\| > 0.1, \forall i = 1, ..., n_0$), and $N = 40d$ queries overall.

To mitigate randomness, we performed 100 independent runs: for each run, the two methods share the same set of n_0 randomly sampled queries.

Acquisition functions (i.e., just $\mu(\mathbf{x})$ for MLE-free GP-BO) are optimized via L-BFGS-B with 10 restarts.

Experiments were run on an Intel(R) Core(TM) i7-7700HQ @2.80GHz, 16GB RAM (Win10 Pro 64bits); R version 4.2.3 (2023-03-15 ucrt) "Shortstop Beagle".

Result 1. In terms of AUGC (**Table 1**) there is not a clear winner on the one-dimensional test problems. On the other hand, MLE-free GP-BO shows significantly higher AUGC values on the two-dimensional problems, meaning that it is more effective and efficient (i.e., converges closer to the optimum and in less iterations) than GP-LCB.

Table 1. Area Under the Gap Curve (AUGC)

d	**Test problem**	**MLE-free BO**	**GP-LCB**	**U test p-value**
1	f2	**31.87** (2.85)	30.25 (2.00)	<0.001 ***
	f4	**34.99** (0.02)	34.91 (0.18)	<0.001 ***
	f5	25.96 (7.00)	**27.58** (03.83)	0.210
	f6	18.04 (9.11)	**23.96** (6.43)	<0.001 ***
	gramacyleesc	**13.54** (9.42)	7.69 (6.83)	<0.001 ***
2	braninsc	**59.74** (7.67)	39.66 (15.26)	<0.001 ***
	goldprsc	**40.74** (17.68)	23.38 (18.91)	<0.001 ***
	rosenbrocksc	**63.53** (4.27)	53.71 (4.20)	<0.001 ***
	schwefelsc	**27.62** (13.82)	8.20 (10.39)	<0.001 ***
	stybtangsc	**64.50** (4.77)	52.71 (4.16)	<0.001 ***

Result 2. The higher AUGC values of MLE-free are motivated by a higher exploration, that is a significantly lower L2-discrepancy from the Uniform distribution (**Table 2**). The lower the L2-discrepancy, the higher the exploration.

Table 2. Discrepancy from Uniform distribution

d	Test problem	MLE-free BO	GP-LCB	U test p-value
1	f2	**0.04936** (0.01525)	0.21768 (0.00639)	<0.001 ***
	f4	**0.05974** (0.01735)	0.28804 (0.00027)	<0.001 ***
	f5	**0.04709** (0.01303)	0.16720 (0.01100)	<0.001 ***
	f6	**0.04567** (0.01133)	0.05015 (0.01385)	<0.001 ***
	gramacyleesc	**0.04687** (0.01225)	0.11301 (0.02733)	<0.001 ***
2	braninsc	**0.01969** (0.00442)	0.04752 (0.00182)	<0.001 ***
	goldprsc	**0.02115** (0.00605)	0.02599 (0.00196)	<0.001 ***
	rosenbrocksc	**0.03342** (0.00676)	0.04021 (0.00199)	<0.001 ***
	schwefelsc	**0.02288** (0.00497)	0.06070 (0.00278)	<0.001 ***
	stybtangsc	**0.02901** (0.00467)	0.05875 (0.00313)	<0.001 ***

Result 3. Importantly, the time saving given by MLE-free GP-BO is quite astonishing (**Table 3**), especially in the light of the better performances provided by MLE-free GP-BO against GP-LCB. To provide a more complete overview, we remark that, on 1D test problems, MLE-free's runtime is, on average, 0.5 seconds, against 15 seconds for GP-LCB. As far as 2D test problems are concerned, MLE-free requires a runtime between 3 and 10 seconds, depending on the problem, against 83 to 190 seconds required by GP-LCB.

Table 3. Wall-clock time ratio: MLE-free BO's runtime / GP-LCB's runtime, separately for GP training, acquisition, and overall.

d	Test problem	GP training	Acquisition	Overall
1	f2	6.85% (1.04%)	3.57% (0.90%)	3.83% (0.88%)
	f4	6.56% (1.00%)	3.01% (0.52%)	3.36% (0.54%)
	f5	6.73% (0.92%)	4.17% (1.35%)	4.39% (1.23%)
	f6	7.49% (1.06%)	5.48% (1.91%)	5.66% (1.75%)
	gramacylee	7.58% (0.71%)	4.37% (0.95%)	4.66% (0.89%)
2	braninsc	11.46% (0.92%)	5.73% (2.75%)	6.07% (2.59%)
	goldprsc	10.57% (0.90%)	10.91% (2.31%)	10.89% (2.19%)
	rosenbrocksc	11.91% (0.96%)	1.76% (0.44%)	1.97% (0.43%)
	schwefelsc	16.13% (1.20%)	4.73% (1.51%)	4.91% (1.49%)
	stybtangsc	12.46% (1.21%)	2.18% (0.56%)	2.62% (0.56%)

Result 4. Finally, we report both the Gap metric curve and the cumulative regret curve of the two methods, for the one-dimensional test cases (**Figure 5**) and the two dimensional ones (**Figure 6**), separately.

As far as the one-dimensional test cases are concerned, MLE-free GP-BO always shows a higher cumulative regret due to a larger exploration implied by the need to avoid instability and ill-conditioning. On the other hand, as already discussed, this allows MLE-free GP-BO to achieve a significantly larger AUGC on three out of the five test cases (i.e., f2, f4, and gramacyleesc).

As far as the two-dimensional test cases are concerned, cumulative regret of MLE-free GP-BO is usually lower than GP-LCB's one. It just increases after the Gap Metric achieves one, meaning that MLE-free GP-BO is just exploring because there is no chance to improve further. Finally, AUGC for MLE-free GP-BO is always significantly larger than GP-LCB's one.

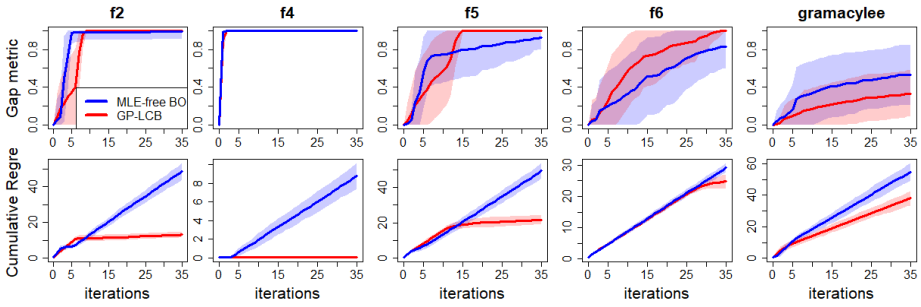


Fig. 5. Gap metric curves (top) and cumulative regret (bottom) for MLE-free GP-BO and GP-LCB on five $d = 1$ test problems (mean and standard deviation on 100 runs).

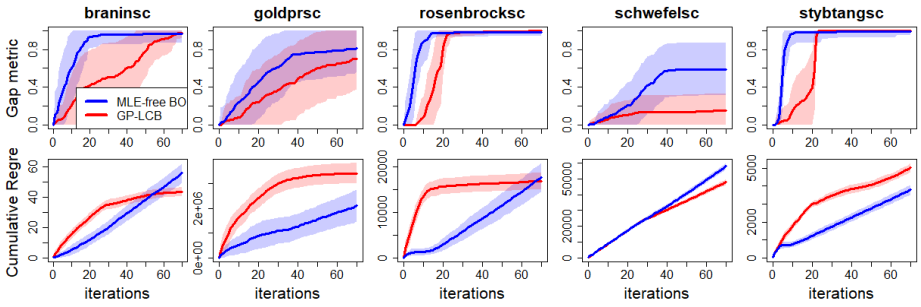


Fig. 6. Gap metric curves (top) and cumulative regret (bottom) for MLE-free GP-BO and GP-LCB on five $d = 2$ test problems (mean and standard deviation on 100 runs).

5 Conclusions

The proposed MLE-free GP-BO algorithm proved to be more effective and efficient than the well-known GP-LCB. Indeed, the MLE-free GP-BO converges to

solutions closer to the optimum than GP-LCB (i.e., significantly higher AUGC values). Moreover, the higher efficiency is not only reflected into a lower number of queries but, more important, also into the computational costs, with a runtime that is, on average, 5% that required by GP-LCB.

Since we have not assumed to work in the (impractical) realizable setting, convergence of GP-LCB is not guaranteed, as empirically demonstrated by the cumulative regret observed in our experiments. Importantly, MLE-free GP-BO has shown a lower cumulative regret than GP-LCB until the Gap Metric can be improved. Otherwise, it increases due to the over exploration triggered by the impossibility to further improve, close to the current best solution, without incurring into ill-conditioning. As an important consequence, MLE-free GP-BO is not only a more effective and efficient BO algorithm; it represents a novel framework in which GP learning and exploration-exploitation trade-off are simultaneously addressed by tuning of the GP kernel's hyperparameters.

Moreover, since the GP model is now learned at a constant time, that is $\mathcal{O}(1)$ instead on $\mathcal{O}(n^3)$, the power of look-ahead acquisition functions can be now disclosed. Indeed, they are known to be inefficient due to the need for retraining many temporary GPs, at each BO iteration, given a set of *hallucinations* (also known as *fantasies*) from the current GP model. Thanks to our approach, this process can be now performed with a really cheap computational cost.

Although limited to one-dimensional and two-dimensional test problems, results are promising and justify further investigation with respect to at least the three following directions: (*i*) high-dimensional search spaces, (*ii*) anisotropic kernels, and (*iii*) theoretical convergence proof (not necessarily based on regret).

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Archetti, F., Candelieri, A.: Bayesian optimization and data science, vol. 849. Springer (2019)
2. Benjamins, C., Raponi, E., Jankovic, A., van der Blom, K., Santoni, M.L., Lindauer, M., Doerr, C.: Pi is back! switching acquisition functions in bayesian optimization. arXiv:2211.01455 (2022)
3. Berk, J., Gupta, S., Rana, S., Venkatesh, S.: Randomised gaussian process upper confidence bound for bayesian optimisation. arXiv:2006.04296 (2020)
4. Berkenkamp, F., Krause, A., Schoellig, A.P.: Bayesian optimization with safety constraints: safe and automatic parameter tuning in robotics. Machine Learning **112**(10), 3713–3747 (2023)
5. Berkenkamp, F., Schoellig, A.P., Krause, A.: No-regret bayesian optimization with unknown hyperparameters. arXiv:1901.03357 (2019)
6. Binois, M., Wycoff, N.: A survey on high-dimensional gaussian process modeling with application to bayesian optimization. ACM Transactions on Evolutionary Learning and Optimization **2**(2), 1–26 (2022)
7. Bogunovic, I., Krause, A.: Misspecified gaussian process bandit optimization. Advances in Neural Information Processing Systems **34**, 3004–3015 (2021)

8. Bull, A.D.: Convergence rates of efficient global optimization algorithms. Journal of Machine Learning Research **12**(10) (2011)
9. Candelieri, A.: Mastering the exploration-exploitation trade-off in bayesian optimization. arXiv:2305.08624 (2023)
10. Candelieri, A., Perego, R., Giordani, I., Ponti, A., Archetti, F.: Modelling human active search in optimizing black-box functions. Soft Computing **24**(23), 17771–17785 (2020)
11. Candelieri, A., Ponti, A., Archetti, F.: Uncertainty quantification and exploration–exploitation trade-off in humans. Journal of Ambient Intelligence and Humanized Computing pp. 1–34 (2021)
12. Candelieri, A., Ponti, A., Archetti, F.: Explaining exploration–exploitation in humans. Big Data and Cognitive Computing **6**(4), 155 (2022)
13. Candelieri, A., Ponti, A., Archetti, F.: Fair and green hyperparameter optimization via multi-objective and multiple information source bayesian optimization. arXiv:2205.08835 (2022)
14. Colliandre, L., Muller, C.: Bayesian optimization in drug discovery. In: HPC for Drug Discovery and Biomedicine, pp. 101–136. Springer (2023)
15. Contal, E., Perchet, V., Vayatis, N.: Gaussian process optimization with mutual information. In: ICML. pp. 253–261. PMLR (2014)
16. De Ath, G., Everson, R.M., Rahat, A.A., Fieldsend, J.E.: Greed is good: Exploration and exploitation trade-offs in bayesian optimisation. ACM Transactions on Evolutionary Learning and Optimization **1**(1), 1–22 (2021)
17. Garnett, R.: Bayesian optimization. Cambridge University Press (2023)
18. Gramacy, R.B.: Surrogates: Gaussian process modeling, design, and optimization for the applied sciences. CRC press (2020)
19. Karras, A., Karras, C., Schizas, N., Avlonitis, M., Sioutas, S.: Automl with bayesian optimizations for big data management. Information **14**(4), 223 (2023)
20. Karvonen, T., Oates, C.J.: Maximum likelihood estimation in gaussian process regression is ill-posed. Journal of Machine Learning Research **24**(120), 1–47 (2023)
21. Sorokin, A., Zhu, X., Lee, E.H., Cheng, B.: Sigopt mulch: An intelligent system for automl of gradient boosted trees. Knowledge-Based Systems **273**, 110604 (2023)
22. Srinivas, N., Krause, A., Kakade, S.M., Seeger, M.W.: Information-theoretic regret bounds for gaussian process optimization in the bandit setting. IEEE transactions on information theory **58**(5), 3250–3265 (2012)
23. Wabersich, K.P., Toussaint, M.: Advancing bayesian optimization: The mixed-global-local (mgl) kernel and length-scale cool down. arXiv:1612.03117 (2016)
24. Wang, Z., de Freitas, N.: Theoretical analysis of bayesian optimisation with unknown gaussian process hyper-parameters. arXiv preprint arXiv:1406.7758 (2014)
25. Wang, Z., Hutter, F., Zoghi, M., Matheson, D., De Feitas, N.: Bayesian optimization in a billion dimensions via random embeddings. Journal of Artificial Intelligence Research **55**, 361–387 (2016)
26. Williams, C.K., Rasmussen, C.E.: Gaussian processes for machine learning, vol. 2. MIT press Cambridge, MA (2006)
27. Yan, L., Duan, X., Liu, B., Xu, J.: Bayesian optimization based on k-optimality. Entropy **20**(8), 594 (2018)
28. Zhang, J., Zeng, H., Li, X., Xu, G., Li, Y., Song, A.: Bayesian optimization for assist-as-needed controller in robot-assisted upper limb training based on energy information. Robotica **41**(10), 3101–3115 (2023)

A Stochastic Dynamic Programming Approach for Request Acceptance and Unsplittable Scheduling Decisions under Uncertainty

Marvin Caspar(✉) [0000−0003−1025−9772], Konstantin Kloster[0000−0002−2538−9900], and Oliver Wendt[0000−0002−7102−3141]

Business Information Systems & Operations Research (BISOR),
RPTU Kaiserslautern-Landau, Postfach 3049,
Erwin-Schrödinger-Str., 67653 Kaiserslautern, Germany
{marvin.caspar,konstantin.kloster,wendt}@wiwi.uni-kl.de
https://wiwi.rptu.de/fgs/bisor

Abstract. The trade-off between gaining revenue through accepting a random incoming request and its scheduling against a higher potential revenue in the future needs sophisticated deliberations and decision-making. To gain deeper insights and to examine such decisions under uncertainty, request acceptance and scheduling should be made simultaneously. This paper introduces the *Dynamic Request Acceptance and Unsplittable Scheduling Problem* (DRAUSP), where we consider a single resource with a given capacity across multiple time slots. In each decision period of a finite time horizon, a single request arrives, and the objective is to either accept or reject the request and, if accepted, simultaneously schedule it onto available time slots without exceeding the capacity to maximize revenue. We formulate the DRAUSP as a *Markov Decision Process* and present a *Stochastic Dynamic Programming* algorithm to solve it optimally. In addition, we introduce a *Dimension Compression* algorithm to compute upper and lower bounds. In our experiments, we demonstrate the effectiveness of these bounds and also show that simple heuristics, such as a *First Come First Serve* heuristic, are not sufficient to cope with the complexity of the DRAUSP. Therefore, more sophisticated heuristics become necessary, since well-planned acceptance and scheduling decisions are essential for revenue maximization.

Keywords: Dynamic Optimization · Request Acceptance and Scheduling · Revenue Maximization · Stochastic Dynamic Programming.

1 Introduction

A central task of revenue management is to manage the trade-off between receiving immediate revenue for selling a fixed and perishable capacity or product against a higher potential revenue in the future [21]. However, determining revenue-maximizing policies is particularly challenging in complex and competitive environments, where disruptions, unforeseen events, and other uncertainties

make predicting the required capacity difficult. Airlines and hotels, for example, often face sudden diverse incoming requests, forcing them to make decisions about accepting or rejecting such requests on short notice to maximize their revenue (see [8]). In addition, such decisions may also affect subsequent scheduling tasks. Scheduling problems arise when scarce resources must be allocated to a set of tasks while optimizing some objective, and are widespread across industries [13].

In this paper, we study the *Dynamic Request Acceptance and Unsplittable Scheduling Problem* (DRAUSP), where the task is to schedule accepted requests onto the limited capacity of a single resource across different time slots. For this problem, we distinguish between an upfront decision horizon, during which $T_d \in \mathbb{N}_{>0}$ requests arrive successively, and a subsequent service period, where the accepted requests get fulfilled. The service period is divided into $K \in \mathbb{N}_{>0}$ discrete time slots, each with a capacity $C_k \in \mathbb{N}_{>0}$, where $k = 1, ..., K$. Each request is defined by a tuple $[r, q]$, where r represents a positive revenue yielded upon acceptance, and $q = (q_1, ..., q_{len(q)}) \in \{0, 1, ..., C_k\}^{len(q)}$ is a discrete, non-negative vector with length $len(q) \in \{1, 2, ..., K\}$ representing the demanded capacity per time slot. Upon acceptance, requests must be scheduled onto the different time slots such that all accepted requests can be fulfilled non-preemptively during the service period without exceeding the capacity C_k of any time slot k. Depending on the available capacity and the length of a request $len(q)$, there are up to $K - len(q) + 1$ possibilities to schedule the request. At each of the T_d periods of the decision horizon, a single request is independently and identically drawn from a stationary distribution over a finite set of possible requests N. The decision to accept or decline the current request, along with its scheduling if accepted, must be made immediately, i.e., before the next request arrives. The objective of the DRAUSP is to maximize the total revenue over the decision horizon.

Figure 1 shows an example with $K = 6$ time slots and a capacity of $C_k = 5$ for these time slots $k = 1, ..., K$. In the current time step of the decision horizon, a request with a certain revenue r and a demand vector $q = (1, 2, 2, 1)$ arrives, and we have to decide whether to accept this request and, if so, on which time slots to schedule it. Since several requests have already been accepted in this example we cannot schedule this request to time slots $[1; 4]$ due to insufficient capacity, while possible time slots are $[2; 5]$ and $[3; 6]$. In this example, we decide to accept the request and schedule it into time slots $[3; 6]$. However, the request arriving in the next time step of the decision horizon with demand vector $q = (2, 1, 1)$ cannot be accepted, as it would exceed the capacity.

The DRAUSP has practical applications including job acceptance and scheduling decisions on high-performance computing clusters, where jobs might be non-preemptive and may have different demand profiles over time. However, defining and describing the problem alone is not sufficient for practical use; we also need to develop solution approaches. Consequently, we present a *Stochastic Dynamic Programming* (SDP) algorithm to solve the DRAUSP optimally. In addition, we introduce a *Dimension Compression* algorithm to compute lower and upper bounds on the optimal solution.

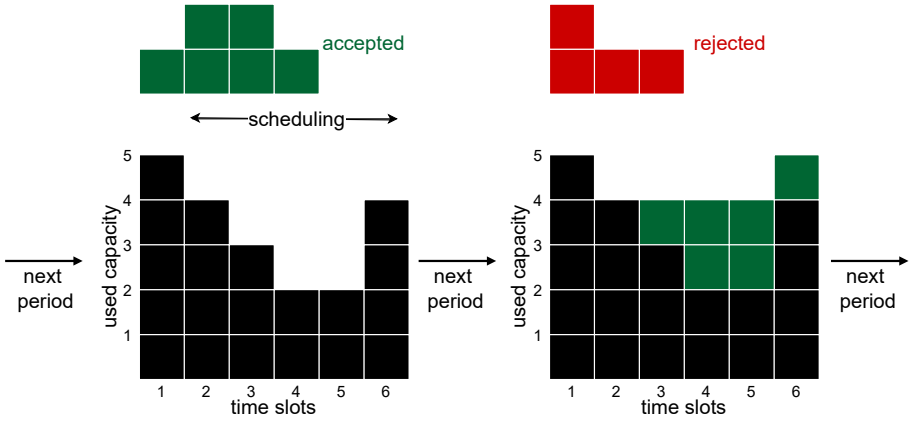


Fig. 1. An illustration of the DRAUSP with $K = 6$ time slots and an initial capacity of $C_k = 5$ for these time slots $k = 1, ..., K$. The first request with $q = (1, 2, 2, 1)$ gets accepted and scheduled onto time slots $[3; 6]$ (time slots $[2; 5]$ would also be possible, while $[1; 4]$ is not possible). The following request in the next decision period with $q = (2, 1, 1)$ cannot be accepted due to insufficient capacity.

The remainder of this paper is structured as follows. Section 2 interlinks the DRAUSP with the relevant literature and distinguishes it from similar problems. In Section 3, we formalize the problem as a *Markov Decision Process* (MDP). Afterward, we present an SDP algorithm for solving the DRAUSP optimally and demonstrate how to obtain lower and upper bounds using a DC algorithm in Section 4. Section 5 provides detailed results of our computational experiments, which we conducted to analyze the tightness of the DC bounds and to compare our SDP with a *First Come First Serve* (FCFS) heuristic. Finally, we conclude this work and discuss future research implications in Section 6.

2 Related Literature

Request acceptance decisions under uncertainty have already been addressed in the revenue management literature, particularly in the context of *Airline Revenue Management* (ARM). Numerous papers, such as [6, 18, 3], consider request acceptance decisions within ARM, incorporating elements like different fare classes, random cancellations, and overbooking. These studies utilize *Reinforcement Learning* (RL) to obtain approximate solutions. Similar problems can also be found in *Cargo Revenue Management*. Seo et al. [17], for example, investigate request acceptance decisions of ad-hoc and contractual containers in a sea cargo revenue management environment, using SDP and RL to solve the problem.

Schwind and Wendt [16] study the less constrained problem of maximizing revenue by accepting or rejecting stochastic requests that arrive dynamically

during an upfront decision horizon, subject to the capacity constraints of one or more resources. To solve this problem, they introduced an RL algorithm and benchmarked its performance against optimal solutions obtained by SDP. The latter problem is closely related to the *Dynamic and Stochastic Knapsack Problem with Deadlines* [12] that belongs to the class of *Dynamic and Stochastic Knapsack Problems* (DSKP) (see [9] and references therein), where a capacitated knapsack is given, and items arrive sequentially as value-size pairs. Each item must either be irrevocably accepted or rejected to maximize the total value of items without exceeding the knapsack's capacity. Yang et al. [24] consider a multidimensional knapsack with the objective of maximizing the total value while respecting the capacities of the different dimensions, and introduce two online algorithms using reservation functions to solve this problem.

While all of the papers mentioned so far study request acceptance decisions, they do not incorporate scheduling considerations. In contrast, the class of *Online Scheduling Problems* (OSP) addresses scheduling decisions without the prior step of accepting or rejecting requests (see [14]). The *Stochastic Resource Constrained Project Scheduling Problem* (SRCPSP) belongs to this OSP class, where a set of jobs with random processing times, precedence constraints, and different resource consumption requirements is scheduled across multiple resources, minimizing a project cost function or the project makespan. Stork [20] employs different scheduling policies and various branch-and-bound algorithms to solve the SRCPSP.

The *Unsplittable Flow Problem* (UFP) is a unifying framework for a variety of scheduling and load-balancing problems such as the *Interval Scheduling Problem* [10], *Storage Allocation Problem* [11] and the *Geometric Strip Packing Problem* [5], that is closely related to the DRAUSP. In the UFP with bag constraints [4], there is a resource with a capacity varying over time and a set of jobs. Each job is specified by a certain demand that is constant over time, a profit, and a set of job instances. The job instances define different possible time intervals for each job, and the objective of the UFP with bag constraints is to select at most one job instance for each job such that the profit is maximized. In contrast to the offline case, literature covering online algorithms for UFP scheduling, to the best of our knowledge, is limited to special variants (see, e.g., [7]), and even for those, competitive analysis does not yield promising bounds.

A research stream that deals with request acceptance and scheduling decisions under uncertainty in the context of manufacturing is the *Stochastic Order Acceptance and Scheduling Problem* (SOASP) [19]. In the SOASP, decisions about which orders to accept for processing and how to schedule them on machines must be made jointly and in real time. Usually, in such SOASPs, accepted orders must be scheduled according to their processing times and due dates and are not subject to capacity restrictions as in the DRAUSP. Xu et al. [23] present a SOASP with sequence-dependent setup times and stochastic orders with fixed revenues for certain order types. The authors formulate the problem as an MDP and develop an order acceptance rule based on opportunity costs, which is com-

pared to a *First Come First Accept* heuristic and the optimal solutions of the corresponding deterministic problems with full information.

Finally, there are several related streams of research in the class of *Online Allocation Problems* (OAP), where real-time decisions about the allocation of accepted requests subject to resource constraints must be made (see [1] for related OAP literature) and therefore our DRAUSP can also be included in this problem class. Balseiro et al. [1] studied a generic OAP with resource constraints where resource consumption can be allocated to resources as required, which is not a fact for the DRAUSP. They present a numerical study for an online linear program with stochastic inputs where action vectors control the allocation of different resources. The authors propose a dual mirror descent algorithm to solve the OAP and compare the results with the hindsight optimum.

The DRAUSP with request acceptance and scheduling decisions under uncertainty is closely related to different problem categories such as ARM problems, DSKPs, OSPs, UFPs, SOASPs, and OAPs. However, to the best of our knowledge, no literature treats such request acceptance decisions of incoming requests with stochastic demand and revenue over a finite time horizon and simultaneous unsplittable scheduling on a resource over different time slots to maximize the aggregate revenue. Thus, we address this research gap with the DRAUSP and provide an MDP formulation in the following section.

3 Markov Decision Process

Recall that in the DRAUSP, $T_d \in \mathbb{N}_{>0}$ requests, independently and identically drawn from a stationary distribution over a finite set of possible requests N, arrive sequentially over a finite time horizon. Each request $[r, q]$ must either be accepted or rejected and, if accepted, non-preemptively scheduled onto $K \in \mathbb{N}_{>0}$ different time slots with capacities $C_k \in \mathbb{N}_{>0}$, where $k = 1, ..., K$. In this section, we introduce an MDP model for the DRAUSP. To this end, we define $T = \{0, ..., T_d\}$ as the time horizon for the MDP. Besides that, an MDP consists of a set of states, a set of actions, a reward function, and the state transitions [15]. We define these elements in the following paragraphs.

States: The state space $S = \{0, 1, ..., C_1\} \times ... \times \{0, 1, ..., C_K\}$ consists of all possible combinations of remaining capacities. We use $s_t \in S$ to represent the state in period t, where $t \in \{T_d, ..., 0\}$ is the down-counting index of periods indicating the number of remaining requests. Furthermore, we define the initial state $s_{T_d} = (C_1, ..., C_K)$, where all time slots have their initial capacity C_k, and the terminal state s_0, where no more requests arrive. Finally, we use $cap \in S$ to refer to the currently available capacity, i.e., $s_t = cap$.

Actions: Within each decision period $t \in T \backslash \{0\}$, a random request $[r, q] \in N$ arrives in the current state s_t and determines the actual action space $A_{[r,q]}$. This set $A_{[r,q]}$ contains actions represented by tuples $a = [a_r, a_q]$ consisting of a revenue $a_r \in \{0, r\}$, which is obtained when the action is chosen, and a vector $a_q \in Q_a$ of length K, which specifies the required demand for all K time slots. The set $Q_a = \{\vec{0}, (q_1, ..., q_{len(q)}, 0, ..., 0), (0, q_1, ..., q_{len(q)}, 0, ..., 0), ..., (0, ..., 0, q_1, ..., q_{len(q)})\}$

with $|Q_a| = K - len(q) + 2$ results from scheduling the demand vector q onto all possible time slots. Q_a also includes $\vec{0}$ to represent the reject action $a = [0, \vec{0}]$, which yields no revenue and demands no capacity. Finally, we use A to denote the set of action spaces $A_{[r,q]}$ for all requests $[r, q] \in N$, i.e., $A = \bigcup_{[r,q] \in N} A_{[r,q]}$.

Transitions: Between two states s_t and s_{t-1} with $t \in T \setminus \{0\}$ a transition occurs, which depends on the selected action $a = [a_r, a_q] \in A_{[r,q]}$. However, we only consider actions that do not exceed the capacity, i.e., actions for which $cap - a_q \geq 0$ holds since we do not allow overbooking. For all valid actions, including the reject action $[0, \vec{0}]$, the state transition principle for the DRAUSP becomes $s_{t-1} = cap - a_q$. Reaching the terminal state s_0 signifies the end of the decision process, as no further requests arrive and no more transitions occur.

Rewards: Transitioning to the next state by choosing an action $a = [a_r, a_q]$ yields the reward a_r in state s_t, where $t \in T \setminus \{0\}$. The reward is thus defined by the reward function $R(s_t, a) = a_r$.

Policy: A policy $\pi(s_t) : S \longrightarrow A_{[r,q]}$ is a function indicating which action to take in each state. In the DRAUSP, the objective is to find the optimal policy, i.e., the policy giving us the optimal action in each state to maximize the cumulative reward over the time horizon T. Thus, the objective function of the DRAUSP becomes:

$$\max_{\pi(s_t)} \mathbb{E} \left\{ \sum_{t \in T \setminus \{0\}} R(s_t, \pi(s_t)) \right\} \tag{1}$$

Value function: The value function $V_t(cap)$ can be used to calculate the expected objective function value when starting in state $cap \in S$ at time step $t \in T$ and choosing the best action until the terminal state s_0 is reached at $t = 0$. Given that $Pr([r, q])$ represents the probability that the request $[r, q] \in N$ arrives in period t, the calculation of $V_t(cap)$ can be expressed as:

$$V_t(cap) = \begin{cases} -\infty & \forall cap \notin (\mathbb{R}_0^+)^K, t \in T \\ 0 & \forall cap \in (\mathbb{R}_0^+)^K,\ t = 0 \\ \sum\limits_{[r,q] \in N} Pr([r,q]) \max\limits_{[a_r, a_q] \in A_{[r,q]}} \{a_r + V_{t-1}(cap - a_q)\} & \text{else} \end{cases} \tag{2}$$

4 Solution Methods for the DRAUSP

4.1 Stochastic Dynamic Programming

One method to solve the value function of the DRAUSP presented in Section 3 is SDP. In this section, we present a forward recursion SDP algorithm, which has the advantage that we only need to compute the values of states that are actually needed, i.e., we do not compute the values for the complete state space.

The function `recursive_sdp` in Algorithm 2 takes the number of decision periods T_d, the vector of initial capacities cap, and the set of action spaces A as input. For each decision period t, where $t = 1, ..., T_d$ represents the number

Algorithm 1: Value computation (for Algorithm 2)

```
1  Function get_value(t, cap, A, Ω):
2      if t = 0 then
3          return 0
4      end
5      if Ω contains [t, cap] then
6          return Ω[t, cap]
7      end
8      value ← 0
9      foreach A_[r,q] ∈ A do
10         best_action_value ← −∞
11         foreach [a_r, a_q] ∈ A_[r,q] do
12             new_state ← cap − a_q
13             if new_state is valid then
14                 action_value ← a_r + get_value(t − 1, new_state, A, Ω)
15             end
16             else
17                 action_value ← −∞
18             end
19             if action_value > best_action_value then
20                 best_action_value ← action_value
21             end
22         end
23         value ← value + best_action_value
24     end
25     Ω[t, cap] ← value/|A|
26     return value/|A|
```

of remaining requests, we compute the optimal expected value $V_t^\star$ for the initial capacity *cap* when there are t incoming requests remaining by calling the function `get_value`. Throughout the computation, we utilize a hashmap Ω to memoize computed values. Finally, the function returns $V_{T_d}^\star$, which is the optimal expected value obtainable when starting with the initial capacity *cap* and facing T_d requests.

Within the `get_value` function in Algorithm 1, we select the best action for each request in the current state by recursively calling the `get_value` function and update the expected value according to equation (2), assuming a discrete uniform distribution, where each request is equally likely to arrive in a period.

When using a thread-safe hashmap, this algorithm can be accelerated by parallelizing the for-each loop over the requests (see line 9 of Algorithm 1) within the `get_value` function. However, this loop should only be parallelized when the `get_value` is called from the `recursive_sdp` function, i.e., not when the function `get_value` is called recursively.

Algorithm 2: Forward recursion stochastic dynamic programming for the DRAUSP

```
1  Function recursive_sdp(T_d, cap, A):
2  |  Ω ← ∅ ;                                          // init hashmap
3  |  for t ← 1 to T_d do
4  |  |  V_t^⋆ ← get_value (t, cap, A, Ω)              // see Algorithm 1
5  |  end
6  |  return V_{T_d}^⋆
```

4.2 Dimension Compression Algorithm

SDP algorithms, like the one presented in Section 3.1, can quickly become computationally intractable due to the curse of dimensionality [2]. To overcome this problem, heuristics are frequently used to find good, though not necessarily optimal, solutions. In this section, we present a *Dimension Compression* (DC) algorithm to compute lower and upper bounds for the DRAUSP (see Algorithm 3), which may be used to evaluate the quality of heuristics if the SDP cannot be solved anymore. The main idea of the DC algorithm is to reduce the number of time slots K. To be more precise, we halve K and adjust the capacity and demand vectors accordingly. The reduced problem can then be solved using the `recursive_sdp` function.

We start by checking if K is an even number. If not, we take the last element of the capacity vector *cap*, append it to *cap*, and increase K by one. Afterward, we create two vectors $\overline{cap}$ and $\underline{cap}$ of length K/2, which will represent our new capacity vectors to compute the upper and the lower bound, respectively. To compute $\overline{cap}$, we look at all non-overlapping pairs of consecutive elements in *cap* and take the maximum value of each pair multiplied by two. Similarly, we take the minimum value of each non-overlapping pair of consecutive elements in *cap* and add it to $\underline{cap}$.

In the next part of the algorithm (see lines 12 - 26 of Algorithm 3), we compute the compressed demand vectors. We iterate through all pairs of revenues and demand vectors $[a_r, a_q] \in A_{[r,q]}$ for all $A_{[r,q]} \in A$. If the length of the current demand vector a_q is not even, we will append 0 to a_q. Then, we create the vectors $\overline{a_q}$ and $\underline{a_q}$ of length $K/2$, representing the new demand vectors for the current request to compute the upper and lower bound, respectively. Afterwards, we look at all non-overlapping pairs of consecutive elements in a_q. To fill $\overline{a_q}$, we take the sum of the elements of each such pair, and to fill $\underline{a_q}$, we take the maximum element of each pair. We add each $\overline{a_q}$ and $\underline{a_q}$ to the sets $\overline{A_{[r,q]}}$ and $\underline{A_{[r,q]}}$, respectively, and each set $\overline{A_{[r,q]}}$ and $\underline{A_{[r,q]}}$ is subsequently added to the new sets of action spaces $\overline{A}$ and $\underline{A}$, respectively.

Finally, we can compute an upper bound $\overline{Z}$ by calling the `recursive_sdp` function with T_d, the updated capacity vector $\overline{cap}$, and the new set of action spaces $\overline{A}$. Similarly, we call `recursive_sdp` with T_d, $\underline{cap}$, and $\underline{A}$ to compute a lower bound $\underline{Z}$ for the DRAUSP. In the next section, we will evaluate the quality

Algorithm 3: A dimension compression algorithm for the DRAUSP

Function `get_bounds`(T_d, cap, A, K)**:**
  init empty sets $\underline{A}$ and $\overline{A}$
  if K *is not even* **then**
    append cap_K to cap
    $K \leftarrow K + 1$
  end
  init empty vectors $\overline{cap}$ and $\underline{cap}$ of length $K/2$
  for $k \leftarrow 1$ **to** $K/2$ **do**
    $\overline{cap}_k \leftarrow 2 \cdot \max\{cap_{2k-1}, cap_{2k}\}$
    $\underline{cap}_k \leftarrow \min\{cap_{2k-1}, cap_{2k}\}$
  end
  foreach $A_{[r,q]} \in A$ **do**
    init empty sets $\overline{A_{[r,q]}}$ and $\underline{A_{[r,q]}}$
    foreach $[a_r, a_q] \in A_{[r,q]}$ **do**
      if $len(a_q)$ *is not even* **then**
        append 0 to a_q
      end
      init empty vectors $\overline{a_q}$ and $\underline{a_q}$ of length $K/2$
      for $k \leftarrow 1$ **to** $K/2$ **do**
        $\overline{a_q}_k \leftarrow a_{q_{2k-1}} + a_{q_{2k}}$
        $\underline{a_q}_k \leftarrow \max\{a_{q_{2k-1}}, a_{q_{2k}}\}$
      end
      add $[a_r, \overline{a_q}]$ to $\overline{A_{[r,q]}}$ and $[a_r, \underline{a_q}]$ to $\underline{A_{[r,q]}}$
    end
    add $\overline{A_{[r,q]}}$ to $\overline{A}$ and $\underline{A_{[r,q]}}$ to $\underline{A}$
  end
  $\overline{Z}$ = `recursive_sdp`$(T_d, \overline{cap}, \overline{A})$
  $\underline{Z}$ = `recursive_sdp`$(T_d, \underline{cap}, \underline{A})$
  return $\overline{Z}, \underline{Z}$

of these bounds and compare the runtime of the DC algorithm with the SDP algorithm.

5 Computational Studies

In this section, we evaluate the quality of the bounds returned by the DC algorithm by comparing them with the optimal expected values returned by the SDP algorithm. In addition, we will assess the effectiveness of a simple FCFS heuristic that successively accepts stochastic incoming requests, if possible, and schedules them to the first possible time slot. Before presenting the results, we will describe the experimental environment.

5.1 Experimental Environment

In the revenue management literature, arrivals of requests and their demands are often assumed to be Poisson distributed [22]. We follow this assumption and generated each element of the demand vector q of any request $[r, q]$ in the finite pool of requests N using a Poisson process:

$$Pr(x) = \frac{e^{-\lambda_p}\lambda_p^x}{x!} \quad \forall x = 0, ..., \min_{1 \leq k \leq K}(C_k).$$

The parameter λ_p states the expected demand value in any time slot and also its variance. Within our experiments, we distinguish between small ($\lambda_p = 1$) and large ($\lambda_p = 3$) request demands. For each request $[r, q]$, we furthermore generated the revenue r by taking the quotient of 1 and an integer random number sampled from a discrete uniform distribution within the interval $[1, 1000]$.

For our experiments, we generated 51 problem instances with the following parameters: number of time slots $K \in \{4, 8, 12\}$, number of requests in the requests pool $|N| \in \{50, 1000\}$, expected demand value in any time slot $\lambda_p \in \{1, 3\}$, capacity per time slot $C \in \{10, 20\}$, and the number of incoming requests $T_d \in \{10, 20, 50\}$. In 21 of those instances, the length of the demand vectors $len(q)$ is equal to the number of time slots K, which means that the requests do not have to be scheduled, and we will only examine the accept/reject decisions for those instances. In the remaining instances, we consider $len(q) \in \{2, 3, 4, 6, 8\}$, with $len(q) < K$. The problem instances used in our experiments are available online (see https://doi.org/10.5281/zenodo.10640203).

We implemented the SDP algorithm and the DC algorithm in C++ and compiled them with GCC 13.1. To speed up the computation, we parallelized the SDP algorithm using OpenMP. All experiments were run on a shared compute cluster equipped with Intel Xeon Gold 6126 processors providing 24 CPU cores and 375 GB of RAM to each experiment. We enforced a time limit of 48 hours on each experiment.

5.2 Computational Results

In Table 1, we present the results for the instances where the length of the requests $len(q)$ is equal to K, i.e., without considering scheduling decisions in these instances. We can observe that the number of time slots K and the available capacity per time slot C are major factors influencing the runtime of the SDP algorithm. For instances with $K = 8$, the runtime quickly becomes unreasonably high when $\lambda_p = 1$. However, using demand vectors with higher demands by increasing λ_p to 3 reduces the runtime significantly because fewer requests can be accepted without exceeding the capacity. Despite the 375GB of RAM of our computing environment, we were not able to solve instances with $K = 12$, $|N| = 1000$, and $C = 20$, as not only the runtime but also the memory requirements grow exponentially with increasing C.

On average, the DC algorithm returns lower bounds that are within 14% from the optimal solution obtained by the SDP algorithm in about a tenth of the time.

Table 1. Numerical results for the DRAUSP instances without the possibility of scheduling accepted requests.

Instance	FCFS	DCH				SDP		Δ^F_{SDP}
Id $(K, \lvert N\rvert, \lambda_p, C, T_d)$	Z^F	$\underline{Z}$	t	$\overline{Z}$	t	Z^*	t	%
WA1 (4, 50, 1, 10, 10)	3.80	3.87	0.01 s	4.63	0.02 s	4.33	0.15 s	14.0
WA2 (4, 50, 1, 10, 20)	4.85	5.41	0.02 s	6.90	0.03 s	6.43	0.43 s	32.6
WA3 (4, 50, 1, 10, 50)	5.82	7.48	0.06 s	10.08	0.07 s	9.49	1.06 s	63.1
WA4 (4, 50, 1, 20, 10)	4.56	5.02	0.02 s	5.08	0.04 s	5.07	0.90 s	11.2
WA5 (4, 50, 1, 20, 20)	8.23	8.20	0.04 s	9.61	0.09 s	9.23	4.60 s	12.2
WA6 (4, 50, 1, 20, 50)	10.45	12.39	0.08 s	16.05	0.24 s	15.36	18.1 s	47.0
WA7 (4, 1000, 1, 10, 10)	3.81	3.91	0.07 s	4.59	0.12 s	4.35	2.12 s	14.2
WA8 (4, 1000, 1, 10, 20)	4.90	5.53	0.15 s	7.04	0.30 s	6.58	4.23 s	34.3
WB1 (8, 50, 1, 10, 10)	3.46	3.50	0.08 s	4.51	0.93 s	4.08	261 s	18.0
WB2 (8, 50, 1, 10, 20)	4.01	4.54	0.25 s	6.30	3.06 s	5.47	1076 s	36.4
WB3 (8, 50, 1, 10, 50)	4.38	5.83	0.61 s	8.44	9.70 s	7.19	3567 s	64.2
WB4 (8, 50, 3, 10, 10)	1.18	1.55	0.03 s	1.95	0.08 s	1.69	0.07 s	43.2
WB5 (8, 50, 3, 10, 20)	1.27	1.86	0.03 s	2.38	0.17 s	2.08	0.11 s	63.8
WB6 (8, 50, 3, 20, 10)	2.45	2.91	0.22 s	3.57	2.83 s	3.22	33.4 s	31.4
WB7 (8, 50, 3, 20, 20)	2.65	3.60	0.63 s	4.48	8.13 s	4.06	132 s	53.2
WC1 (12, 50, 3, 10, 10)	1.02	1.45	0.02 s	1.83	0.06 s	1.54	0.03 s	51.0
WC2 (12, 50, 3, 10, 20)	1.07	1.65	0.03 s	2.19	0.16 s	1.78	0.07 s	66.4
WC3 (12, 50, 3, 10, 50)	1.12	1.84	0.06 s	2.54	0.37 s	2.13	0.14 s	90.2
WC4 (12, 1000, 3, 10, 10)	1.00	1.38	10.4 s	1.83	364 s	1.53	299 s	53.0
WC5 (12, 1000, 3, 10, 20)	1.05	1.65	22.3 s	2.19	901 s	1.79	855 s	70.5
WC6 (12, 1000, 3, 10, 50)	1.09	1.87	67.3 s	2.59	2393 s	2.17	2236 s	99.1
Average	3.44	4.07	4.88 s	5.18	176 s	4.74	404 s	46.1

The upper bound exceeds the optimal solution by 9% on average. However, as with SDP, computing the upper bound quickly becomes intractable because, even though we halve the number of time slots, we double the available capacity to compute the upper bound. This problem is particularly evident for instances WC4-WC6, where the runtime to compute the upper bounds even exceeds the runtime of the SDP algorithm.

Let us define Δ^F_{SDP} as the relative improvement of an SDP solution (Z^*) compared to an FCFS solution (Z^F) of the same instance. As the number of incoming requests T_d and time slots K increases, while keeping all other parameters equal, we see that the value Δ^F_{SDP} becomes larger. The overall average of Δ^F_{SDP} for the instances in Table 1 is 46.1%.

The results of the experiments with $len(q) < K$, which require accepted requests to be scheduled onto the time slots, are shown in Table 2. We see that, again, the number of time slots K and the capacity per time slot C have strong effects on the runtime of the SDP algorithm. For example, doubling the number of time slots K from 4 to 8, without changing the other parameters, results in a 5000-fold increase in runtime for instances SA1 and SB1. The results of instances

Table 2. Numerical results for the DRAUSP instances with the possibility of scheduling accepted requests.

Instance	FCFS	DCH				SDP		Δ^F_{SDP}
Id $(K, len(q), \lvert N\rvert, \lambda_p, C, T_d)$	Z^F	$\underline{Z}$	t	$\overline{Z}$	t	Z^*	t	%
SA1 (4, 2, 50, 1, 10, 10)	3.43	5.03	0.03 s	5.08	0.06 s	5.08	0.42 s	48.1
SA2 (4, 2, 50, 1, 10, 20)	3.46	8.21	0.06 s	8.88	0.13 s	8.80	1.21 s	154.3
SA3 (4, 2, 50, 1, 10, 50)	3.48	12.99	0.20 s	13.52	0.42 s	13.44	3.53 s	286.2
SA4 (4, 3, 50, 1, 10, 10)	3.18	3.88	0.01 s	4.75	0.02 s	4.35	0.14 s	36.8
SA5 (4, 3, 50, 1, 10, 20)	3.20	5.07	0.02 s	6.70	0.05 s	5.97	0.29 s	86.6
SA6 (4, 3, 50, 1, 10, 50)	3.21	6.50	0.04 s	8.59	0.09 s	7.80	0.78 s	143.0
SA7 (4, 2, 50, 1, 20, 10)	4.53	5.08	0.05 s	5.08	0.11 s	5.08	1.41 s	12.1
SA8 (4, 2, 50, 1, 20, 20)	6.86	10.14	0.11 s	10.16	0.49 s	10.16	14.4 s	48.1
SA9 (4, 2, 50, 1, 20, 50)	6.86	18.95	0.27 s	20.46	2.01 s	20.38	60.3 s	197.1
SA10 (4, 3, 50, 1, 20, 10)	4.60	5.06	0.02 s	5.08	0.12 s	5.08	0.78 s	10.4
SA11 (4, 3, 50, 1, 20, 20)	6.44	8.16	0.05 s	9.74	0.13 s	9.02	3.37 s	40.1
SA12 (4, 3, 50, 1, 20, 50)	6.61	11.27	0.10 s	14.90	0.31 s	13.43	11.1 s	103.2
SA13 (4, 2, 1000, 1, 10, 10)	3.38	4.88	0.21 s	4.90	0.92 s	4.90	9.43 s	45.0
SA14 (4, 2, 1000, 1, 10, 20)	3.48	8.32	0.50 s	8.88	2.40 s	8.79	22.6 s	152.6
SB1 (8, 2, 50, 1, 10, 10)	3.46	5.08	1.59 s	5.08	10.2 s	5.08	2192 s	46.8
SB2 (8, 2, 50, 1, 10, 20)	3.48	10.14	5.92 s	10.16	103 s	10.16	42830 s	192.0
SB3 (8, 4, 50, 1, 10, 10)	3.17	5.03	0.45 s	5.08	3.87 s	5.08	1019 s	60.3
SB4 (8, 4, 50, 1, 10, 20)	3.18	8.16	1.65 s	8.94	14.0 s	8.76	4321 s	175.5
SB5 (8, 4, 50, 1, 10, 50)	3.18	12.01	3.45 s	12.97	45.1 s	12.60	14008 s	296.2
SB6 (8, 6, 50, 1, 10, 10)	3.09	4.26	0.21 s	4.55	1.30 s	4.44	102 s	43.7
SB7 (8, 6, 50, 1, 10, 20)	3.13	5.68	0.48 s	6.00	3.70 s	5.85	355 s	86.9
SB8 (8, 6, 50, 1, 10, 50)	3.16	7.52	1.41 s	7.55	10.9 s	7.53	1114 s	138.3
SB9 (8, 4, 50, 3, 10, 10)	1.26	3.04	0.33 s	3.73	3.50 s	3.38	685 s	168.3
SB10 (8, 4, 50, 3, 10, 20)	1.34	3.84	0.79 s	4.71	11.2 s	4.28	2126 s	219.4
SC1 (12, 4, 50, 3, 10, 5)	1.22	2.53	11.0 s	2.54	102 s	2.54	3368 s	108.2
SC2 (12, 6, 50, 3, 10, 5)	1.07	2.18	3.21 s	2.48	63 s	2.32	233 s	116.8
SC3 (12, 8, 50, 3, 10, 5)	1.02	1.53	0.94 s	1.98	10.4 s	1.74	5.78 s	70.6
SC4 (12, 8, 50, 3, 10, 10)	1.08	1.89	2.09 s	2.47	54.6 s	2.17	26.8 s	100.9
SC5 (12, 8, 50, 3, 10, 20)	1.14	2.24	5.13 s	2.85	160 s	2.55	72.2 s	123.7
SC6 (12, 8, 50, 3, 10, 50)	1.20	2.67	16.6 s	5.26	450 s	4.13	216 s	244.2
Average	3.26	6.38	1.90 s	7.10	35.1 s	6.83	2427 s	118.5

SB1 and SB2 show that, in some cases, an increasing number of incoming requests T_d can also lead to an exponentially increasing runtime. However, with an increasing length of demand vectors $len(q)$, the runtime decreases significantly, as there are fewer possibilities to schedule requests with longer demand vectors. Given our compute resources, we were unable to solve instances with more than 5 incoming requests when $K = 12$ and $len(q) = 4$.

On average, the DC algorithm finds a lower and upper bound in less than 0.08% and 1.5% of the runtime of the SDP algorithm, respectively. The obtained

lower bound is within 7% of the optimal solution on average, and the upper bound exceeds it by about 4% on average.

With an average relative improvement Δ^F_{SDP} of over 118.5% for the instances with $len(q) < K$ (see Table 2), the performance of the FCFS heuristic becomes even worse than for instances with $len(q) = K$ (see Table 1). Again, the performance of the FCFS heuristic is mainly affected by the number of incoming requests T_d. However, in contrast to the results in Table 1, an increasing number of time slots K does not necessarily lead to a higher average Δ^F_{SDP}.

In conclusion, our experiments have shown that the scheduling decisions in the DRAUSP significantly increase the complexity of the problem. Nevertheless, the DC algorithm was still capable of finding good upper and lower bounds, which can be useful for evaluating the quality of heuristics. Our experiments have also shown that the FCFS heuristic is not sufficient to obtain satisfactory results, especially when acceptance and scheduling decisions are required, and thus more sophisticated heuristics are needed for the DRAUSP.

6 Conclusion and Future Research

In this paper, we studied the *Dynamic Request Acceptance and Unsplittable Scheduling Problem* (DRAUSP). We formulated the DRAUSP as a *Markov Decision Process* (MDP) and showed how *Stochastic Dynamic Programming* (SDP) can optimally solve this problem (see Algorithm 2). Furthermore, we contributed a *Dimension Compression* (DC) algorithm (see Algorithm 3) for the DRAUSP that provides lower and upper bounds to gauge the optimal solution of a DRAUSP instance.

Our experiments showed that the DC algorithm provides promising lower and upper bounds, and the runtime in doing so is relatively short in comparison to the runtime of the SDP algorithm. However, our experiments also demonstrated the complexity of the DRAUSP and showed that the FCFS heuristic leads to poor results. Therefore, future work should examine more sophisticated heuristic approaches, and since we provided an MDP model for the DRAUSP, *Reinforcement Learning* might be a promising direction.

References

1. Balseiro, S.R., Lu, H., Mirrokni, V.: The best of many worlds: Dual mirror descent for online allocation problems. Operations Research **71**(1), 101–119 (2023)
2. Bellman, R.E.: Dynamic Programming. Princeton University Press (1957)
3. Bondoux, N., Nguyen, A.Q., Fiig, T., Acuna-Agost, R.: Reinforcement learning applied to airline revenue management. Journal of Revenue and Pricing Management **19**(5), 332–348 (2020)
4. Chakaravarthy, V.T., Choudhury, A.R., Gupta, S., Roy, S., Sabharwal, Y.: Improved algorithms for resource allocation under varying capacity. Journal of Scheduling **21**, 313–325 (2018)
5. Gálvez, W., Grandoni, F., Ameli, A.J., Khodamoradi, K.: Approximation algorithms for demand strip packing. arXiv preprint arXiv:2105.08577 (2021)

6. Gosavi, A., Bandla, N., Das, T.: A reinforcement learning approach to airline seat allocation for multiple fare classes with overbooking" iie transactions, 34. Special issue on advances on large-scale optimization for logistics, production and manufacturing systems (2002)
7. Jahanjou, H., Kantor, E., Rajaraman, R.: Improved algorithms for scheduling unsplittable flows on paths. Algorithmica **85**(2), 563–583 (2023)
8. Kimes, S.E.: Yield management: a tool for capacity-considered service firms. Journal of operations management **8**(4), 348–363 (1989)
9. Kleywegt, A.J., Papastavrou, J.D.: The dynamic and stochastic knapsack problem. Operations research **46**(1), 17–35 (1998)
10. Kolen, A.W., Lenstra, J.K., Papadimitriou, C.H., Spieksma, F.C.: Interval scheduling: A survey. Naval Research Logistics (NRL) **54**(5), 530–543 (2007)
11. Mömke, T., Wiese, A.: A $(2+\epsilon)$-approximation algorithm for the storage allocation problem. In: International Colloquium on Automata, Languages, and Programming. pp. 973–984. Springer (2015)
12. Papastavrou, J.D., Rajagopalan, S., Kleywegt, A.J.: The dynamic and stochastic knapsack problem with deadlines. Management Science **42**(12), 1706–1718 (1996)
13. Pinedo, M.L.: Scheduling. Springer (2012)
14. Pruhs, K., Sgall, J., Torng, E.: Online scheduling. In: Leung, J., Kelly, L., Anderson, J.H. (eds.) Handbook of Scheduling: Algorithms, Models, and Performance Analysis. CRC Press (2004)
15. Puterman, M.L.: Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons (2014)
16. Schwind, M., Wendt, O.: Dynamic pricing of information products based on reinforcement learning: A yield-management approach. In: KI 2002: Advances in Artificial Intelligence. pp. 51–66. Springer (2002)
17. Seo, D.W., Chang, K., Cheong, T., Baek, J.G.: A reinforcement learning approach to distribution-free capacity allocation for sea cargo revenue management. Information Sciences **571**, 623–648 (2021)
18. Shihab, S.A.M., Logemann, C., Thomas, D.G., Wei, P.: Autonomous airline revenue management: A deep reinforcement learning approach to seat inventory control and overbooking. arXiv preprint arXiv:1902.06824 (2019)
19. Slotnick, S.A.: Order acceptance and scheduling: A taxonomy and review. European Journal of Operational Research **212**(1), 1–11 (2011)
20. Stork, F.: Stochastic resource-constrained project scheduling. Ph.D. thesis, TU Berlin (2001)
21. Talluri, K.T., Van Ryzin, G.: The theory and practice of revenue management. Springer (2004)
22. Weatherford, L.: The history of unconstraining models in revenue management. Journal of Revenue and Pricing Management **15**, 222–228 (2016)
23. Xu, L., Wang, Q., Huang, S.: Dynamic order acceptance and scheduling problem with sequence-dependent setup time. International Journal of Production Research **53**(19), 5797–5808 (2015)
24. Yang, L., Zeynali, A., Hajiesmaili, M.H., Sitaraman, R.K., Towsley, D.: Competitive algorithms for online multidimensional knapsack problems. Proceedings of the ACM on Measurement and Analysis of Computing Systems **5**(3), 1–30 (2021)

Efficient vertex linear orderings to find minimal Feedback Arc Sets (minFAS)

Claudia Cavallaro[1 [0000−0003−3938−0947]], Vincenzo Cutello[1 [0000−0002−7521−3516]], and Mario Pavone[1 [0000−0003−3421−3293]]

Department of Mathematics and Computer Science, University of Catania, 95125 Catania, Italy

Abstract. Given a directed graph, we present and experimentally validate new heuristics to find good linear orderings of its vertices so to obtain Feedback Arc Sets which are minimal, i.e. such that none of the arcs can be reintroduced in the graph without disrupting acyclicity. We will also show that, in many cases, the newly proposed linear orderings along with an improved heuristic algorithm to reinsert eliminated arcs, which has a good polynomial upper bounds, produce, in fact, a minimum FAS.

Keywords: Minimal Feedback Arc Set · Heuristics · Optimization Problem · Experimental Analysis

1 Introduction

The Minimum Feedback Set Problems, both in the vertex and in the arc case, are among the best known $\mathcal{NP}$-complete problems, and are listed in the seminal paper written by Karp in 1972 [12]. Due to their hard computational nature, many heuristics and approximation algorithms have been produced over more than 50 years. In particular, a thorough review of the Minimum Feedback Arc Set problem (MFAS), which also includes a description of the special cases for which we do have a polynomial time algorithm, such as for instance planar graphs [14], can be found in the recent monography [13].

The problem can simply be defined as follows: given a directed graph $G = (V, A)$, find the smallest possible subset of its set of arcs, i.e. a subset $F \subseteq A$, whose removal from G makes the graph acyclic. There are different ways to try to tackle the MFAS problem. Following the equivalent Linear Arrangement formulation, one can try to find a good ordering of the vertices, among the $|V|!$ possible orderings, such that the number of forward (respectively backward) arcs, i.e. arcs directed from left to right (respectively right to left) with respect to the given ordering, is minimum. Clearly, both the set of forward arcs and the set of backward arcs are a Feedback Arc Set, which in turn implies that any minimum Feedback Arc Set has size at most $\frac{|A|}{2}$ (see also [3]). Such an approach was followed quite successfully in [9], where the authors developed a linear time algorithm to find a good ordering so to put all backward arcs in the Feedback

Arc Set. Different approaches could be used for specific types of graphs. For sparse graphs, for instance, one can try to enumerate all the simple cycles and remove them, see the recent papers [2] and [10].

Here, we want to extend the work in [4] and introduce a new heuristic to find good linear arrangements of the set of vertices of the graph. Our main goal is to produce, in polynomial time, feedback arc sets which are minimal and, at the same time, prove that on a set of given benchmarks, they actually produce a minimum feedback arc set.

2 Linear orderings and minimal feedback arc sets

Let us briefly describe the general approach of producing a minimal Feedback Arc Set (minFAS) using a linear ordering of the vertices. We recall that a minimal FAS (minFAS) is a feedback arc set which verifies the property that none of its arcs can be reintroduced into the graph without disrupting acyclicity.

First, we recall that given a directed graph $G = (V, A)$ with $|V| = n$ and $|A| = m$ the vertices which participate in any cycle belong to the same Strongly Connected Component (SCC) of G. Therefore, we can concentrate just on finding a minFAS for the SCC's of G and the union of the found minFAS's will be a minFAS for the entire graph.

Thus, let us work on Strongly Connected Graphs. Given a linear ordering $L = (v_1, v_2, \ldots, v_n)$ of the set of vertices V of a directed strongly connected graph $G = (V, A)$, we can produce 4 different Feedback Arc Sets as follows:

1. Eliminate all the forward arcs, i.e. all the arcs of type (v_i, v_j) with $j > i$. We denote the obtained FAS with F_0.
2. Eliminate all the backward arcs, i.e. all the arcs of type (v_i, v_j) with $i > j$. We denote the obtained FAS with B_0.
3. Following the list L, eliminate, vertex by vertex, all the forward arcs, i.e. all the arcs of type (v_i, v_j) with $j > i$, but checking after every vertex v_i if the graph has become acyclic. We denote the obtained FAS with F_1.
4. Following the list L, eliminate vertex by vertex, all the backward arcs, i.e. all the arcs of type (v_i, v_j) with $i > j$, but checking after every vertex v_i if the graph has become acyclic. We denote the obtained FAS with B_1.

Clearly, $F_1 \subseteq F_0$ and $B_1 \subseteq B_0$. Moreover, if L is the linear order which will give us a minimum FAS, it will either be $F_1 = F_0$ or $B_1 = B_0$. The computational cost of finding B_0 and F_0 is linear in the size of the graph, i.e. $\mathcal{O}(|V| + |A|)$, since for each vertex we can simply go through its adjacency list and search for all the vertices which follow (or precede) in the given ordering L.

If we want to find B_1 or F_1 the computational cost is higher. In particular, since acyclicity can be tested in $\mathcal{O}(|V|+|A|)$, an upper bound for the overall cost in such cases is $\mathcal{O}(|V|(|V| + |A|))$. Such an asymptotic extra cost is amortized by the fact that, in general, we may stop much earlier in the search of a FAS. The pseudo-codes in Algorithms 1 and 2 show how the two feedback arc sets F_1 and B_1 are produced.

There are two simple in place optimization procedures applied by the two Algorithms: $MLF(L)$ and $MLB(L)$. Starting at the first element of the list L (index 0 following the standard C or Python notation) they modify in place the input list L as follows: for $i = 0, \ldots, |L| - 1$

$MLF(L)$ if there is an arc from $(L[i+1], L[i])$ but no arc $(L[i], L[i+1])$ then it swaps $L[i]$ and $L[i+1]$. As a consequence of the swapping of the two values, there is one less backward arc.

$MLB(L)$ if there is an arc from $(L[i], L[i+1])$ but no arc $(L[i+1], L[i])$ then it swaps $L[i]$ and $L[i+1]$. As a consequence of the swapping of the two values, there is one less forward arc.

Algorithm 1: RFA

Inputs : Directed strongly connected graph $G = (V, A)$, ordered list of vertices L
Output: Ordered list $F_1 \subseteq A$, FAS for G

```
1  F1 = []
2  MLF(L)
3  for i = 1 to |L| do
4  |  for j = i + 1 to |L| do
5  |  |  if (L[i], L[j]) ∈ A then
6  |  |  |  add (L[i], L[j]) to F1
7  |  |  |  remove it from A
8  |  |  end
9  |  end
10 |  if G is acyclic then
11 |  |  exit
12 |  end
13 end
14 return F1
```

Algorithm 2: RBA

Inputs : Directed strongly connected graph $G = (V, A)$, ordered list of vertices L
Output: Ordered list $B_1 \subseteq A$, FAS for G

```
1  B1 = []
2  MLB(L)
3  for i = 1 to |L| do
4  |  for j = i + 1 to |L| do
5  |  |  if (L[j], L[i]) ∈ A then
6  |  |  |  add (L[j], L[i]) to B1
7  |  |  |  remove it from A
8  |  |  end
9  |  end
10 |  if G is acyclic then
11 |  |  exit
12 |  end
13 end
14 return B1
```

3 Producing a minimal FAS

Given an acyclic directed graph $G = (V, A)$, let us suppose that by using either the algorithm RFA or RBA we have now a set of arcs F which is a FAS for G, i.e. the graph $G' = (V, A \setminus F)$ is acyclic. Now, we want to try to add to G' as many arcs as possible from F, so that the remaining set F' will be a minimal FAS. The simplest way is to go through the arcs in F and, one by one, add them to the graph. If, after adding one arc, the graph is no longer acyclic, we remove it again. Every arc needs to be tested only once, since the insertion of other arcs will not change the fact that the arc introduces a cycle. Finally, since $F \subset A$ and, as a consequence, $|F| < |A|$, the overall cost of such a simple approach is $\mathcal{O}(|A|(|V| + |A|))$.

A different approach to the problem of adding arcs from F into the graph G and, as a result minimizing a given set of arcs that were removed from the graph to make it acyclic, can be obtained using topological sorting. Given an acyclic graph we can topologically sort the set of its vertices and add back to the graph all the arcs in F which are forward arcs, i.e. which follow the order of the topological sorting. Such an approach is quite fast, since its overall time complexity once we have computed the topological sorting of the vertices, is $\mathcal{O}(|F|)$, however there is no guarantee that the obtained FAS is minimal.

If F is a FAS obtained by removing forward or backward arcs, such as the outputs of Algorithms 1 and 2, we do know that the arcs in F follow the order in which they were eliminated, until the graph became acyclic. Thus, it is obvious that when trying to reinsert arcs it is more convenient to try to reinsert the first ones that were eliminated not the last ones. So, we will now describe an algorithm which is quite effective, by using such a property of F and still using, when possible, the topological sort of the vertices. Experimentally, we saw that the way to add back as many arcs as possible is to avoid to reinsert all the arscs from a vertex, but, instead, try to reinsert arcs from as many vertices as possible.

More in details, we will go through the list F in many rounds, but at round i, if previously h arcs had been added to the graph, we go from the arc in position i to the arc in position $i + h$.

Such a heuristic for adding arcs, which we call minimizeFAS and which improves the heuristic to minimize the FAS presented in [4], is formally described in Algorithm 3. It ensures that no more than half of the arcs related to a specific vertex can be added at any round, and, experimentally, it has given us excellent results. Algorithm 3 is shown using Python 3 code. The way Python handles deletion in lists, guarantees what above described. We recall, for clarity, that the Python function *nx.topological_sort(G)* computes the topological sort of the list of vertices of an acyclic graph, while the function *nx.is_directed_acyclic_graph(G)* checks if a graph is acyclic.

Let us now study the computational complexity of minimizeFAS. Although it is done in more rounds, each arc of F is tested only once. So, considering that the cost of checking acyclicity is $\mathcal{O}(|V| + |E|)$ as well as the cost of computing the topological sort of the vertices, the overall complexity is $\mathcal{O}(|F|(|V| + |E|)) = \mathcal{O}(|E|(|V| + |E|))$.

Putting it all together, we have an overall heuristic algorithm shown in Algorithm 4 to produce a minimal feedback arc set given a strongly connected directed graph and given a linear ordering of its vertices, which extends and improve, as we will see experimentally, the heuristic algorithm presented in [4].

3.1 From linear orderings to minimal FAS

Given a directed graph $G = (V, E)$ which has more than one strongly connected component (with at least 2 vertices), and given s criteria to order the vertices of each component, we then apply the general Algorithm described in pseudo-code Algorithm 5. Simply put, given a directed graph G and s ordering criteria, for each strongly connected component, the algorithm OminFAS finds a minimal

Algorithm 3: Algorithm minimizeFAS

Inputs : Acyclic graph $G = (V, E)$, list of arcs F, not in G
Output: Sublist of F, which is a minimal Feedback Arc Set for graph G.

```
ad=1, ad1=0, tested =[]
ts=list(nx.topological_sort(G)) *ts=topological sort
                                   of the vertices of G
while ad==1 :
  for e in F:
      if e not in tested :
          if ts.index(e[0]) <  ts.index(e[1]):
              G.add_edge(*e)
              F.remove(e)
              ad1=1
          else:
              G.add_edge(*e)
              if nx.is_directed_acyclic_graph(G):
                  F.remove(e)
                  ad1=1
                  ts=list(nx.topological_sort(G))
              else:
                  G.remove_edge(*e)
                  tested.append(e)
  ad=ad1
  ad1=0
return F
```

Algorithm 4: Algorithm minFAS

Inputs : Strongly connected graph $G = (V, E)$, permutation L of V
Output: Subset F of E, which is a minimal Feedback Arc Set for graph G.

$F = RFE(G, L)$
$G' = (V, E \setminus F)$
$minF = minimizeFAS(G', F)$
$B = RBE(G, L)$
$G' = (V, E \setminus B)$
$minB = minimizeFAS(G', B)$
if $|minF| \leq |minB|$ **then**
  return $minF$
else
  return $minB$
end

FAS for each obtained linear ordering, and it chooses, for the component, the minimum among all the found minimal FAS. Finally, it returns the union of all the found minimal FAS.

Algorithm 5: Algorithm OminFAS

Inputs : Directed graph $G = (V, E)$, $c_1, \ldots, c_s$ ordering criteria
Output: Subset F of E, which is a minimal Feedback Arc Set for graph G.

Compute the Strongly Connected Components of G :
$C_1 = (V_1, E_1), C_2 = (V2, E_2), \ldots C_k = (V_k, E_k)$
for $i = 1$ *to* k **do**
  for $j = 1$ *to* s **do**
    Compute $L_{i,j}$, linear order of V_i using criterion c_j
    $FAS(i, j) = minFAS(C_i, L_{i,j})$
  end
  $FAS(i) = minimum(FAS(i, 1), \ldots, FAS(i, s))$
end
return $\bigcup_{i=1}^{k} FAS(i)$

Algorithm OminFAS depends heavily upon the linear orderings which are used. As we already remarked in Section 2, if we have a minimum FAS F for a given graph G, after the removal of the arcs in F, the obtained graph G' is acyclic and therefore we can topologically sort its vertices. The obtained ordering L_{ts} is such that all the arcs in G' go from left to right in it. Thus, given L_{ts} Algorithm OminFAS will return in output exactly the minimum FAS F. It follows that we need to find some good and efficient heuristics to produce linear orderings which could be as close as possible to the optimal one.

4 Heuristics for linear orderings

Let $G = (V, E)$ be a strongly connected graph. In our approach in [4], we ordered the vertices using the 2 simplest and most intuitive criteria: out-degree of a vertex, in-degree of a vertex. In turn, we obtained 4 different orderings: decreasing out-degree (do), increasing out-degree (io), decreasing in-degree (di), increasing in-degree (ii) and for each ordering, we produced 2 different FAS, one eliminating forward arcs and the other eliminating backward arcs. The initial asymptotic cost of ordering the vertices is $\mathcal{O}(|V| \log |V|)$. If the graph is not sparse, and so $|A| \sim |V|^2$ such an extra cost is absorbed into the $\mathcal{O}(|V| + |A|)$ cost of computing all the in-degrees and out-degrees of the vertices. The obtained results were compared to the results obtained in [9] and also in [11], where the authors propose a $\mathcal{O}(|V||E|^4)$–heuristic and by empirical validation they achieve an approximation rate of $r \leq 2$ with $r \approx 1.3606$ being a lower bound for the APX–hardness.

As noted, the algorithm *OFAS* presented in [4] represented a good balance between the ability to reach good minimal values (often minimum) and the

possibility of application to problems of high dimensions, like the ones described in [16]. Moreover, the algorithm was clearly performing a lot better when dealing with scale free graphs and, in turn, performed more poorly for graphs whose vertices had all approximately the same degrees.

4.1 Level-2 linear orderings

The 4 different linear orderings of the vertices we introduce now extend the set of orderings previously introduced and try to overcome their inherent weaknesses. If we look at the out-degree of a vertex as the number of paths of length 1 leaving the vertex and, analogously, the in-degree as the number of paths of length 1 entering the vertex, then we can extend such a definition to length 2.

Given a vertex $v \in V$ let us denote, for simplicity, as *successor* of v any vertex w such that $(v, w) \in E$ and as *predecessor* of v any vertex u such that $(u, v) \in E$. Let us also denote for any vertex $v \in V$ with $id(v)$ and $od(v)$ respectively the in-degree and the out-degree of vertex v. These two values measure the number of vertices at distance 1, respectively, forward and backward. Thus, if $id(v) = h$ and $od(v) = k$ then the v has h predecessors (distance 1 backward) and k successors (distance 1 forward).

We now go at distance 2, trying to overcome some of the problems with the previous linear orderings. In details, let $\pi(v)$ and $\sigma(v)$ denote, respectively, the set of predecessors and the set of successors of v. Given a vertex $v \in V$, the value

$$\alpha(v) = \sum_{w \in \sigma(v)} od(w)$$

measures the number of paths of length 2 starting from v. Analogously, the value

$$\beta(v) = \sum_{u \in \pi(v)} id(u)$$

measures the number of paths of length 2 ending in v.

We note that in such cases a vertex might be both at distance 1 and 2. Moreover, a vertex might be reachable by more than one path of length 2, both forward and backward.

Both values above can be weighted using in-degrees and out-degrees. In details, given a vertex v we multiply its α value by the ratio $\frac{od(v)}{id(v)}$ or by the ratio $\frac{id(v)}{od(v)}$. In the first case, the obtained value $\alpha_1(v)$ will be higher for vertices that can can reach a high number of vertices in two steps and moreover have a high out-degree and a low in-degree. In the second case, the obtained value $\alpha_2(v)$ will be, again, higher for vertices that can can reach a high number of vertices in two steps and moreover have a high in-degree and a low out-degree. Analogously for the β values.

Assuming that the graph G is strongly connected and, as a consequence, every vertex has both in-degree and out-degree greater than 0, to each vertex v

we assign the following 4 values:

$$\alpha_1(v) = \frac{od(v)}{id(v)}\alpha(v)$$
$$\alpha_2(v) = \frac{id(v)}{od(v)}\alpha(v)$$
$$\beta_1(v) = \frac{od(v)}{id(v)}\beta(v)$$
$$\beta_2(v) = \frac{id(v)}{od(v)}\beta(v)$$

If we keep in mind the optimal linear orderings L_b and L_f which would give us, respectively, a minimum FAS by removing backward arcs (L_b) or by removing forward arcs (L_f), we can reasonably assume that the first element of L_b would be a vertex with high out-degree, low in-degree, thus maximum $\frac{od(v)}{id(v)}$ value. Moreover, among all the vertices with the highest $\frac{od(v)}{id(v)}$ it would probably be the vertex which the highest number of forward connections of length 2. Therefore, it would probably be a vertex with the highest α_1 value.

Analogously, the first element of L_f would be a vertex with high in-degree, low out-degree and maximum $\frac{id(v)}{od(v)}$ value. Moreover, also the vertex with probably the highest number of backward connections of length 2, i.e. it would probably be an element with the highest β_2 value.

We note here that, although, using the ratio $\frac{id(v)}{od(v)}$ as ordering criterion would certainly be an interesting idea, already used in [18] for instance, for the specific test cases we are studying it would not give us any improvements.

We now define the 8 ordering criteria we are going to use for our tests:

O1: decreasing order of α_1 values, denoted with $d\alpha_1$;
O2: increasing order of α_1 values, denoted with $i\alpha_1$;
O3: decreasing order of α_2 values, denoted with $d\alpha_2$;
O4: increasing order of α_2 values, denoted with $i\alpha_2$;
O5: decreasing order of β_1 values, denoted with $d\beta_1$;
O6: increasing order of β_1 values, denoted with $i\beta_1$;
O7: decreasing order of β_2 values, denoted with $d\beta_2$;
O8: increasing order of β_2 values, denoted with $i\beta_2$.

5 Results

To prove the efficacy of our newly proposed linear orderings, we ran OminFAS on the same test cases used in [11] and [4] for comparison. The used ISCAS circuit testing dataset [8] is available at https://github.com/alidasdan/graph-benchmarks/tree/master/iscas.

The results where the algorithm and the linear orderings proposed in [4], column denoted as OFAS, did not reach the known minimum value or where the

minimum value is unknown, are shown in Table 1. We can see that in 4 of these 6 cases, namely cases $1, 2, 5, 6$, the OminFAS algorithms, using the new linear orderings, improves the results obtained in [4] and in 3 of them, namely $1, 5, 6$, reaches the minimum value.

Another useful set of tests, was introduced and described in [1], also available at the URL above cited, where one can find much larger graphs. Following [11], we ran our algorithms on 3 of them. In one case the algorithm in [4] reached the minimum value, the other 2 cases, where the minimum is unknown, are shown in Table 1 under IBM code.

For the graphs with just one SCC, we can easily highlight some extra information. Table 2 shows the best values obtained by each of the 8 orderings. For the test case denoted "dsip" we show the results for each of its two strongly connected components (denoted dsip1 and dsip2). Although small, such a sample of test cases is quite significant as we can see of variety of sizes for the graphs involved. We can see that the average value of the results for the 8 orderings is very close to the minimum value, except for the first graph which has a relatively small number of vertices and arcs. In other words, the proposed orderings and the algorithm OminFAS are giving very reliable and stable results. In particular, in 3 of the test cases, the minimal results is obtained by more than one ordering.

Table 3 also shows how we got the minimum value, either by eliminating forward or backward arcs. It shows also the number of swaps and the percentage of arcs which remain in the FAS after we minimize the initially obtained FAS. Here we can see the efficacy of the proposed minimization heuristics. In most cases, we can see how we can reinsert in the graph more than 50% of the arcs which were initially eliminated. We can also see, how significant the proposed simple heuristic of swapping vertices in the ordering before removing forward or backward arcs.

Table 1. Comparing algorithms

ISCAS Code	Vertices-Arcs	SCC	Min	OFAS	OminFAS
1 s953	730-1090	1	6	**7**	**6**
2 s5378	3076-4589	1	30	**34**	**32**
3 s9234	3083-4298	21	90	**91**	**91**
4 dsip	4079-6602	2	-	**159**	**159**
5 s38584	20349-34562	1	1080	**1089**	**1080**
6 s38417	24255-34876	437	1022	**1023**	**1022**
IBM Code	Vertices-Arcs	SCC	Min	OFAS	OminFAS
ibm01	12752-36048	6	-	**1840**	**1815**
ibm02	19601-57753	1	-	**3837**	**3804**

Table 2. Comparing the 8 linear orderings

Code	Vertices-Arcs	$O1$	$O2$	$O3$	$O4$	$O5$	$O6$	$O7$	$O8$	Avg	Avg/Min
s953	730-1090	6	7	12	23	12	10	10	6	10.75	1.79
s5378	3076-4589	36	46	39	33	37	32	41	33	37.125	1.16
dsip 1	1120-1456	79	81	84	103	79	114	81	79	87.5	1.10
dsip 2	1120-1456	82	81	80	99	82	102	81	82	86.125	1.07
s38584	20349-34562	1080	1249	1149	1158	1209	1169	1110	1095	1152.375	1.06
ibm02	19601-57753	3804	3832	4083	4108	3877	3883	3848	3847	3910.25	1.03

Table 3. From initial to minimal FAS

Code	F/B	Initial FAS	Min FAS	Order	Swaps	% Initial FAS
s953 (1)	B	18	**6**	**O1**	4	33%
s953 (2)	B	19	**6**	**O8**	1	31%
s5378(1)	F	165	**32**	**O6**	95	19%
s5378(2)	B	350	**32**	**O6**	96	9%
dsip 1	B,B,B	103, 103, 103	**79**	**O1,O5,O8**	0,1,0	76%
dsip 2	B,F	107, 107	**80**	**O3**	0,1	74%
s38584	B	2176	**1080**	**O1**	904	49%
ibm02	B	7071	**3804**	**O1**	744	53%

6 Conclusions and future work

In our present work, we have proposed an improvement of the algorithm presented in [4]. By using different heuristics to weight the vertices of a given directed graph, we were able to produce linear orderings which allowed us to obtain minimal Feedback Arc Sets for strongly connected graphs.

Our algorithm described in Subsection 3.1, lends itself naturally to a possible parallel execution. We could have 8 parallel threads working on each SCC and the minimum of the 8 results would be taken into account.

As a future research project, we also intend to work on extending our algorithm by finding efficient heuristics to improve the produced linear orderings by taking into account as different factors as:

- Dynamic properties of the vertices, i.e. if for instance we are eliminating forward arcs and the next vertex in the linear ordering has now in-degree equal to 0, all its outgoing forward arcs could be eliminated and then put back safely, without introducing new cycles. We have performed some initial experiments using this properties and the results are quite promising.
- A more theoretical characterization of the linear orderings which produce a minimum FAS and try to make the initially given linear orderings or the orderings produced thereafter consistent with such properties.
- Dynamically compute the SCC's of the graph, after the elimination of outgoing forward arcs or incoming backward arcs of a vertex. This will allow us to concentrate only on arcs within the same SCC. We have already performed

some tests on it and obtained some slight improvements. In pursuing such an extension and testing its efficacy, we will generate test cases of known minimum FAS using the algorithm introduced in [15].

.

Finally, another line of research we intend to follow, is related to the work done in [5, 7], where the *twin* Minimum Feedback Vertex Set Problem was tackled by means of evolutionary metaheuristics and, in particular an optimization Immune Algorithm [19], which has been proven to be among the best derivative free optimization algorithms for many problems, even not so common ones such as [17] or [6], and for which many related interesting problem could be addressed, such as the lifespan and the age assignment of an individual in the population [20]. It would also be quite interesting to see parallel evolutions of different populations of individuals (solutions), each one of them randomly initiated using the different linear orderings.

Acknowledgments. This research is supported by the project Future Artificial Intelligence Research (FAIR) – PNRR MUR Cod. PE0000013 - CUP: E63C22001940006

References

1. Alpert, C.J.: The ispd98 circuit benchmark suite. In: Proceedings of the 1998 International Symposium on Physical Design. p. 80–85. ISPD '98, Association for Computing Machinery, New York, NY, USA (1998). https://doi.org/10.1145/274535.274546
2. Baharev, A., Schichl, H., Neumaier, A., Achterberg, T.: An exact method for the minimum feedback arc set problem. ACM J. Exp. Algorithmics **26** (apr 2021). https://doi.org/10.1145/3446429, https://doi.org/10.1145/3446429
3. Berger, B., Shor, P.W.: Approximation alogorithms for the maximum acyclic subgraph problem. In: Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms. p. 236–243. SODA '90, Society for Industrial and Applied Mathematics, USA (1990)
4. Cavallaro, C., Cutello, V., Pavone, M.: Effective heuristics for finding small minimal feedback arc set even for large graphs. In: CEUR Workshop Proceedings. vol. 3606 (2023), https://ceur-ws.org/Vol-3606/paper56.pdf
5. Cutello, V., Oliva, M., Pavone, M., Scollo, R.A.: An immune metaheuristics for large instances of the weighted feedback vertex set problem. In: 2019 IEEE Symposium Series on Computational Intelligence (SSCI). pp. 1928–1936 (2019). https://doi.org/10.1109/SSCI44817.2019.9002988
6. Cutello, V., Fargetta, G., Pavone, M., Scollo, R.A.: Optimization algorithms for detection of social interactions. Algorithms **13**(6) (2020). https://doi.org/10.3390/a13060139, https://www.mdpi.com/1999-4893/13/6/139
7. Cutello, V., Oliva, M., Pavone, M., Scollo, R.A.: A hybrid immunological search for the weighted feedback vertex set problem. In: Matsatsinis, N.F., Marinakis, Y., Pardalos, P. (eds.) Learning and Intelligent Optimization. pp. 1–16. Springer International Publishing, Cham (2020)
8. Dasdan, A.: Experimental analysis of the fastest optimum cycle ratio and mean algorithms. ACM Trans. Des. Autom. Electron. Syst. **9**(4), 385–418 (oct 2004). https://doi.org/10.1145/1027084.1027085

9. Eades, P., Lin, X., Smyth, W.: A fast and effective heuristic for the feedback arc set problem. Information Processing Letters **47**(6), 319–323 (1993). https://doi.org/https://doi.org/10.1016/0020-0190(93)90079-O
10. Grötschel, M., Jünger, M., Reinelt, G.: Comments on "an exact method for the minimum feedback arc set problem" **27** (jul 2022). https://doi.org/10.1145/3545001, https://doi.org/10.1145/3545001
11. Hecht, M., Gonciarz, K., Horvát, S.: Tight localizations of feedback sets. ACM J. Exp. Algorithmics **26** (mar 2021). https://doi.org/10.1145/3447652
12. Karp, R.M.: Reducibility among Combinatorial Problems, pp. 85–103. Springer US, Boston, MA (1972). https://doi.org/10.1007/978-1-4684-2001-2_9
13. Kudelić, R.: Feedback Arc Set: A History of the Problem and Algorithms. Springer Nature (2022)
14. Lucchesi, C.L., Younger, D.H.: A minimax theorem for directed graphs. Journal of the London Mathematical Society **s2-17**(3), 369–374 (1978). https://doi.org/https://doi.org/10.1112/jlms/s2-17.3.369
15. Saab, Y.: A fast and effective algorithm for the feedback arc set problem. Journal of Heuristics **7**(3), 235–250 (May 2011). https://doi.org/10.1023/A:1011315014322
16. Simpson, M., Srinivasan, V., Thomo, A.: Efficient computation of feedback arc set at web-scale. Proc. VLDB Endow. **10**(3), 133–144 (nov 2016). https://doi.org/10.14778/3021924.3021930
17. Stracquadanio, G., Greco, O., Conca, P., Cutello, V., Pavone, M., Nicosia, G.: Packing equal disks in a unit square: an immunological optimization approach. In: 2015 International Workshop on Artificial Immune Systems (AIS). pp. 1–5 (2015). https://doi.org/10.1109/AISW.2015.7469239
18. Sun, J., Ajwani, D., Nicholson, P.K., Sala, A., Parthasarathy, S.: Breaking cycles in noisy hierarchies. In: Proceedings of the 2017 ACM on Web Science Conference. p. 151–160. WebSci '17, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3091478.3091495, https://doi.org/10.1145/3091478.3091495
19. Timmis, J., Hone, A., Stibor, T., Clark, E.: Theoretical advances in artificial immune systems. Theoretical Computer Science **403**(1), 11–32 (2008). https://doi.org/https://doi.org/10.1016/j.tcs.2008.02.011
20. Vitale, A., Di Stefano, A., Cutello, V., Pavone, M.: The influence of age assignments on the performance of immune algorithms. In: Lotfi, A., Bouchachia, H., Gegov, A., Langensiepen, C., McGinnity, M. (eds.) Advances in Computational Intelligence Systems. pp. 16–28. Springer International Publishing, Cham (2019)

A Real-Time Adaptive Tabu Search for Handling Zoom In/Out in Map Labeling Problem

Vincenzo Cutello[0000−0002−7521−3516], Alessio Mezzina, Mario Pavone[0000−0003−3421−3293], and Francesco Zito[0000−0003−1374−0510]

Department of Mathematics and Computer Science
University of Catania
viale Andrea Doria 6, 95125 Catania, Italy
cutello@unict.it, mpavone@dmi.unict.it, francesco.zito@phd.unict.it

Abstract. The paper delves into the analysis of optimization problems that involve changing problem constraints over time and proposes a method to adapt metaheuristic algorithms for real-time applications. The dynamic map labeling problem serves as a case study. In the static version of the problem, the goal is to position labels without overlap for maximum readability. However, in dynamic scenarios where the map view changes, the task becomes more challenging, requiring labels to be adjusted in real-time. To address this challenge, a Tabu Search algorithm is adapted to function in a dynamic environment where user interactions can alter the map view, changing the solution space of possible label placements on the current map view.

Keywords: Real-Time Optimization · Dynamic Map Labeling Problem · Metaheuristics · Tabu-Search · Dynamic Constraints Optimization Problems

1 Introduction

Optimization algorithms play a pivotal role in enhancing a variety of industrial processes, including logistics, transportation, scheduling, and human behavior [1,11,9]. These algorithms lead to more efficient resource utilization, reducing costs. An optimization problem is defined by constraints and an objective function, and aims at finding a solution, within the set of feasible solutions, that maximizes the objective function. Metaheuristic algorithms have been used in real-world problems in diverse domains, offering efficient and adaptable solutions [16]. These algorithms, which draw inspiration from nature or heuristic strategies such as warm optimization, evolutionary algorithms, and artificial immune systems, are particularly effective in addressing optimization challenges where conventional exact methods may prove impractical or computationally very expensive, as in the case of NP-hard problems [21,17].

These algorithms are typically ad-hoc designed to address problems with well-defined constraints and objectives [2,4,10,3], such as optimizing hyperparameters in machine learning models [22]. While metaheuristics offer flexibility

and adaptability, their features make them less suitable for real-time applications where predictability, optimality, and bounded execution times are critical [7]. Real-time applications involve interactions with external entities, such as humans, that change the settings of an optimization problem over time. The dynamic nature of these interactions necessitates that a metaheuristic algorithm is able to adapt when constraints and objectives change over time. The presence of external agents therefore generates a dynamic landscape in which the optimal solution found by the algorithm at a specific time may no longer be optimal in the next one [18].

In this paper, we focus on enhancing a metaheuristic algorithm, specifically Tabu Search as described in [5], for real-time applications such as dynamic map labeling problem. The algorithm is designed to dynamically adjust the process of inserting labels on a map in response to user interactions, such as zooming in or out. A key innovation of our approach is its ability to operate across multiple solution spaces, rather than being limited to a single search space. This enables a more adaptable and efficient response, ensuring continuous optimization of map labels in real-time.

2 Dynamic Constraints Optimization Problems

An optimization problem is defined by two elements: objective function, and constraints. The objective function typically seeks to either maximize or minimize a certain quantity, such as profit, cost, efficiency, or any other measurable metric [19]. The constraints define the search space and regions of feasible solutions, which the algorithm explores so to find the optimal solution based on the objective function [20]. These constraints can be equality constraints, inequality constraints, or a combination of both, depending on the specific problem being addressed [13]. The final goal is to find the optimal solution that either maximizes or minimizes the objective function while satisfying all constraints. Formally, a typical optimization problem, for instance minimization, is defined as:

$$\min_{x} \; f(x), \quad \text{s.t.} \quad x \in X, \tag{1}$$

where $f : X \to \mathbb{R}$ is the objective function,and $X \subseteq \mathbb{R}^n$ is the solution space with n representing the number of dimensions.

Optimization algorithms explore the search space by using different strategies in order to find a solution that optimizes the objective function (minimization in case of eq. 1). If the constraints of the problem change, then it is necessary to update the solution space to a new one that contains all the feasible solutions that satisfy the new constraints. This type of optimization problem with dynamic constraints can be modeled as follows:

$$\min_{x^{(t)}} \; f\left(x^{(t)}\right), \quad \text{s.t.} \quad x^{(t)} \in X^{(t)}, \tag{2}$$

where $X^{(t)} \subseteq \mathbb{R}^n$ is the solution space at time t. Let us consider two solution spaces at two distinct instants of time, t_1 and t_2, with $t_1 < t_2$; the two solution

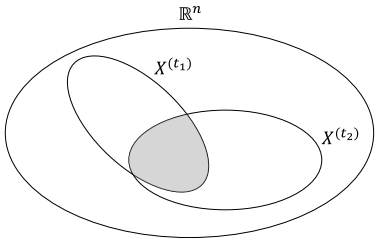


Fig. 1: Representation of two solution spaces, before $X^{(t_1)}$ and after $X^{(t_2)}$ changing the constraints of the problem.

spaces are respectively denoted with $X^{(t_1)}$ and $X^{(t_2)}$. The intersection of the two solution spaces can be a non-empty set (Figure 1), indicating that there are common feasible solutions that satisfy the constraints at both time t_1 and t_2. On the other hand, it is possible that a solution x at time t_1, contained in the solution space $X^{(t_1)}$, may not be a feasible solution at time t_2 because it does not satisfy the constraints that define the new solution space $X^{(t_2)}$. As a consequence, all the solutions in $X^{(t_1)}$ that are not contained in $X^{(t_2)}$ must be discarded at time t_2 due to the fact that they do not satisfy the new constraints.

In the formulation made above, we considered the fact that the solution spaces over time are contained in the same vector space with dimension n, that is $X^{(t)} \subseteq \mathbb{R}^n \; \forall t > 0$. However, in real-time optimization, this may no longer be true. Indeed, changing the constraints of the problem can change the dimensions of the vector space. Formally, if $X \subseteq \mathbb{R}^n$ is a solution space at time t_1, and if a change of constraints of the optimization problem alters the dimensions of the solution space, then the new solution space will be defined as $Y \subseteq \mathbb{R}^m$, where m is the new dimension of the solution space. Therefore, it might be necessary to map a solution from a space of dimension n to a solution from a space with dimension m. To this end, we define a function $\phi : \mathbb{R}^n \to \mathbb{R}^m$. When the problem constraints are modified, all solutions within the solution space X obtained by the algorithm up to that point must be mapped to the new solution space Y to adhere to the updated constraints (Figure 2). Subsequently, the optimization algorithm can proceed to enhance these solutions in order to identify the optimal solutions within the new solution space. It can be stated that two solutions, $x \in X$ and $y \in Y$, obtained at different times, may have different objective function values. This contrasts with Equation 2, where the domain of the objective function remains constant despite changes in problem constraints. In this new formulation, the objective function is also influenced by changes in constraints. This can then be formally defined as:

$$\min_{x} \; f_X(x), \quad \text{s.t.} \quad x \in X, \tag{3}$$

where $f_X : X \to \mathbb{R}$ is the objective function to be minimized; $X \in \Omega$ is the current solution space that the algorithm is exploring; and Ω is the set of all possible solution spaces that we can have.

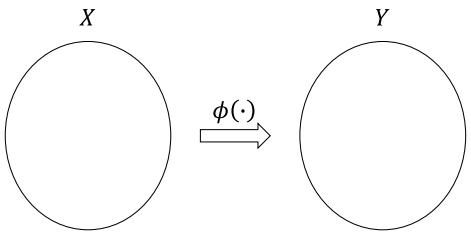


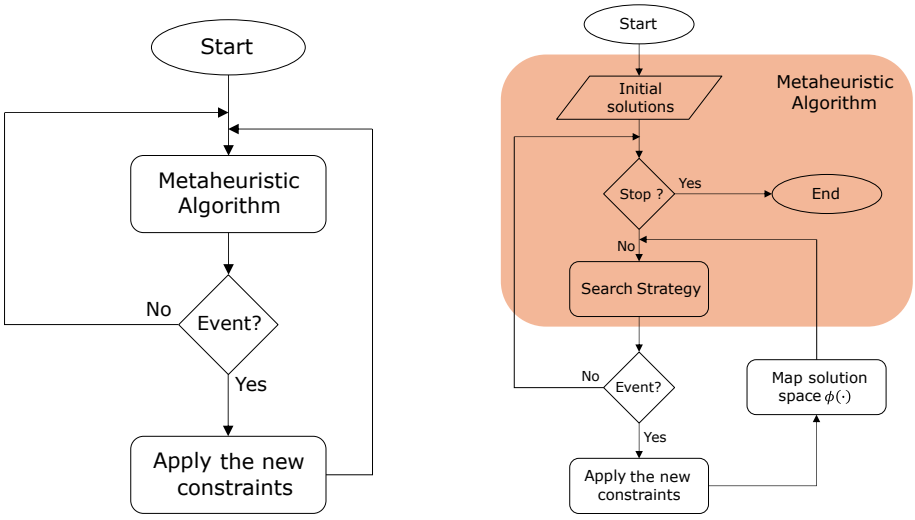
Fig. 2: Mapping a solution from solution space X to solution space Y after having modified the constraints of the optimization problem.

3 Metaheuristics in Real Time Optimization

Traditional optimization algorithms are typically not well-suited for real-time problems that involve dynamic interactions with external agents capable of dynamically changing the parameters to be optimized. As a result, several strategies need to be employed to adapt an optimization algorithm so to be used in such scenarios [6]. Real-Time Optimization (RTO) is a branch of optimization that focuses on applying optimization algorithms to real-world problems where the constraints of the problem evolve over time [14].

Metaheuristics are specifically developed to address optimization problems [16]. They iteratively enhance a solution until either the optimal solution is reached, or a stopping criterion is met [8]. Traditionally, when the constraints of the problem change and consequently the search space, the algorithm is stopped and restarted with the new constraints and objectives (Figure 3a). In [15], for instance, various strategies for addressing the Dynamic Vehicle Routing Problem are addressed. Among the different techniques explored, the authors discussed a Parallel Services (PS) framework referred to as ContDVRP. This framework involves restarting an optimization algorithm each time the parameters of the problem change. However, this approach can be inconvenient, especially in applications where the environment in which the algorithm operates is complex and expensive. Moreover, restarting the algorithm from the beginning results in the loss of previous knowledge, leading to restarting exploration from scratch on the new search space.

In order to adapt a metaheuristic in real-time optimization, it is essential to address how the algorithm should respond to external events that occur. These events typically involve changes in the constraints of the optimization problem, often triggered by human actions. Unlike metaheuristics based on a stopping criterion, the ones employed for real-time optimization must operate continuously without having a specific stopping condition. They are designed to constantly optimize solutions. In the absence of any external events, when a metaheuristic algorithm converges toward an optimal solution, if the lower or upper bound in the case of minimization or maximization problems are known a priori, it proceeds to look for alternative optimal solutions. Another strategy involves pausing the algorithm upon reaching an optimal solution, if it is known,



(a) Common Strategy (b) Proposed strategy

Fig. 3: Two strategies for adapting a metaheuristic algorithm for real-time optimization.

resuming its operation each time an event occurs that alters the solution space. Otherwise, if the lower or upper bound is unknown, the algorithm attempts to improve the current solution (Figure 3b). Therefore, a metaheuristic algorithm must be able to quickly adapt to these changes introduced by external agents and adjust its strategy in real-time to find new solutions that align with the new problem constraints.

4 Dynamic Map Labeling Problem

To demonstrate the adaptability of metaheuristics to real-time applications, as shown in Figure 3b, we examine the task of labeling buildings on a map as a case study. Previous research has explored the use of optimization algorithms for similar challenges, such as for instance the one in [12], where a trajectory-based dynamic map labeling algorithm was proposed. This algorithm predicts the future positions of objects and assigns labels based on their trajectories, utilizing a combination of greedy heuristics and simulated annealing optimization methods.

This section focuses on assessing the response time of a metaheuristic in real-time scenarios. To demonstrate this, we have enhanced and adapted an existing Tabu Search algorithm, proposed in [5], to accurately position building labels on a map, thereby preventing label overlaps and ensuring readability. The placement of labels on the map is dynamic, adjusting as users zoom in or out on the map. The enhancement involves evolving the algorithm to handle multiple continually

changing solution spaces, each corresponding to different map zoom levels. This evolution marks a significant shift from handling a single static solution space to managing a series of dynamic solution spaces, requiring the algorithm to rapidly adjust to new conditions and find optimal label placements in real-time.

4.1 Problem formulation

In the context of map labeling, the solution format includes details such as label positions on the map, label orientations, and text distribution across multiple lines. All this information is represented by a vector of integer numbers of size $3 \times n$, where n is the total number of labels to be placed on the map [5]. Hence, a solution is encoded as:

$$(p_1, o_1, d_1, p_2, o_2, d_2, \ldots, p_n, o_n, d_n)\,, \tag{4}$$

where p_i, o_i, and d_i denote the position, orientation, and text distribution of the i-th label, respectively. The quality of a solution depends on the overlapping among labels on the map. To this end, we define a matrix denoted as $Q \in \mathbb{R}^{n \times n}$, which contains the percentage of overlap between the i-th and j-th labels. The objective is to minimize the overlap area among labels. For further information on how label placements are evaluated, refer to [5]. In the original version of the algorithm, a map always contains the same number of labels to be placed on the buildings. However, in this new version, zoom levels have been introduced. Each zoom level represents a reduction or expansion of the original map, corresponding therefore to either a subset or a superset of the buildings to be labeled. An action of zooming in or out on a map means changing the elements of the map itself and, consequently, altering the solution space in which the algorithm works. According to Equation 3, the set of all possible solution spaces is then defined as:

$$\Omega = \{X_1, X_2, \ldots, X_z\}, \tag{5}$$

where z represents the number of zoom levels considered, and X_i with $i = (1, \ldots, z)$ represents the solution space created based on the elements visible on the map at zoom level i. The user's interaction with the map, specifically the zoom event, is simulated through the modification of displayed elements on the map at time t. Two distinct zoom strategies are employed, referred to as *sequential* and *random*. In the sequential strategy, following a zoom event, the new zoom level may either increase or decrease by one level. Conversely, in the random strategy, upon a zoom event, the zoom level is chosen randomly.

The transition between two levels of zoom is possible through a mapping function (denoted with ϕ and described in Section 2) which is responsible for modifying the solution space by mapping the current solution to a new solution space. Indeed, in the case of a zoom in, the reduction in the number of labels on the map decreases label overlaps. Conversely, zooming out may increase the cost of the solution as the overlapping between labels increases with the growing number of labels. The zoom mechanism works by utilizing the map's coordinates, specifically by identifying the four corners of the map along with its center.

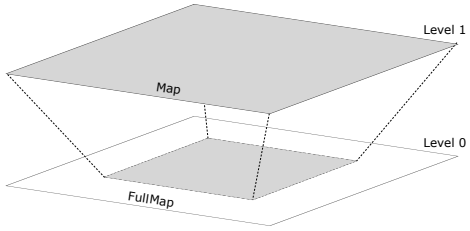


Fig. 4: This figure illustrates how zoom-in works on a map.

Subsequently, it generates z maps by progressively zooming towards the center. Thus, for example, it transitions from displaying the full map at 100% (zoom level 0) to a different zoom level, such as 80% of the map. The zoom is carried out taking into account the center of the map and the main diagonal, subsequently calculating the new corners to be considered (Figure 4). When there is a zoom in or zoom out event, the buildings displayed on the map change, affecting the labels as well. Consequently, the solution format (Equation 4) differs as it now considers a different subset or superset of buildings. The process involves repositioning the labels on the map and transforming a solution encoded in the previous format, before a zoom event, into a new solution encoded in the updated format, to align with the new solutions space. Labels that were not visible on the map before the event, but are now shown after the zoom, are included in the solution using default values.

4.2 Dynamic Map Labeling Algorithm

In this subsection, we present the pseudo code (Algorithm 1) for the dynamic map labeling algorithm that can be employed for real-time optimization in dynamic environments. We used the same objective and neighborhood functions introduced in [5]. Here, they are denoted as $f_X(\cdot)$ and $nh_X(\cdot)$, respectively, and depend upon the solution space $X \in \Omega$ under consideration.

The core of this algorithm lies in its ability to effectively manage label placements and switch solution spaces using the function $\phi(\cdot)$ as explained earlier. If no zoom events occur during the execution of our version of the tabu search algorithm, it will employ the same exploration strategy as a standard tabu search optimization. The key difference is that even if the best solution is found, the optimization cycle will not be interrupted until the maximum time t_{max} is reached.

User interactions play a critical role in how the algorithm continues to try to optimize the current solution and are simulated by adjusting the zoom level in the map at regular intervals of time denoted by δ_t. During the search process of the algorithm, particularly during the calculation and evaluation of the neighborhood of the current solution, the algorithm consistently monitors any changes in the zoom level (lines from 9 to 14 in Algorithm 1). Upon detecting a change, the algorithm interrupts the current neighbor search process, facilitating a transition to a new solution space by mapping the current and tabu-listed

Algorithm 1: Dynamic Map Labeling Algorithm

```
   Input: Initial label placement x, Initial zoom level i, Set of solutions spaces Ω, Tabu list
          size L, Maximum time of execution t_max
   Output: Best label placement x*
 1 x* ← x; tabu_list ← ∅;
 2 Let X be the solution space contained in Ω associated with the i-th level of zoom;
 3 while max execution time is not reached do
 4 |   best_candidate ← None; best_candidate_f ← float('inf');
 5 |   for y in nh_X(x) do
 6 |   |   if y not in tabu_list and f_X(y) < best_candidate_f then
 7 |   |   |   best_candidate = y; best_candidate_f = f_X(y);
 8 |   |   end
 9 |   |   if a zoom event occurred then
10 |   |   |   Let X̂ ∈ Ω represent the solution space associated with the new zoom level j;
11 |   |   |   Map the current best solution (x*) and the elements of the tabu_list from
   |   |   |    the solution space X to the solution space X̂ by using the function φ;
12 |   |   |   X ← X̂; i ← j;
13 |   |   |   Stop this iteration and go to the next iteration;
14 |   |   end
15 |   end
16 |   if best_ candidate is not equal to None then
17 |   |   if best_candidate_f < f_X(x*) then
18 |   |   |   x* ← best_candidate;
19 |   |   end
20 |   |   Add best_candidate to tabu_list;
21 |   end
22 |   if tabu list size exceeds L then
23 |   |   Remove oldest move from tabu_list;
24 |   end
25 end
```

solutions to this new context using the function $\phi(\cdot)$. The *if* statement within the *for* loop, at line 9 in Algorithm 1, improves the computational efficiency by preventing to process irrelevant neighbors of x and, in turn, minimizing wasted time and resources. As a result, the algorithm may experience a slight delay in solution updates during the neighborhood calculation phase. The maximum delay occurs when a zoom event happens while the algorithm is evaluating a neighborhood solution. Therefore, the interval of time in which a zoom event occurs needs to be greater than or equal to the time it takes for the algorithm to evaluate a solution. To address this, we introduce a waiting time parameter denoted as φ_t such that the interval of time between two consecutive events is defined as:

$$\delta_t = \varphi_t \cdot e_t, \tag{6}$$

where e_t is the estimated time necessary to evaluate a solution. To minimize delays, one potential strategy is to pre-calculate label placements once and reuse them each time the user views a map at the same zoom level. Such an approach could significantly speed up the transition between zoom levels by eliminating the need to recalculate the default solution of the new labels, but it would lead to higher data storage demands for multiple zoom levels, increasing memory usage. Saving the current state at each instance fails to address real-time optimization requirements, and as such, this approach will not be further explored in this paper.

Table 1: Parameters of the algorithm

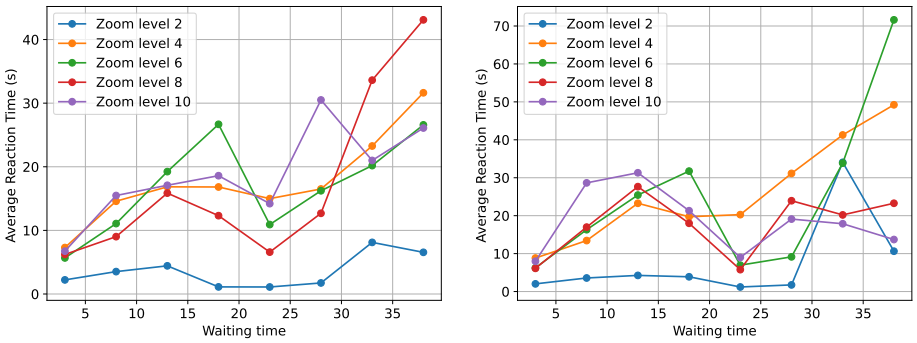
Symbol	Description	Value
Ω	Set of solution spaces	
t_{max}	Time of the simulation	1200s
z	Specifies the number of zoom levels available	6
s	Specifies the zoom strategy	Sequential
φ_t	Waiting time used to calculate the duration between two consecutive zoom events (Equation 6)	3
L	Tabu list size	30
i	Initial zoom level	0

Table 2: Statistical analysis on reaction times measured across various maps.

Map	Labels	Events	R_t avg (s)	R_t max (s)	R_t min (s)	R_t std (s)
Milan	63	143	6.2397	9.5781	0.0	3.2479
Paris	66	109	10.3913	12.75	0.0	1.9571
Rome	172	33	35.3945	43.4375	30.0	3.184
Taormina	61	142	6.8724	10.3125	0.0	3.2247
Venice	155	40	31.4538	37.7813	23.2656	3.3392

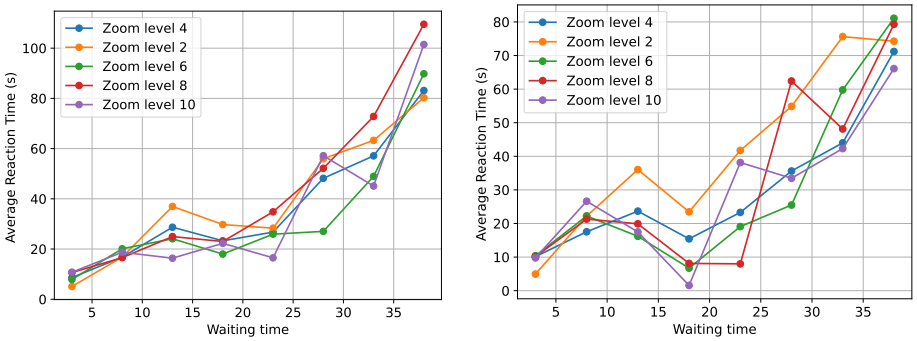
5 Results

In order to evaluate the algorithm's adaptability within the solution space, we used reaction time as a metric. The reaction time (R_t) represents the time taken by the algorithm to adjust map labels to optimize fitness following a zoom event and preceding another event. Thus, it is the time interval between a zoom event and when the algorithm identifies the best solution. If a zoom event occurs at time t_{z_1} and the algorithm finds a solution at time t ($t > t_{z_1}$), without further improving it before another zoom event at time t_{z_2}, the difference between t and t_{z_1} is defined as the reaction time, while the waiting time is the difference between t_{z_2} and t_{z_1}. In the worst-case scenario, the algorithm may take the entire time interval between two consecutive zoom events to improve a solution. In such cases, the reaction time will be equal to the waiting time, which can be viewed as an upper bound for the reaction time. The average reaction time is calculated as the mean of the reaction times throughout the simulation. We tested our algorithm on five different maps using the parameters outlined in Table 1. These parameters are selected empirically but we will provide a careful analysis on some of them later in this section. Instead, the maps selected for our experiments were sourced from OpenStreetMap, an open-source platform for exporting maps in OSM format. Table 2 displays the average reaction times for each map under analysis. Zoom events are triggered at regular intervals based on the parameter φ_t defined in Equation 6. Additionally, we investigated the distribution of reaction times, which indicate the time required to achieve an



(a) Random (b) Sequential

Fig. 5: Average reaction time based on zoom levels, waiting time and zoom strategy by considering Milan city map.

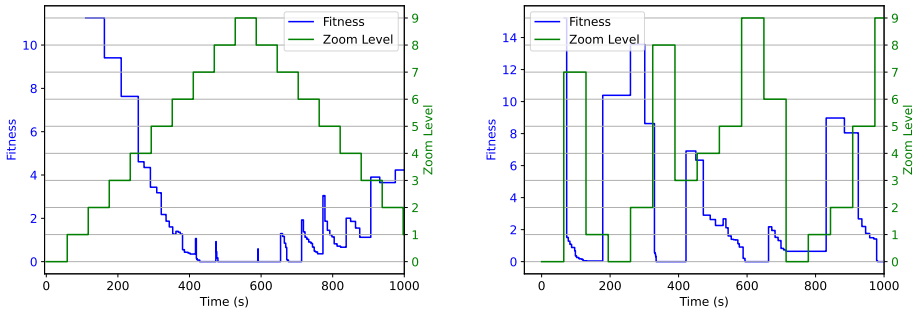


(a) Random (b) Sequential

Fig. 6: Average reaction time based on zoom levels, waiting time and zoom strategy by considering Paris city map.

optimal label placement after a change in zoom level. The columns in the table also include the minimum and maximum reaction times, along with the standard deviation, in addition to the mean value. It is evident that the number of zoom events occurring during the simulation varies, even when using the same waiting time φ_t. As can be inferred from Equation 6, the real waiting time depends on the time to evaluate a solution, which depends on the map's complexity and is proportional to the number of layers (check Rome in Table 2).

To better understand the impact of the parameters *waiting time* (φ_t), *number of zoom levels* (z), and *zoom strategies* (s), on average reaction time, additional experiments were conducted. Figures 5 and 6 illustrate the relationship between average reaction time and the algorithm parameters mentioned earlier. It is evident that the average reaction time is directly proportional to the number of

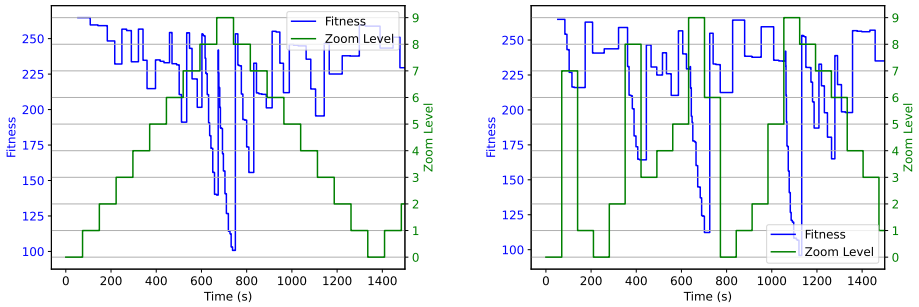


(a) Sequential (b) Random

Fig. 7: Comparing fitness evolution over time using sequential 7a and random 7b zoom strategies on Milan city map.

levels. Interestingly, with an increase in waiting time, the reaction time remains relatively consistent up to a certain threshold, beyond which it sharply increases. This behavior can be explained by the fact that increasing the waiting time results in a larger time window between consecutive zoom events. Consequently, the algorithm has more time to refine a solution and potentially find a slightly better label placement. On the other hand, when the waiting time is shorter, the interval between consecutive events is reduced. Therefore, when a zoom event occurs, the algorithm breaks the current iteration, updates the solution space, and resumes the search in the new solution space. Regarding the relationship between the average reaction time of the algorithm and the zoom strategies employed (random or sequential), there does not appear to be a clear correlation, as evidenced by the plots. The same consideration can also be applied to other maps that, due to space limitations, cannot be included in this paper.

To further explore in detail how the algorithm handles solutions and adapts to a new solution space, we conducted an analysis of how the fitness evolves over time. This analysis highlighted the sensitivity of fitness values to changes in dynamic contexts. The figures 7 and 8 depict the changes in fitness over time during zoom events. These plots illustrate the evolution of fitness over time and the corresponding zoom levels during zoom events. To generate these plots, we set a maximum time of 20 minutes, a waiting time of 30, and 10 zoom levels. Due to space constraints, we only included plots for the maps of Milan and Paris. The plot on the left displays the fitness evaluation results of using a sequential zoom strategy, while the plot on the right shows the fitness evaluation results of using a random zoom strategy. The green peaks in these plots represent operations of zooming in or out, and they have a significant impact on the effectiveness of the current solution as it is possible to see from the blue line. In the random zoom strategy (Figures 7b and 8b), the algorithm is more vulnerable to sudden changes in fitness due to the solution space undergoing drastic changes. For instance, there could be a shift from maximum zoom level, where the number



(a) Sequential (b) Random

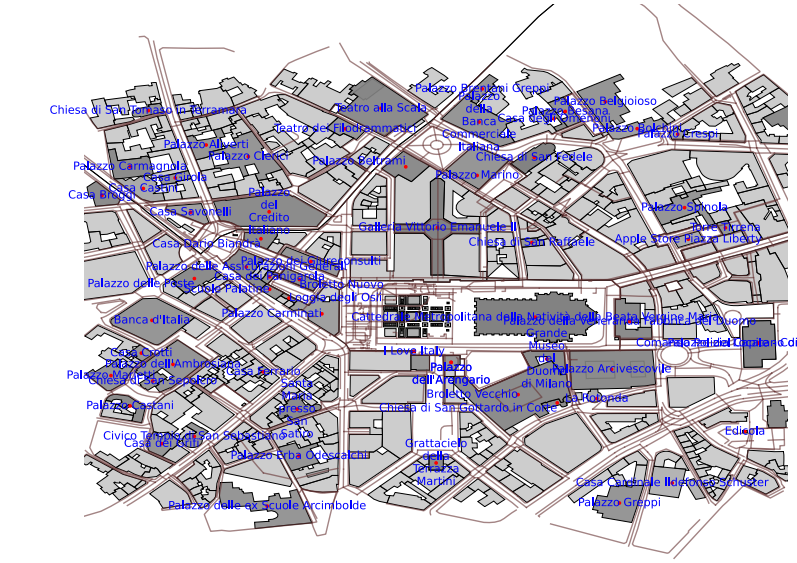
Fig. 8: Comparing fitness evolution over time using sequential (8a) and random zoom (8b) strategies on Paris city map.

of labels is significantly smaller, to a scenario with no zoom applied, resulting in the map containing many more labels and requiring the algorithm to start with default label placements. On the other hand, in sequential zoom strategies (Figures 7a and 8a), this transition is not as evident. The shift from one zoom level to another is gradual as two sequential zoom levels share some elements viewed on the map. As a result, the algorithm only needs to reposition a smaller number of labels than when two levels of zoom are distant as in the random zoom strategy can happen. When the zoom level increases from a lower value to a higher value, the number of labels displayed on a map gradually decreases as the map is zoomed in. Consequently, the algorithm must adjust a smaller number of labels to prevent overlaps between them. This results in a decrease in the fitness value, as fewer labels overlap due to the reduced number of labels on the map. Conversely, when the zoom level decreases from a higher value to a lower value, an increase in fitness is anticipated due to the placement of new labels in default positions, thereby increasing the cost function. The straight lines in the fitness plots symbolize periods of zooming, indicating moments when the algorithm pauses its normal operation to accommodate changes in zoom level.

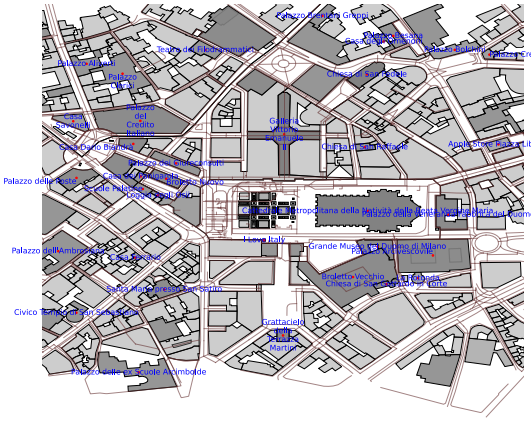
As observed in image 7b slightly before the time 200, the zoom level shifts from 7 to 1, denoting a significant zoom-out action that drastically changes the solution space and the number of labels to be placed. Consequently, due to such a large zoom-out action, the algorithm responds only after a few iterations, as illustrated by the straight-line ascent of the fitness curve. This phenomenon exemplifies the concept of reaction time, which has been previously discussed, highlighting the system's slightly delayed adjustment to rapid drastic changes in the solution space. When transitioning from a low zoom level to a higher zoom level (zoom-in), fewer labels need to be placed. Conversely, when transitioning from a high zoom level to a lower zoom level (zoom-out), a significant number of labels are added to the map. Consequently, a zoom-out action drastically changes the solution space. This observation can be seen in Figure 8a, in the range of time from 700 to 800. Every time there are zoom actions, there are drastic changes

Table 3: Number of labels for each of the 10 zoom levels considered for the Paris city map.

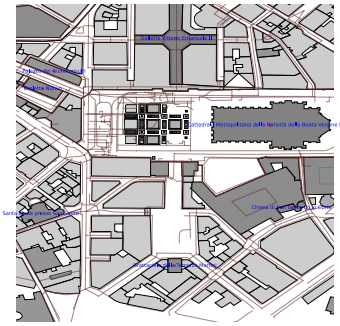
Zoom levels	0	1	2	3	4	5	6	7	8	9
Number of labels	66	62	61	58	57	55	52	45	38	28



(a) Full Map (Zoom level 0)



(b) Map (Zoom level 3)



(c) Map (Zoom level 8)

Fig. 9: Different perspectives of Milan city map.

in the fitness value. To provide more detail, Table 3 shows the number of labels for each zoom level. To conclude this discussion, Figure 9 presents the city map

of Milan at three different zoom levels: the full map without zoom (Figure 9a), middle zoom level (Figure 9b), and the maximum zoom level (Figure 9c).

6 Conclusions

This research study examines real-time optimization problems that involve changing constraints of the addressed problem over time. These challenges arise because, in order to be efficient, algorithms must adapt to evolving constraints and quickly provide optimal, or at least acceptable, solutions, and this is a typical behavior that can be easily find in many real-world problems. In this paper we present a method for adapting metaheuristic algorithms for real-time optimization applications. To illustrate this, an existing tabu search algorithm, originally designed for map labeling, was modified to address dynamic map labeling problems, where zoom levels change altering the solution space. Experimental results highlight the effectiveness of adapting tabu search for real-time optimization tasks. Future research could explore integrating machine learning to predict label importance based on user behavior and implementing a caching system to enhance the dynamic labeling process.

References

1. Metaheuristic and Evolutionary Computation: Algorithms and Applications. Springer Singapore (2021). https://doi.org/10.1007/978-981-15-7571-6
2. Bianchi, L., Dorigo, M., Gambardella, L.M., Gutjahr, W.J.: A survey on metaheuristics for stochastic combinatorial optimization. Natural Computing **8**(2), 239–287 (Sep 2008). https://doi.org/10.1007/s11047-008-9098-4
3. Cavallaro, C., Crespi, C., Cutello, V., Pavone, M., Zito, F.: Group dynamics in memory-enhanced ant colonies: The influence of colony division on a maze navigation problem. Algorithms **17**(2) (2024). https://doi.org/10.3390/a17020063
4. Cavallaro, C., Cutello, V., Pavone, M., Zito, F.: Machine learning and genetic algorithms: A case study on image reconstruction. Knowledge-Based Systems **284**, 111194 (2024). https://doi.org/https://doi.org/10.1016/j.knosys.2023.111194
5. Cavallaro, C., Cutello, V., Pavone, M., Zito, F.: A tabu search algorithm for the map labeling problem. In: Villani, M., Cagnoni, S., Serra, R. (eds.) Artificial Life and Evolutionary Computation. pp. 16–28. Springer Nature Switzerland, Cham (2024). https://doi.org/https://doi.org/10.1007/978-3-031-57430-6_2
6. Chachuat, B., Srinivasan, B., Bonvin, D.: Adaptation strategies for real-time optimization. Computers & Chemical Engineering **33**(10), 1557–1567 (2009). https://doi.org/https://doi.org/10.1016/j.compchemeng.2009.04.014, selected Papers from the 18th European Symposium on Computer Aided Process Engineering (ESCAPE-18)
7. Chopard, B., Tomassini, M.: Performance and Limitations of Metaheuristics, p. 191–203. Springer International Publishing (2018). https://doi.org/10.1007/978-3-319-93073-2_11
8. Corominas, A.: On deciding when to stop metaheuristics: Properties, rules and termination conditions. Operations Research Perspectives **10**, 100283 (2023). https://doi.org/https://doi.org/10.1016/j.orp.2023.100283

9. Crespi, C., Scollo, R.A., Fargetta, G., Pavone, M.: A sensitivity analysis of parameters in an agent-based model for crowd simulations. Applied Soft Computing **146**, 110684 (2023). https://doi.org/https://doi.org/10.1016/j.asoc.2023.110684
10. Cutello, V., Oliva, M., Pavone, M., Scollo, R.A.: An immune metaheuristics for large instances of the weighted feedback vertex set problem. In: 2019 IEEE Symposium Series on Computational Intelligence (SSCI). pp. 1928–1936 (2019). https://doi.org/10.1109/SSCI44817.2019.9002988
11. Cutello, V., Fargetta, G., Pavone, M., Scollo, R.A.: Optimization algorithms for detection of social interactions. Algorithms **13**(6) (2020). https://doi.org/10.3390/a13060139
12. Gemsa, A., Niedermann, B., Nöllenburg, M.: Trajectory-based dynamic map labeling. In: Cai, L., Cheng, S.W., Lam, T.W. (eds.) Algorithms and Computation. pp. 413–423. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
13. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning. Springer New York (2009). https://doi.org/10.1007/978-0-387-84858-7
14. Krishnamoorthy, D., Skogestad, S.: Real-time optimization as a feedback control problem - a review. Computers & Chemical Engineering **161**, 107723 (2022). https://doi.org/https://doi.org/10.1016/j.compchemeng.2022.107723
15. Okulewicz, M., Mańdziuk, J.: A metaheuristic approach to solve dynamic vehicle routing problem in continuous search space. Swarm and Evolutionary Computation **48**, 44–61 (2019). https://doi.org/https://doi.org/10.1016/j.swevo.2019.03.008
16. Osaba, E., Villar-Rodriguez, E., Del Ser, J., Nebro, A.J., Molina, D., LaTorre, A., Suganthan, P.N., Coello Coello, C.A., Herrera, F.: A tutorial on the design, experimentation and application of metaheuristic algorithms to real-world optimization problems. Swarm and Evolutionary Computation **64**, 100888 (2021). https://doi.org/https://doi.org/10.1016/j.swevo.2021.100888
17. Rajwar, K., Deep, K., Das, S.: An exhaustive review of the metaheuristic algorithms for search and optimization: taxonomy, applications, and open challenges. Artificial Intelligence Review **56**(11), 13187–13257 (Nov 2023)
18. Richter, H., Dietel, F.: Solving Dynamic Constrained Optimization Problems with Asynchronous Change Pattern, pp. 334–343. Springer Berlin Heidelberg (2011). https://doi.org/10.1007/978-3-642-20525-5_34
19. Slowik, A., Kwasnicka, H.: Evolutionary algorithms and their applications to engineering problems. Neural Computing and Applications **32**(16), 12363–12379 (Mar 2020). https://doi.org/10.1007/s00521-020-04832-8
20. TEKIN, E., SABUNCUOGLU, I.: Simulation optimization: A comprehensive review on theory and applications. IIE Transactions **36**(11), 1067–1081 (Nov 2004). https://doi.org/10.1080/07408170490500654
21. Tzanetos, A., Dounias, G.: Nature inspired optimization algorithms or simply variations of metaheuristics? Artificial Intelligence Review **54**(3), 1841–1862 (Aug 2020). https://doi.org/10.1007/s10462-020-09893-8
22. Zito, F., Cutello, V., Pavone, M.: A machine learning approach to simulate gene expression and infer gene regulatory networks. Entropy **25**(8), 1214 (Aug 2023). https://doi.org/10.3390/e25081214

An SMC Sampler for Decision Trees with Enhanced Initial Proposal for Stochastic Metaheuristic Optimization

Efthyvoulos Drousiotis[1], Alessandro Varsi[1], Paul G, Spirakis[2], and Simon Maskell[1]

[1] Department of Electrical Engineering and Electronics, University of Liverpool, Liverpool L69 3GJ, UK; {E.Drousiotis, A.Varsi, S.Maskell}@liverpool.ac.uk
[2] Department of Computer Science, University of Liverpool, Liverpool L69 3BX, UK
P.Spirakis@liverpool.ac.uk

Abstract. Bayesian Decision Trees (DTs) have emerged as a significant tool for Machine Learning (ML) tasks, offering better performance compared to traditional approaches. However, the convergence of Bayesian Trees can be affected by slow initialization, particularly when employing the random method. This paper addresses this limitation by proposing an enhanced initialization strategy for the Sequential Monte Carlo (SMC) sampler applied to DTs. We demonstrate both experimentally and theoretically that our approach accelerates convergence to the posterior distribution. Our contribution lies in introducing an SMC sampler for DTs with a refined initialization strategy. By leveraging a more sophisticated approach, we achieve faster convergence to the posterior distribution, as substantiated by experimental results and theoretical analysis. Unlike conventional random initialization methods, our proposed approach unlocks the full potential of the underlying model.

1 Introduction

1.1 Motivation

Bayesian approaches to the Decision Tree (DT) model have recently emerged as powerful tools to tackle Machine Learning (ML) classification and regression tasks. This family of methodologies often provides better classification/regression scores than traditional approaches, including decision forest [8]. The Bayesian framework inherits its principles from statistical sampling and probability theory, having the appealing property of incorporating prior knowledge from the very beginning of the training process. Consequently, the integration of Bayesian algorithms in ML has gained substantial attention within the scientific community with wide-ranging applicability including (but not limited to) psychology [5], education [7], and chemistry [15, 26]. Despite the state-of-the-art performance, Bayesian Trees sometimes need a substantial amount of time to converge and produce acceptable solutions. One of the reasons for slow convergence is the poor initialization of the population using the random method.

1.2 Related Work

Researchers during the past years have focused on enhancing the efficiency of the MCMC methods. The improvements can be categorized into various categories [24]. One category focuses on the geometric aspects of the target density function. Hamiltonian Monte Carlo (HMC) [13], for instance, utilizes an auxiliary variable called momentum, updating it based on the density function's gradient. While HMC produces less correlated samples than Metropolis-Hastings, it necessitates the availability and computational feasibility of the density function's gradient.

Another category the researchers have explored is the division of complex problems [22] into smaller and more manageable components. The first technique employed is the Divide-and-Conquer, which partitions big datasets into batches, executing independent MCMC algorithms on each subset. Partitioning the data space, which is already implemented in the context of Bayesian DTs [11], or partitioning the parameter space [1] into simpler pieces that can process independently are proven not to be very effective. The second technique is the Sub-Sampling, which aims to reduce the number of individual datapoint likelihood evaluations operated at each iteration towards accelerating MCMC algorithms.

Chipman and Denison [2, 3] first proposed the Bayesian Decision Trees with the main difference lying between their priors. Both proposed a model that generates N samples to estimate the true DT by using the MCMC algorithm with a relatively simple but general purpose and very restrictive prior. Their prior aim was to maintain the tree's shape and structure using some hyperparameters. Bayesian DTs need many iterations to converge on the "true" DT, especially when the dataset is large with complex relationships between the features (the same holds for the conventional MCMC). In general, MCMC generates samples from a given distribution by proposing new samples and deciding to accept or reject them based on the evaluation of the posterior distribution. Fundamentally, the new sample of the chain is based on the previous one, which makes the MCMC chain inherently sequential, very difficult, and problem-specific on how to process a single chain MCMC using multiple processing elements. A method in [10] proposed a single chain parallelization on MCMC DTs; however, the speedup is not always guaranteed. Most recently, in [12], a different Bayesian numerical approach is proposed, Sequential Monte Carlo Decision Trees (SMC DTs), which is inherently parallel and showed great speedup results against MCMC DTs without compromising its accuracy.

A different method for enhancing the efficiency of the Bayesian methods focuses on enhancing the proposal function, which is what we study in this paper in the context of the SMC DTs when it comes to the initialization of the population at step $t = 0$. Researchers studied the improvements in the proposals for step $t + 1$, which is right after the initialization of the samples. This can be achieved through techniques such as simulated tempering [20], adaptive MCMC [4], or multi-proposal MCMC [18]. In the domain of probabilistic modeling and generally in Bayesian statistics, the initialization of the populations holds significant importance. This also applies to the Sequential Monte Carlo algorithms (SMC),

applied to Decision Trees (DTs) [6,12]. The most common practice often relies on randomness to generate initial proposals, a method that, while straightforward, may not exploit the full potential of the underlying model.

1.3 Contribution and Paper Outline

In this paper, we present an implementation of the SMC sampler for the DTs, which improves the initialization of the proposals while exploiting the full potential of the algorithm. More precisely, we show experimentally and prove theoretically that by initializing the proposals more sophisticatedly, our model converges faster to the posterior distribution. We not only advance the understanding of SMC DTs but also provide a valuable contribution to the broader field of Bayesian statistics by addressing the critical aspect of population initialization as the suggested method can be applied in other domains SMC is used.

The remainder of this paper is organized as follows: Section 2 describes the DT in general. Section 3 describes the SMC algorithm in the DTs. Section 4 describes the MIC and how it is used in our context following a proof showing that our method approximates the posterior better than the random method. Then, in Section 5, we present the numerical results with a discussion. Section 6 summarises the paper.

2 Decision Trees

In this section, we briefly describe the anatomy of a DT, in all its components, and how this model is used for regression tasks. The reader is referred to [14] for further details.

Given a dataset comprising a target vector of l records, $\mathbf{Y} \in \mathbb{Z}^l$, and the corresponding matrix of data points, $\mathbf{x} \in \mathbb{R}^{l \times r}$, the DT model is trained to classify a datum, $\mathbf{Y}_j$, given the corresponding j-th row in $\mathbf{x}$. In other words, every data point, $\mathbf{x}_j$ $\forall j = 0, 1, 2, \ldots, l-1$, contains r thresholds of the generic vector of r features, ϕ. Table 1 gives a generic illustration of a dataset.

	ϕ_0	ϕ_1	ϕ_2	$\ldots$	ϕ_{r-1}	target
$\mathbf{x}_0$	-	-	-	-	-	$\mathbf{Y}_0$
$\mathbf{x}_1$	-	-	-	-	-	$\mathbf{Y}_1$
$\mathbf{x}_2$	-	-	-	-	-	$\mathbf{Y}_2$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\mathbf{x}_{l-1}$	-	-	-	-	-	$\mathbf{Y}_{l-1}$

Table 1: Generic representation of a dataset. The $-$ are used for illustration purposes but represent generic thresholds in $\mathbf{x}_j$ relating to a specific feature ϕ_n.

For a given tree, $\mathbf{T}$, we define $d(\mathbf{T})$ to be the depth of the tree, $D(\mathbf{T})$ to be the set of the m non-leaf nodes, and $L(\mathbf{T})$ to be the set of leaf nodes. Each tree,

$\mathbf{T}$, will then have a total of m non-leaf nodes, each node containing a feature, $\mathbf{k}_j$ $\forall j = 0, 1, \ldots, m-1$, and a threshold, $\mathbf{c}_j$ $\forall j = 0, 1, \ldots, m-1$, both selected from the dataset. A DT operates by descending a tree. The process of outputting a regression outcome for a given datum starts at the root node. At each non-leaf node, a decision as to which child node to progress to is made based on the datum and the parameters of the node. This process continues until a leaf node is reached. At the leaf node, a node-specific and datum-independent regression output is generated. Figure 1 exemplifies a DT with five non-leaf nodes and six leaf nodes.

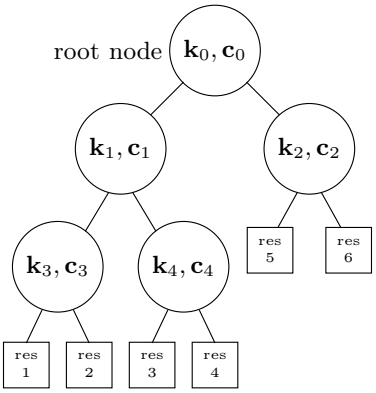


Fig. 1: An example of a DT with $d(\mathbf{T}) = 3$, $m = 5$, and six-leaf nodes each with a possible quantitative outcome (or result) expressed as a number in $\mathbb{R}$.

3 Sequential Monte Carlo

MC methods can be used to make estimations of the state of a statistical model, which is described by a posterior distribution (or simply posterior; the terms are used interchangeably), i.e., the Probability Density Function (PDF) of the state of the model given some data. The idea is to generate a population of random samples representing the posterior, with a view to using the samples to make estimates. However, posterior distributions are often challenging to sample from directly. SMC is one of the most popular Bayesian classes of methodologies for sampling from a posterior distribution.

When it comes to Bayesian approaches to DTs, we are interested in making estimations of the true tree, $\mathbf{T}$, and its set of features and thresholds, θ, given a dataset described by the target vector, $\mathbf{Y}$, and its related data-points, $\mathbf{x}$, for a regression problem. Therefore, $\pi(\mathbf{T}, \theta|\mathbf{Y}, \mathbf{x}) \propto \pi(\mathbf{T}, \theta, \mathbf{Y}|\mathbf{x})$ is the posterior from which we want to generate random samples. In this section, we describe how SMC can be implemented in the specific case of sampling a population of DTs. The reader is referred to [2, 3, 21].

The general idea behind SMC samplers is to perform sampling and resampling in sequence for K iterations, $\forall k = 0, 1, \ldots, K-1$, each of which will generate and update N independent, weighted samples, $\mathbf{t}_k^i$, $\forall i = 0, 1, 2, \ldots, N-1$. The samples are first initialized from the initial proposal and weighted as follows:

$$\mathbf{t}_0^i \sim q_0(\cdot), \quad \forall i = 0, 1, 2, \ldots, N-1, \tag{1a}$$

$$\mathbf{w}_0^i = \frac{\pi\left(\mathbf{t}_0^i, \mathbf{Y}|\mathbf{x}\right)}{q_0\left(\mathbf{t}_0^i\right)}, \quad \forall i = 0, 1, 2, \ldots, N-1. \tag{1b}$$

Then, during each SMC iteration, $\forall k = 1, 2, \ldots, K-1$, the samples are updated from the proposal and weighted as follows:

$$\mathbf{t}_k^i \sim q(\cdot|\mathbf{t}_{k-1}^i), \quad \forall i = 0, 1, 2, \ldots, N-1, \tag{2a}$$

$$\mathbf{w}_k^i = \mathbf{w}_{k-1}^i \frac{\pi\left(\mathbf{t}_k^i, \mathbf{Y}|\mathbf{x}\right)}{\pi\left(\mathbf{t}_{k-1}^i, \mathbf{Y}|\mathbf{x}\right)} \frac{q\left(\mathbf{t}_{k-1}^i|\mathbf{t}_k^i\right)}{q\left(\mathbf{t}_k^i|\mathbf{t}_{k-1}^i\right)}, \quad \forall i = 0, 1, 2, \ldots, N-1. \tag{2b}$$

The weights, $\mathbf{w}_k^i$ $\forall i = 0, 1, \ldots, N-1$, are then normalized as follows:

$$\tilde{\mathbf{w}}_k^i = \frac{\mathbf{w}_k^i}{\sum_{j=0}^{N-1} \mathbf{w}_k^j}, \quad \forall i = 0, 1, 2, \ldots, N-1, \tag{3}$$

such that $\sum_i \tilde{\mathbf{w}}_k^i = 1$, i.e., 100% of the total probability space.

Since the samples are generated from $q(\cdot|\cdot)$, and not directly from the posterior distribution of interest, this sampling strategy may suffer from a numerical error called degeneracy, which could make most of the weights equal to 0, leading towards poor estimations of the state. This problem is commonly tackled by using resampling when the effective sample size (ESS) of the weights,

$$N_{eff} = \frac{1}{\sum_{i=1}^{N} \left(\tilde{\mathbf{w}}_k^i\right)^2}, \tag{4}$$

falls down below a user-defined threshold, usually set $\frac{N}{2}$. Resampling overwrites the samples with low weights with copies of the samples with high weights. Several variants of resampling can be found in the literature [17]. In this work, we use systematic resampling, which performs three steps.

Systematic resampling first computes $\mathbf{cdf} \in \mathbb{R}^N$, the cumulative sum of $N\tilde{\mathbf{w}}$, as follows:

$$\mathbf{cdf}^i = N \sum_{j=0}^{i} \tilde{\mathbf{w}}_k^j, \quad \forall i = 0, 1, 2, \ldots, N-1. \tag{5}$$

Then, each $\mathbf{t}_k^i$ gets copied as many times as

$$\mathbf{ncopies}^i = \lceil \mathbf{cdf}^{i+1} - u \rceil - \lceil \mathbf{cdf}^i - u \rceil, \quad \forall i = 0, 1, \ldots, N-1, \tag{6}$$

where $u \sim \texttt{Uniform}[0, 1]$, and $\lceil \cdot \rceil$ is the ceiling operator. From (5) and (6) it is relatively straightforward to infer that

$$0 \leq \mathbf{ncopies}^i \leq N, \quad \forall i = 0, 1, 2, \ldots, N-1, \tag{7a}$$

$$\sum_{i=0}^{N-1} \mathbf{ncopies}^i = N, \tag{7b}$$

meaning that, after resampling, we will always have N samples, and those for which $\mathbf{ncopies}^i = 0$ will be deleted. In the end, resampling resets each weight, $\mathbf{w}^i$, to $1/N$.

After resampling, a new iteration starts from (2) and, after $K-1$ iterations, each of the final samples, $\mathbf{t}^i_{K-1}$, is used to make a prediction with respect to a given test dataset. After that, the final estimate is generated by either computing the mean or the majority voting of the predictions.

3.1 Posterior and Proposal for Bayesian Decision Trees

In this section, we briefly describe how to compute the posterior, the proposal, and the initial proposal in the specific case of DTs.

Posterior Distribution The posterior is proportional (up to a normalization constant) to the product of the likelihood and the prior. In other words:

$$\pi(\mathbf{T}, \theta|\mathbf{Y}, \mathbf{x}) \propto \pi(\mathbf{T}, \theta, \mathbf{Y}|\mathbf{x}) = p(\mathbf{Y}|\mathbf{T}, \theta, \mathbf{x})p(\theta, \mathbf{T}) = p(\mathbf{Y}|\mathbf{T}, \theta, \mathbf{x})p(\theta|\mathbf{T})p(\mathbf{T}), \tag{8}$$

where $p(\mathbf{Y}|\mathbf{T}, \theta, \mathbf{x})$ is the likelihood and $p(\theta, \mathbf{T})$ is the prior. More precisely, for each individual term in (8), we use the following expressions:

$$p(\mathbf{Y}|\mathbf{T}, \theta, \mathbf{x}) = \prod_{j=0}^{l-1} p(\mathbf{Y}_j|\mathbf{x}_j, \mathbf{T}, \theta), \tag{9}$$

$$p(\theta|\mathbf{T}) = \prod_{j=0}^{m-1} p(\mathbf{k}_j|\mathbf{T})p(\mathbf{c}_j|\mathbf{k}_j, \mathbf{T}), \tag{10}$$

$$p(\mathbf{T}) = \frac{\lambda^m}{(e^\lambda - 1)m!}. \tag{11}$$

Using the Poisson distribution for the $p(\mathbf{T})$ term is a common practice in the literature [3, 16, 19]. As we can see, (11) only requires tuning of the λ hyperparameter. Another valid expression can be found in [23, 27]. This alternative, however, requires tuning of two hyperparameters. Therefore, we employ (11) for simplicity.

Proposal Distribution The proposal distribution, $q(\cdot|\cdot)$, consists of a set of four possible moves, which may either change the dimensionality of the tree or update its nodes. The possible moves are:

- *Grow*. This move increases the dimensionality of the tree by uniformly selecting one of the leaf nodes and appending two new child nodes to the selected leaf.
- *Prune*. This move decreases the dimensionality of the tree by uniformly selecting one of the non-leaf nodes and removing its child nodes.

- *Change*. This move picks uniformly a non-leaf node, j, and changes its feature, $\mathbf{k}^j$, and threshold, $\mathbf{c}^j$. In other words, this move neither increases nor decreases the dimensionality of the tree.
- *Swap*. This move picks two non-leaf nodes, i and j, uniformly, and swaps their features, $\mathbf{k}^i$ and $\mathbf{k}^j$, and their thresholds, $\mathbf{c}^i$ and $\mathbf{c}^j$. Therefore, this move also maintains the dimensionality of the tree unchanged.

The probabilities of selecting any of these four moves, $p(Grow)$, $p(Prune)$, $p(Change)$, and $p(Swap)$, are user-defined, but they must sum up to 1, and a typical choice is to make these probabilities equal to 25%. The value of $q(\cdot|\cdot)$ to be used in (2b) is described in detail in [9], and omitted here for brevity.

Initial Proposal Distribution In the case of the initial proposal distribution, the typical approach, which is described in several related works such as [9, 12] consists of sampling from the following initial proposal:

$$q_0(\cdot) = p_{\mathcal{U}}(\phi)p(\mathbf{T}), \tag{12}$$

where $p(\mathbf{T})$ is the Poisson distribution given by Equation (11), and ϕ is the array of r possible features to select from the given dataset, and $p_{\mathcal{U}}(\phi)$ is a uniform distribution, such that:

$$p_{\mathcal{U}}(\phi_j) = \frac{1}{r}, \quad \forall j = 0, 1, 2, \ldots, r-1. \tag{13}$$

In other words, the initial proposal in (12) grows each tree according to a Poisson process, and, every time a new non-leaf node is created, it selects uniformly one of the features in ϕ to be assigned to the said non-leaf node's feature. This uniform process of selecting the features is rather naive, and, in Section 5 will be named random proposal. In the next section, we propose an alternative approach, which we prove can initialize the trees much more efficiently and promote a faster convergence.

4 Initial Proposal with Maximal Information Coefficient

In this section, we present an initial proposal distribution, $q_0(\cdot)$, that makes use of the MIC to decide how to select new features. To provide context, Section 4.1 briefly describes MIC and how to use it to assign a probability to each feature, while Section 4.2 presents and provides proof of the validity of our approach.

4.1 Maximal Information Coefficient

MIC is a powerful approach used to measure the correlation between two variables. MIC deals with the correlation analysis of linear, nonlinear, and potential nonfunctional relationships in large datasets. The fundamental idea of MIC is that if a specific relationship exists between two features, a grid can be drawn on

the scatter-plot to partition the features. Then, it will be able to encapsulate the mutual information of the two features according to the approximate probability density distribution in the grid. The MIC approach is summarized below.

MIC is calculated based on mutual information and the grid partition method. Given two independent features with l samples, $\mathbf{x} = \{\mathbf{x}_j | j = 1, ..., l\}$ and the related target $\mathbf{Y} = \{\mathbf{Y}_j | j = 1, .., l\}$, a finite set $D = (\mathbf{x}_j, \mathbf{Y}_j | j = 1, .., l)$ of ordered pairs can be obtained. Given a grid G, we can partition the $\mathbf{x}_j$ values of D into l bins and the $\mathbf{Y}_j$ values of D into l bins. MIC is obtained according to the following equations:

$$MI(D, \mathbf{x}, \mathbf{Y}) = \max MI(D|G), \tag{14}$$

where $MI(D, \mathbf{x}, \mathbf{Y})$ denotes the maximum mutual information of D over grids G. $D|G$ represents the distribution induced by the points in D on the cells of grid G. The following equation defines the characteristic matrix of D:

$$M(D)_{\mathbf{xY}} = \frac{MI(D, \mathbf{x}, \mathbf{Y})}{\log \min\{\mathbf{x}, \mathbf{Y}\}}. \tag{15}$$

The MIC of D with grid size less than $B(\nu)$ is defined as:

$$MIC(D)_{\mathbf{xY}} = \max_{\mathbf{xy} < B(\nu)} \{M(D)_{\mathbf{xY}}\}, \tag{16}$$

where $B(\nu)$ is the upper limit of the mesh division $\mathbf{xY}$. In general $B(\nu) = \nu^{0.6}$. In simple words, for every feature, $\phi_j \ \forall j = 0, 1, \ldots, r-1$, (16) gives us a score in the range $[0, 1]$, which encapsulates the degree of correlation between the said feature and the related target. This means that a higher MIC value indicates a stronger correlation between the variables. In order to obtain a PDF for the features, ϕ, we normalize these scores such that these values sum up to 1.0 (i.e. 100% of the total probability space). We name $h(\phi)$ the PDFs of the features. In order to sample from $h(\phi)$, we use a standard Markov process, where we first compute $\mathbf{ch} \in \mathbb{R}^{r+1}$, the prefix sum of $h(\phi)$. That means the first element in $\mathbf{ch}$ is equal to $\mathbf{ch}^0 = 0$, and any n-th element in $\mathbf{ch}$, for $n > 0$, is equal to $\mathbf{ch}^i = \sum_{i=1}^{n}$. Then, we draw a uniform random number, a, in the range of $[0, 1]$. We then find the two consecutive elements in $\mathbf{ch}$ such that $\mathbf{ch}^j \leq a \leq \mathbf{ch}^{j+1}$, and we then pick the j-th feature, ϕ_j.

4.2 Initialisation of the Population

We want to design an effective initial proposal, $q_0(\cdot)$, which initializes the trees from a favorable initial position (i.e., topology). Ideally, we would want to sample from the optimal initial proposal, $q_0^{opt}(\cdot) = \pi(\mathbf{T}, \theta | \mathbf{x}, \mathbf{Y})$. In other words, we would like to sample directly from the posterior distribution. However, this is often impossible in practice. Nevertheless, we can still define an initial proposal that approximates some portions of the posterior well. This will help initialize the DTs more sophisticatedly and promote faster convergence after the initialization.

Our approach is to define an initial proposal that efficiently selects the features to assign to each node of the DTs. In other words, by Bayes' rule, we can consider the posterior expressed as follows:

$$\pi(\mathbf{T}, \theta|\mathbf{x}, \mathbf{Y}) = p(\phi)p(\cdots|\phi),$$

where

$$\sum_{j=0}^{r-1} p(\phi_j) = 1.0,$$

and we design an initial proposal that samples approximately well from $p(\phi)$.

As we have explained in Section 3.1, the typical approach found in the literature consists of using (12), which selects a feature uniformly for every non-leaf node of a tree. For obvious reasons, the uniform $p_{\mathcal{U}}(\phi)$ term is far from approximating well $p(\phi)$.

In the posterior, each feature, ϕ_j, has an increased selection probability depending on its correlation to the target vector, $\mathbf{Y}$. This makes $h(\phi)$ a suitable PDF to sample from in order to evaluate which feature to select. Indeed, $h(\phi)$ tries to sample more frequently the features with the highest MIC correlation with the target. However, in some rare cases with probability $\omega \in [0, 1]$ and $\omega \ll 1.0$, $h(\phi)$ may occasionally significantly over-sample less important features and under-sample (just as significantly) more important features. This happens, for example, when a feature ϕ_j with probability $p(\phi_j) \ll \frac{1}{r}$ (i.e., a non-important feature) is sampled more frequently than $\frac{1}{r} \times 100$ percent of the time, or when a feature ϕ_j with probability $p(\phi_i) \gg \frac{1}{r}$ (i.e. an important feature) is sampled less frequently than $\frac{1}{r} \times 100$ percent of the time. In this rare scenario, sampling from the $p_{\mathcal{U}}(\phi)$ would be a better idea. Therefore, instead of sampling the features directly from $h(\phi)$, we will sample from:

$$p_{MIC}(\phi) = (1 - \omega)h(\phi) + \omega p_{\mathcal{U}}(\phi), \tag{17}$$

such that the initial proposal becomes:

$$q_0(\cdot) = p(\mathbf{T})p_{MIC}(\phi) = p(\mathbf{T})((1 - \omega)h(\phi) + \omega p_{\mathcal{U}}(\phi)). \tag{18}$$

Under the condition of pre-tuning ω, (17) bounds the worst case of our approach to $p_{\mathcal{U}}(\phi)$. Therefore, (17) approximates $p(\phi)$ better than $p_{\mathcal{U}}(\phi)$, as proven by the following theorem.

Theorem 1. *Let ϕ be a list of r features relating to a given dataset, $\{\mathbf{x}, \mathbf{Y}\}$, with PDF, $p(\phi)$, which is such that a given feature ϕ_j has higher probability depending on its correlation with the target vector, $\mathbf{Y}$. Therefore, a proposal distribution based on the MIC coefficient as in (17) approximate $p(\phi)$ better than a uniform proposal $p_{\mathcal{U}}(\phi)$.*

Proof. To prove Theorem 1, we use the Kullback–Leibler (KL) divergence for discrete functions (as is the case for PDFs relating to Bayesian DTs), which

measures the divergence of a given PDF, $\hat{p}(\phi)$, from another PDF, $p(\phi)$, as follows:

$$d(\hat{p} \parallel p) = \sum_{j=0}^{r-1} \hat{p}(\phi_j) \ln\left(\frac{\hat{p}(\phi_j)}{p(\phi_j)}\right). \tag{19}$$

In the ideal scenario, we have $\hat{p}(\phi) = p(\phi)$, which makes $d(q_0 \parallel p) = 0$. Therefore, Theorem 1 is proven if $d(p_{MIC} \parallel p)$ is closer to 0 than $d(p_{\mathcal{U}} \parallel p)$, or, in other words, if

$$0 \leq |d(p_{MIC} \parallel p)| \leq |d(p_{\mathcal{U}} \parallel p)|. \tag{20}$$

Before computing $d(p_{MIC} \parallel p)$ and $d(p_{\mathcal{U}} \parallel p)$, we note that for this proof we will use the following notation. Since the features are not continuous values in $\mathbb{R}$, it is convenient to relabel the indexes j in ϕ in such a way that:

$$p(\phi_0) \leq p(\phi_1) \leq \cdots \leq p(\phi_{t-1}) < \frac{1}{r} \leq p(\phi_t) \leq p(\phi_{t+1}) \leq ... \leq p(\phi_{r-1})) \tag{21}$$

In other words, we use the index $0 \leq j < t$ to name those features whose probability is below the average $\frac{1}{r}$, and index $t \leq j < r$ to name those features whose probability is equal or above the average $\frac{1}{r}$. We do this to simplify the explanation of this proof.

Intuitively, the KL divergence of $p_{\mathcal{U}}$ from $p(\phi)$ is:

$$d(p_{\mathcal{U}} \parallel p) = \frac{1}{r} \sum_{j=0}^{r-1} \ln\left(\frac{\frac{1}{r}}{p(\phi_j)}\right). \tag{22}$$

Sampling the features from (17) means that, in the worst-case, $d(p_{MIC} \parallel p) = d(p_{\mathcal{U}} \parallel p)$. In the other cases, we sample from $h(\phi)$, such that we propose features with higher correlation with the target vector, $\mathbf{Y}$, which means that $p_{MIC}(\phi) \approx p(\phi)$ as the MIC captures both linear and nonlinear correlations between variable pairs of $\mathbf{x}$ and $\mathbf{Y}$.

Hence, there are only two possible intermediate cases that may occur. The first one is when the MIC proposal, $h(\phi)$, identifies a feature with high correlation, ϕ_n for $n \geq t$, and will assign $p_{MIC}(\phi_n) = \frac{1}{r} + \epsilon$ with $\epsilon > 0$, such that $\frac{1}{r} < p_{MIC}(\phi_n) < p(\phi_n)$. The MIC proposal will compensate for this by selecting at least one feature, ϕ_i for $i < t$, (or more features, in the most general case) with a low correlation and reducing its selection probability (or the combined selection probability, in the most general case) by ϵ, starting with the feature with the lowest correlation. For brevity, we assume only one feature, ϕ_i for $i < t$, is selected, but the proof works in the most general case as well. Therefore, $h(\phi)$ will assign $p_{MIC}(\phi_i) = \frac{1}{r} - \epsilon$, such that $p(\phi_i) < p_{MIC}(\phi_i) < \frac{1}{r}$, and keep every other $p_{MIC}(\phi_j) = \frac{1}{r}$, for $j \neq i, k$. This is because, we must always have

$\sum_j p_{MIC}(\phi_j) = 1$. In such case, we need to check if

$$|d(p_{MIC} \parallel p)| = \left| p_{MIC}(\phi_i) \ln\left(\frac{p_{MIC}(\phi_i)}{p(\phi_i)}\right) + p_{MIC}(\phi_n) \ln\left(\frac{p_{MIC}(\phi_n)}{p(\phi_n)}\right) + \right.$$
$$\left. + \frac{1}{r} \sum_{\forall j \neq i,k} \ln\left(\frac{\frac{1}{r}}{p(\phi_j)}\right) \right| \quad (23)$$

is lower or equal than

$$|d(p_{\mathcal{U}} \parallel p)| = \left| \frac{1}{r} \ln\left(\frac{\frac{1}{r}}{p(\phi_i)}\right) + \frac{1}{r} \ln\left(\frac{\frac{1}{r}}{p(\phi_n)}\right) + \frac{1}{r} \sum_{\forall j \neq i,k} \ln\left(\frac{\frac{1}{r}}{p(\phi_j)}\right) \right|. \quad (24)$$

The sums on the right-hand side of (23) and (24) are equal for obvious reasons. This means we only have to compare the remaining terms.

First, by comparing the two terms for ϕ_i, it is straightforward to show that they are both positive and that:

$$p_{MIC}(\phi_i) \ln\left(\frac{p_{MIC}(\phi_i)}{p(\phi_i)}\right) < \frac{1}{r} \ln\left(\frac{\frac{1}{r}}{p(\phi_i)}\right), \quad (25)$$

because $1 < \frac{p_{MIC}(\phi_i)}{p(\phi_i)} < \frac{\frac{1}{r}}{p(\phi_i)}$, since $p(\phi_i) < p_{MIC}(\phi_i) < \frac{1}{r}$.

Second, by comparing the two terms for ϕ_n, it is straightforward to show that they are both negative and that:

$$\frac{1}{r} \ln\left(\frac{\frac{1}{r}}{p(\phi_n)}\right) < p_{MIC}(\phi_n) \ln\left(\frac{p_{MIC}(\phi_n)}{p(\phi_n)}\right) < 0, \quad (26)$$

because $0 < \frac{\frac{1}{r}}{p(\phi_n)} < \frac{p_{MIC}(\phi_n)}{p(\phi_n)} < 1$, and $\frac{1}{r} < p_{MIC}(\phi_n) < p(\phi_n)$. Therefore, (20) is valid in this intermediate case.

In the other possible intermediate case, the MIC proposal assigns $p_{MIC}(\phi_i) = \frac{1}{r} - \epsilon$ and $p_{MIC}(\phi_n) = \frac{1}{r} + \epsilon$, such that $p_{MIC}(\phi_i) < p(\phi_i) < \frac{1}{r}$ and $\frac{1}{r} < p(\phi_n) < p_{MIC}(\phi_n)$. In this case, it is still possible to prove (20) by repeating the analogous steps to (23), (24), (25), and (26), which are omitted here for brevity. □

In the next section, we compare an SMC sampler for DTs with (18) as the initial proposal vs an identical SMC sampler for DTs with (12) as the initial proposal and show that our proposed approach initialized the DTs trees better and converge faster. In the next section, we will refer to this approach as MIC proposal.

5 Numerical Results

As described in 4.1, the main objective of our proposed algorithm is to make informed initialization of the DTs using insights inherited from the dataset,

demonstrating the overall performance and convergence efficiency. In this section, we will first present the results and the accuracy improvement of the proposed method, followed by a discussion.

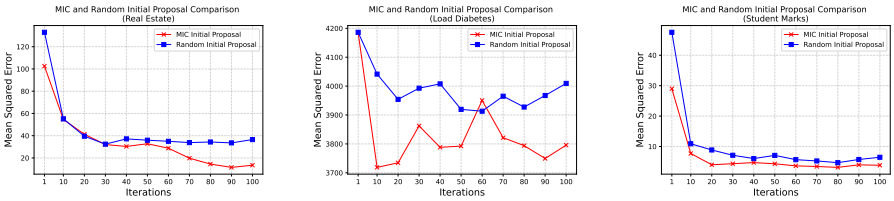
In order to show the overall algorithm improvement, we experiment on three publicly available datasets on the UCI machine learning repository. More precisely, the first dataset is called Real Estates, which has 157 records and 8 features and provides data to train a model to predict house prices. The second dataset is called Load Diabetes, which has 442 records and 10 features, where the model is trained to predict the level of blood glucose. The third dataset is called Students' Marks, which has 100 records and 2 features, and the predictive outcome is the students' performance.

We have run the contesting algorithms for 100 iterations for all datasets and reported the Mean Squared Error (MSE) for every 10 iterations. Experiments have shown that 100 iterations are enough for the algorithm to converge and produce acceptable predictions. The goal is to monitor and report the lowest MSE and examine the influence of the initial proposals when the model is trained on different numbers of iterations. We note that, the hyperparameter ω in (18) has been empirically pre-tuned to 0.01.

For the first dataset, Real Estates, we observe a significant improvement in initializing the proposals when MIC proposal is utilized. The MSE using MIC is 102.5 compared to 132.9 using the random method. Figure 2a demonstrates that as we run the algorithm for more iterations, the MSE achieved with the MIC proposal is progressively lower than the MSE achieved with the random proposal. The MSE score using the MIC proposal is 11.5, and the MSE using the random proposal is 33.5. For the second dataset, Load Diabetes, the improvement in the initialization is less prudent when the MIC proposal is utilized. The MSE using the MIC proposal is 4184.6 compared to 4186.3 using the random proposal. Figure 2b shows that as we run the algorithm for more iterations, the MSE with the MIC proposal is progressively lower than the one with the random proposal. Indeed, we observe that the best MSE score with the MIC proposal is 3749.1, while the MSE score using the random proposal is 3927.6. The third dataset, Students' Marks, produces results in line with the first dataset. We observe a significant improvement in initializing the DTs when the MIC proposal is utilized. The MSE achieved with the MIC proposal is 29.1 compared to 47.5 MSE achieved by using the random proposal. Indeed, Figure 2c shows that the MSE using the MIC proposal is consistently lower than the MSE using the random proposal. The minimum MSE scores are 3. and 6.5, for the MIC proposal and the random proposal, respectively.

The experiments have shown that implementing a more data-driven and informed initialization strategy for the samples helps to explore the solution space more efficiently and converge to a satisfactory solution faster. Specifically, for the Real Estates dataset and the Student Marks, the MIC proposal improved impressively the starting position of the samples, resulting in a lower MSE and faster convergence. Regarding the Load Diabetes dataset, the initialization of

the proposals using the MIC was not as significant; however, overall, the model benefits from a more sophisticated initialization strategy in the long run.



(a) Real Estates Dataset (b) Load Diabetes Dataset (c) Student Marks Dataset

Fig. 2: Comparison between MIC and Random Initial Proposals

6 Conclusion

In conclusion, this paper addresses a critical challenge in the convergence of Bayesian DTs by presenting an enhanced initialization strategy for SMC samplers for DTs using an initial proposal distribution that selects features based on their MIC coefficient. Indeed, despite the superior performance of Bayesian DTs in supervised learning tasks, previous work employing a purely random initialization suffers from slow convergence. Hence, our refined initialization strategy has been experimentally and theoretically demonstrated to accelerate convergence to the posterior distribution. More precisely, on three publicly available datasets for regression problems, our proposed approach shows better MSE by a 1.5× factor after the initialization of the population of DTs, and a reduction of the MSE by up to a factor of three after convergence.

Even though the results are encouraging, there are several new avenues for future work to explore. First, we could improve the quality of the predictions by proposing more sophisticated sampling mechanisms for Bayesian DTs, such as [25], to be adopted after the initialization. Improving the proposal mechanism typically affects the run-time of the algorithm. This could be addressed by exploring state-of-the-art parallel processing architectures, such as GPUs.

References

1. Guillaume Basse, Aaron Smith, and Natesh Pillai. Parallel markov chain monte carlo via spectral clustering. In Arthur Gretton and Christian C. Robert, editors, *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 1318–1327, Cadiz, Spain, 09–11 May 2016. PMLR.
2. Hugh A Chipman, Edward I George, and Robert E McCulloch. Bayesian cart model search. *Journal of the American Statistical Association*, 93(443):935–948, 1998.

3. David GT Denison, Bani K Mallick, and Adrian FM Smith. A bayesian cart algorithm. *Biometrika*, 85(2):363–377, 1998. doi:10.1093/biomet/85.2.363.
4. Randal Douc, Arnaud Guillin, J-M Marin, and Christian P Robert. Convergence of adaptive mixtures of importance sampling schemes. *The Annals of Statistics*, 35(1):420–448, 2007.
5. Efthyvoulos Drousiotis, Dan W. Joyce, Robert C. Dempsey, Alina Haines, Paul G. Spirakis, Lei Shi, and Simon Maskell. Probabilistic Decision Trees for Predicting 12-Month University Students Likely to Experience Suicidal Ideation. In Ilias Maglogiannis, Lazaros Iliadis, John MacIntyre, and Manuel Dominguez, editors, *Artificial Intelligence Applications and Innovations*, pages 475–487, Cham, 2023. Springer Nature Switzerland.
6. Efthyvoulos Drousiotis, Alexander M. Phillips, Paul G. Spirakis, and Simon Maskell. Bayesian Decision Trees Inspired from Evolutionary Algorithms. In Meinolf Sellmann and Kevin Tierney, editors, *Learning and Intelligent Optimization*, pages 318–331, Cham, 2023. Springer International Publishing.
7. Efthyvoulos Drousiotis, Lei Shi, and Simon Maskell. Early predictor for student success based on behavioural and demographical indicators. In Alexandra I. Cristea and Christos Troussas, editors, *Intelligent Tutoring Systems*, pages 161–172, Cham, 2021. Springer International Publishing.
8. Efthyvoulos Drousiotis, Lei Shi, Paul G. Spirakis, and Simon Maskell. Novel decision forest building techniques by utilising correlation coefficient methods. In Lazaros Iliadis, Chrisina Jayne, Anastasios Tefas, and Elias Pimenidis, editors, *Engineering Applications of Neural Networks*, pages 90–102, Cham, 2022. Springer International Publishing.
9. Efthyvoulos Drousiotis and Paul G. Spirakis. Single mcmc chain parallelisation on decision trees. In Dimitris E. Simos, Varvara A. Rasskazova, Francesco Archetti, Ilias S. Kotsireas, and Panos M. Pardalos, editors, *Learning and Intelligent Optimization*, pages 191–204, Cham, 2022. Springer International Publishing.
10. Efthyvoulos Drousiotis and Paul G Spirakis. Single mcmc chain parallelisation on decision trees. In *Learning and Intelligent Optimization: 16th International Conference, LION 16, Milos Island, Greece, June 5–10, 2022, Revised Selected Papers*, pages 191–204. Springer, 2023.
11. Efthyvoulos Drousiotis, Paul G Spirakis, and Simon Maskell. Parallel approaches to accelerate bayesian decision trees. *arXiv preprint arXiv:2301.09090*, 2023.
12. Efthyvoulos Drousiotis, Alessandro Varsi, Paul G. Spirakis, and Simon Maskell. A shared memory smc sampler for decision trees. In *2023 IEEE 35th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 209–218, 2023. doi:10.1109/SBAC-PAD59825.2023.00030.
13. Simon Duane, Anthony D Kennedy, Brian J Pendleton, and Duncan Roweth. Hybrid monte carlo. *Physics letters B*, 195(2):216–222, 1987.
14. Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.
15. Belinda Hernández, Stephen R Pennington, and Andrew C Parnell. Bayesian methods for proteomic biomarker development. *EuPA Open Proteomics*, 9:54–64, 2015. doi:https://doi.org/10.1016/j.euprot.2015.08.001.
16. Wenbiao Hu, Rebecca A. O'Leary, Kerrie Mengersen, and Samantha Low Choy. Bayesian classification and regression trees for predicting incidence of cryptosporidiosis. *PLOS ONE*, 6(8):1–8, 08 2011. doi:10.1371/journal.pone.0023903.

17. Tiancheng Li, Miodrag Bolic, and Petar M. Djuric. Resampling methods for particle filtering: Classification, implementation, and strategies. *IEEE Signal Processing Magazine*, 32(3):70–86, 2015. doi:10.1109/MSP.2014.2330626.
18. Jun S Liu, Faming Liang, and Wing Hung Wong. The multiple-try method and local optimization in metropolis sampling. *Journal of the American Statistical Association*, 95(449):121–134, 2000.
19. Amal Saki Malehi and Mina Jahangiri. Classic and bayesian tree-based methods. In Petrică Vizureanu, editor, *Enhanced Expert Systems*, chapter 3. IntechOpen, Rijeka, 2019. doi:10.5772/intechopen.83380.
20. Enzo Marinari and Giorgio Parisi. Simulated tempering: a new monte carlo scheme. *EPL (Europhysics Letters)*, 19(6):451, 1992.
21. Pierre Del Moral, Arnaud Doucet, and Ajay Jasra. Sequential monte carlo samplers. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 68(3):411–436, 2006. URL: http://www.jstor.org/stable/3879283.
22. Per Mykland, Luke Tierney, and Bin Yu. Regeneration in markov chain samplers. *Journal of the American Statistical Association*, 90(429):233–241, 1995.
23. Matthew T. Pratola. Efficient Metropolis–Hastings Proposal Mechanisms for Bayesian Regression Tree Models. *Bayesian Analysis*, 11(3):885 – 911, 2016. doi:10.1214/16-BA999.
24. Christian P Robert, Víctor Elvira, Nick Tawn, and Changye Wu. Accelerating mcmc algorithms. *Wiley Interdisciplinary Reviews: Computational Statistics*, 10(5):e1435, 2018.
25. Malcolm Strens. Efficient hierarchical mcmc for policy search. In *Proceedings of the Twenty-First International Conference on Machine Learning*, ICML '04, page 97, New York, NY, USA, 2004. Association for Computing Machinery. doi:10.1145/1015330.1015381.
26. Patrik Waldmann. Genome-wide prediction using bayesian additive regression trees. *Genetics Selection Evolution*, 48, 12 2016. doi:10.1186/s12711-016-0219-8.
27. Yuhong Wu, Håkon Tjelmeland, and Mike West. Bayesian cart: Prior specification and posterior simulation. *Journal of Computational and Graphical Statistics*, 16(1):44–66, 2007. doi:10.1198/106186007X180426.

Multi-Assignment Scheduler: A New Behavioral Cloning Method for the Job-Shop Scheduling Problem

Imanol Echeverria[1][0000−0001−6053−0589], Maialen Murua[1][0000−0001−7922−6771], and Roberto Santana[2][0000−0002−0430−2255]

[1] TECNALIA, Basque Research and Technology Alliance (BRTA), Donostia-San Sebastian, Spain
[2] Computer Science and Artificial Intelligence Department, University of the Basque Country (UPV/EHU), Donostia-San Sebastian, Spain

Abstract. Recent advances in applying deep learning methods to address complex scheduling problems have highlighted their potential in learning dispatching rules. However, most studies have predominantly focused on deep reinforcement learning (DRL). This paper introduces a novel methodology aimed at learning dispatching policies for the job-shop scheduling problem (JSSP) by employing behavioral cloning and graph neural networks. By leveraging optimal solutions for the training phase, our approach sidesteps the need for exhaustive exploration of the solution space, thereby enhancing performance compared to DRL methods proposed in the literature. Additionally, we introduce a novel modelling of the JSSP with the aim of improving efficiency in terms of solving an instance in real time. This involves two key aspects: firstly, the creation of an action space that allows our policy to assign multiple operations to machines within a single action, substantially reducing the frequency of model usage; and secondly, the definition of a state space that only includes significant operations. We evaluated our methodology using a widely recognized open JSSP benchmark, comparing it against four state-of-the-art DRL methods and an enhanced metaheuristic approach, demonstrating superior performance.

Keywords: Job-shop scheduling problem · Behavioral cloning · Markov process · Graph neural networks.

1 Introduction

In the realm of manufacturing and production, efficiently solving the job-shop scheduling problem (JSSP) is a cornerstone for achieving optimal operational performance [16]. The JSSP, a well-known combinatorial optimization (CO) problem, is crucial in various industries, particularly those involving complex and diverse manufacturing procedures. The objective in JSSP is to optimally assign a series of jobs to a limited number of machines, typically with the goal of minimizing the overall completion time, known as makespan. This problem,

however, is notoriously difficult due to its NP-hard nature, meaning that finding the optimal solution within a reasonable timeframe becomes increasingly impractical as the problem size grows [15].

Traditional strategies for tackling the JSSP predominantly rely on exact techniques, such as mixed-integer linear programming (MILP) and branch-and-bound algorithms [36]. These methods, while effective in certain scenarios, often struggle with the scalability and complexity inherent in real-world scheduling problems. As a result, heuristics and metaheuristics, including tabu search [25], genetic algorithms [10], and simulated annealing [11], have been widely adopted. However, they still face challenges in consistently achieving near-optimal solutions, especially in highly complex scheduling environments.

Recent advances in deep learning (DL) and, more specifically, deep reinforcement learning (DRL), have opened new avenues for tackling the JSSP [7]. The ability of DLR to learn and adapt to complex patterns makes it a promising tool for developing efficient dispatching rules in scheduling problems. Unlike traditional methods, DRL can learn end-to-end policies that map the state of the scheduling environment directly to the dispatching actions, potentially offering improved solutions. This is particularly relevant in the context of JSSP, where the scheduling environment can be exceedingly intricate, and the standard heuristic rules may not suffice [24].

The application of DRL in solving scheduling problems faces several challenges. First, the size of the solution space makes the process of finding optimal solutions computationally demanding. This complexity can hinder the policies ability to train on optimal instances, thereby limiting its effectiveness. Additionally, for smaller problem instances, exact methods such as constraint programming (CP) can yield optimal solutions [8], but RL-based methods cannot effectively integrate such knowledge. Methods based on behavioral cloning (BC) [19] allow for the imitation of assignments of operations to machines that generate an optimal solution, but their use is much more limited in scheduling problems. This is because by using only optimal instances, the solution space may not be fully explored, potentially leading to inferior performance.

Furthermore, the modeling of the state and action spaces presents its own set of limitations. The action space generally entails assigning an operation to a machine, which implies that the model will be used as many times as operations the problem instance has. For larger instances, this approach might be too time-consuming as it could fail to generate solutions in real-time. Additionally, while recent methods represent the problem instance through graphs [30], this approach may not be feasible for larger problems, due to the number of operations to process. In such scenarios, taking into account operations that are not imminent in the scheduling process may not be necessary for immediate decision-making. Hence, there is a clear need for more efficient representation in DRL-based methods to improve decision-making in the face of complex scheduling challenges.

Therefore, in this paper, we focus on overcoming these limitations by proposing a new method for modeling the JSSP as a Markov Process (MP) and finding an optimal policy. Our contributions are as follows:

- We introduce a training approach for DL models using heterogeneous graph neural networks (GNNs) and BC, utilizing fewer but optimal CP-generated solutions from complete and partially solved instances, achieving superior results compared to DRL methods.
- We propose a new way of representing the JSSP as a MP. This involves defining a state space that limits the number of operations and an action space that allows assigning multiple operations in a single action instead of a single assignment, reducing the number and cost of state transitions and the number of model usages in the inference phase.
- We have conducted experiments using the Taillard benchmark [31], a well-known benchmark with instances of varying sizes. Our method has been compared against four state-of-the-art DRL methods, and an enhanced meta-heuristic approach, outperforming all of them.

The organization of the paper is as follows: We begin with a review of DL methods in scheduling in Section 2. The formulation of the JSSP is detailed in Section 3. An in-depth explanation of our proposed method is provided in Section 4. The discussion of experiments conducted and their results is found in Section 5. Finally, the paper concludes with a summary of our findings and potential avenues for future work in Section 6.

2 Related work

In this section, we focus on the application of DL methods to CO problems. We begin by discussing how DL has been applied to routing problems, such as the traveling salesman problem (TSP) and the capacitated vehicle routing problem (CVRP). Our discussion then shifts to scheduling, where we examine the application of DL in the field.

Regarding routing problems, in [34], the Pointer Network (Ptr-Net), a DL model that uses neural attention, was applied to various CO problems, including the TSP. The model was trained using a set of optimal solutions. In [2], this work was expanded using DRL, specifically the REINFORCE algorithm, to train Ptr-Net for the TSP, while the authors of [26] adapted it for the CVRP. A significant advancement was the application of attention models to CO problems, replacing the Ptr-Net with a transformer, enabling better understanding of relationships within the components of CO problems and thus enhancing performance [20].

The application of DL, particularly DRL, has recently been extended to scheduling problems. Early approaches encapsulated state information in a vector containing features. The authors in [32] proposed an RL-based method, where each job was represented by a feature vector, and a multi-layer perceptron (MLP) based policy was used to decide whether a job should be assigned to a machine. The major limitation of this approach is its lack of size-agnosticism; the network

cannot solve instances of varying sizes without retraining. A subsequent step presented in [23] involved using convolutional networks that utilized three matrices to represent the problem, with the action being the selection among various dispatching rules. This approach, however, was constrained by its simplistic action space, limiting the quality of solutions.

To overcome these challenges, the authors in [38] modeled the JSSP using a disjunctive graph and introduced an end-to-end DRL method for learning dispatching rules. The use of graphs and GNNs addressed the size-agnosticism challenge and captured a wide range of information for complex scheduling problems. GNNs facilitate information sharing between operations connected by conjunctive and disjunctive edges within a disjunctive graph. Several approaches have been introduced with varying reward functions, training algorithms, masking techniques, and model structures [5, 17, 21, 27, 33].

Another method for modeling different nodes and their relationships is the use of heterogeneous graphs. Song et al. [30] proposed using heterogeneous graph attention networks (HGATs) to address the flexible job-shop scheduling problem, a variant of the JSSP in which operations can be assigned to multiple machines, creating nodes for operation and machine types with various edge types to indicate precedence and machine-operation relations. This method simplifies the action space compared to disjunctive graphs by reducing the number of edges.

In scheduling, several approaches have employed optimally generated solutions for model training with BC, using optimal solutions to train the neural networks (NNs). This stands in contrast to RL, which can be more time-consuming, as RL training necessitates exploring sub-optimal experiences to discover optimal solutions. There is a clear disparity in the number of contributions that employ BC compared to those using RL. In [19], the authors introduced an approach that utilizes a MILP solver to produce optimal demonstrations, aiming to acquire an efficient dispatching rule for the JSSP. The authors of [6] proposed a method for the flexible job-shop scheduling problem by dividing it into two separate problems and training distinct convolutional networks with instances solved by the IBM CP-Optimizer solver. However, these approaches are not size-agnostic, requiring a model to be trained for each problem instance, thus limiting their applicability. Lastly, the flow shop scheduling problem, a simpler variant of the JSSP, was addressed in [22], using GNNs to replicate actions and states generated by the NEH [28] algorithm, surpassing NNs trained through RL.

3 JSSP Formulation

The JSSP can be formally defined as follows. Let there be a set of jobs $\mathcal{J} = \{j_1, j_2, ..., j_n\}$ and a set of machines $\mathcal{M} = \{m_1, m_2, ..., m_m\}$. Each job j_i consists of a sequence of operations $\{o_{i1}, o_{i2}, ..., o_{ik}\}$ that need to be processed in a specific order. Each operation o_{ij} is to be processed on a specific machine m_{ij} for a duration of p_{ij}, which is the processing time. The objective is to find a schedule S, which is a set of start times for each operation o_{ij}, such that the following conditions are met: 1) Sequence constraint: each operation of a job must be

completed before the next operation of the same job begins. 2) No overlapping constraint: each machine can process only one operation at a time. The most common objective is to find a schedule that minimizes the makespan, which is the completion time of the last operation in the schedule.

Table 1. JSSP instance with 3 jobs and 3 machines where the processing times of each operation are indicated.

Jobs	Operations	m_1	m_2	m_3
j_1	o_{11}	-	4	-
	o_{12}	-	-	3
	o_{13}	4	-	-
	o_{14}	-	3	-
j_2	o_{21}	-	2	-
	o_{22}	2	-	-
j_3	o_{31}	-	-	2
	o_{32}	-	2	-
	o_{33}	1	-	-

In Table 1, a simple instance of the JSSP with 3 jobs and 3 machines is presented, where their operations and processing times are listed. Within Figures 1 and 2, two solutions to the problem are depicted. In these images, each distinct color corresponds to a job, while the x-axis represents time, and the y-axis machines. It can be observed that the second solution, which is optimal, achieves a lower makespan as operations are parallelized more efficiently in this solution.

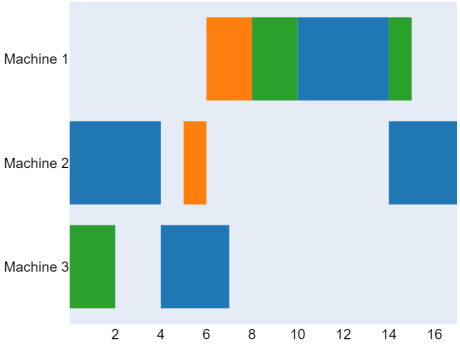


Fig. 1. A solution to the JSSP instance with a makespan of 17.

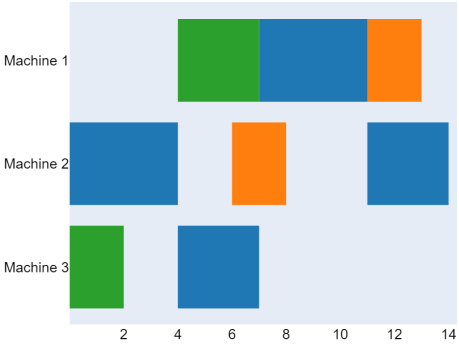


Fig. 2. An optimal solution with a makespan of 14.

4 Proposed method: Multi-assignment scheduler

In this section, we present our method for generating a policy trained based on instances solved optimally by a CP-based method, which we call Multi-assignment scheduler (MAS). First, we will describe how the JSSP has been modeled as a Markov process, followed by a description of the GNN used, and finally, the instance generation method and model training.

4.1 The JSSP as a Markov Process

The first steps consist of framing the problem as a Markov process, distinguishing it from current approaches that use Markov decision processes by not requiring a reward function. This simplifies the modelling to define a state space, an action space, and a transition function.

State space. Building upon the previously introduced state space representation in [12], we employ heterogeneous graphs. At each timestep t, the state s_t is represented as a heterogeneous graph $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$, consisting of three node types (operations, jobs, and machines) and six edge types, both directed and undirected. The number of operations considered per job within the state is constrained to a fixed number (in our experiments, 10). This limitation reduces the amount of information to be processed at each step, particularly beneficial in larger instances, where the influence of operations with many unassigned tasks is deemed insignificant in current assignments. For a more detailed description of the features of the nodes and edges, we refer the reader to the paper [12] (section 4.2, pages 7-8).

Action space. At each timestep t, the action space $\mathcal{A}_t$ consists of compatible job-machine pairs. When selecting a job, the first unscheduled operation of that job is selected. To prevent over-selection, the set of pairs is limited in two ways. Firstly, we define st as the timestamp at which the first operation can be assigned to a machine, and actions are masked accordingly, making operations starting later than $st * p$ illegal. Here, p is close to one (1.05 in our experiments). Secondly, the number of possibilities is constrained to the number of machines in the instance. This particularly affects larger instances with many jobs and few machines to avoid considering too many options simultaneously.

Transition function. The solution is built through the sequential assignment of operations to machines. Upon the assignment of an operation, the operation is removed, and the edges of the corresponding job are updated to indicate the next operation to be executed.

In Figure 3, a state transition with the different types of nodes and their relationships is depicted, where the maximum number of operations per job is set to 2. For simplicity, only jobs j_2 and j_3 are represented. As can be observed, even though j_3 has 3 operations, only the first two are processed, and once the action is taken, the assigned operations are removed.

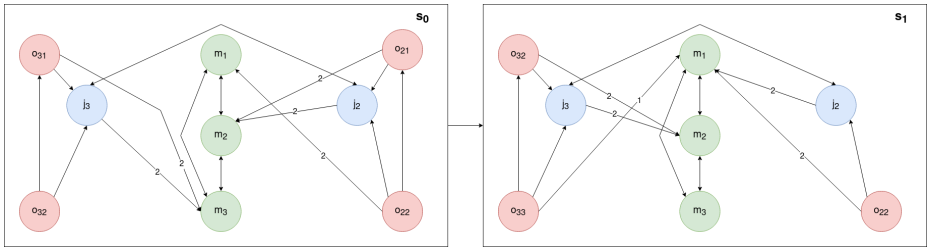


Fig. 3. Diagram depicting the example from Table 1 and a state transition where operations o_{31} and o_{21} have been assigned.

4.2 Graph Neural Network Architecture

The use of GNNs is common for extracting features from a state of a Markov process representing a CO problem, particularly those using attention mechanisms [30, 35]. In a previous work [12], we used the architecture proposed in [3], GATv2. In this case, we will adapt the architecture proposed by [29] and [18], inspired by transformers, for heterogeneous graphs because in preliminary experimentation we have identified that this is more robust than the one previously used.

To make its use clearer, we will show how to calculate the embedding of the operation node. The operation node is related to the operation that follows, indicating precedence, and the machines on which it can be executed. The initial representation of this node is called $\boldsymbol{h_{o_{ij}}} \in \mathbb{R}^{d_{\mathcal{O}}}$, where $d_{\mathcal{O}}$ is the initial number of node features. First, we need to calculate the attention coefficients for the two relationships. If the operation has a successor, between operations o_{ij} and o_{ij+1}, the coefficient $\alpha_{o_{ijj+1}}$ is defined as:

$$\alpha_{o_{ijj+1}} = \text{softmax}\left(\frac{\left(\mathbf{W}_1^{\mathcal{O}}\boldsymbol{h_{o_{ij}}}\right)^\top \left(\mathbf{W}_2^{\mathcal{O}}\boldsymbol{h_{o_{ij+1}}}\right)}{\sqrt{d'_{\mathcal{O}}}}\right) \tag{1}$$

where $\boldsymbol{W_1^{\mathcal{O}}}, \boldsymbol{W_2^{\mathcal{O}}} \in \mathcal{R}^{d'_{\mathcal{O}} \times d_{\mathcal{O}}}$ are learned linear transformations, and $d'_{\mathcal{O}}$ is the dimension given to the hidden space of the embeddings. To calculate this value for the relationship between operations and machines, it is similar, only adding the edge feature. For operation o_{ij} and m_k, the edge feature is denoted $\boldsymbol{h_{om_{ijk}}} \in \mathbb{R}^{d_{\mathcal{OM}}}$, where $d_{\mathcal{OM}}$ is the dimension of the feature, and $\boldsymbol{h_{m_k}} \in \mathbb{R}^{d_{\mathcal{M}}}$ is the representation of machine m_k, and $d_{\mathcal{M}}$ is its dimension. The attention coefficient $\alpha_{om_{ijk}}$ is computed as:

$$\alpha_{om_{ijk}} = \text{softmax}\left(\frac{\left(\mathbf{W}_1^{\mathcal{OM}}\boldsymbol{h_{o_{ij}}}\right)^\top \left(\mathbf{W}_2^{\mathcal{OM}}\boldsymbol{h_{m_k}} + \mathbf{W}_3^{\mathcal{OM}}\boldsymbol{h_{om_{ijk}}}\right)}{\sqrt{d'_{\mathcal{OM}}}}\right) \tag{2}$$

where $\boldsymbol{W_1^{\mathcal{OM}}} \in \mathcal{R}^{d'_{\mathcal{OM}} \times d_{\mathcal{OM}}}$, $\boldsymbol{W_2^{\mathcal{OM}}} \in \mathcal{R}^{d'_{\mathcal{OM}} \times d_{\mathcal{M}}}$, and $\boldsymbol{W_3^{\mathcal{OM}}} \in \mathcal{R}^{d'_{\mathcal{OM}} \times d_{\mathcal{OM}}}$. Once the attention coefficients have been calculated, the new embedding of o_{ij},

$\boldsymbol{h'_{o_{ij}}}$ is computed as:

$$\boldsymbol{h'_{o_{ij}}} = \mathbf{W}_3^{\mathcal{O}} \boldsymbol{h_{o_{ij}}} + \alpha_{o_{ijj+1}} \mathbf{W}_4^{\mathcal{O}} \alpha_{o_{ijj+1}} \boldsymbol{h_{o_{ij+1}}} + \sum_{m_k \in \mathcal{O}_{m_k}} \alpha_{om_{ijk}} (\mathbf{W}_4^{\mathcal{OM}} \boldsymbol{h_{m_k}} + \mathbf{W}_5^{\mathcal{OM}} \boldsymbol{h_{om_{ijk}}}) \tag{3}$$

There are two ways to add complexity to the GNN to capture more meaningful relationships between nodes: multiple attention heads and increasing the number of layers. Using multiple attention heads is a common technique in various domains such as neural machine translation or computer vision [1, 4], and in CO problems [26, 35], as it allows us to focus on important nodes and relations. Assuming we have K attention heads, $\boldsymbol{h'^{k}_{o_{ij}}}, k = 1, 2, \ldots K$, K embeddings are calculated, and each attention head has its own attention coefficients with its respective linear transformations. Once calculated, they are concatenated to obtain the final embedding:

$$\boldsymbol{h'_{o_{ij}}} = \|_{k=1}^{K} \boldsymbol{h'^{k}_{o_{ij}}} \tag{4}$$

The second way to add complexity is to use multiple layers, i.e., repeating the procedure L times but with different learnable parameters in each layer. Having a sufficient number of layers is essential for information from nodes that are not directly connected to propagate, as in layer l, embeddings calculated in layer $l-1$ are used. The output of this layer is called $\boldsymbol{h^{L}_{o_{ij}}}$.

4.3 Training procedure with BC

The information from the GNN embeddings is used to generate a policy. MLPs are utilized to define the policy on edges connecting compatible jobs and machines. Specifically, if the set of edges for the state s_t is denoted as $\mathcal{JM}_t$, the score of selecting an edge $e \in \mathcal{JM}_t$, where this edge connects job j_i and machine m_k, and the edge features are denoted as $\boldsymbol{h_{j_i m_k}}$, is defined as follows:

$$\mu_\theta(e|s_t) = \text{MLP}_\theta(\boldsymbol{h^{L}_{j_i}} || \boldsymbol{h^{L}_{m_k}} || \boldsymbol{h_{j_i m_k}}) \tag{5}$$

$$\pi_\theta(e|s_t) = \text{softmax}\left(\mu_\theta(e|s_t) - \frac{\sum_{x \in \mathcal{JM}_t} \mu_\theta(x|s_t)}{|\mathcal{JM}|}\right) \tag{6}$$

In Equation 6, the average is taken to ensure that at least one action will be selected (only positive actions are selected), and the softmax function is employed to constrain the values between 0 and 1.

After defining the MP and selecting the GNN architecture, training the agent using BC involves generating a set of optimal solutions for n_i instances, denoted $\mathcal{I}$. This is achieved by synthetically generating a set of instances that can be solved using CP techniques in less than 10 seconds (in order to lower the computational cost of generating the dataset), followed by constructing a trajectory of states and actions. Additionally, in this paper, it is proposed to start the construction of an instance solution by randomly selecting operations until at

least a partial solution has been completed. The length of the partial solution is determined by a percentage parameter. This parameter is randomly chosen for each instance uniformly in $[0, 0.3]$. The aim of this procedure is to create a more diverse dataset in which the model can learn from different scheduling scenarios that may arise.

Given the constructive nature of the approach, feasible assignments of operations to machines must be established. Initially, operations are sorted by their start time and end time, facilitating the construction of a sequence of states and actions. Each step will consist of all possible operations until there is an operation that must be assigned to a machine to which another operation has already been assigned. That is, in a single action, two operations cannot be assigned to the same machine. Formally, for each instance $i_i \in \mathcal{I}$, the set $\mathcal{D}_i = \{(s_1, a_1), (s_2, a_2), \ldots, (s_{d_i}, a_{d_i})\}$ is defined, where d_i represents the number of actions to solve the instance. For all $t = \{1, 2, \ldots d_i\}$, s_t represents a heterogeneous graph, and a_t denotes the action to be taken, which is a vector with zeros in the pairs of jobs and machines that are not to be executed and ones in those that are. The final dataset is defined as the union of all $\mathcal{D}_i$, that is $\mathcal{D} = \bigcup_{i_i \in \mathcal{I}} \mathcal{D}_i$.

These trajectories constitute the dataset used to train the GNN as a supervised learning model, specifically as a multi-label classification problem. Training occurs over n_e epochs, with the dataset divided into batches of size n_b for each epoch. Binary cross-entropy is used as the loss function, since this is a classification problem.

5 Experimental results

In this section, we describe the experimental setup and present the results of our method in comparison to other approaches.

5.1 Experimental Setup

Configuration. For the implementation of our proposed method, Python 3.10 was employed and PyTorch Geometric for the implementation of the GNNs [14]. For the CP method, OR-Tools[3] was selected due to its open-source nature and widespread use in scheduling problems [9]. The experiments were conducted on a machine equipped with an AMD Ryzen 5 5600X processor and an NVIDIA GeForce RTX 3070 Ti. The code is publicly available at `https://github.com/Echever/MAS_Scheduler`.

The configuration parameters were set as follows: the number of the GNN layers to 4, the number of all the hidden channels to 128, the number of attention heads to 4, the number of epochs to 25, the batch size to 64 and the learning rate to 0.0002. Regarding the action mask, the number of actions considered was limited to a maximum of the number of machines in the instance, and only those operations that could start at 1.05 times the current timestamp were considered.

[3] We obtained the implementation from `https://github.com/google/or-tools/blob/stable/examples/python/flexible_job_shop_sat.py`.

Dataset. For generating the training dataset, a set of instances were created and solved using OR-Tools, using 5300 in the training phase and 50 as the validation set. This set was generated by modifying the method proposed by Taillard [31] to ensure greater diversity in the training set. Specifically, the quantity of jobs, n_j, was sampled from $U(10, 20)$, the number of machines, n_m, from $U(10, n_j)$, the number of operations per job from $\sim U(n_m, n_m + 5)$, and the processing time for each operation sampled from $U(5, 90)$. As mentioned earlier, each instance is solved until a certain percentage of operations are assigned (between 0 and 0.3 of the total). For the test set, the well-known JSSP benchmark dataset by Taillard [31] was used. It consists of 80 instances ranging from 15 jobs and 15 machines (with 225 operations) to 100 jobs and 20 machines (2000 operations), providing a comprehensive overview of a method's performance across instances of varying sizes.

Baselines. Our approach is compared to several methods that focus on learning a priority rule using DRL. Among the various DRL-based approaches, we compare with the four state-of-the-art approaches: the method proposed in [33], which uses a CP method as the input of their method and for feedback defining the reward (in this paper we refer to this approach as RLCP); [21], which employs attention mechanisms similar to this paper (AttSch); [17], which introduces a method called residual scheduling that eliminates unnecessary operations and machines (ResSch); and the method proposed in [37], which uses a new state representation using the state features of bidirectional scheduling (BiSch). Additionally, our approach is also compared with an enhanced metaheuristic [13], which has been shown to outperform traditional JSSP metaheuristics. Of the three types of policies they trained, the best for the Taillard benchmark, NLS_{AN}, is chosen. Of the methods mentioned, all use the Taillard benchmark and present their results on the instances, except for RLCP, for which their open-source implementation was used.

Evaluation Metric. Performance is evaluated based on the optimal gap, with the makespan serving as the optimization target, which is defined as:

$$OG = \left(\frac{C_\pi}{C_{\text{ub}}} - 1 \right) \cdot 100 \tag{7}$$

where C_π is the makespan produced by a policy and C_{ub} is either the instance's optimal or best-known makespan.

5.2 Results on the Taillard JSSP benchmark

Performance Analysis. This subsection presents an analysis of our method's results on the Taillard benchmark. Table 2 shows the optimal gap achieved by our method and the state-of-the-art techniques. Each column of this table represents the average of the results for ten instances with the same number of jobs and machines. The first row of the table presents the results from the enhanced metaheuristic method NLS_{AN}. Following this, the table includes results from various constructive methods that utilize DRL. For each method, the table displays the

average gap from the optimal solution. The final column of the table aggregates these averages to present the average optimal gap when the complete benchmark is considered. A key observation from the table is that our method maintains a

Table 2. Optimal gap comparison with an enhanced meta-heuristic, DRL methods, and MAS in the Taillard benchmark.

Method	15×15	20×15	20×20	30×15	30×20	50×15	50×20	100×20	Mean
NLS_{AN}	**10.32**	**13.18**	**12.95**	14.91	17.78	11.87	12.02	6.22	12.41
BiSch	17.85	20.59	19.53	22.39	23.32	16.03	17.41	8.91	18.25
ResSch	13.70	18.00	16.50	17.30	18.10	8.40	11.40	4.00	13.40
AttSch	13.10	17.90	16.10	19.60	22.80	12.90	16.80	9.30	16.06
RLCP	16.27	19.75	18.61	18.29	22.81	10.13	14.03	4.56	15.55
MAS	13.54	13.96	13.40	**14.84**	**17.40**	**7.08**	**9.51**	**2.31**	**11.50**

consistently lower average gap from the optimal solution compared to both DRL-based and the enhanced metaheuristic NLS_{AN}. This is particularly noticeable in larger instances, especially those with 30 jobs or more, where our method outperforms all the other methods. In comparison with other DRL methods, our approach yields better results in nearly all the instance groups, with the only exception being the very first group, the smallest. Additionally, our method demonstrates effective generalization capabilities, as it maintains low optimal gap values in instance sizes that are larger than those in the training dataset.

Analysis of the quantity of model usages. Figure 4 offers an analysis comparing how frequently different models are used in the Taillard benchmark across various approaches. The graph plots the number of operations in an instance (shown on the x-axis) against the number of inferences made by the model (displayed on the y-axis). This analysis compares the BiSch, ResSch, and AttSch methods —wherein each method assigns a single operation to a machine— against the RLCP method, which also permits multiple assignments concurrently, as well as our MAS approach.

The graph illustrates that for methods employing a single assignment strategy, the frequency of model usage increases linearly, akin to the number of operations. The RLCP approach displays a similar rate of increase. Nonetheless, our MAS approach reveals a distinctly more efficient pattern, showcasing a significantly less pronounced rise in model usage as the number of operations escalates. This denotes a more scalable and efficient utilization of the model in complex scenarios.

Execution Time Analysis. The final part of our analysis compares the execution times. Focusing on RLCP [37], which is the only algorithm for which the code was available for comparison, Table 3 displays the average execution times, categorized by the size of the instances. Our approach not only solves instances in less time but also, it is important to note, employs a more complex GNN architecture. By potentially simplifying this architecture, we could achieve even

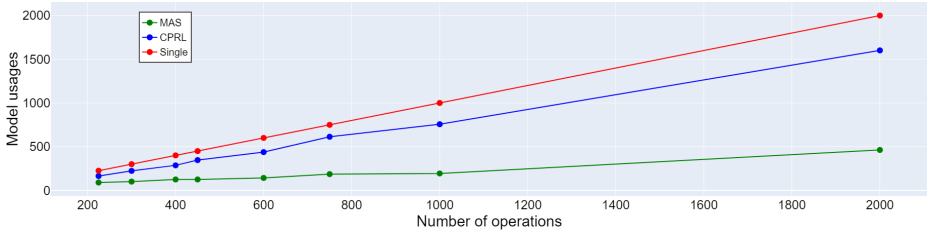


Fig. 4. Comparison of the number of inferences between our approach, the DRL approaches in which a single operation is assigned, and the RLCP in the Taillard benchmark.

Table 3. Execution time for CPRL and MAS approaches in the Taillard benchmark.

Method	15×15	20×15	20×20	30×15	30×20	50×15	50×20	100×20	Mean
RLCP	5.60	6.32	7.29	8.99	10.90	16.96	20.64	63.75	18.81
MAS	**3.10**	**3.78**	**5.02**	**5.41**	**6.72**	**12.07**	**13.83**	**58.24**	**13.52**

faster execution times. The recorded execution times align with those reported for the RLCP approach.

6 Conclusions and future work

In this paper we have introduced a novel DL method for addressing the JSSP, utilizing BC and GNNs. This approach effectively avoids the extensive exploration of the solution space, which is a requisite in traditional RL algorithms, enhancing performance. In addition, we have refined the modeling of the JSSP to enhance efficiency. This is achieved by enabling the assignment of multiple operations to machines in a single action, thereby reducing the frequency of model usage, and by focusing on operations that are immediately relevant to the scheduling process.

The effectiveness of our method has been tested on the Taillard benchmark. Our approach not only surpassed several state-of-the-art DRL learning methods but also outperformed an enhanced metaheuristic approach, particularly in larger instances. Additionally, the analysis performed to compare the execution time and model uses showed that the execution times are lower than other DRL methods and that there was a significant reduction in the number of model usages.

Moving forward, our work will explore the application of this methodology to various other scheduling problems to assess its adaptability and efficiency in different contexts. Moreover, we will also explore the design of GNNs capable of extracting information more efficiently in larger instances of the problem.

Acknowledgements

This work was partially financed by the Basque Government through their Elkartek program (SONETO project, ref. KK-2023/00038) and the Gipuzkoa Provicial Council through their Gipuzkoa network of Science, Technology and Innovation program (KATEAN project, ref. 2023-CIEN-000053-01). R. Santana thanks the Misiones Euskampus 2.0 programme for the financial help received through Euskampus Fundazioa. Partial support by the Research Groups 2022-2024 (IT1504-22) and the Elkartek Program from the Basque Government, and the PID2019-104966GB-I00 and PID2022-137442NB-I00 research projects from the Spanish Ministry of Science is acknowledged.

References

1. Behnke, M., Heafield, K.: Losing heads in the lottery: Pruning transformer attention in neural machine translation. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP). pp. 2664–2674 (2020)
2. Bello, I., Pham, H., Le, Q.V., Norouzi, M., Bengio, S.: Neural combinatorial optimization with reinforcement learning. arXiv preprint arXiv:1611.09940 (2016)
3. Brody, S., Alon, U., Yahav, E.: How attentive are graph attention networks? arXiv preprint arXiv:2105.14491 (2021)
4. Chen, K., Wang, J., Chen, L.C., Gao, H., Xu, W., Nevatia, R.: ABC-CNN: An attention based convolutional neural network for visual question answering. arXiv preprint arXiv:1511.05960 (2015)
5. Chen, R., Li, W., Yang, H.: A deep reinforcement learning framework based on an attention mechanism and disjunctive graph embedding for the job-shop scheduling problem. IEEE Transactions on Industrial Informatics **19**(2), 1322–1331 (2022)
6. Chen, W., Khir, R., Van Hentenryck, P.: Two-stage learning for the flexible job shop scheduling problem. arXiv preprint arXiv:2301.09703 (2023)
7. Cunha, B., Madureira, A.M., Fonseca, B., Coelho, D.: Deep reinforcement learning as a job shop scheduling solver: A literature review. In: Hybrid Intelligent Systems: 18th International Conference on Hybrid Intelligent Systems (HIS 2018) Held in Porto, Portugal, December 13-15, 2018 18. pp. 350–359. Springer (2020)
8. Da Col, G., Teppan, E.: Google vs IBM: A constraint solving challenge on the job-shop scheduling problem. arXiv preprint arXiv:1909.08247 (2019)
9. Da Col, G., Teppan, E.C.: Industrial size job shop scheduling tackled by present day CP solvers. In: Principles and Practice of Constraint Programming: 25th International Conference, CP 2019, Stamford, CT, USA, September 30–October 4, 2019, Proceedings 25. pp. 144–160. Springer (2019)
10. Davis, L.: Job shop scheduling with genetic algorithms. In: Proceedings of the first International Conference on Genetic Algorithms and their Applications. pp. 136–140. Psychology Press (2014)
11. Defersha, F.M., Obimuyiwa, D., Yimer, A.D.: Mathematical model and simulated annealing algorithm for setup operator constrained flexible job shop scheduling problem. Computers & Industrial Engineering **171**, 108487 (2022)
12. Echeverria, I., Murua, M., Santana, R.: Solving large flexible job shop scheduling instances by generating a diverse set of scheduling policies with deep reinforcement learning. arXiv preprint arXiv:2310.15706 (2023)

13. Falkner, J.K., Thyssens, D., Bdeir, A., Schmidt-Thieme, L.: Learning to control local search for combinatorial optimization. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. pp. 361–376. Springer (2022)
14. Fey, M., Lenssen, J.E.: Fast graph representation learning with PyTorch Geometric. In: ICLR Workshop on Representation Learning on Graphs and Manifolds (2019)
15. Garey, M.R., Johnson, D.S., Sethi, R.: The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research **1**(2), 117–129 (1976)
16. Gupta, A.K., Sivakumar, A.I.: Job shop scheduling techniques in semiconductor manufacturing. The International Journal of Advanced Manufacturing Technology **27**, 1163–1169 (2006)
17. Ho, K.H., Jheng, R.Y., Wu, J.H., Chiang, F., Chen, Y.C., Wu, Y.Y., Wu, I., et al.: Residual scheduling: A new reinforcement learning approach to solving job shop scheduling problem. arXiv preprint arXiv:2309.15517 (2023)
18. Hu, Z., Dong, Y., Wang, K., Sun, Y.: Heterogeneous graph transformer. In: Proceedings of the web conference 2020. pp. 2704–2710 (2020)
19. Ingimundardottir, H., Runarsson, T.P.: Discovering dispatching rules from data using imitation learning: A case study for the job-shop problem. Journal of Scheduling **21**, 413–428 (2018)
20. Kool, W., Van Hoof, H., Welling, M.: Attention, learn to solve routing problems! arXiv preprint arXiv:1803.08475 (2018)
21. Lee, J., Kee, S., Janakiram, M., Runger, G.: Attention-based reinforcement learning for combinatorial optimization: Application to job shop scheduling problem. arXiv preprint arXiv:2401.16580 (2024)
22. Li, L., Liang, S., Zhu, Z., Cao, X., Ding, C., Zha, H., Wu, B.: Learning to optimize permutation flow shop scheduling via graph-based imitation learning. arXiv preprint arXiv:2210.17178 (2022)
23. Liu, C.L., Chang, C.C., Tseng, C.J.: Actor-critic deep reinforcement learning for solving job shop scheduling problems. IEEE Access **8**, 71752–71762 (2020)
24. Luo, S., Zhang, L., Fan, Y.: Dynamic multi-objective scheduling for flexible job shop by deep reinforcement learning. Computers & Industrial Engineering **159**, 107489 (2021)
25. Martínez-Puras, A., Pacheco, J.: MOAMP-Tabu search and NSGA-II for a real bi-objective scheduling-routing problem. Knowledge-Based Systems **112**, 92–104 (2016)
26. Nazari, M., Oroojlooy, A., Snyder, L., Takác, M.: Reinforcement learning for solving the vehicle routing problem. Advances in Neural Information Processing Systems **31** (2018)
27. Park, J., Bakhtiyar, S., Park, J.: Schedulenet: Learn to solve multi-agent scheduling problems with reinforcement learning. arXiv preprint arXiv:2106.03051 (2021)
28. Sharma, M., Sharma, S.: An improved NEH heuristic to minimize makespan for flow shop scheduling problems. Decision Science Letters **10**(3), 311–322 (2021)
29. Shi, Y., Huang, Z., Feng, S., Zhong, H., Wang, W., Sun, Y.: Masked label prediction: Unified message passing model for semi-supervised classification. arXiv preprint arXiv:2009.03509 (2020)
30. Song, W., Chen, X., Li, Q., Cao, Z.: Flexible job-shop scheduling via graph neural network and deep reinforcement learning. IEEE Transactions on Industrial Informatics **19**(2), 1600–1610 (2022)
31. Taillard, E.: Benchmarks for basic scheduling problems. European Journal of Operational Research **64**(2), 278–285 (1993)

32. Tassel, P., Gebser, M., Schekotihin, K.: A reinforcement learning environment for job-shop scheduling. arXiv preprint arXiv:2104.03760 (2021)
33. Tassel, P., Gebser, M., Schekotihin, K.: An end-to-end reinforcement learning approach for job-shop scheduling problems based on constraint programming. arXiv preprint arXiv:2306.05747 (2023)
34. Vinyals, O., Fortunato, M., Jaitly, N.: Pointer networks. Advances in Neural Information Processing Systems **28** (2015)
35. Wang, R., Wang, G., Sun, J., Deng, F., Chen, J.: Flexible job shop scheduling via dual attention network based reinforcement learning. arXiv preprint arXiv:2305.05119 (2023)
36. Xie, J., Gao, L., Peng, K., Li, X., Li, H.: Review on flexible job shop scheduling. IET Collaborative Intelligent Manufacturing **1**(3), 67–77 (2019)
37. Yuan, E., Cheng, S., Wang, L., Song, S., Wu, F.: Solving job shop scheduling problems via deep reinforcement learning. Applied Soft Computing **143**, 110436 (2023)
38. Zhang, C., Song, W., Cao, Z., Zhang, J., Tan, P.S., Chi, X.: Learning to dispatch for job shop scheduling via deep reinforcement learning. Advances in Neural Information Processing Systems **33**, 1621–1632 (2020)

An imitation-based learning approach using DAgger for the Casual Employee Call Timing Problem

Prakash Gawas[1,2,3][0009−0002−3870−302X], Antoine Legrain[1,2,3][0000−0003−2903−9593], and Louis-Martin Rousseau[1,2][0000−0001−6949−6014]

[1] Polytechnique Montréal, Montréal, QC H3T 1J4, Canada
[2] CIRRELT, Montréal, QC H3T 1J4
[3] GERAD, Montréal, QC H3T 2A7

Abstract. Predictive models are increasingly important in enhancing decision-making processes. This study proposes an innovative approach utilizing DAgger, an imitation learning algorithm, to iteratively train a policy for addressing stochastic sequential decision problems. These problems can be challenging, especially when expert input is costly or unavailable. Our focus lies in crafting an effective expert within the DAgger framework, drawing from deterministic solutions derived from contextual scenarios generated at each decision point. Subsequently, a predictive model is developed to mimic the expert's behavior, aiding real-time decision-making. To illustrate the applicability of this methodology, we address a dynamic employee call-timing issue concerning the scheduling of casual personnel for on-call work shifts. The key decision involves determining the optimal time to contact the next employee in seniority order, allowing them to select a preferred shift. Uncertainty arises from the varying response times of employees. The goal is to strike a balance between minimizing schedule changes induced by early notifications or calls and avoiding unassigned shifts due to late notifications. Unlike traditional predict-and-optimize approaches, our method utilizes optimization to train learning models that establish connections between the system's current state and the expert's wait time. We apply our algorithm using data provided by our industrial partner to derive an operational policy. Results demonstrate the superiority of this policy over the current heuristic method in use.

Keywords: on-call scheduling · operations research · imitation learning.

1 Introduction

In operations research (OR), the integration of machine learning (ML) has led to significant progress, improving decision-making through adaptive intelligence and accurate predictions. Recent efforts from ML and operations research communities to apply ML to solve stochastic optimization problems have notably

increased. From the perspective of stochastic optimization, ML can enhance an algorithm's performance on a range of problem instances in two ways. Firstly, it can substitute certain resource-intensive computations with rapid approximations. Secondly, it can help explore the decision space and learn the most effective behavior (policy) from the experience. This paper is motivated by the desire to use ML to alleviate the computational challenges in stochastic sequential decision problems.

A conventional method of using ML for stochastic optimization is the predict-then-optimize approach, where ML's predictive capabilities feed OR's optimization strategies. The predict-then-optimize framework initially utilizes ML models to predict the values of uncertain parameters based on available auxiliary data. Subsequently, these point predictions are incorporated into downstream optimization problems, transforming uncertain problems into more manageable deterministic ones. Recently, progress has been made to train prediction models that effectively utilize the structure of the downstream optimization problem to reduce decision error rather than prediction error. Examples of such applications can be observed in areas such as Retail Inventory Management [5], Healthcare Resource allocation [6], and Logistics [11]. Furthermore [10] combines the learning and optimization, wherein ML models consider decision error in the learning phase.

When one cares about discovering new policies, i.e., optimizing an algorithmic decision function from the ground up, the policy may be learned by reinforcement learning through experience. Agents accumulate knowledge over time by experiencing various states, taking actions, and receiving feedback in the form of rewards. The major reinforcement learning algorithms are approximate dynamic programming and policy gradients.

An alternative approach involves employing ML to generate solutions or decisions from input instances directly, a method commonly known as End-to-End Learning [7], [8], [9]. This can be achieved through demonstrations, where an expert illustrates the expected behavior to the ML model, a process called Imitation Learning. In this paradigm, the learner focuses not on optimizing a performance measure but on mimicking the expert's actions. Imitation learning has proven successful in diverse domains, including robotics, autonomous vehicles, and game-playing, where human expertise directly contributes to model training. However, imitation learning presents challenges, notably the need for diverse and representative expert demonstrations, which can be resource-intensive. Additionally, issues such as distributional shifts and the potential impact of suboptimal expert behavior on model performance must be addressed. Despite these challenges, imitation learning has numerous applications in operations research, demonstrating its versatility and effectiveness in various operational contexts [12].

In the work by [1], a novel neural network architecture is employed to tackle the Euclidean Traveling Salesman Problem. The authors train the model using supervised learning, with pre-computed solutions to the traveling salesman problem serving as targets. Similar work can also be found in [13]. Another instance is

seen in the research by [4], where a neural network is trained to predict solutions for a stochastic load planning problem with a mixed-integer linear programming formulation. The authors utilize operational solutions, specific solutions to the deterministic version of the problem, aggregating them to provide tactical solution targets for the ML model. In an approach similar to ours, [2] adopts a prediction-based approach for online dynamic radiotherapy scheduling. They propose an Integer Programming (IP) model to derive optimal offline schedules from many instances. Subsequently, a model is trained, with the offline solution serving as expert decisions in the learning process. In all the above cases, the authors use offline deterministic solutions as an expert.

In this article, we propose an imitation learning approach to tackle the recently identified Employee Call Timing Problem (ECTP) as discussed in[3]. Our focus is scheduling on-call shifts for casual employees, addressing the dynamic sequential decision problem of formulating a policy for dispatching calls/notifications to employees based on their seniority, rendering them eligible to select work shifts. The challenge stems from the uncertainty surrounding the time it takes for an employee to respond to system calls, leading to frequent replacements or bumps governed by seniority rules. The primary goal is to minimize bumps while ensuring the assignment of all shifts. In the paper [3], the authors present the Employee Call Timing Problem as an $\mathcal{NP}$-complete problem. Their approach involves the development of dynamic policies through the aggregation of offline solutions at each decision epoch. Notably, certain crucial components of the system state are overlooked in their decision-making process. In contrast, a pivotal difference in our study lies in including comprehensive state information to formulate a policy and guide decision-making. Our contributions to the literature can be summarized as follows:

1. Our study tackles the emerging Employee Call Timing Problem by employing ML-driven operating policies. The ML model is trained using an expert function derived from mathematical optimization, effectively emulating the expert policy without imposing the computational burden during online operations. We experiment with different expert functions in a DAgger framework and choose the best one.
2. The results indicate that the machine-learned model outperforms a tuned version of the existing heuristic policy in use while showing almost similar performance to another heuristic in [3].

2 Problem Description

A new flexible on-call scheduling system is gaining popularity in North America. This system is designed for use by casual (very dynamic) workers, such as part-time students. They seek highly flexible, low-skilled jobs in the service industry (e.g., hospitality, security, entertainment) as a supplemental source of income. A Workforce Scheduler is tasked with scheduling L on-call shifts for a given day within a planning horizon of H hours among M casual employees who

prefer flexible schedules. These shifts are typically scheduled only 1 to 3 days in advance. The M employees have seniority, and all are eligible for all L shifts. The manager initiates an electronic system that sends notifications or phone calls to employees at discrete periods within the scheduling horizon. An employee can see the available set of shifts and select a shift when he or she receives a notification or system call. From now on, we will refer to these actions as calls.

Upon receiving a call, employees are free to take their time before responding to the system and accepting any available shifts not selected by other employees. Sometimes employees choose not to respond, indicating that they are not available to work that day. The time between the system's communication and the employee's response is called the *response delay*. Employees can choose any unassigned shift. The system also allows senior employees to select shifts that are available but occupied by junior employees, provided they respond within a specified time interval known as the *cutoff*, denoted by D. This action results in a schedule change, with the previously scheduled employee considered *bumped*. This happens when a junior employee responds earlier than the senior. The cutoff time starts when an employee is first called. If an employee responds after the cutoff time, he or she can only choose from the remaining unselected shifts and loses the advantage of seniority. Bumped employees are notified and can choose from the remaining available shifts at the time of bumping. The scheduling process is complete when all employees have responded or when the planning horizon is reached.

The uncertainty within the system is a result of employee shift preferences and delays in their responses. Tracking preferences is especially difficult. The preferences are expected to change regularly depending on their personal lives. In addition, since most of the employees are casual workers, they are constantly joining and leaving their positions. Thus, predicting shift preferences is challenging given the inherent variability among individuals and the non-stationary nature of these factors. Our industry partner, Merinio, which operates the system, also does not explicitly collect employee preferences. It would be inconvenient to ask the employees to rank their shifts daily. Therefore, to test our methodology for this new problem, we proceed with the assumption of identical preferences. This can even be seen as a worst-case analysis of the problem [3]. For example, suppose there are 5 employees and they all respond in reverse order of seniority. In that case, there will be a total of 10 (= 1+2+3+4) bumps where a bumped employee will bump other employees due to the same preferences. Note that a bumped employee is allotted the next shift in the preference order, potentially causing more bumps. The response delays are uncertain, thus calling too many employees early in the horizon can lead to multiple bumps if some senior employees take more time to decide. Conversely, calling too few employees to avoid bumps may not allow enough time to get all responses within the planning horizon, potentially resulting in unassigned shifts. Frequent schedule changes, or bumps, can lead to frustration among junior employees and may cause them to leave their jobs. Consequently, the system faces the challenge of determining the optimal number of calls to make at any given time, given the current state of

the system. This problem, formally known as the dynamic employee call timing problem (DECTP), revolves around determining the best times to initiate employee calls.

2.1 Deterministic Formulation

We introduce the deterministic/offline formulation of the DECTP, under specific assumptions and simplifications. The response delays r_i for each employee $i \in \mathcal{E}$ are assumed to be known at the start of the planning process. Here, $\mathcal{E} = \{1, \ldots, N\}$ represents the entire employee set. Employee response delays denoted as r_i, are considered independent and identically distributed (i.i.d.) random variables, following a shared probability distribution denoted as $\mathbb{P}$. All employees are eligible for all available shifts. Given the inherent complexity of the problem and lack of explicit data, incorporating stochastic (data-mined) individual preferences would significantly complicate the associated MIP when the problem is known to be NP-hard. Hence we also assume that all employees possess identical preferences, known at the commencement of planning. Consequently, we operate assuming that shifts are ranked based on a unique criterion such as monetary benefit. This also implies that when responding, an employee i will consistently have the opportunity to select a shift ranked at least i in their preferences. Due to operational constraints, a maximum of W simultaneous calls can be dispatched anytime. Table 1 provides the definitions for the notations used in the problem.

Table 1. Model, Parameters, and Variables of the DECTP

Sets	
$\mathcal{E}$	set of employees in the order of seniority with 1 being most senior
Parameters	
H	planning horizon
M	total number of available employees
L	total number of available shifts
D	bump cutoff
W	maximum number of calls sent per time epoch
$R = [r_i]_{i \in \mathcal{E}}$	set of response delays for employee i
δ_{ij}	response delay difference $r_i - r_j$ between two employees $i, j \in \mathcal{E}$
G	reward for filling a shift
Variables	
s_i	time of system call to employee $i \in \mathcal{E}$
y_{ij}	binary variable to indicate if employee $i \in \mathcal{E}$ bumps $j \in \mathcal{E}$
z_i	binary variable equal to 1 if employee $i \in \mathcal{E}$ responds within the horizon else 0
θ	number of shifts vacant

We use the MIP formulation of [3] for the offline DECTP, given by the equations (1)–(10). This formulation only decides when to call employees. It does not explicitly assign shifts to employees. One must post-process the solution obtained to know the shifts assigned to the employees. The objective in (1)

minimizes the shift vacancies and the total bumps. We set G to a very high value to guarantee complete shift assignment in the objective. The constraint in (2) ensures that employee seniority is respected when making calls. (3) sets z_i to 1 if an employee answers before the horizon, while (4) sets z_i to 0 if an employee answers after the horizon. (5) is a Big-M constraint that tracks a bump for any pair of senior and junior employees. For a senior employee i to be eligible to bump a junior employee j, the response delay for i must satisfy $r_i \leq D$ and $r_i \geq r_j$. The optimization ensures that there are at most W calls at each decision epoch, as per (6). Furthermore, (7) counts the number of vacant shifts. A solution to the presented formulation yields a call schedule $S = [s_i]_{i \in \mathcal{E}}$.

$$(MIP_{DECTP-O}) \quad \min \quad G\theta + \sum_{i \in \mathcal{E}} \sum_{j \in \mathcal{E}: i<j} y_{ij} \tag{1}$$

Constraints

$$s_i \leq s_j \qquad \forall i, j \in \mathcal{E}, i < j \tag{2}$$

$$s_i + r_i \geq (H+1)(1 - z_i) \qquad \forall i \in \mathcal{E} \tag{3}$$

$$s_i + r_i \leq H + r_i(1 - z_i) \qquad \forall i \in \mathcal{E} \tag{4}$$

$$s_i - s_j + \delta_{ij} \leq \delta_{ij} y_{ij} + (H + r_i)(1 - z_i) \qquad \forall i, j \in \mathcal{E}, i < j, r_i \leq D, r_i \geq r_j \tag{5}$$

$$s_i + 1 \leq s_{i+W} \qquad \forall i \in \mathcal{E}, i + W \in \mathcal{E} \tag{6}$$

$$\sum_{i \in \mathcal{E}} z_i + \theta \geq L \tag{7}$$

$$y_{ij} \in \{0, 1\} \qquad \forall i, j \in \mathcal{E}, i < j \tag{8}$$

$$s_i \in \mathbb{R}^+ \qquad \forall i, \in \mathcal{E} \tag{9}$$

$$z_i \in \{0, 1\} \qquad \forall i \in \mathcal{E} \tag{10}$$

2.2 Sequential Decision Model

We now introduce an event-based sequential decision model for the DECTP. In this model, the agent's main decision is how long to wait before making the next call. An event is therefore triggered if this waiting time is exhausted or an employee responds to the system. However, the events are considered to occur at discrete periods in the planning period.

Decision Epochs: Let $\mathcal{K} = \{0, 1, \ldots, K\}$ be the set containing all decision moments within the planning horizon H for some list of shifts denoted by L.

State Variables: $X_k = (t_k, x_k^n, x_k^s, x_k^r)$ gives the system's state. Here x_k^n is the number of calls made until epoch k, and x_k^s is a list of indicators for each employee with a value of 1 if that employee has responded and 0 otherwise. $x_k^r = [x_k^{ri}]_{i \in \mathcal{E}}$ keeps track of the bump cutoff and is used to know if an employee can bump others or not. t_k refers to the time elapsed at epoch k.

Decision Variable: For every decision epoch $k \in \mathcal{K}$, the policy decides the time to wait $a_k \geq 0$ before calling the next employee in the seniority order given the state X_k.

Exogenous Information: In the DECTP, the callback from employees represents the exogenous information. Let $W_k = \{\bar{t}_k, \bar{x}^s_k\}$ represent it. $\bar{t}_k$ is the next time of the next decision epoch. $\bar{x}^s_k$ which is an array of indicators for each employee with a value of 1 if an employee made a callback in that epoch. Multiple employees may call back at the same point due to discretization.
Transition Function: The transition function, $X^M(.)$ describes the transition from state X_k to X_{k+1} such that $X_{k+1} = X^M(X_k, a_k, W_k)$. This function captures the evolution of the system state based on the decisions made, the current state, and the exogenous information received. More specifically,

- $t_{k+1} = \bar{t}_k$.
- $x^n_{k+1} = x^n_k + \mathbb{1}_{\{a_k=0\}}$.
- $x^s_{k+1} = x^s_k + \bar{x}^s_k$.
- $x^r_{k+1} = [x^{ri}_{k+1}]_{i\in\mathcal{E}}$ such that

$$x^{ri}_{k+1} = \begin{cases} 0, & \text{if } \bar{x}^{si}_k = 1, \\ D-1, & \text{if } x^n_k < i < x^n_k + \mathbb{1}_{\{a_k=0\}}, \\ \max(x^{ri}_k - (t_{k+1} - t_k), 0), & \text{else.} \end{cases}$$

The objective we want to optimize is defined by equation 1.

3 Methodology

In practical scenarios, people commonly reach out to a few employees and wait for their responses before communicating with others. This sequential process often leads to delays. However, there is an opportunity to improve efficiency by proactively initiating additional calls based on expected responses from previously contacted employees. This proactive approach can save time in the long run. Anticipatory actions present challenges for humans due to inherent uncertainty, variability, and limited information, making it difficult to predict future states. Analyzing complex probability distributions and considering numerous interacting factors may overwhelm human cognitive capabilities. Hence, there is a need for ML. The problem setting for the DECTP is characterized by its parameters, including the number of employees M, the number of shifts to be scheduled L, the planning period H, the distribution of employee response delay $\mathbb{P}$ and the allowed response cutoff time D. An optimal offline calling schedule can be derived using the DECTP formulation ($MIP_{DECTP-O}$) for any instance within a given setting.

Our approach involves using a learned model to predict employee responses indirectly and strategically placing calls early. To accomplish this, we are seeking an expert agent who acts perfectly in a given instance. However, since our underlying problem is stochastic, seeking this expert is expensive. We propose utilizing offline solutions with perfect future knowledge to guide the learning instead. The aim is to identify patterns between system states and an intelligently constructed target action based on offline solutions. An ML model is intended to

capture and learn these patterns effectively. Once trained, the model can seamlessly integrate into real-time decision-making scenarios, offering a streamlined and efficient approach to online decision-making.

3.1 Constructing training examples and training policies - DAgger

[14] introduced the DAgger algorithm that employs an iterative policy training approach by adapting online learning principles that require access to an expert. In each iteration, the primary classifier undergoes retraining using all previously encountered states by the learner. This allows the learner to rectify past errors at each iteration. Initially, the first policy, π_0, is comprehensively instructed by the expert or is at least a good heuristic policy. Subsequently, π_0 is executed and the configurations visited by the learner are observed. A new dataset is compiled containing insights into rectifying the errors made by π_0 using the expert. To ensure information from both π_0 and the subsequent policy, π_1, is incorporated, π_1 is trained using a combination of the initial expert-only trajectories and the newly generated trajectories.

The issue with directly using DAgger for dynamic sequential decision problems in operations research is that getting an expert is highly expensive even for offline training. We rely on a pseudo-expert constructed using offline solutions to replace the actual expert. This pseudo-expert is still expensive to compute in real-time but is viable for offline training. We derive our main idea for the pseudo-expert from the OCP policy in [3]. This policy is derived by first solving several instances offline using ($MIP_{DECTP-O}$). Next, the optimal solutions, the number of calls to be made at each epoch are aggregated using an aggregator function. The aggregator functions used are simple functions like mean, and percentiles. We run DAgger algorithms for each of the aggregator functions and the best-performing one is chosen among them. The algorithm used to generate trajectories and train models is given by Algorithm 1.

During each iteration, we generate I sets of K-step trajectories. For each decision epoch within a trajectory governed by a policy π, we additionally create n contextual scenarios based on the current system state. This state encompasses information on all employees contacted by the system and those who have selected shifts. Consequently, for employees who have chosen shifts, we possess precise data regarding their response delays. For those yet to respond to calls, we have information on the time elapsed since they were contacted. Each scenario generated is tailored to this contextual data.

Each of the n scenarios is then solved offline using ($MIP_{DECTP-O}$) to obtain a vector of optimal actions $\bar{A}_k$ for a given decision epoch. Note ($MIP_{DECTP-O}$) gives the optimal number of calls at each epoch. However, with this information, one can then easily calculate the optimal wait time between calls. This effectively provides us with n distinct experts for each scenario. Subsequently, an aggregator function is applied to this vector to define a singular target $\hat{a}_k$. The state variable, denoted by X_k, encapsulates comprehensive information. Our objective is to distill this wealth of data into a fixed set of relevant features $\mathcal{F}$, elaborated upon in the results section. For each decision epoch within the planning horizon, a

training example is constructed, represented as $(F_k, \hat{a}_k)$, where F_k signifies the input features and $\hat{a}_k$ denotes the label indicating the expert action at epoch k. Note that we skip the iteration subscript used in the algorithm to denote the feature vector and the target. Ultimately, the best policy π_i is selected based on a set of validation instances.

Algorithm 1: DAgger algorithm to generate training examples and train policy

Initialize $\mathcal{D} \leftarrow \phi, \pi_0 \leftarrow OCP$
for $i \in \{0, \ldots, itr\}$ **do**
  for $j \in \{0, \ldots, I\}$ **do**
    Sample K-step trajectories using π_i ;
    for $k \in \{1, \ldots, K\}$ **do**
      Generate n scenarios ;
      Get optimal actions for each scenario $\bar{A}_k^{ij} = [\bar{a}_{kl}^{ij}]_{l \in \{1,\ldots,n\}}$ using $MIP_{DECTP-O}$;
      Get the input feature vector F_k^{ij};
      Get target action $\bar{a}_k^{ij} = Agg(\bar{a}_k^{ij})$;
    end
    Collect $\mathcal{D}_{ij} = \{(F_k^{ij}, \hat{a}_k^{ij})\}$ dataset of visited states and target actions ;
    $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_{ij}$;
  end
  Train policy π_{i+1} on $\mathcal{D}$;
end

3.2 Using the ML model

The trained model is integrated into the prediction-based algorithm to facilitate the call decision-making process. A decision epoch occurs when an employee responds to the system or the wait time is exhausted. At each decision epoch, an input feature vector is constructed based on the current state of the system. This vector is then input into the trained model, which outputs the recommended wait time at that epoch. If the wait time predicted is 0 then an employee i, next in the seniority order has to be called. We immediately then run the same model to get the wait time of employee $i + 1$ and so on. This incorporation of the ML model allows for dynamically determining the optimal number of calls in real-time based on the evolving system state.

4 Results

All experiments and models were implemented in Python, and GUROBI 10.0 was employed to solve the offline optimization problem instances. A maximum

solving time of 4 minutes was set for each instance. GUROBI consistently demonstrated efficient performance, often finding the optimal solution within seconds for all instances. In rare cases, it required additional time to confirm optimality.

4.1 Data and Scenarios

In all conducted experiments, a consistent planning horizon $H = 6$ hours and a fixed number $L = 50$ of available shifts were employed. A group of $M = 150$ employees, all possessing identical preferences, were available to fill these shifts. We consider a cutoff of 3 hours. The problem setting is defined by $\mathcal{I} = \{N, M, L, \mathbb{P}, D\}$ for all our experiments. We set $G = 200$, that is the company prioritises assigning all shifts. This value is considerably higher than the average bumps by a single employee in the optimal offline solutions. The approach is tested on real-world data from our industrial partner, collected from 2018-2020. We cleaned the data and censored response delays of more than 1000 minutes in the horizon. Response delays were then sampled for each employee from this dataset to generate instances. Our experiments consider scheduling personnel over a single day of operation with a six-hour planning period (H). The time horizon is discretized into 1-minute intervals. Figure 1 shows the CDF plot for the response delay distributions. Around 20% of the employees respond within the first 5 minutes, while only 50% of them eventually call back. Thus majority of the employees who respond, do so quickly.

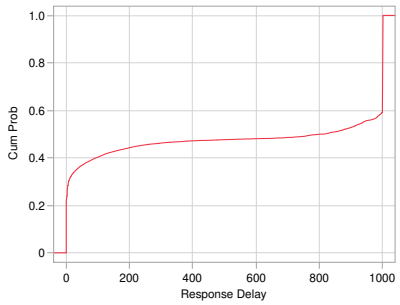


Fig. 1. Real World data Response Delay

A simulation model was developed to assess different policies. The real-world response delay data was divided into train and test sets in a ratio of 80:20. The train set was utilized to select the best-performing models, and then the test set was used to assess performance. We also simulate the currently employed policy by our industrial partner, a linear call-and-wait (CAW) policy that calls a fixed number of employees $\eta = 5$ every $\tau = 1$ time unit. Additionally, we attempted to improve this heuristic policy (TCAW) through brute force search on its parameters, simulating and testing all possible combinations. Furthermore, a

call-all (CA) policy was implemented, which calls all employees at the beginning as soon as possible, respecting the maximum call per decision epoch constraint. This policy was included to emphasize its adverse impact on bumps. Finally, we also compare our results with OCP from [3]. The OCP is a heuristic policy based on the aggregation of pure offline solutions. This policy will make calls if the current cumulative calls do not meet a dynamic threshold obtained from offline solutions for each decision epoch. Figure 2 shows the calling profile of the OCP across time. After making a few calls at the start the calling rate slows down and is then later accelerated when the end of the planning period is close.

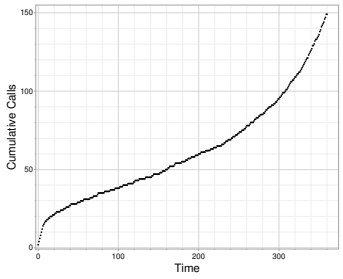


Fig. 2. Calling profile

For the DAgger algorithm, we employ gradient-boosted decision trees (GBDT) with XGBoost to build a learning model. Mean and percentile are used as aggregator functions. The percentile functions utilized are spaced 5 units apart, namely $\{45, 50, 55\}$. Such policies are referred to as percentile DAgger policies. Our regression model predicts the waiting time before making the next call as a continuous value, which is then rounded down to the nearest integer to obtain a discrete number.

The features utilized in our models are detailed in Table 2. While most features are self-explanatory, additional detail is provided for some. The *cumulative leftover cutoff* represents the sum of the remaining cutoff time for all employees. Next, to obtain the *delay buckets*, we first divide the interval $[0, D]$ into six intervals to define delay buckets. For example, when $D = 180$, intervals such as $[(0, 10), (10, 30), (30, 60), (60, 120), (120, 150), (150, 180)]$ are used to create six buckets. For any interval (l, u), the feature captures the count of employees whose current delay usage falls between l and u.

OCP cumulative calls represent the fixed number of employees that would have been called by the OCP policy until the current epoch. Meanwhile, the *to call now* feature represents the number of employees waiting to be called at the current epoch. This scenario occurs when the model suggests calling an employee and immediately receives the decision for the next employee in the seniority order. We enforce a restriction of not calling more than 2 employees, similar to the OCP policy.

Table 2. Set of features used to make predictions

Features	
time remaining (tr)	time since since last call made (tslc)
shifts vacant (sv)	cumulative employees called (cc)
ratio of time remaining and shifts vacant (rtbysv)	cumulative leftover cutoff (cl)
action at last epoch (la)	delay buckets (db_j)
OCP cumulative calls (cc_ocp)	number of response at last epoch (lr)
to call now (tc)	

4.2 Results across aggregator functions

Table 4.2 displays the average cost across 500 test instances obtained for different policies. Algorithm 1 is executed with various aggregator functions, gathering different operating policies while utilizing $n = 9$ scenarios to define the target. The cost function remains consistent with the objective defined in $MIP_{DECTP-O}$. The primary observation indicates that the DAgger policy DT_{45} slightly outperforms the OCP in terms of cost. While, DT_{50}, exhibits nearly identical performance in cost compared to the OCP.

However, DT_{μ} significantly underperforms compared to DAgger policies. This observation is noteworthy, indicating that an average offline decision derived from a set of contextual scenarios is notably inadequate for the stochastic problem. Both the current policy CAW and the CA policy demonstrate considerably higher costs. Even a tuned version (TCAW) of the current policy exhibits notably weaker performance than the percentile DAgger policies.

Figure 3 illustrates the bumps and vacancies for each policy. It is noticeable that as the number of bumps increases within the various policies, the number of vacancies decreases. The current heuristic (CAW) and call-all (CA) policies perform poorly in terms of bumps to ensure a complete shift allocation. Compared to the OCP, DT_{45} results in fewer vacancies, which may be of greater interest to decision-makers.

It's also important to highlight the substantial value of stochastic information in this problem. Solving 1000 instances offline under complete information, where all uncertainty is known at the start of the planning period, yielded an average of about 5.5 bumps with 0 average vacancies. Thus, if all response delays are known at the start, nearly all bumps can be eliminated, and all shifts can be scheduled.

Table 3. Cost for different policies

	Policies							
	CA	**CAW**	**TCAW**	**OCP**	$\boldsymbol{DT_{\mu}}$	$\boldsymbol{DT_{45}}$	$\boldsymbol{DT_{50}}$	$\boldsymbol{DT_{55}}$
Cost	668.8	418.0	122.9	103.7	143.06	**99.48**	103.4	114.8

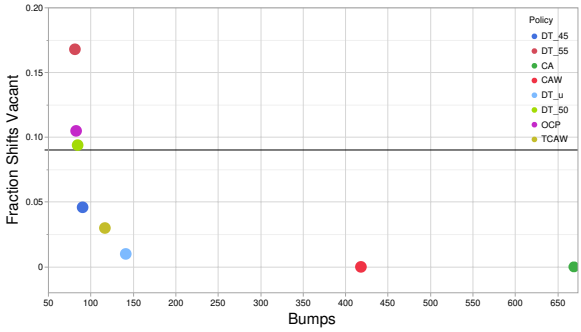


Fig. 3. Bumps vs Vacancy for all policies

4.3 Effect of number of scenarios

Next, we also study the effect of the number of scenarios (n) on the learning. We try 3 different settings for $n = \{1, 5, 9\}$ and present the results obtained from the best policies. We see that the performance in five scenarios is not as bad as compared to 9 scenarios. However, having only 1 scenario means there may be vacant shifts. As noted in the previous subsection, the value of stochastic information is high, and as a result having 1 scenario completely fails in ensuring sufficient shift occupancy. The results show that the greater the number of scenarios, the better the performance in terms of cost.

Table 4. Cost, Bumps, and Vacancy across different scenarios

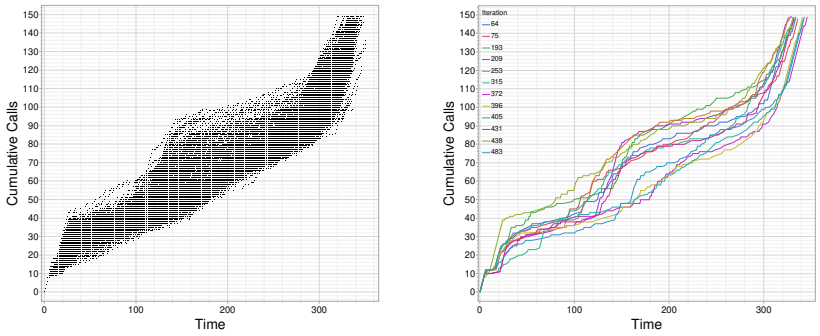
	Scenarios		
	1	**5**	**9**
Cost	275.9	119.5	99.4
Bumps	65.9	106.5	90.2
Vacancy	1.05	0.065	0.046

4.4 Calling Profiles

Figure 4 illustrates the distribution of cumulative calls and calling profiles when vacancies for DT_{45} occur. Figure 4a presents the distribution of the cumulative number of calls made at each discrete point in time. Upon comparison with Figure 2, it becomes evident that the distributions are generally similar. The distribution appears tightly packed, indicating the absence of divergent trajectories.

Furthermore, Figure 4b displays the calling profile across 12 episodes featuring empty shifts out of the 500 test scenarios. The trajectory reveals that the

model indeed attempted to call all 150 employees. However, due to inadequate responses within the allotted time, some shifts remained vacant.



(a) Distribution of cumulative calls for 500 simulations

(b) Call profile over 12 simulation runs with empty shifts

Fig. 4. Call profiles over the horizon for DT_{45}

4.5 Feature importance

The graph in figure 5 displays the feature importance in the model. The y-axis denotes the feature names, with lower positions indicating greater influence on the output. The analysis reveals that features such as time remaining, cumulative leftover cutoff delay, the ratio of time remaining to shift vacant, and the first two delay buffers are the most important features.

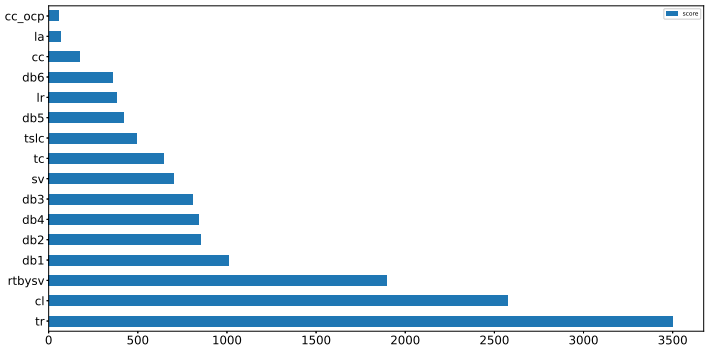


Fig. 5. Feature Importance

5 Conclusion

This paper introduces an imitation-based learning approach that uses the DAgger framework to address the Employee Call Timing problem for casual employees, a sequential decision-making challenge aimed at minimizing schedule changes (referred to as bumps) while ensuring sufficient staffing for all shifts. Our learning-based approach is designed to imitate an aggregated target obtained from offline solutions and dynamically make decisions based on the current system state. The methodology is trained and evaluated using real-world data from our industrial partner. The prediction-based approach shows reduced average bumps without significantly increasing shift vacancies compared to the existing heuristic practice. Our findings reveal that learned models perform almost as well as compared to OCP, a tuned heuristic based on offline solutions. Further analysis has been also shown regarding the type of calling profiles obtained. This approach seems promising and it seems that with more careful tuning of ML models may be able to beat the heuristic OCP.

References

1. Vinyals, O. Meire, F., Jaitly, N.: Pointer networks. Advances in neural information processing systems **28**, (2015)
2. Pham, T. S., Legrain, A., De Causmaecker, P., Rousseau, L-M.: A prediction-based approach for online dynamic appointment scheduling: A case study in radiotherapy treatment. INFORMS Journal on Computing. **35**(4), 844–868 (2023)
3. Gawas, P. S., Legrain, A., Rousseau, L-M.: Personnel Scheduling with flexibility for On-Call Shifts. arXiv preprint arXiv:2312.06139. (2023)
4. Larsen, E., Lachapelle, S., Bengio, Y., Lacoste-Julien, S., Lodi, A.: Predicting Tactical Solutions to Operational Planning Problems Under Imperfect Information. INFORMS Journal on Computing. **34**(1), 227–242 (2021)
5. Ferreira, K. J., Lee, B. H. A., Simchi-Levi, D.: Analytics for an online retailer: Demand forecasting and price optimization. Manufacturing & service operations management. **18**(1), 69–88 (2015)
6. Chan, C., Farias, V., Bambos, N., Escobar, G.: Optimizing intensive care unit discharge decisions with patient readmissions. Operations research. **60**(6), 1323–1341 (2012)
7. Bengio, Y., Lodi, A., Prouvost, A.: Machine learning for combinatorial optimization: a methodological tour d'horizon. European Journal of Operational Research. **290**(2), 405–421 (2021)
8. Donti, P., Amos, B., Kolter, J. Z.: Task-based end-to-end model learning in stochastic optimization. Advances in neural information processing systems. **30**, (2017)
9. Kotary, J., Fioretto, F., Van Hentenryck, P., Wilder, B.: End-to-End Constrained Optimization Learning: A Survey. In: Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21, pp. 4475–4482 (2021).
10. Elmachtoub, A., Grigas, P.: Smart "predict, then optimize". Management Science. **68**(1), 9–26 (2022)
11. Chu, H., Zhang, W., Bai, P., Chen, Y.: Data-driven optimization for last-mile delivery. Complex & Intelligent Systems. 1–14, (2021)

12. Hussein, A., Gaber, M., Elyan, E., Jayne, C.: Imitation learning: A survey of learning methods. ACM Computing Surveys. **50**(2), 1–34, (2017)
13. Joshi, C., Laurent, T., Bresson, X.: An efficient graph convolutional network technique for the travelling salesman problem. arXiv preprint arXiv:1906.01227. (2019)
14. Ross, S., Gordon, G., Bagnell, D.: A reduction of imitation learning and structured prediction to no-regret online learning. In: Proceedings of Machine Learning Research. **9**, 661–668 (2010)

Conditional Importance Resampling for an Enhanced Sequential Monte Carlo Sampler

Soodeh Habibi[1], Efthyvoulos Drousiotis[1], Alessandro Varsi[1], Simon Maskell[1], Robert Moore[1], and Paul G, Spirakis[2]

[1] Department of Electrical Engineering and Electronics, University of Liverpool, Liverpool L69 3GJ, UK; {S.Habibi, E.Drousiotis, A.Varsi, S.Maskell}@liverpool.ac.uk

[2] Department of Computer Science, University of Liverpool, Liverpool L69 3BX, UK P.Spirakis@liverpool.ac.uk

Abstract. Sequential Monte Carlo (SMC) samplers are a family of powerful Bayesian inference methods that combine sampling and resampling to sample from challenging posterior distributions. This makes SMC widely used in several application domains of statistics and Machine Learning. The aim of this paper is to introduce a new resampling framework, called Conditional Importance Resampling (CIR) that reduces the quantization error arising in the application of traditional resampling schemes. To assess the impact of this approach, we conduct a comparative study between two SMC samplers, differing solely in their resampling schemes: one utilizing systematic resampling and the other employing CIR. The overall improvement is demonstrated by theoretical results and numerical experiments for sampling a forest of Bayesian Decision Trees, focusing on its application in classification and regression tasks.

1 Introduction

1.1 Motivation

Collecting and computing random samples from a posterior distribution is a common challenge in Bayesian statistics. Sequential Monte Carlo (SMC) is one of the most commonly used algorithms to tackle this issue, given its state-of-the-art performance. Indeed, SMC samplers are an attractive choice to be used across various domains, ranging from Bayesian Optimization [21], Machine Learning [15], Target Tracking [16], to Image Processing [19]. The overall idea is to employ Importance Sampling (IS) [6] to generate a population of N weighted samples, and then use the resampling algorithm to address the degeneracy that inevitably arises from repeated use of IS. This combination of IS and resampling is indeed quite flexible. However, despite addressing the samples' degeneracy effectively, the resampling step also introduces a quantization error to the weights. This error poses a significant issue as it increases the variability of the resampled set of samples. Therefore, we argue that an alternative resampling scheme that reduces this quantization error could significantly produce better estimates.

1.2 Related Work

The original idea of using resampling as a solution to the degeneracy of the samples in an SMC method was proposed in [5], and presents a resampling scheme called multinomial resampling. The objective of resampling is to replace the current population of N weighted samples with a new degeneracy-free population of N samples by removing the samples that have low weights and replacing them with copies of the samples with high weights (i.e. the healthy samples), in such a way that the estimates after resampling remain unbiased.

Several alternative resampling schemes have been proposed since the original multinomial resampling in [5]. Some of the most commonly used schemes are stratified resampling [3, 8], and systematic resampling [1, 8]. These algorithms achieve $O(N)$ time complexity, use a multinomial distribution, and then divide the population into strata or intervals, and use one or more random variables to draw samples from these intervals. Another popular method is residual resampling, presented in [12] and improved in [11]; this approach also takes $O(N)$ steps and separately considers the integer and decimal parts of the weights multiplied by N to decide how many times each sample is to be copied.

In [2], another resampling scheme based on the knapsack problem is presented. While this alternative scheme offers improved accuracy compared to systematic resampling, it does so at the price of a much greater time complexity (up to $O(N^2)$, depending on the size of the knapsack). This complexity appears to have limited the adoption of this resampling algorithm and we do not therefore consider it further herein.

The resampling algorithms that have $O(N)$ complexity have been compared in several studies [7, 9, 10], and the consistent conclusion is that systematic resampling is to be preferred for its ability to maximize accuracy and reduce the variance of the estimation. Note that systematic resampling can be performed in parallel, achieving up to $O(\log_2 N)$ parallel time complexity on both shared [13, 14, 18, 24] and distributed memory architectures [20, 22, 23].

Whichever resampling strategy is chosen, the weight of each member of the new degeneracy-free population of samples will always be reset to $\frac{1}{N}$. This introduces a quantization error with respect to the samples' weights before resampling, which affects the quality of the estimation produced by the SMC sampler.

1.3 Contribution and Paper Outline

In this paper, we present Systematic-Conditional Importance Resampling (S-CIR), an alternative $O(N)$ resampling scheme for the SMC sampler. We first prove theoretically that S-CIR achieves a lower quantization error compared to systematic resampling. We then compare two versions of the same SMC sampler, one using S-CIR and one using systematic resampling, in the context of sampling Bayesian Decision Trees (DTs). The experimental results on a wide range of publicly available datasets for supervised learning tasks show that by designing a more sophisticated resampling scheme, our approach can significantly improve performance.

The rest of the paper is therefore organized as follows: Section 2 introduces the SMC sampler algorithm. Section 3 first provides a high-level description of any resampling scheme and then gives a brief overview of systematic resampling. Section 4 describes our proposed resampling scheme in detail and proves its validity. Section 5 presents the numerical results on both classification and regression problems. Section 6 concludes and discusses possible future improvements.

2 Sequential Monte Carlo Samplers

SMC samplers are Bayesian inference algorithms that generate N random samples from a posterior distribution of interest proportional to $\pi(\mathbf{X}, \mathbf{Y})$, i.e., the product of the prior and the likelihood. Here, we briefly describe the algorithmic details of SMC samplers. The reader is referred to [17] for further details.

The key idea behind SMC samplers is to use a combination of IS and resampling for a total of K iterations, $\forall k = 0, 1, \ldots, K-1$. Each iteration generates or updates, $\mathbf{x}_k$, a population of N samples. The $i-$th sample, $\mathbf{x}_k^i$ $\forall i = 0, 1, \ldots, N-1$, is also weighted by an importance weight, $\mathbf{w}_k^i \in \mathbb{R}$, which represents how well the sample is approximating the posterior distribution.

In order to initialize the samples at $k = 0$, the SMC sampler draws each sample $\mathbf{x}_0^i$, from an arbitrary initial proposal distribution (or, simply, initial proposal, the terms are used interchangeably), $q_0(\cdot)$, from which it is easy to sample. Then, each importance weight is initialized by computing the ratio between the posterior and the initial proposal computed in each sample. More precisely, the samples and weights are initialized as follows:

$$\mathbf{x}_0^i \sim q_0(\cdot), \qquad \forall i = 0, 1, 2, \ldots, N-1, \tag{1a}$$

$$\mathbf{w}_0^i = \frac{\pi\left(\mathbf{x}_0^i, \mathbf{Y}\right)}{q_0\left(\mathbf{x}_0^i\right)}, \qquad \forall i = 0, 1, 2, \ldots, N-1. \tag{1b}$$

After the initialization step, each sample is updated by sampling from another arbitrary proposal distribution (or, simply, proposal, the terms are used interchangeably), $q(\cdot|\mathbf{x}_{k-1}^i)$, from which it is also easy to sample. Each weight is updated given the previous weight. More precisely the samples and weights are updated as follows:

$$\mathbf{x}_k^i \sim q(\cdot|\mathbf{x}_{k-1}^i), \qquad \forall i = 0, 1, 2, \ldots, N-1, \tag{2a}$$

$$\mathbf{w}_k^i = \mathbf{w}_{k-1}^i \frac{\pi\left(\mathbf{x}_k^i, \mathbf{Y}\right)}{\pi\left(\mathbf{x}_{k-1}^i, \mathbf{Y}\right)} \frac{q\left(\mathbf{x}_{k-1}^i|\mathbf{x}_k^i\right)}{q\left(\mathbf{x}_k^i|\mathbf{x}_{k-1}^i\right)}, \qquad \forall i = 0, 1, 2, \ldots, N-1. \tag{2b}$$

The validity of (2) is proven in [17] and is omitted here for brevity.

After (2) each weight is normalized as follows:

$$\tilde{\mathbf{w}}_k^i = \frac{\mathbf{w}_k^i}{\sum_{j=0}^{N-1} \mathbf{w}_k^j}, \quad \forall i = 0, 1, 2, \ldots, N-1, \tag{3}$$

such that $\sum \tilde{\mathbf{w}}_k^i = 1$, i.e., 100% of the total probability. It is then possible to estimate an expectation of interest as a weighted sum across the samples:

$$\mathbb{E}[f(\mathbf{x})] = \int \pi(\mathbf{x}|\mathbf{Y}) f(\mathbf{x}) d\mathbf{x} \approx \sum_{i=0}^{N-1} \tilde{\mathbf{w}}_k^i f\left(\mathbf{x}_k^i\right) \tag{4}$$

In general, the approach described up to this point provides useful estimates. However, since the samples are generated from the proposal and not directly from the posterior distribution of interest, the samples are subject to a numerical error, called degeneracy. Degeneracy makes the variance of the normalized weights diverge. More precisely, if not corrected, degeneracy makes all normalized weights but one tends to 0, while a single sample's normalized weight will converge to 1. For obvious reasons, in this scenario, it is impossible to generate meaningful weighted estimates of the posterior from the population of samples. To correct degeneracy, the typical approach is to use a resampling algorithm. In the next sections, we will provide details about systematic resampling, and our proposed approach, called Conditional Importance Resampling (CIR). However, any resampling scheme deletes the samples with low weights and replaces these samples with copies of the samples with higher weights. Then, the weights are reset to $1/N$. In SMC samplers, resampling is not used in every iteration, but only if needed. More precisely, the SMC sampler first measures the variance of the normalized weights by computing the Effective Sample Size (ESS), as follows:

$$N_{eff} = \frac{1}{\sum_{i=0}^{N-1} \left(\tilde{\mathbf{w}}_k^i\right)^2}. \tag{5}$$

Resampling is typically used if the ESS drops below an arbitrary threshold, typically set to $N/2$. Then the SMC samplers repeats steps (2), (3), (5), and resampling for K iterations. After K iterations, we can produce estimates based on the final samples using (4).

3 Conventional Resampling Algorithms

In this section, we first give a general description of the strategy of any resampling algorithm in Section 3.1. Then Section 3.2 provides details about systematic resampling, the baseline method considered in the results presented in section 5.

3.1 General Description

The general idea behind a conventional resampling algorithm consists of performing three steps: choice; redistribution; reset.

Step 1 - Choice. This step is utilized to infer the number of times each sample needs to be replicated. To do this, we sample indices N times from a distribution, $g_{\tilde{\mathbf{w}}}(\cdot)$ (a multinomial distribution), parameterized by the normalized weights, $\tilde{\mathbf{w}}$.

This sampling procedure will generate a vector, $\mathbf{ncopies} \in \mathbb{N}^N$, such that

$$\sum_{i=0}^{N-1} \mathbf{ncopies}^i = N. \tag{6}$$

Therefore, the i-th entry, $\mathbf{ncopies}^i$, indicates how many times the i-th sample, $\mathbf{x}^i$, is required to be replicated, meaning that those samples for which $\mathbf{ncopies}^i = 0$ will be eliminated.

Step 2 - Redistribution. This step replicates each $\mathbf{x}^i$ as many times as $\mathbf{ncopies}^i$, such that we get the new population of samples, $\mathbf{x}_{new}$. This operation is described

Algorithm 1 Redistribution

Input: $\mathbf{x}$, $\mathbf{ncopies}$
Output: $\mathbf{x}_{new}$, R

```
N ←size(ncopies)
i ← 0
for j ∈ {0, 1, 2, ..., N − 1} do
    for copy ∈ {0, 1, 2, ..., ncopies^j − 1} do
        x^i_new ← x^j
        R^i ← j
        i ← i + 1
    end for
end for
```

by Algorithm 1. For example, if we have:

$$\mathbf{ncopies} = [0, 0, 2, 0, 1, 4, 0, 1]$$

and we want to redistribute an array of samples, $\mathbf{x}$, defined as:

$$\mathbf{x} = [\mathbf{x}^0, \mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3, \mathbf{x}^4, \mathbf{x}^5, \mathbf{x}^6, \mathbf{x}^7],$$

then

$$\mathbf{x}_{new} = [\mathbf{x}^2, \mathbf{x}^2, \mathbf{x}^4, \mathbf{x}^5, \mathbf{x}^5, \mathbf{x}^5, \mathbf{x}^5, \mathbf{x}^7],$$

and

$$R = [2, 2, 4, 5, 5, 5, 5, 7], \tag{7}$$

where R is the vector of indices of the samples that have been replicated. This vector will simplify the notation in Section 4.

Step 3 - Reset. In the final step, as mentioned in Section 2, the weights of the new samples are reset to $1/N$.

3.2 Systematic Resampling

Systematic resampling uses a statistically efficient mechanism for sampling N times from a multinomial distribution to find **ncopies**. This is achieved by first

sampling a uniform random variable:

$$u \sim \texttt{Uniform}[0, 1), \tag{8}$$

the algorithm then computes the cumulative density function of $N\tilde{\mathbf{w}}$, $\mathbf{cdf} \in \mathbb{R}^N$, as follows:

$$\mathbf{cdf}^i = N \sum_{j=0}^{i} \tilde{\mathbf{w}}^j, \quad \forall i = 0, 1, 2, \ldots, N-1, \tag{9}$$

such that each particle, $\mathbf{x}^i$, is copied as many times as:

$$\mathbf{ncopies}^i = \lceil \mathbf{cdf}^i + \tilde{\mathbf{w}}^i - u \rceil - \lceil \mathbf{cdf}^i - u \rceil, \tag{10}$$

where $\lceil \cdot \rceil$ is the ceiling function. The steps described by (8) (9) and (10) describe a sampling procedure from a multinomial distribution $g_{\tilde{\mathbf{w}}}(\cdot)$, parameterized by the normalized weights, $\tilde{\mathbf{w}}$. After that, as previously mentioned, each sample, $\mathbf{x}^i$ is replicated $\mathbf{ncopies}^i$ times, and the weights are all reset to $1/N$. These steps are summarized in Algorithm 2.

After systematic resampling, unbiased estimates of the posterior can be calculated as follows:

$$\sum_{i=0}^{N-1} f\left(\mathbf{x}^i\right) \tilde{\mathbf{w}}^i \approx \frac{1}{N} \sum_{i=0}^{N-1} f\left(\mathbf{x}^i_{new}\right). \tag{11}$$

Algorithm 2 Systematic Resampling

Input: $\mathbf{x}$, $\tilde{\mathbf{w}}$, N

Output: $\mathbf{x}_{new}$, $\mathbf{w}_{new}$

1: $\mathbf{ncopies} \sim g_{\tilde{\mathbf{w}}}(\cdot)$, See Equations (8), (9), and (10)
2: $\mathbf{x}_{new}, R \leftarrow$`Redistribution`($\mathbf{x}$, $\mathbf{ncopies}$), See Algorithm 1
3: $\mathbf{w}^i_{new} \leftarrow \frac{1}{N}\ \forall i = 0, 1, 2, \ldots, N-1$

4 Proposed Approach

In this section, we describe our approach and prove its validity and its improvement over the other conventional resampling approaches.

4.1 Conditional Importance Resampling

The approach we propose aims to reduce the quantization error that systematic resampling introduces when all samples' importance weights are reset to $1/N$.

Definition 1 (Quantization error). *Let* $\mathbf{x}$ *be a population of* N *samples, where each sample,* $\mathbf{x}^i$, $\forall i = 0, 1, 2, \ldots, N-1$, *is weighted by a normalized weight,* $\tilde{\mathbf{w}}^i \in \mathbb{R}$. *Let* $\mathbf{x}_{new}$ *and* $\tilde{\mathbf{w}}_{new}$ *be the new population of* N *samples, and the new*

weights generated by any resampling scheme, which replicates $\mathbf{x}$ *according to a vector,* $\mathbf{ncopies} \in \mathbb{N}^N$*, for which* (6) *holds. For each weight in* $\tilde{\mathbf{w}}$*, the quantization error,* e_q^i*, is computed as follows:*

$$e_q^i = \begin{cases} 0, & \text{if } \mathbf{ncopies}^i = 0, \\ |\tilde{\mathbf{w}}^i - \sum_{\mathbf{x}_{new}^j = \mathbf{x}^i} \tilde{\mathbf{w}}_{new}^j|^2, & \text{if } \mathbf{ncopies}^i > 0, \end{cases} \tag{12}$$

where the term $\sum_{\mathbf{x}_{new}^j = \mathbf{x}^i} \tilde{\mathbf{w}}_{new}^j$ *is the total weight across all the copies of* $\mathbf{x}^i$ *being generated by resampling (and, for notational convenience, we assume that the proposal is such that each value of* $\mathbf{x}^i$ *is unique).*

For example, if a sample $\mathbf{x}^i$ has a weight such that $N\tilde{\mathbf{w}}^i = 3.1$, systematic resampling will generate either 3 or 4 copies of $\mathbf{x}^i$ and do so such that the expected number of copies (when averaging across multiple realizations of the algorithm's operation) is 3.1. However, in any single realization of the algorithm, the total weight of all copies of $\mathbf{x}^i$ would be quantized to either $3 \times \frac{1}{N}$, or $4 \times \frac{1}{N}$, and will never be equal to $3.1 \times \frac{1}{N}$. This quantization error will affect the final estimate and its variance. Furthermore, this quantization error will only be zero if the weights are all multiples of $\frac{1}{N}$. Our approach to reducing this quantization error is described in the text that follows.

The overall idea is to split the normalized weights (and the corresponding samples) into two subsets, one containing small weights and another containing big weights. We will also identify one of the big weights that we will divide into two parts, a small weight and a weight that is still a big weight. In so doing, we will ensure that the sum of the small weights is equal to an integer multiple of $\frac{1}{N}$. We will then use systematic resampling over the samples with small weights. After that, we will use a novel resampling scheme over the samples with big weights, which is designed such that it ensures that no quantization error is introduced for this subset of samples. By doing this resampling, the total quantization error over the full population of samples will be no larger than systematic resampling alone. We now describe this process in more detail.

Step 1 - Preliminaries. We identify $\tilde{\mathbf{w}}_p$, which is non-zero for a subset of the normalized weights, such that:

$$\tilde{\mathbf{w}}_p^i = \begin{cases} \tilde{\mathbf{w}}^i \text{ If } \tilde{\mathbf{w}}^i < \frac{1}{N}, \\ 0 \quad \text{If } \tilde{\mathbf{w}}^i \geq \frac{1}{N}. \end{cases} \tag{13}$$

We refer to this vector as the vector of preliminary small weights.

Step 2 - Divide. As anticipated above, we want to perform systematic resampling over the preliminary small weights in (13). However, given that systematic resampling will result in a total weight that is a multiple of $\frac{1}{N}$, we define the notion of the (not preliminary) small weights to be such that the total of the small weights is a multiple of $\frac{1}{N}$. To achieve this, we compute, $\alpha \in [\frac{0}{N}, \frac{1}{N}]$, the fraction such that $\sum_{i=0}^{N-1} \tilde{\mathbf{w}}_p^i + \alpha$ is a multiple of $\frac{1}{N}$: α is then an additional weight that we seek to identify by dividing one of the weights that is not a preliminary small weight. To identify the weight to divide, we look for an index j, such that

$\tilde{\mathbf{w}}^j > \frac{2}{N}$. This weight can be divided into a weight that is equal to α and a weight that is greater than or equal to $\frac{1}{N}$. We then define the small weights as:

$$\tilde{\mathbf{w}}_s^i = \begin{cases} \tilde{\mathbf{w}}^i \text{ If } \tilde{\mathbf{w}}^i < \frac{1}{N}, \\ \alpha \quad \text{If } i = j, \\ 0 \quad \text{If } \tilde{\mathbf{w}}^i \geq \frac{1}{N} \text{ and } i \neq j, \end{cases} \tag{14}$$

and the big weights as

$$\tilde{\mathbf{w}}_b^i = \begin{cases} 0 & \text{If } \tilde{\mathbf{w}}^i < \frac{1}{N}, \\ \tilde{\mathbf{w}}^j - \alpha & \text{If } i = j, \\ \tilde{\mathbf{w}}^i & \text{If } \tilde{\mathbf{w}}^i \geq \frac{1}{N} \text{ and } i \neq j. \end{cases} \tag{15}$$

We note that if no weight higher than $\frac{2}{N}$ is found, we use systematic resampling over the full input population of samples.

Step 3 - Systematic Resampling. We perform systematic resampling to resample $N_s \leq N$ samples from the samples with non-zero small weights, $\tilde{\mathbf{w}}_s$, where N_s is computed as follows:

$$N_s = N \sum_{i=0}^{N-1} \tilde{\mathbf{w}}_s^i. \tag{16}$$

We note that the weights, $\tilde{\mathbf{w}}_s$, input to systematic resampling need to be re-normalized over the set of non-zero small weights. This step will generate N_s samples, $\mathbf{x}_{s,new}$. We reset each weight, $\tilde{\mathbf{w}}_{s,new}^i$, to $1/N$.

Step 4 - Conditional Importance Resampling. In this step, we want to resample $N - N_s$ copies of the samples with non-zero big weights, $\tilde{\mathbf{w}}_b$, without contributing any quantization error in the context of the big weights. To achieve this, we use a combination of two multinomial distributions to decide how many times we replicate each sample with a big weight. We now explain this process in more detail.

Step 4.1 - First Multinomial. We first re-normalize the weights $\tilde{\mathbf{w}}_b$, such that each $\hat{\mathbf{w}}_b^i = \frac{\tilde{\mathbf{w}}_b^i}{\sum_{i'=0}^{N-1} \tilde{\mathbf{w}}_b^{i'}}$. Then, we use systematic resampling to draw a random vector $\theta \in \mathbb{N}^{N-N_s}$ comprising the empirical distribution of $N - N_s$ draws from a multinomial distribution, $g_{\hat{\mathbf{w}}_b}(\theta)$, parameterized by the normalized big weights, $\hat{\mathbf{w}}_b$. Therefore, we can compute $\mathbf{q} = \frac{\theta}{N-N_s} \in \mathbb{R}^N$, which represents the (total) quantized normalized weights, i.e., the weights we would have if we had just used systematic resampling to resample using $\hat{\mathbf{w}}_b$. We note that all these quantized weights will, by design, be non-zero and (equivalently) that $N - N_s$ is greater than or equal to the number of big weights (with the equality occurring in the unusual circumstance that all the big weights have a weight of exactly $\frac{1}{N}$).

Step 4.2 - Second Multinomial. We then use systematic resampling again to get a new vector, **ncopies**, representing the number of copies of the samples we will generate from the samples with big weights. However, this second resampling process will be parameterized by $\mathbf{q}$, such that $\mathbf{ncopies} \sim g_{\mathbf{q}}(\cdot)$. Note that, since every weight parameterizing $\mathbf{q}$ is a multiple of $\frac{1}{N}$, systematic resampling will

only ever produce one output given $\mathbf{q}$, i.e. the realization of u in (8) has no effect on **ncopies**.

Step 4.3 - Redistribution. We replicate each sample corresponding to a non-zero $\tilde{\mathbf{w}}_b^i$ as many times as $\mathbf{ncopies}^i$ by using Algorithm 1, such that we generate a new set of $N - N_s$ samples $\mathbf{x}_{b,new}$.

Step 4.4 - Reset. In this step, we reset the big weights. To simplify the exposition, we re-introduce R^i as the index in the original population of samples corresponding to the $i-$th member of $\mathbf{x}_{b,new}$. We then reset each new weight to:

$$\tilde{\mathbf{w}}_{b,new}^i = \tilde{\mathbf{r}}^i \times \frac{N - N_s}{N}, \quad i = 0, 1, 2, \ldots, N - N_s - 1. \tag{17}$$

where each $\tilde{\mathbf{r}}^i$ is a normalized version of $\mathbf{r}^i$ which is itself defined based on IS (since we know we sampled from $\mathbf{q}$ and not $\hat{\mathbf{w}}_b$) as:

$$\mathbf{r}^i = \frac{\hat{\mathbf{w}}_b^{R^i}}{\mathbf{q}^{R^i}} = \frac{\hat{\mathbf{w}}_b^{R^i} \times (N - N_s)}{\theta^{R^i}}, \quad \forall i = 0, 1, 2, \ldots, N - N_s - 1. \tag{18}$$

In Lemma 1, we show that (17) will allow us to preserve the original weights $\mathbf{w}_b$ in the expectation of the estimate, leading to no quantization error for the $N - N_s$ big weights as shown in Theorem 1.

Step 5 - Merge. Finally, we merge the outputs from *Step 3* and *Step 4*, such that: $\tilde{\mathbf{w}}_{new} = \{\tilde{\mathbf{w}}_{s,new}, \tilde{\mathbf{w}}_{b,new}\}$, and $\mathbf{x}_{new} = \{\mathbf{x}_{s,new}, \mathbf{x}_{b,new}\}$.

The steps above are summarized in Algorithm 3 which, as previously mentioned in the introduction, we name Systematic-Conditional Importance Resampling (S-CIR). To be valid, Algorithm 3 must be unbiased. The next lemma and theorem prove not only that our approach is unbiased but also that it has a variance caused by quantization that is never greater than that introduced by systematic resampling. We note that S-CIR achieves $O(N)$, as it consists of two steps (one for $\mathbf{w}_s$ and one for $\mathbf{w}_b$) each of which performs $O(N)$ operations.

Lemma 1. *Let* $\mathbf{x}$ *be a population of* N *samples, and let* $\tilde{\mathbf{w}} \in \mathbb{R}^N$ *be the set of the normalized importance weights associated with the samples. By using Algorithm 3, it is possible to generate a new population of* N *samples such that the estimate remains unbiased.*

Proof. To prove that S-CIR is unbiased it is enough to show that the expectation of estimated value after resampling is the same as the estimated value before. Indeed, we seek to prove that:

$$\sum_{i=0}^{N-1} f(\mathbf{x}^i)\tilde{\mathbf{w}}^i = \mathbb{E}\left[\sum_{i=0}^{N-1} f(\mathbf{x}_{new}^i)\tilde{\mathbf{w}}_{new}^i\right]. \tag{19}$$

Given that Algorithm 3 first splits the weights into $\tilde{\mathbf{w}}_b$ and $\tilde{\mathbf{w}}_s$, we can start by expressing the estimate before computing Algorithm 3 as follows:

$$\sum_{i=0}^{N-1} f\left(\mathbf{x}^i\right)\tilde{\mathbf{w}}^i = \underbrace{\sum_{j=0}^{N-1} f\left(\mathbf{x}^j\right)\tilde{\mathbf{w}}_s^j}_{(I)} + \underbrace{\sum_{n=0}^{N-1} f\left(\mathbf{x}^n\right)\tilde{\mathbf{w}}_b^n}_{(II)}. \tag{20}$$

Algorithm 3 Systematic Conditional Importance Resampling

Input: $\mathbf{x}$, $\tilde{\mathbf{w}}$, N

Output: $\mathbf{x}_{new}$, $\tilde{\mathbf{w}}_{new}$

for $i \in \{0, 1, 2, \ldots, N-1\}$ **do**

  $\tilde{\mathbf{w}}_p^i \leftarrow \begin{cases} \tilde{\mathbf{w}}^i \text{ If } \tilde{\mathbf{w}}^i < \frac{1}{N} \\ 0 \quad \text{If } \tilde{\mathbf{w}}^i \geq \frac{1}{N} \end{cases}$

end for

Find (any) j s.t. $\tilde{\mathbf{w}}^j > \frac{2}{N}$

if j is not found **then**

  $\mathbf{x}_{new}, \tilde{\mathbf{w}}_{new} \leftarrow$`Systematic Resampling`$(\mathbf{x}, \tilde{\mathbf{w}}, N)$, See Algorithm 2

  return

end if

$\alpha \leftarrow \lceil \sum_{i=0}^{N-1} \tilde{\mathbf{w}}_s^i \rceil - \sum_{i=0}^{N-1} \tilde{\mathbf{w}}_s^i$

for $i \in \{0, 1, 2, \ldots, N-1\}$ **do**

  $\tilde{\mathbf{w}}_s^i \leftarrow \begin{cases} \tilde{\mathbf{w}}^i \text{ If } \tilde{\mathbf{w}}^i < \frac{1}{N} \\ \alpha \quad \text{If } i = j \\ 0 \quad \text{If } \tilde{\mathbf{w}}^i \geq \frac{1}{N} \text{ and } i \neq j \end{cases}$

  $\tilde{\mathbf{w}}_b^i \leftarrow \begin{cases} 0 & \text{If } \tilde{\mathbf{w}}^i < \frac{1}{N} \\ \tilde{\mathbf{w}}^j - \alpha & \text{If } i = j \\ \tilde{\mathbf{w}}^i & \text{If } \tilde{\mathbf{w}}^i \geq \frac{1}{N} \text{ and } i \neq j \end{cases}$

end for

$N_s \leftarrow N \sum_{i=0}^{N-1} \tilde{\mathbf{w}}_s^i$

for $i \in \{0, 1, 2, \ldots, N-1\}$ **do**

  $\hat{\mathbf{w}}_s^i \leftarrow \frac{\tilde{\mathbf{w}}_s^i}{\sum_{j=0}^{N-1} \tilde{\mathbf{w}}_s^j}$

end for

$\mathbf{x}_{s,new}, \hat{\mathbf{w}}_{s,new} \leftarrow$`Systematic Resampling`$(\mathbf{x}, \hat{\mathbf{w}}_s, N_s)$, See Algorithm 2

for $i \in \{0, 1, 2, \ldots, N_s-1\}$ **do**

  $\tilde{\mathbf{w}}_{s,new}^i \leftarrow \frac{1}{N}$

end for

for $i \in \{0, 1, 2, \ldots, N-1\}$ **do**

  $\hat{\mathbf{w}}_b^i \leftarrow \frac{\tilde{\mathbf{w}}_b^i}{\sum_{j=0}^{N-1} \tilde{\mathbf{w}}_b^j}$

end for

$\theta \sim g_{\hat{\mathbf{w}}_b}(\cdot)$, See Equations (8), (9), and (10)

for $i \in \{0, 1, 2, \ldots, N-1\}$ **do**

  $\mathbf{q}^i \leftarrow \frac{\theta^i}{N - N_s}$

end for

ncopies $\sim g_{\mathbf{q}}(\cdot)$, See Equations (8), (9), and (10)

$\mathbf{x}_{b,new}, R \leftarrow$`Redistribution`$(\mathbf{x}$, **ncopies**$)$, See Algorithm 1

for $i \in \{0, 1, 2, \ldots, N - N_s - 1\}$ **do**

  $\mathbf{r}^i \leftarrow \frac{\hat{\mathbf{w}}_b^{R^i}}{\mathbf{q}^{R^i}}$,

  $\tilde{\mathbf{r}}^i \leftarrow \frac{\mathbf{r}^i}{\sum_{j=0}^{N-N_s-1} \mathbf{r}^j}$,

  $\tilde{\mathbf{w}}_{b,new}^i \leftarrow \tilde{\mathbf{r}}^i \times \frac{N-N_s}{N}$,

end for

$\mathbf{x}_{new} \leftarrow \{\mathbf{x}_{s,new}, \mathbf{x}_{b,new}\}$

$\tilde{\mathbf{w}}_{new} \leftarrow \{\tilde{\mathbf{w}}_{s,new}, \tilde{\mathbf{w}}_{b,new}\}$

Since we adopt two resampling schemes, one for the non-zero values of $\tilde{\mathbf{w}}_s$, and one for the non-zero values of $\tilde{\mathbf{w}}_b$, we split the rest of the proof into two parts. We use systematic resampling for the non-zero values of $\tilde{\mathbf{w}}_s$, such that the first term (I) of the right hand-side in (20) is:

$$\sum_{j=0}^{N-1} f\left(\mathbf{x}_s^j\right) \tilde{\mathbf{w}}_s^j = \frac{N_s}{N} \sum_{j=0}^{N-1} f\left(\mathbf{x}_s^j\right) \hat{\mathbf{w}}_s^j = \frac{N_s}{N} \mathbb{E}\left[\frac{1}{N_s} \sum_{i=0}^{N_s-1} f(\mathbf{x}_{new}^i)\right], \tag{21}$$

where the fraction immediately preceding the expectation is the result of the sum of the small weights being less than unity. The proof of the remainder of (21) is provided in [5], and here is omitted for brevity.

For the term (II) in the right hand-side of (20), we consider IS conditional on the value of θ such that we pose an estimate of a function of $\mathbf{x}$, by averaging over samples of θ, as follows:

$$\begin{aligned}\sum_{n=0}^{N-1} f\left(\mathbf{x}^n\right) \tilde{\mathbf{w}}_b^n &= \frac{N - N_S}{N} \int \mathrm{g}_{\hat{\mathbf{w}}_b}(\theta) \sum_{n=0}^{N-1} f\left(\mathbf{x}^n\right) \hat{\mathbf{w}}_b^n d\theta \\ &= \frac{N - N_S}{N} \mathbb{E}\left[\frac{1}{(N - N_s)} \sum_{n=0}^{N-1} \theta^n f\left(\mathbf{x}^n\right) \frac{\hat{\mathbf{w}}_b^n}{\theta^n/(N - N_s)}\right]. \end{aligned} \tag{22}$$

Note that, by splitting the weights into small and big weights, we have been able to only apply IS in contexts where the proposal, $\mathbf{q}$, is non-zero irrespective of the value of θ. This ensures that using IS results in an estimator with finite variance, which cannot be ensured if IS is also applied to remove the quantization error of the small weights, as their corresponding θ might be 0.

After *Step 4 - Importance Resampling*, the new samples, $\mathbf{x}_{b,new}$, will be resampled from $\mathbf{x}$, such that the expectation of the estimate in (22) would result in the following equality:

$$\sum_{n=0}^{N-1} f\left(\mathbf{x}^n\right) \tilde{\mathbf{w}}_b^n = \frac{N - N_s}{N} \mathbb{E}\left[\frac{1}{(N - N_s)} \sum_{j=0}^{N-N_s-1} f\left(\mathbf{x}^j\right) \frac{\hat{\mathbf{w}}_b^j}{\theta^j/(N - N_s)}\right]. \tag{23}$$

Since for any random variables X_1, X_2, we have $\mathbb{E}[X_1 + X_2] = \mathbb{E}[X_1] + \mathbb{E}[X_2]$, (21) and (23) prove (19), i.e., that Algorithm 3 is unbiased. Also, (23) shows that the total of the weights, $\tilde{\mathbf{w}}_b^i$, for each big weight (in the original population) is preserved by the resampling.

Theorem 1. *Let* $\mathbf{x}$ *be a population of* N *samples, which we want to resample according to normalized importance weights,* $\tilde{\mathbf{w}} \in \mathbb{R}^N$*. The quantization error introduced if using S-CIR (Algorithm 3) can never be worse than the quantization error introduced if using systematic resampling (Algorithm 2).*

Proof. For obvious reasons, the quantization error for S-CIR associated with the small weights, $\tilde{\mathbf{w}}_s$, is identical to the quantization error for systematic resampling, as $\tilde{\mathbf{w}}_{s,new}^i = \frac{1}{N}$, $\forall i = 0, 1, 2, \ldots, N_s - 1$, as is the case in systematic resampling.

Lemma 1 shows that the big weights before resampling are preserved after resampling, meaning that the quantization error for these weights is 0.

Therefore, since $N_s \leq N$, we have:

$$\sum_{i=0}^{N-1} e^i_{q,s-cir} \leq \sum_{i=0}^{N-1} e^i_{q,sr}, \tag{24}$$

where $e^i_{q,s-cir}$ and $e^i_{q,sr}$ are, respectively, the quantization errors on the i-th weight when using S-CIR and systematic resampling. Hence, the two errors, $e^i_{q,s-cir}$ and $e^i_{q,sr}$, can only be equal when $N_s = N$ or when there is no weight such that $\tilde{\mathbf{w}}^j > \frac{2}{N}$: in these cases S-CIR and systematic resampling are identical.

5 Numerical Results

In this section, we compare two SMC samplers differing only in the resampling scheme being used; one SMC sampler uses systematic resampling and the other SMC sampler uses S-CIR. The results are provided for an exemplary discrete model, which is designed to sample a forest of N Bayesian Decision Trees (DTs) for supervised learning tasks (classification and regression). To provide context, we first briefly describe SMC DTs, and then we provide and discuss the results.

5.1 Model Description

Bayesian DTs represent a powerful framework for probabilistic modeling and decision-making. The main characteristic of Bayesian DTs is the incorporation of uncertainty by assigning probability distributions to both the observed data and model parameters, allowing for a more sophisticated and accurate representation of the uncertainty compared to deterministic DTs. The nodes of the tree specify the features used for the decision made at the node, and the edges represent possible outcomes, with conditional probability distributions associated with each datum then deduced by descending the tree. For this paper, we consider both classification and regression problems. For more information and specifications about the model we use, the reader is referred to [4].

5.2 Results

As described in Section 4, our main objective is to show that the S-CIR reduces the quantization error that systematic resampling introduces when all the resampled weights are set to $1/N$. To demonstrate the accuracy improvement of the S-CIR compared to systematic resampling in supervised learning problems, we conducted experiments on eight publicly available datasets (five datasets for classifications and three for regression) from the UCI Machine Learning Repository[3]: an overview of the datasets is provided in Table 1.

We have run the contesting algorithms for 10 MC runs, each time for 10 iterations sampling $N = 150$ DTs per iteration for all datasets and reported the

[3] *https://archive.ics.uci.edu/*

Name	Type	Number of Records	Feature Count	Feature Types
Glass	C	214	9	R
Balance Scale	C	625	4	C
Liver Disorders	C	345	5	C,I,R
Arrhythmia	C	452	279	C,I
Heart	C	303	13	C,I
Forest fires	R	517	12	R
Synchronous Machines	R	557	5	R
Student Marks	R	145	31	I

Table 1: Datasets considered, including type ((C)lassification or (R)egression) and feature types ((R)eal-valued, (I)nteger and (C)ategorical).

accuracy (and associated runtime) for the output of iteration $K = 10$. We have calculated classification accuracy for classification datasets and Mean Squared Error (MSE) for regression datasets. Experiments have shown that 10 iterations are enough for the algorithm to converge and produce acceptable predictions. The numerical results are shown in Table 2a for classification, and in Table 2b for regression, and represent averages over the MC runs.

The improvement in classification accuracy that results from using S-CIR ranges from 0.23% (for Arrhythmia) to 2.03% (for Balance Scale) with an average improvement of 1.27%. In the context of the regression tasks, the percentage reductions in MSE resulting from using S-CIR are 4.4%, 1.4%, and 2.1% for the Forest Fires, Synchronous Machines, and Students Marks datasets, respectively: the average percentage improvement in MSE is 2.6%. We also note that the run-

(a) Classification Results

Dataset	**Method**	**Accuracy**	**Time**
Glass	**S-CIR**	63.23%	10.63s
	SR	62.30%	10.27s
Balance Scale	**S-CIR**	74.68%	4.88s
	SR	72.65%	4.58s
Liver Disorder	**S-CIR**	62.01%	6.05s
	SR	60.57%	5.08s
Arrhythmia	**S-CIR**	59.62%	15.56s
	SR	59.39%	15.55s
Heart	**S-CIR**	75.43%	4.97s
	SR	73.70%	4.66s

(b) Regression Results

Dataset	**Method**	**MSE**	**Time**
Forest Fires	**S-CIR**	5440.94	26.82s
	SR	5694.34	20.16s
Synchronous Machines	**S-CIR**	19486.85	8.37s
	SR	19764.61	8.01s
Student Marks	**S-CIR**	8.67	2.76s
	SR	8.86	2.60s

Table 2: Results for Supervised Learning Tasks. In the tables, SR stands for (S)ystematic (R)sampling. The algorithms are run on a Intel Core i7-10875H.

time of both methods is roughly the same, as S-CIR and systematic resampling take $O(N)$ steps. To sum up, the experiments have shown that implementing a more sophisticated resampling scheme helps improve the overall performance of SMC when sampling DTs for all classification and regression tasks considered.

6 Conclusion

In conclusion, this paper presents a novel resampling strategy, S-CIR, applicable to all SMC methods. The proposed approach reduces the quantization error relative to the pre-existing state-of-the-art, systematic resampling. We applied the novel resampling scheme to SMC DTs to show the performance improvement in real-world scenarios. Relative to state-of-the-art results on supervised learning tasks, SMC DTs' performance is significantly improved by using the novel resampling algorithm: Our improved resampling scheme has experimentally shown an average 1.27% improvement in classification accuracy problems and an average reduction in MSE of 2.6% in the context of regression problems. The implication of the theoretical and empirical results is that improvements are likely in the many other contexts where SMC methods are applied.

As well as quantifying this benefit in contexts beyond those considered herein, future work should include the exploration of alternative strategies for further reducing the errors introduced by resampling. Ideally, any such novel scheme should be synergistic to the novel resampling technique described herein.

References

1. M. Sanjeev Arulampalam, Simon Maskell, Neil J. Gordon, and Tim Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, 2002.
2. Ahmet Bacak and Ali Köksal Hocaoğlu. A novel resampling algorithm based on the knapsack problem. *Signal Processing*, 170:107436, 2020. doi:https://doi.org/10.1016/j.sigpro.2019.107436.
3. Arnaud Doucet, Adrian Smith, Nando de Freitas, and Neil J. Gordon. *Sequential Monte Carlo Methods in Practice*. Information Science and Statistics. Springer New York, 2001.
4. Efthyvoulos Drousiotis, Alessandro Varsi, Paul G. Spirakis, and Simon Maskell. A shared memory smc sampler for decision trees. In *2023 IEEE 35th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 209–218, 2023. doi:10.1109/SBAC-PAD59825.2023.00030.
5. Neil J. Gordon, David J. Salmond, and Adrian Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. *IEE Proc. F Radar Signal Process. UK*, 140(2):107, 1993.
6. Peter L Green and Simon Maskell. Parameter estimation from big data using a sequential monte carlo sampler. In *Proceedings of ISMA2016 International Conference On Noise and Vibration Engineering and USD2016 International Conference On Uncertainty In Structural Dynamics*, pages 4111–4119, 2016.
7. Jeroen D. Hol, Thomas B. Schon, and Fredrik Gustafsson. On resampling algorithms for particle filters. In *2006 IEEE Nonlinear Statistical Signal Processing Workshop*, pages 79–82, 2006.
8. Genshiro Kitagawa. Monte carlo filter and smoother for non-gaussian nonlinear state space models. *Journal of Computational and Graphical Statistics*, 5(1):1–25, 1996.
9. Chanin Kuptametee and Nattapol Aunsri. A review of resampling techniques in particle filtering framework. *Measurement*, 193:110836, 2022. doi:https://doi.org/10.1016/j.measurement.2022.110836.

10. Tiancheng Li, Miodrag Bolic, and Petar M. Djuric. Resampling methods for particle filtering: Classification, implementation, and strategies. *IEEE Signal Processing Magazine*, 32(3):70–86, 2015. doi:10.1109/MSP.2014.2330626.
11. Tiancheng Li, Tariq Pervez Sattar, and Shudong Sun. Deterministic resampling: Unbiased sampling to avoid sample impoverishment in particle filters. *Signal Processing*, 92(7):1637–1645, 2012. doi:https://doi.org/10.1016/j.sigpro.2011.12.019.
12. Jun S. Liu and Rong Chen. Sequential monte carlo methods for dynamic systems. *Journal of the American Statistical Association*, 93(443):1032–1044, 1998.
13. Felipe Lopez, Lixun Zhang, Joseph Beaman, and Aloysius Mok. Implementation of a particle filter on a gpu for nonlinear estimation in a manufacturing remelting process. In *2014 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pages 340–345, July 2014. doi:10.1109/AIM.2014.6878102.
14. Felipe Lopez, Lixun Zhang, Aloysius Mok, and Joseph Beaman. Particle filtering on gpu architectures for manufacturing applications. *Computers in Industry*, 71:116 – 127, 2015. doi:https://doi.org/10.1016/j.compind.2015.03.013.
15. Xiao Ma, Peter Karkus, David Hsu, and Wee Sun Lee. Particle filter recurrent neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5101–5108, 2020.
16. Lyudmila Mihaylova, Avishy Y Carmi, François Septier, Amadou Gning, Sze Kim Pang, and Simon Godsill. Overview of bayesian sequential monte carlo methods for group and extended object tracking. *Digital Signal Processing*, 25:1–16, 2014.
17. Pierre Del Moral, Arnaud Doucet, and Ajay Jasra. Sequential monte carlo samplers. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 68(3):411–436, 2006.
18. Lawrence M. Murray, Anthony Lee, and Pierre E. Jacob. Parallel resampling in the particle filter. *Journal of Computational and Graphical Statistics*, 25(3):789–805, 2016. doi:10.1080/10618600.2015.1062015.
19. Miroslav Radojević and Erik Meijering. Automated neuron reconstruction from 3d fluorescence microscopy images using sequential monte carlo estimation. *Neuroinformatics*, 17(3):423–442, 2019.
20. Conor Rosato, Alessandro Varsi, Joshua Murphy, and Simon Maskell. An O(log2N) SMC2 algorithm on distributed memory with an approx. optimal l-kernel. In *2023 IEEE Symposium Sensor Data Fusion and International Conference on Multisensor Fusion and Integration (SDF-MFI)*, pages 1–8, 2023. doi:10.1109/SDF-MFI59545.2023.10361452.
21. Andreas Svensson, Johan Dahlin, and Thomas B. Schön. In *2015 IEEE 6th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, pages 477–480, 2015. doi:10.1109/CAMSAP.2015.7383840.
22. Alessandro Varsi, Lykourgos Kekempanos, Jeyarajan Thiyagalingam, and Simon Maskell. Parallelising particle filters with deterministic runtime on distributed memory systems. In *IET 3rd International Conference on Intelligent Signal Processing (ISP 2017)*, pages 1–10, 2017. doi:10.1049/cp.2017.0357.
23. Alessandro Varsi, Simon Maskell, and Paul G Spirakis. An O(log2N) fully-balanced resampling algorithm for particle filters on distributed memory architectures. *Algorithms*, 14(12):342–362, 2021.
24. Alessandro Varsi, Jack Taylor, Lykourgos Kekempanos, Edward Pyzer Knapp, and Simon Maskell. A fast parallel particle filter for shared memory systems. *IEEE Signal Processing Letters*, 27:1570–1574, 2020. doi:10.1109/LSP.2020.3014035.

Applying Instance Space Analysis to Optimize the Construction of Matheuristics

Sophie Hildebrandt[1,2][0000−0001−9606−3695] and
Guido Sand[1][0009−0005−3162−4726]

[1] Pforzheim University of Applied Science, Germany
[2] University of Twente, Netherlands

Abstract. The combination of mathematical programming and heuristics is known as matheuristic. Constructing a matheuristic comprises two steps: First selecting its components, second parameterizing those components. The construction of a matheuristic is often done manually by experts, which leads to expensive engineering processes. To bring matheuristics into practice, their construction should be automated. In this paper a matheuristic construction approach is proposed to close this gap. The approach is based on instance space analysis for the selection and Bayesian optimization for the parametrization of the matheuristic. First computational studies show, how Bayesian optimization can be used to parameterize an original heuristic.

Keywords: Matheuristic Construction · Instance Space Analysis · Bayesian Optimization · Mixed Integer Programming.

1 Optimizing the Construction of Matheuristics

Electroplating plants give rise to hoist scheduling problems (HSPs), which are proven to be NP-complete. HSPs can be modelled as Mixed Integer Programs (MIPs). However, for real-world sized problems monolithic MIPs cannot be solved by standard solvers in reasonable time. A widely applied solution are original heuristics, which reduce the computational cost of solving HSPs by decomposing the monolithic into a polylithic model. The combination of mathematical programming with heuristics, e.g. metaheuristics or original heuristics, is known as matheuristic. In our research, we focus on matheuristics, that combine an MIP model with original heuristics and a standard solver. The individual components already exist in literature in several variants.

Every plant has individual characteristics resulting in a multitude of different HSPs. Selecting appropriate matheuristic components with their corresponding parametrization i.e. constructing a matheuristic, is a time-consuming task often done manually by experts. Usually, only one matheuristic is constructed and used for different HSPs. However, specific matheuristic component combinations with specific parametrizations i.e. matheuristic setups, may be more appropriate for individual HSPs. To bring matheuristics into practice an automated engineering process to adapt a matheuristic with low effort to new instances, is needed. [1]

The construction consists of two consecutive steps: First the selection of the matheuristic components (MIP model, original heuristic, and standard solver); second the parameterization of the selected original heuristic and the standard solver.

In the first step all hyperparameter are categorical. For example, possible parameter values for the component standard solver are e.g. CPLEX, Gurobi or Xpress. Therefore, the first step corresponds to an algorithm selection problem since no ordering of the possible values exists. Additionally, the parameters of the first step can depend on each other. For example, in hoist scheduling not all combinations of MIP models and original heuristics are feasible.

In the second step, the hyperparameters comprise the parameters of the selected standard solver and original heuristic. These parameters can be of different types (continuous, integer, ordinal, categorical). They can also depend on each other or be subject to constraints. While several methods for parametrizing standard solvers already exist, the parameters of original heuristics are usually adjusted manually or with grid search. In general, two conflicting objectives occur: First minimizing computational cost and second maximizing solution quality. The first objective corresponds to the time the standard solver needs to solve the MIP model. The second objective corresponds to the objective of the underlying optimization problem e.g. minimizing makespan. Both objectives are functions of the hyperparameters of the matheuristic. They can be evaluated pointwise but no prior knowledge about the objective functions like gradients can be obtained.

2 Algorithmic Setup

In the artificial intelligence (AI) domain various methods for the two steps of selecting and parametrizing algorithms were studied in the last years and applied e.g. to metaheuristics and neural networks. One successfully applied method for algorithm selection is Instance Space Analysis (ISA) [3], for algorithm parametrization Bayesian Optimization (BO) is successfully applied [4].

Since constructing matheuristics also comprises algorithm selection and parametrization problems, identical methods as used in the AI domain, may be applicable. To the best knowledge of the authors no application of ISA or BO to matheuristic construction have been investigated in literature.

2.1 Selecting the Matheuristic Components

For the first step of the matheuristic construction, namely the selection of components, ISA is appropriate since all hyperparameters are categorical. Different matheuristic components exploit different characteristics of the problem instances. Therefore, with ISA the performance of the matheuristic components is not evaluated on average over a set of instances but evaluated based on the instance characteristics. This approach is probably better with regard to the two objectives minimizing computational cost and maximizing solution quality.

2.2 Parameterizing the Matheuristic Components

After selecting the matheuristic components, the original heuristic must be parameterized. In contrast to grid or random search, BO efficiently samples the solution space and is considered as the state of the art for algorithm parametrization. As mentioned, the hyperparameters of the original heuristics can be of different types. However, BO in its original form is intended to parameterize solely continuous hyperparameters, but extensions of the BO, considering integer and categorical parameters, are studied as well. With reference to these studies the authors expect that the BO can be used to parameterize the original heuristics. Nevertheless, it is expected, that the BO takes too much time for real-world-sized problems, if the BO is performed from scratch for every new instance. Further research directions like transfer learning, multifidelity and parallel BO should be taken. [2]

3 Computational studies on Bayesian Optimization to parameterize Original Heuristic

In first computational studies the application of BO to the parametrization of an original heuristics at the second step are studied by Hildebrandt and Sand [2]. Different tools for BO exist: The python package SMAC3 was used, since it supports different acquisition functions and surrogate models and also supports different types of hyperparameters, as well as conditions and constraints on hyperparameters. The implementation is structured as follows: SMAC3 executes the BO by calling the matheuristic, comprising the original heuristic, MIP model and standard solver; The original heuristic manages the solution process of the MIP model implemented with Pyomo and calls the standard solver. The BO uses a Gaussian Process as surrogate model and the expected improvement as acquisition function. This setup is the standard configuration for BO. However, other BO setups should be investigated in further studies as well. The time limit and the number of evaluations of the BO are restricted. The BO terminates after 16 retries of searching new values of the hyperparameters. The computational studies involve three small HSPs. The original heuristic consists of a rolling horizon heuristic combined with an improvement heuristic. Three different integer-valued hyperparameters of the original heuristic are parameterized.

A complete enumeration of all possible parameter combinations serves as reference: Both objective values, computational cost and solution quality, are obtained. For the BO those objectives are combined into a single weighted sum using the inverse mean of the a posteriori calculated points from the complete enumeration, as weights.

For two HSPs the BO finds that it is better to solve the monolithic rather than the polylithic model. For the third HSP the BO finds the optimal configuration of the hyperparameters evaluating 25% of all points. Detailed information on the algorithmic setup can be found in [2].

4 Instance Space Analysis for selecting Bayesian Optimization Hyperparameters

The BO, which is used to parameterize the original heuristic at the lower level, also possesses hyperparameters itself. For example, the surrogate model, the acquisition function and the acquisition function optimizer must be selected. Besides using ISA to select the components of the matheuristic, it could also be used outside the matheuristic construction at a higher level to set up the BO.

Based on the studies on BO for the parameterization of original heuristics described in section 3, the application of ISA to BO will be studied. The original heuristic is parameterized using the following algorithmic setup: ISA is used to select the components of the BO for the instance at hand; BO parameterize the original heuristic for the instance at hand; the original heuristic manages the solution process and solves the underlying optimization problem using the MIP formulation and the standard solver.

Like for the BO, again two conflicting objectives must be handled: computational cost and solution quality. In real-world applications a limited amount of time is available. Therefor in a first approach, the time limit of the BO is also set to a predefined value. By doing so, the computational cost is handled as a constraint and the solution quality can be used as a single objective to evaluate the performance of a specific BO setup. To ensure that enough evaluations during one BO can be conducted, the time limit of the standard solver for solving the underlying optimization problem (here HSP) should be set according to the time limit of the BO. One possible solution is to define the number of evaluations i.e. runs of the matheuristic, used by the BO, and calculate the time limit for solving one HSP by dividing the time limit of the BO by the number of evaluations.

The research on how ISA and BO can be used to automate the matheuristic construction tend to shift the human interaction to a higher level. This would simplify the engineering of matheuristics for the automation vendor and therefore advance the application of matheuristics in practice.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Reimschüssel, S., Fuchs, U., Sand, G.: Electroplating scheduling: Closing a research gap from an automation vendor's perspective. ESCAPE 2023, CACE, vol. 52, pp. 125-130, Greece (2023). https://doi.org/10.1016/B978 -0-443-15274-0.50021-4
2. Hildebrandt, S., Sand, G.: Hyperparameter Optimization of Matheuristics for Hoist Scheduling. ESCAPE 2024, CACE, vol. 53, Italy (2024).
3. Smith-Miles, K., Muñoz, M.: Instance Space Analysis for Algorithm Testing: Methodology and Software Tools. ACM Computing Surveys, vol. 55, pp. 1-31, (2023). https://doi.org/10.1145/3572895
4. Victoria, A., Maragatham, G.: Automatic tuning of hyperparameters using Bayesian optimization. Evolving Systems, vol. 12, pp. 217–223, (2021). https://doi.org/10.1007/s12530-020-09345-2

WANCE: Learnt Clause Evaluation Method for SAT Solver Using Graph Structure

Yoichiro Iida[1], Tomohiro Sonobe[2], and Mary Inaba[1]

[1] Graduate School of Information Science and Technology, The University of Tokyo
yoichiro-iida@g.ecc.u-tokyo.ac.jp, mary@is.s.u-tokyo.ac.jp
[2] National Institute of Informatics, Japan
tomohiro_sonobe@nii.ac.jp

Abstract. Effective evaluation of learnt clauses is crucial for the performance of conflict-driven clause learning SAT solvers. We propose a novel learnt clause evaluation method using the information of the graph structure of SAT problem, called weight to adjacent node-based clause evaluation (WANCE). The graph representation of SAT problems is recognized as useful for understanding the structure of the problems, and it is known that the efficiency of SAT solvers is related to this structure. WANCE utilizes the problem structure of the input CNF, which can identify valuable clauses independently of their search, contrary to existing methods (e.g., literal block distance, LBD) reliant on a search tree state for their evaluation values. First, to reveal the relationship between learnt clauses and graph structure in more detail, we investigated the impact of a learnt clause on the graph structure through correlation analysis with its quality measured by LBD. Findings indicated that high-quality clauses had heavy edges to their adjacent variables on the graph, enhancing their propagation potential. Second, assuming that that feature implied clause quality, we implemented WANCE using the feature as the primary metric for clause management. The results of performance evaluation experiments demonstrated that a solver equipped with WANCE outperformed those using LBD-based methods. Our findings underscore the potential of utilizing the graph-structural properties of SAT problems for clause evaluation.

Keywords: SAT solver · Structure of SAT · Learnt clause

1 Introduction

Conflict-driven clause learning (CDCL) satisfiability (SAT) solvers are widely used to solve SAT problems owing to their efficacy in addressing industrial SAT problems derived from real-world problems. Clause learning is a key component of a CDCL solver, enabling it to learn from failures. These learning results are stored as learnt clauses. Modern solvers generate a vast number of learnt clauses; however, it is inefficient to retain all of them. Therefore, evaluation method to determine ones that should be deleted is important for the performance of solvers. "Activity" (or the clause version of a variable state independent decaying

sum) [23], one of the evaluation metrics, gauges how often a clause is used in the conflict and its resolution; Clauses with a higher frequency of use are evaluated as more valuable ones. "Literal block distance (LBD)" [5], widely used in many solvers owing to their high performance, measures the number of literal blocks in the learnt clause. LBD evaluates substantially shorter clauses by treating variables used in a decision and its associated propagation as a single variable (literal block).

We propose a new method for evaluating learnt clauses called weight-to-adjacent-node-based clause evaluation (WANCE), utilizing a graph-structural property inherent to the conjunctive normal form (CNF) of the input SAT problem. In existing methods (e.g., Activity and LBD), the evaluation value of a learnt clause varies depending on the state of the search. While this has the advantage of obtaining an optimized evaluation value for that search, the evaluation values may be inappropriate in environments with different search states (e.g., when a period of time has passed after the evaluation and the search environment has changed, or when clauses are shared between parallel searches where each worker employs a different search strategy). Our proposal method can identify valuable clauses independent of the search state and thus has the potential to perform better, at least under certain conditions. Application SAT problems derived from real-world problems have remarkable structural properties [1] compared to the randomly generated ones. The difference was observed by quantifying the structural properties such as centrality, treewidth, or modularity, which are usually on the variable incident graph (VIG). Nodes of VIG are variables, and edges connect variables in a clause with some designated weight. The relationship between the degree of the graph-structural properties and the runtime of the solver (performance) is also known [12]. In addition, several studies have addressed the relationship between the quality of clauses and structural properties of SAT problems. For example, the LBD value correlates to the number of communities to which variables in clause belongs [24], and clauses with low LBD values (thus valuable clauses) increase the modularity value of the graph, that is, these clauses reinforce the community structure of graph[14].

First, we conducted a correlation analysis to reveal the relationship between learnt clauses and graph structure in more detail. Given a learnt clause, we quantified the implication of a clause on the graph structure using properties such as the clause's connectivity, treewidth, and the weights of edges within variables. Then, we performed a correlation analysis between them and the quality of the clause measured by LBD. This analysis aimed to examine the implication of valuable clauses on the graph structure. Results indicated that valuable clauses tend to have heavy edges to their adjacent variables on the graph, indicating that they have higher probability to be used in propagation. Further, the experiments revealed that this metric predicts LBD's minimum value (updated value) more accurately than its maximum value. Therefore, using this property, we propose WANCE because we assume that this feature can intrinsically evaluate valuable clauses, and is computationally efficient among the structural features.

Next, we conducted performance evaluation experiments using this feature as the primary metric for clause evaluation. Experimental results verified the effectiveness of our proposal method in the clause deletion task in sequential solvers.

The contributions of this study are listed below:

1. Through correlation analysis, we extensively investigated the properties of learnt clauses in the graph that valuable clauses have.
2. We revealed that the quality of clauses measured by LBD relates to the weight of edge to adjacent variables and, thus, their propagation probability.
3. We demonstrated that the clause evaluation method utilizing the structure is effective in the task of clause deletion in sequential solvers.

The remainder of this paper is organized as follows: Section 2 introduces SAT solvers and their techniques. Section 3 presents our observations of the correlation analysis. Section 4 proposes the implementation of WANCE. Section 5 presents the results of performance evaluation. Section 6 discusses related work, and Section 7 summarizes the paper and suggests future research directions.

2 Preliminaries

2.1 SAT Problem and SAT Solver

The propositional SAT problem involves determining if a given propositional logic formula can be satisfied by a Boolean value assignment of variables. The formula is provided in conjunctive normal form (CNF), wherein variables are combined into clauses with disjunctions, and the clauses are combined with conjunctions. A formula is in CNF if it has the form $(C_1 \wedge C_2 \wedge \cdots \wedge C_m)$, where C_i represents a clause. Each clause is a disjunction of literals $(L_{i,1} \vee L_{i,2} \vee \cdots \vee L_{i,n})$, where $L_{i,j}$ represents a literal in clause C_i. An example of a CNF formula is $(x \vee y) \wedge (\neg y \vee z)$. Here, $(x \vee y)$ and $(\neg y \vee z)$ are clauses, and x, y, $\neg y$, and z are literals where y and $\neg y$ represent the positive and negative forms of variable y. If at least one valid Boolean assignment of variables satisfies all clauses, the formula is satisfiable (SAT); otherwise, it is unsatisfiable (UNSAT).

SAT solvers are programs used to solve SAT problems. SAT is a typical NP-complete problem; no polynomial-time algorithm is believed to solve all SAT instances. Despite this worst-case complexity, modern SAT solvers are remarkably efficient and can handle large and complex instances. Therefore, they are used to solve real-world problems such as computer-aided theorem proof [13] and binary neural network verification [8] encoded as SAT problems. Most modern SAT solvers are CDCL SAT solvers.

2.2 Clause learning

In SAT solvers, learning refers to deriving new clauses for the formula to avoid revisiting unsatisfiable assignments. The learnt clause, the learning output, is an

inference derived from the original SAT instance represented as a clause. The following is the process of SAT solver search: The solver performs a search by making an assumption that assigns a Boolean (True or False) value to a variable called a decision. Subsequently, the solver assigns the Boolean values of other variables as the logical consequence of the decision, called propagation. If the decision is incorrect, an inconsistency occurs during the propagation, referred to as conflict. Then, the solver analyzes the conflict's root cause and learns the cause's counter-examples as clauses (learnt clause) to avoid the same incorrect assumptions and conflicts in subsequent searches. Finally, the solver cancels the decision, which is called backtrack.

Recent SAT solvers obtain many learnt clauses, and it often learns over a million clauses. Maintaining such an excessive number of clauses degrades performance because of the increased propagation time for checking more clauses. Clause management systems are implemented in SAT solvers to balance the benefits from learnt clauses to prune the search space and the performance degradation caused by increased clauses. This system evaluates the quality of learnt clauses using evaluation metrics such as size and LBD, and it removes low-quality clauses according to their metric value. Most CDCL SAT solvers perform this operation after a specific search period (e.g., a certain number of conflicts or restarts). LBD [5] is the most popular clause evaluation metric widely adopted by most state-of-the-art SAT solvers. The LBD of a clause c can be defined as $LBD(c) = |\{d(l) : l \in c\}|$ where $d(l)$ represents the decision level of literal l where decision level indicates the depth of the branching decision tree. A clause with a lower LBD is considered more valuable because it usually involves a smaller set of decision levels. Besides clause evaluation, the LBD is also used in other heuristics such as restart strategy [6] and decision branching [9].

2.3 Structure of the SAT Problem

An industrial SAT problem has remarkable structural properties compared to randomly generated problems. The clause variable ratio (CVR) [10] is an initial idea that links the structure of the SAT problems and its satisfiability in the context of random instances. The specific boundary of the CVR value where the probability of SAT or UNSAT changes suddenly is known as the phase transition. Focusing on the graph of the problem, Ferrara et al. [11] showed that graphs of industrial SAT problems have much small treewidth compared to random ones. Newsham et al. [18] stated that the industrial SAT problem has a clear community structure and a relationship between performance and structure was observed. Zulkoski et al. [27] focused on the features of the resolution principle, such as mergeability between clauses. Further, Williams et al. [25] focused on backdoor variables, stating that a problem is polynomially tractable if the Boolean assignment of backdoor variables can be determined.

2.4 Graph representation of SAT problem

A variable incidence graph (VIG) is often used to represent the SAT problem as a graph. Let ϕ be a SAT instance with variables V and clauses C. The nodes of the VIG $G(\phi)$ are denoted by V, and edges E denote the existence of a clause in C relating two variables v_i, $v_j \in V$. The weight function $w(e_{v_i,v_j})$ of the edge between v_i and v_j is defined as $\sum_{c \in C, v_i, v_j \in c} 1/\binom{|c|}{2}$. Both literals v and $\neg v$ belong to node v in VIG. $w(e)$ sums all weights of pairs when the same pair of variables (v_i, v_j) appear more than once in the set of clauses C. This implies that a shorter clause obtains a higher edge weight and vice versa. A clause variable incidence graph (CVIG) is a bipartite graph where variables and clauses are represented as nodes on each side, respectively, and there exists an edge between a clause and a variable if the variable exists in the clause. In this study, we used VIG as the graph representation of the SAT problem.

3 Correlation analysis on structural properties

We nominated the following ten structural properties (hereafter, features) of the learnt clause to examine the correlation coefficients with LBD. These features are based on the VIG of the original SAT problem (i.e., no learnt clause added). We denote the learnt clause under the evaluation of features as c and the VIG representation of the problem instance ψ as G.

1. size of clause is defined by $|\{v : v \in c\}|$ where v, c, and notation $|A|$ represent a variable, learnt clause and the size of the arbitrary set A, respectively.

2. Number of used is defined by the number of times c is referred to in conflict analysis. Since a higher number of used is expected for better clauses (i.e., lower LBD), it can be a benchmark feature for interpreting the results of the correlation coefficient values in this experiment.

3. Number of adjacent nodes for clause c is defined by $|\bigcup_{v \in c} adj(v)|$ where $adj(v) = \{u \in V : (u, v) \in E, u \notin c\}$, V, E represent all nodes and edges in G, respectively.

4. Treewidth of clause The treewidth measures the tree-likeness of the graph. Given a $g(c)$, a subgraph of G containing all variables $V_c \in V$ in clause c and edges $E_c \in E$ from V_c, a tree decomposition of a graph $g(c)$ is $d(B, T)$ where T is a tree whose nodes are sets B such that:

1. $\bigcup_{i \in I} B_i = V_c$
2. For every edge $(v, w) \in E_c$, there exists an B_k such that $v \in B_k$ and $w \in B_k$.
3. For all vertices $v \in V_c$, the set $\{i \in I \mid v \in B_i\}$ forms a subtree of T.

The width of a tree decomposition $d(B, T)$ is defined as $\max_{i \in I} |B_i| - 1$. Therefore, the treewidth for a clause c is defined by $\min_{d(B,T)} \text{width}(d(B, T))$. We used the minimum degree heuristic for the computation of treewidth value using the networkx python package. [3]

[3] Networkx https://networkx.org/documentation/stable/reference/index.html

5. Number of community is $|\{community(v) : v \in c\}|$ where $community(v)$ presents the community wherein variable v belongs in graph G. The community represents a group of nodes denoting variables within a graph that have a high degree of interconnections among themselves compared to the connections with nodes outside the group. We used Louvain's community detection algorithm [7].

6. Average internal edge weight is defined by $avg(\{w(e)/|c| : e = (v_i, v_j) \in E, v_i \in c, v_j \in c\})$. This notion denotes the degree of strength (the weight of the edge) of the connection between the nodes within the clauses. $e = (v_i, v_j)$, and $v_i, v_j \in c$ indicates that edge e connects to variables v_i, v_j, belonging to the clause c. Function $w(e)$ returns the weight of edge e. $|c|$ denotes the size of c, and function $avg(X)$ returns the means of the set X.

7. Average external edge weight is defined by $avg(\{w(e)/|c| : e = (v_i, v_j) \in E, v_i \in c, v_j \notin c\})$. This feature is similar to the average internal edge weight; however, the difference is $v_j \notin c$, which means that it calculates the edges between a variable v_i in the learnt clause c and the adjacent variable v_j of v_i, which doesn't belong to learnt clause. When it shows a large value, the variables link heavily to the external (not in the learnt clause) adjacent variables.

8. Clustering coefficient for clause is defined by $avg(\{clusterCoef(v) : v \in c\})$. The node's clustering coefficient indicates the degree to which nodes in a graph tend to cluster together. For a given node v, the $clusterCoef(v)$ is given by $2M(v)/k(v)(k(v)-1)$, where $M(v)$ represents the actual number of edges between the neighbors of v and $k(v)$ represents the number of neighbors of v. The $clusterCoef(v)$ represents the strength of local node ties, and its value lies between 0 and 1, where 1 indicates a complete graph.

9. Connectivity is defined by $min\{|s| : s \in c, g(c) - s \text{ is disconnected}\}$ where s represents a set of nodes, and $g - s$ represents the graph after removing the nodes in s, respectively. Connectivity refers to the minimum number of elements that need to be removed for a graph to become disconnected. A graph is connected if there is a path between every pair of nodes and disconnected otherwise.

10. Shared adjacents ratio is defined by $\bigcup_{v,u \in c}(adj(v) \cap adj(u))/\bigcup_{v \in c}(adj(v))$. This notion indicates the ratio of shared (common) adjacent nodes in all adjacent nodes.

3.1 Experiment setup

We investigated the correlation coefficient of learnt clauses between their ten features and the values of LBD. We used Glucose 4.2.1 [20], the first SAT solver with LBD implementation, as the base solver for the experiments. We exported all learnt clauses and their information during the search, such as the literals contained in the clause and their size, LBD, and the number of used. The export was performed when a clause was learned, deleted, updated (e.g., promoting the LBD value), and then, the search finished. We used 400 benchmark instances of the main track in the SAT competition 2021. We set 30 min (1800 s) as the

time limit to finish the search for the solver. We imposed the following rules to select instances and clauses: a) instances with more than 1 million variables or 10 million clauses are excluded because of memory limitations for the graph construction; b) the clauses with LBD values of 30 or more and sizes of 100 or more were excluded to save the calculation cost, assuming it was reasonable because clauses with large LBD or sizes are less likely to be used in practical searches; c) calculations for one instance were terminated at a maximum of 60 h or when the calculation was completed for a maximum of 100,000 sampled clauses, which were randomly selected from all exported clauses.

3.2 Correlation between the LBD and graph features

Table 1 indicates the distribution of correlation coefficient values between the quality measured by LBD and the features defined in the previous subsection. We calculate the coefficient on each clause using the Pearson correlation coefficient. The mean correlation coefficient value of all clauses is the value for an instance. The *size of clause* shows a moderately strong correlation with the value of LBD: on the mean, it is +0.44, while it is +0.35 at the 25th percentile, and it is +0.54 at the 75th percentile. The value of LBD is inherently bounded by its size, and therefore, a certain degree of this correlation is to be anticipated. In contrast, the correlation mean was only -0.09 for *number of used*, though we expected it to be strongly correlated with LBD. The *number of community* correlates only +0.30 in the mean and +0.20/+0.40 in the 25th/75th percentile, respectively. This result suggests that the *number of community* correlates to the value of LBD a little, but not strongly. The *number of adjacents*, *shared adjacents ratio*, and *average external edge weight* showed strong correlations; each mean was +0.48, -0.51, and -0.52, respectively. Interestingly, contrary to LBD, these three features are independent of the search state. They use only information from the subgraph of the original SAT problem; however, they show a strong correlation with the LBD values. Among them, the *average external edge weight*, which showed a robust negative correlation with LBD, was analyzed more deeply in the subsequent section.

3.3 Deep diving to the correlation between AEEW and LBD

Hereafter, we refer to the *average external edge weight* as AEEW. This notion value presents the degree of connection strength (as the weight of edge) between the node in the learnt clause and its adjacent node outside the learnt clause (i.e., external). We propose that AEEW and LBD are potentially connected in measuring the "probability of propagation". LBD reflects the outcome of propagation arising from its decision tree structure, whereas AEEW forecasts the probability of propagation based on the strength of edge connections.

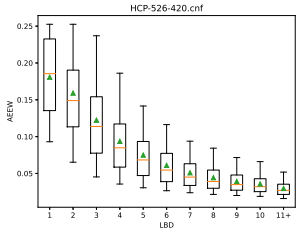
AEEW value at each LBD The previous result in Table 1 shows the distribution the mean value of all clauses in an instance. Next, we analyzed it in

Table 1: Values of the correlation coefficient between the ten features and the value of LBD. Features are sorted according to their mean values.

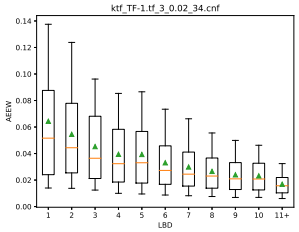
Feature	Min	25%	Mean	75%	Max
Number of adjacents	0.07	0.39	0.48	0.58	0.90
Size of clause	0.06	0.35	0.44	0.54	0.91
Treewidth	-0.12	0.22	0.32	0.43	0.70
Number of community	-0.13	0.20	0.30	0.40	0.65
Clustering coefficient	-0.49	-0.17	-0.06	0.05	0.54
Number of used	-0.38	-0.14	-0.09	-0.03	0.39
Connectivity	-0.43	-0.21	-0.13	-0.03	0.33
Average internal edge weight	-0.53	-0.33	-0.23	-0.05	0.34
Shared adjacents ratio	-0.77	-0.57	-0.51	-0.41	-0.06
Average external edge weight	-0.78	-0.60	-0.52	-0.41	-0.08

more detail by checking the correlation between the AEEW and LBD of a single instance. Figure 1 shows the distribution of the AEEW value at each LBD in an instance. We picked four instances to demonstrate the typical results caused by the space limitation. These charts indicate the following: (1) The mean of AEEW decreases with an increase in the LBD value, which helps verify the correlation in Table 1. (2) The variance of the distribution decreases according to the increase in LBD values. (3) The degree of decrease in the mean and its variance depends on the instances. These can be explained by the definition of AEEW as follows: A heavier edge weight indicates the existence of the pair of variables in the small size clause and/or in multiple clauses. The Boolean values of variables that strongly connected (with a heavier edge weight) are more likely to be assigned in the same propagation. The variables used in the same propagation belong to the same decision level, i.e., in the same literal block. Therefore, the heavier AEEW tends to cause smaller LBD values, suggesting that the AEEW can be an alternative measurement to the LBD value because of its relationship.

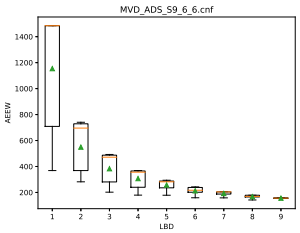
Correlation between AEEW and initial/updated LBD values One characteristic of LBD is that the value is updated during the search. The decision tree determines the LBD value and changes according to the progress of the search. Therefore, the LBD value is updated when a smaller set of decision levels for a clause are found during the search. [21] revealed that this update mechanism is the source of the efficiency of LBD, and turning off this update mechanism of LBD deteriorates the performance of the solver. We compared AEEW with the updated LBD values in the previous experiments. This experiment shows the correlation between the AEEW and initial/updated LBD values. A clause is identified by its containing literals, and the initial/updated LBD is defined by the maximum/minimum LBD value observed from all exported clauses, respectively. Figure 2a shows the differences in the correlation coefficient between AEEW and initial/updated LBD values as a scatter chart. Each dot denotes an instance, and the dot is plotted at the value of its correlation coefficient value



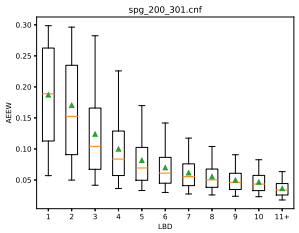
(a) HCP-526-420.cnf



(b) ktf_TF-1.tf_3_0.02_34.cnf



(c) MVD_ADS_S9_6_6.cnf

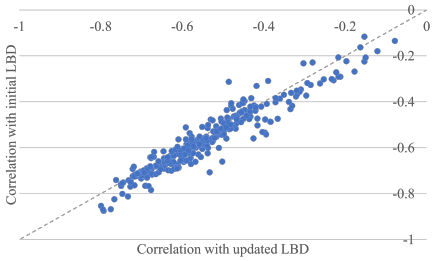


(d) spg_200_301.cnf

Fig. 1: Box-plot of the value of AEEW in the sample 4 instances

of the AEEW of initial/updated values, respectively. A total of 337 instances remained through the setup rules of the previous experiment. A total of 273 instances (81%) exhibit a stronger correlation in the updated LBD than that in the initial LBD. We suggest that this result originates from the fact that LBD calculates its values based on the actual propagation results (in a decision tree). In contrast, AEEW estimates the likelihood of propagation based on the strength of connections between edges. This characteristic of AEEW, which correlates to updated LBD more, though it is based on the original instance, suggests its potential utility in our study.

We visualized the AEEW of each LBD value in an instance in Figure 2b using spg_300_300.cnf as a typical sample to obtain a more detailed correlation between the initial and updated LBD values. The value present in the cell at $(lbd_{init}, lbd_{updated})$ represents the average value of AEEW for a clause where the initial/updated LBD value was lbd_{init}, $lbd_{updated}$, respectively. When comparing the values of cells in any column (i.e., when given an initial LBD value), the higher values are more observed in the cell of the lower updated LBD. This suggests that AEEW indicates the updated (future) LBD value more than the initial value, though AEEW utilizes only the initial (original) SAT structure information.



(a) Scatter plot of the correlation between the value of the AEEW and initial or updated LBD values

Initial LBD value (maximum value)

Updated LBD value (minimum value)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1			0.22	0.21	0.17	0.15	0.12	0.10	0.09	0.07	0.06	0.04	0.07	0.01	
2		0.20	0.27	0.17	0.16	0.13	0.11	0.10	0.08	0.08	0.07	0.10	0.06	0.06	
3			0.14	0.23	0.11	0.11	0.10	0.09	0.08	0.07	0.07	0.05	0.06	0.06	0.02
4				0.11	0.16	0.09	0.08	0.07	0.08	0.07	0.07	0.04	0.05		0.05
5					0.09	0.13	0.07	0.07	0.06	0.06	0.05	0.05	0.05	0.03	0.03
6						0.08		0.06	0.05	0.06	0.04	0.04	0.04	0.05	
7							0.07		0.05	0.05	0.04	0.05	0.04	0.03	0.07
8								0.06		0.05	0.05	0.04	0.04	0.04	0.06
9									0.05		0.04	0.04	0.04	0.03	
10										0.05		0.04	0.04	0.03	0.04
11											0.04		0.04	0.03	0.03
12												0.04		0.04	0.02
13													0.04		0.03
14														0.03	
15															0.03

(b) Value of the AEEW at initial and updated LBD values of the instance spg_300_300.cnf

Fig. 2: AEEW correlation and heatmap with updated/initial LBD values

4 Weight-to-Adjacent-Node-based Clause Evaluation

Through the result of Section 3, we found a strong-correlated feature, AEEW in the correlation analysis. It indicated that valuable clauses tend to have heavy edges to adjacent variables on the graph. It means that these clauses are more easily used in the propagation step and AEEW can capture it. Furthermore, the second experiment revealed that this feature predicts LBD's minimum value (updated value) more accurately than its maximum value. Therefore, we assumed that AEEW is a helpful metric for clause evaluation because of its power to quantify the propagation probability and its capability to maintain clauses that will be updated and evaluated as good in the future.

In this section, we propose a new clause evaluation method, weight-to-adjacent-node-based clause evaluation (WANCE), to manage the learnt clauses. WANCE utilizes the values of AEEW as the quality metric of the clauses. A higher AEEW value indicates a higher quality because of the negative correlation. In implementing WANCE, the AEEW values are discretized by rounding to the nearest ten, ignoring the slight differences. We set 0.0001 based on the results of the preliminary experiments to view the distribution of the AEEW values. We used WANCE only from the learnt clause management perspective and did not replace all usages of LBD, e.g., for the decision heuristics and restart strategy, because replacing LBD can affect the performance of other heuristics, which makes it difficult to isolate the effect of modifying the quality metric and clause management method. Further, we used LBD as the secondary metric in the WANCE method and size as the third one.

Regarding the procedure of the learnt clause evaluation using WANCE, we followed the scheme of a state-of-the-art SAT solver CaDiCaL [3]: All learnt clauses are marked to be maintained or targeted for reduction based on their state. They are sorted according to the values of AEEW, LBD, then, size. A

specific percentage p (in this study, 75% pre-defined) of lower-evaluated clauses are deleted from the database. The instance graph G to calculate the value of AEEW is built from the original SAT instance at the beginning of the search before any learning. The sorting function orders the clauses based on AEEW, LBD, and size values. Given two clauses c_i and c_j, the higher (better) evaluation is provided to the larger AEEW clause. Therefore, clauses with smaller AEEW values are deleted preferentially. If AEEW values are equal, the clauses are deleted based on the LBD value. For equivalent LBD values, the clauses are deleted based on size. The difference from the original implementation of CaDiCaL is the addition of primary evaluation using AEEW. When a clause was given, the rate that other maintained clauses had the same AEEW values was 1.41%. The average percentage of clauses with the same AEEW value was 1.41%; however, among 400 benchmarks, there were three instances where the probability reached 100%. Excluding them, the mean probability was 0.315%, with a distribution ranging from 0.018% to 5.505%. This is a notably lower result compared to the probabilities of having identical LBD and Size values, which are 2.47% and 1.13%, respectively.

5 Performance Evaluation

5.1 Experiment setup

We evaluated the performance of a solver using the WANCE method against a base solver, CaDiCaL 1.4.1 [3]. This state-of-the-art sequential SAT solver has been widely used in the SAT community since 2020 because of its high source-code readability, structural simplicity, and outstanding performance, making it the 2019 SAT race winner. We implemented a solver with a random clause evaluation method to investigate the effect of modifying the clause evaluation method of the base solver, which assigns random values to each clause as its quality indicator with a random range of 30 (1–30).

Our benchmark comprised 1600 instances from the main tracks of SAT competitions held from 2019–2022 (400 instances/year). We conducted experiments on a computer with an AMD Threadripper Pro 3995WX processor (64 core) and 512 GB (128 GB 4 slots, DDR4-3200 MHz) RAM. We used the default CaDiCaL implementation besides the necessary functions for WANCE and the random method. We used no options of CaDiCaL for running solvers except the one for the timeout. For the AEEW calculation, clauses with LBD values of 30 or more, and sizes of 100 or more, were ignored to ensure that the calculation cost is reduced as that for the observation experimental setup. We evaluated the performance of the solver based on the number of instances solved within a time limit of 3600 s on the CPU clock and the PAR-2 per instance score, which represents the mean time required to solve an instance with an additional penalty of 3600 s for each unsolved instance.

5.2 Evaluation result

Table 2 summarizes the results of our experiments, where we compared Base (CaDiCaL without any modifications), Random (CaDiCaL with the random clause evaluation method), and WANCE (CaDiCaL with the WANCE method).

Table 2: Performance evaluation results corresponding to each solver

	sc19		sc20		sc21		sc22		total
Solver	SAT	UNSAT	SAT	UNSAT	SAT	UNSAT	SAT	UNSAT	
Base	129	**75**	**110**	**94**	111	**124**	128	112	883
Random	124	66	95	87	110	117	124	107	830
WANCE	**139**	73	105	92	**120**	**124**	**132**	**120**	**905**

The performance of the solver using WANCE is better than that of the base solver using LBD because of solving 22 more instances (+2.5%) and achieving a PAR-2 per instance score improvement of 38 on average. The improvement was more significant for SAT instances, whereas the number of UNSAT instances solved was almost equivalent to that of the base solver. A 4.6% performance improvement was observed when limited to only SAT. As expected, the random selection solver performed considerably worse, thereby solving 53 instances less (-6.0%). Compared with the results of the random one, the obtained results confirmed that WANCE could be improved via its methodology instead of by modifying only the clause evaluation function.

This result is notable because WANCE relies only on the subgraph information of the original SAT problem, which makes it sufficient to assess the quality of the learnt clauses as accurately as the LBD. This improvement can be attributed to the AEEW value being more correlated to the updated LBD value than the initial one. In the LBD-based solver, the clauses are evaluated and removed based on their LBD values at a certain moment. A lower-rated clause can be removed even if the clause has a possibility to be higher-rated in the future. We observed that clauses learned several times (removed several times in other words) were updated to LBD 2. WANCE can retain these potentially useful clauses, thereby improving the performance of the solver. This study also hints at possible insights for designing and fine-tuning SAT solvers, suggesting that clauses strengthening the current structure make the solver work better.

6 Related Work

Learnt Clause Evaluation Audemard et al. [4] proposed the "freezing and reactivation" strategy for clause management. The solver saves all clauses in its database and selectively activates some clauses learned in the search state, which are similar to the currently learned one. Further, it freezes other clauses instead of deleting them from the database. Sonobe [21] studied the differences

between the same LBD clauses and distinguished the clauses of the same LBD value based on the number of variables in each literal block. Further, the research revealed that disabling the update function of LBD worsened the performance of solving UNSAT instances with a limited impact on the SAT. Jabbour et al. [15] focused on the size metric against LBD and showed that employing a random strategy for clauses above a certain size improves performance. Vallade et al. [24] revealed the correlation between the LBD value and the number of communities in the graph, and they proposed a new clause evaluation method that combines the number of communities and LBD scores. Soos et al. [22] developed a framework to understand heuristics in SAT solvers, particularly about managing learnt clauses. They assessed various clause features, such as those utilized for UIP creation, to measure their significance as "relative importance." Yang et al. [26] introduced CausalSAT, a tool designed to uncover the causal connections between learned clauses and the values of specific heuristics, particularly LBD. Jamali et al. [16] utilized Between Centrality to evaluate learnt clauses and they showed that this measure contributes to Solver's performance improvement.

Relationship between the SAT Problem and Graph Structure Anstegui et al. [2] analyzed the graph of SAT problems and showed that industrial SAT problems have stronger community structures than random ones when using modularity Q to quantify the results. Newsham et al. [18] studied the correlation between the community structure and LBD values and demonstrated that the degree of the community structure is related to its runtime (i.e., the difficulty of the instance). Newsham et al. [19] visualized the graph structure of the SAT problem and showed that the progress of the search (i.e., acquired learnt clause) destroyed the clarity of the structure. Mull et al. [17] studied the theoretical analysis of the hardness from the structure of the SAT problem perspective. Several theoretical attempts were made to explain the effectiveness of the modern SAT solver, and Ganesh and Vardi [12] summarized these attempts.

7 Conclusion

We proposed a new learnt clause evaluation method, WANCE, for SAT solvers. Through the correlation analysis between the quality of clauses measured by LBD and the graph-structural properties of the learnt clauses, we found a property, AEEW that represents the average weight of the edge between nodes in a clause and their adjacent nodes outside the clause, that indicated a strong correlation coefficient of -0.52 with LBD. WANCE was implemented using AEEW as its primary metric for clause evaluation. Experimental results indicated that a solver that implemented WANCE achieved 2.5% better performance in the number of instances solved than the state-of-the-art solver that primarily uses LBD.

As the future work, first, there is potential to explore other structural properties as evaluation metrics other than AEEW. Adopting machine learning techniques such as graph neural networks can help discover better-quality metrics. In

addition, structural evaluation metrics such as WANCE are expected to be more effective in parallel environments owing to their search-independent characteristics. WANCE can be utilized as a criterion for clause-sharing decisions among parallel workers. Finally, we hope that utilizing graph features for clause evaluation will spur further research into the theoretical study of the effectiveness of LBD and other clause evaluation methods.

References

1. Alyahya, T.N., Menai, M.E.B., Mathkour, H.: On the structure of the boolean satisfiability problem: A survey. ACM Comput. Surv. **55**(3), 1–34 (Mar 2022). https://doi.org/10.1145/3491210
2. Ansótegui, C., Giráldez-Cru, J., Levy, J.: The community structure of sat formulas. In: Theory and Applications of Satisfiability Testing – SAT 2012. pp. 410–423. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
3. Armin, B., Fazekas, K., Fleury, M.: CaDiCaL 1.4.1 (2021), https://github.com/arminbiere/cadical, accessed: 2023-1-26
4. Audemard, G., Lagniez, J.M., Mazure, B., Saïs, L.: On freezing and reactivating learnt clauses. In: Theory and Applications of Satisfiability Testing - SAT 2011. pp. 188–200. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
5. Audemard, G., Simon, L.: Predicting learnt clauses quality in modern SAT solvers. In: Proceedings of the 21st international jont conference on Artifical intelligence. pp. 399–404. IJCAI'09, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (Jul 2009)
6. Biere, A., Frohlich, A.: Evaluating cdcl restart schemes. In: Proceedings of Pragmatics of SAT 2015 and 2018. EPiC Series in Computing, vol. 59, pp. 1–17. EasyChair (2019). https://doi.org/10.29007/89dw
7. Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment **2008**(10), P10008 (oct 2008). https://doi.org/10.1088/1742-5468/2008/10/P10008
8. Bright, C., Kotsireas, I., Heinle, A., Ganesh, V.: Complex golay pairs up to length 28: A search via computer algebra and programmatic sat. Journal of Symbolic Computation **102**, 153–172 (2021). https://doi.org/10.1016/j.jsc.2019.10.013
9. Chang, W., Wu, G., Xu, Y.: Adding a lbd-based rewarding mechanism in branching heuristic for sat solvers. In: 2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE). pp. 1–6 (2017). https://doi.org/10.1109/ISKE.2017.8258780
10. Cheeseman, P., Kanefsky, B., Taylor, W.M.: Where the really hard problems are. In: Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume 1. p. 331–337. IJCAI'91, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1991)
11. Ferrara, A., Pan, G., Vardi, M.Y.: Treewidth in verification: Local vs. global. In: Logic for Programming, Artificial Intelligence, and Reasoning. pp. 489–503. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
12. Ganesh, V., Vardi, M.Y.: On the unreasonable effectiveness of SAT solvers. In: Beyond the Worst-Case Analysis of Algorithms, pp. 547–566. Cambridge University Press (Jan 2021). https://doi.org/10.1017/9781108637435.032

13. Heule, M.J.H., Kullmann, O., Marek, V.W.: Solving and verifying the boolean pythagorean triples problem via Cube-and-Conquer. In: Theory and Applications of Satisfiability Testing – SAT 2016. pp. 228–245. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40970-2_15
14. Iida, Y., Sonobe, T., Inaba, M.: Structural impact of learnt clauses in sat solvers: An empirical analysis. In: Pragmatics of SAT (2023)
15. Jabbour, S., Lonlac, J., Saïs, L., Salhi, Y.: Revisiting the learned clauses database reduction strategies. International journal of artificial intelligence tools: architectures, languages, algorithms **27**(08), 1850033 (2018). https://doi.org/10.1142/S0218213018500331
16. Jamali, S., Mitchell, D.: Centrality-based improvements to cdcl heuristics. In: Beyersdorff, O., Wintersteiger, C.M. (eds.) Theory and Applications of Satisfiability Testing – SAT 2018. pp. 122–131. Springer International Publishing, Cham (2018)
17. Mull, N., Fremont, D.J., Seshia, S.A.: On the hardness of sat with community structure. In: Theory and Applications of Satisfiability Testing – SAT 2016. pp. 141–159. Springer International Publishing, Cham (2016)
18. Newsham, Z., Ganesh, V., Fischmeister, S., Audemard, G., Simon, L.: Impact of community structure on sat solver performance. In: Theory and Applications of Satisfiability Testing – SAT 2014. pp. 252–268. Springer International Publishing, Cham (2014)
19. Newsham, Z., Lindsay, W., Ganesh, V., Liang, J.H., Fischmeister, S., Czarnecki, K.: Satgraf: Visualizing the evolution of sat formula structure in solvers. In: Theory and Applications of Satisfiability Testing – SAT 2015. pp. 62–70. Springer International Publishing, Cham (2015)
20. Simon, Laurent; Audemard, G.: About the glucose sat solver (2018), https://www.labri.fr/perso/lsimon/research/glucose/, accessed: 2023-3-8
21. Sonobe, T.: Looking inside literal blocks: Towards mining more promising learnt clauses in sat solving. In: 2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI). pp. 972–979 (2016). https://doi.org/10.1109/ICTAI.2016.0150
22. Soos, M., Kulkarni, R., Meel, K.S.: CrystalBall: Gazing in the black box of SAT solving. In: Theory and Applications of Satisfiability Testing – SAT 2019. pp. 371–387. Springer International Publishing (2019). https://doi.org/10.1007/978-3-030-24258-9_26
23. Sörensson, N., Eén, N.: Minisat v1.13 - a sat solver with conflict-clause minimization. In: Theory and Applications of Satisfiability Testing – SAT 2005 (2005)
24. Vallade, V., Le Frioux, L., Baarir, S., Sopena, J., Ganesh, V., Kordon, F.: Community and lbd-based clause sharing policy for parallel sat solving. In: Theory and Applications of Satisfiability Testing – SAT 2020. pp. 11–27. Springer International Publishing, Cham (2020)
25. Williams, R., Gomes, C., Selman, B.: On the connections between backdoors, restarts, and heavy-tailedness in combinatorial search. In: Theory and Applications of Satisfiability Testing 2003 (2003)
26. Yang, J., Shaw, A., Baluta, T., Soos, M., Meel, K.S.: Explaining sat solving using causal reasoning (2023)
27. Zulkoski, E., Martins, R., Wintersteiger, C.M., Liang, J.H., Czarnecki, K., Ganesh, V.: The effect of structural measures and merges on sat solver performance. In: Principles and Practice of Constraint Programming. pp. 436–452. Springer International Publishing, Cham (2018)

Multi-output regression for travel demand estimation in an urban road network

Alexander Krylatov[1,2*], Anastasiya Raevskaya[2], and Ilya Murzin[2]

[1] Institute of Applied Mathematical Research
of the Karelian Research Centre of the Russian Academy of Sciences,
11 Pushkinskaya str., Petrozavodsk, 185910, Russia
[2] Saint-Petersburg State University,
Universitetskaya emb. 7-9, 199034 Saint-Petersburg, Russia
aykrylatov@yandex.ru,a.krylatov@spbu.ru

Abstract. Nowadays, artificial intelligence systems seem to be urgent and breaking-through tools for efficient traffic management in modern urban road networks. However, these tools are highly sensitive to accurate travel demand data when predicting traffic congestion. Despite the fact that researchers have already developed numerous approaches to dealing with travel demand estimation, in many cases there is still a lack of ability to achieve the required accuracy level. The present paper focuses on the travel demand search task, formulated as an inverse traffic assignment problem. We developed multi-output regression models to estimate travel demand values for a road network with multiple origin-destination pairs. We used solutions to the non-linear optimization problem of equilibrium traffic assignment under different demand patterns to generate data for training and test sets. Our computational study gave several practical recommendations on the best scenarios for the implementation of such a tool as a multi-output regression for travel demand estimation.

Keywords: travel demand estimation, user-equilibrium, multi-output regression

1 Introduction

Nowadays, artificial intelligence systems seem to be urgent and breaking-through tools for efficient traffic management in modern urban road networks. However, these tools are highly sensitive to accurate travel demand data when predicting traffic congestion. Researchers deal with travel demand estimation (or origin-destination (OD) matrix estimation) by virtue of different approaches and techniques. From a mathematical perspective, the simplest approach may refer to

* The work was supported by a grant from the Russian Science Foundation (No. 22-11-20015 Research and development of mathematical models and software for finding equilibrium traffic flows and optimization of a transportation network on the case of Petrozavodsk city).

the gravity methods, which have been well described in the context of spatial regional analysis [9]. Another large class of approaches can be associated with *a priori* economic theorizing (see, for instance, [6]). Such basic concepts have been developed and implemented in various ways by scientists from various fields. Moreover, the development of technical facilities and devices, such as traffic counters or plate scanning sensors, has not completely solved the abovementioned problem but has initiated investigations of emerging issues. Thus, researchers have developed a technique that could possibly help overcome the incompleteness of data gathered by plate scanners [16]. In fact, even back then, incomplete data was a routine input condition when solving the travel demand estimation problem.

The first bi-level formulations of the OD-matrix estimation problem employing the user equilibrium assignment principle were made in the 1980s [7, 17]. In the beginning of the 1990s, this approach was reinforced by a clear demonstration of various alternative bi-level problems with user equilibrium assignment (in the lower-level), which reflected the travel demand estimation process [24]. Since those bi-level programs that had appeared were actually NP-hard, a single-level optimization model was based on the stochastic user equilibrium concept to estimate path flows and, hence, travel demand [2]. In turn, a method for estimating the set of routes and their flows under the static user-equilibrium assignment principle was presented [1]. Researchers offered to cope with the complications of the bi-level problem by means of heuristic methodology [15]. Moreover, a convex approximation of the bi-level program by single-level optimization was offered [21]. The stochastic user equilibrium principle was also employed by researchers in order to avoid a bi-level formulation of the travel demand estimation problem [23].

Researchers still contribute in the field, improving previously proposed solutions and developing new approaches. Sun and Fan study demand estimation using stochastic programming [22], while others develop perimeter control strategies based on the macroscopic fundamental diagram [14]. There are new results on demand estimation for strategic road traffic assignment models with strict capacity constraints and residual queues [5]. However, all these studies are based on the assumption that estimated OD flow should align with *a prior* belief or knowledge of OD demand. We believe that the existence of an *a priori* OD-matrix is a too strict assumption, since in practice, as a rule, no one has any pre-given or *a priori* information on travel demand. The only data traffic engineers can gather in most large cities using primitive observation equipment is road flows. In our research, we deal with the travel demand estimation problem, avoiding *a priori* given demands. Our aim is to develop tools that are able to provide valuable demand information for practical reasons.

In the present paper, we concentrate on the development of approaches for solving the bi-level formulation of the travel demand estimation problem employing the user equilibrium assignment principle. A deep understanding of the relationship between traffic assignment and travel demand estimation allows one to develop artificial intelligence tools of high performance and efficiency for traf-

fic management support. The traffic assignment problem is formulated in the form of a nonlinear optimization program in Section 2. A bilevel travel demand estimation problem is given in Section 3. Up-to-date issues and challenges are discussed. Section 4 is devoted to regression models in the context of the travel demand estimation problem. In particular, we study the sensitivity of key metrics of regression models to the number of OD pairs under consideration. The last section is devoted to conclusions and further plans.

2 Traffic assignment problem

Let us consider an urban road network presented by a directed graph $G = (E, V)$, where V represents a set of intersections, while E represents a set of available roads between the adjacent intersections. Define $W \subseteq V \times V$ as the ordered set of pairs of nodes with non-zero travel demand $F^w > 0$, $w \in W$. W is usually called as the set of origin-destination pairs (OD-pairs), $|W| = m$. Any set of sequentially linked arcs initiating in the origin node of OD-pair w and terminating in the destination node of the OD-pair w we call *route* between the OD-pair w, $w \in W$. The ordered set of all possible routes between nodes of the OD-pair w we denote as R^w, $w \in W$, and the ordered set of all possible routes between all OD-pairs we denote as R, $R = \cup_{w \in W} R^w$. Demand $F^w > 0$ seeks to be assigned between the available routes R^w: $\sum_{r \in R^w} f_r^w = F^w$, where f_r^w is a variable corresponding to a traffic flow through route $r \in R^w$ between nodes of OD-pair $w \in W$. Introduce the vector of demand $F = (F^1, \ldots, F^m)^{\mathrm{T}}$ and the vector f associated with the route-flow traffic assignment pattern, such that $f = f_R = (\ldots, f_r^w, \ldots)^{\mathrm{T}}$ is actually a vector of $\{f_r^w\}_{r \in R^w}^{w \in W}$.

Let us introduce differentiable strictly increasing functions on the set of real non-negative numbers $t_e(\cdot)$, $e \in E$. We suppose that $t_e(\cdot)$, $e \in E$, are non-negative and their first derivatives are strictly positive on the set of real non-negative numbers. By x_e we denote traffic flow on the arc e, while x is an appropriate vector of arc-flows, $x = (\ldots, x_e, \ldots)^{\mathrm{T}}$, $e \in E$. Defined functions $t_e(x_e)$ are used to describe travel time on arcs e, $e \in E$, and they are commonly called arc *delay*, *cost* or *performance* functions. In this section we assume that the travel time function of the route $r \in R^w$ between OD-pair $w \in W$ is the sum of travel delays on all arcs belonging to this route. Thus, we define travel time through the route $r \in R^w$ between OD-pair $w \in W$ as the following separable function

$$t_r^w(f) = \sum_{e \in E} t_e(x_e) \delta_{e,r}^w \quad \forall r \in R^w, w \in W, \tag{1}$$

where, by definition,

$$\delta_{e,r}^w = \begin{cases} 1, \text{ if arc } e \text{ belongs to the route } r \in R^w, \\ 0, \text{ otherwise.} \end{cases} \quad \forall e \in E, w \in W,$$

while, naturally,

$$x_e = \sum_{w \in W} \sum_{r \in R^w} f_r^w \delta_{e,r}^w \quad \forall e \in E, \tag{2}$$

i.e. traffic flow on the arc is the sum of traffic flows through all routes that include this arc.

The *equilibrium traffic assignment problem* under separable travel time functions has the following form of optimization program [18, 20]:

$$\chi(F) = \min_{x} \sum_{e \in E} \int_0^{x_e} t_e(u)du, \tag{3}$$

subject to

$$\sum_{r \in R^w} f_r^w = F^w, \quad \forall w \in W, \tag{4}$$

$$f_r^w \geq 0 \quad \forall r \in R^w, w \in W, \tag{5}$$

where, by definition,

$$x_e = \sum_{w \in W} \sum_{r \in R^w} f_r^w \delta_{e,r}^w \quad \forall e \in E. \tag{6}$$

An arc-flow assignment pattern x and corresponding route-flow assignment pattern f, satisfying (3)–(6), are proved to reflect *user equilibrium* traffic assignment or such an assignment that

$$t_r^w(f) = \sum_{e \in E} t_e(x_e)\delta_{e,r}^w \begin{cases} = \mathfrak{t}^w, \text{ if } f_r^w > 0, \\ \geq \mathfrak{t}^w, \text{ if } f_r^w = 0, \end{cases} \quad \forall r \in R^w, w \in W, \tag{7}$$

where $\mathfrak{t}^w$ is *equilibrium travel time* or travel time on actually used routes between OD-pair w, $w \in W$. Herewith, in case x is searched as a result of solving (3)–(6), then one deals with *arc-flow equilibrium traffic assignment problem* [11]. Otherwise, if f is searched, then *route-flow equilibrium traffic assignment problem* is under consideration [12].

3 Travel demand estimation problem

Travel demand estimation and traffic assignment search are two problems that can be considered mutually inverse. Indeed, the solution variables of the travel demand estimation problem are input data for traffic assignment search, while the solution variables of the traffic assignment problem are input data for travel demand estimation (table 1). The first consideration of the travel demand estimation problem as the inverse traffic assignment problem was made by [4]. However, he formulated the inverse problem in very general terms, where even a matrix of route choice was believed to be given. One of the main purposes of the present paper is to eliminate the excessive use of pre-given and *a priori* information when estimating travel demand. Indeed, when using an a priori matrix in the objective function of the demand evaluation task, the travel demand estimation problem is reduced to a standard deviation search.

Table 1. Input data and solution variables.

	Locations of OD-pairs	Travel demand values	Traffic congestions
Traffic assignment problem	Input data	Input data	Solution variables
Travel demand estimation	Solution variables	Solution variables	Input data

In fact, the problem of travel demand estimation is reduced to the problem of iterative updating of the travel demand information for a given set of origin-destination pairs.

We believe the assumption of the existence of an *a priori* OD-matrix is too strict when considering the travel demand estimation, and we formulate this problem in such a way that we could avoid the pre-given or *a priori* information. Actually, we formulate the inverse traffic assignment problem, where the equilibrium assignment pattern $\bar{x}$ is given, while locations of OD-pairs and travel demand values are to be found [10]. Therefore, the *travel demand estimation problem* has the following form of bi-level optimization program:

$$\min_{F} ||\chi(F) - \bar{x}|| \tag{8}$$

subject to

$$F^w \geq 0 \quad \forall w \in V \times V, \tag{9}$$

where mapping $\chi(F)$ is given by the optimization program (3)–(5) with a definitional constraint (6). However, one can see that the problem (8)–(9) has a trivial solution such that $F^w = \bar{x}_w$ for all $w \in W$ under $W = E$. Moreover, let us consider a transportation network presented by a directed graph with 4 nodes and 4 arcs (Fig. 1).

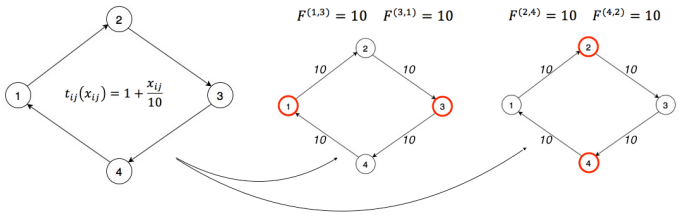


Fig. 1. Transportation network of 4 nodes

We suppose that there is a traffic flow on each arc, which is equal to 10. A solution to the corresponding inverse traffic assignment problem (8)–(9) does

exist. However, there are several solutions to the travel demand estimation problem (8)–(9) in this case: for example, $F^{(1,3)} = 10$ and $F^{(3,1)} = 10$ or $F^{(2,4)} = 10$ and $F^{(4,2)} = 10$ (Fig. 1). Thus, one can see that the travel demand estimation problem (8)–(9) may have multiple solutions. Actually, it has already been revealed previously that [10]:

- generally, the inverse equilibrium traffic assignment problem has no unique solutions;
- if the location of the OD-pairs is known, then the inverse equilibrium traffic assignment problem may have a unique solution.

Hence, once the travel demand locations are specified, the obstacles concerning the OD-pairs search disappear. Otherwise, the upper-level goal function demonstrates fuzzy behavior, and continuous optimization cannot cope with the problem. In other words, *a priori* information on the locations of OD pairs appears to be quite important for travel demand estimation, and it is still not as strict a requirement as *a priori* information on the OD matrix. Therefore, we should improve the formulation (8)–(9) as follows:

$$\min_{F} ||\chi(F) - \bar{x}|| \tag{10}$$

subject to

$$F^w > 0 \quad \forall w \in W, \tag{11}$$

$$F^w = 0 \quad \forall w \in V \times V \backslash W, \tag{12}$$

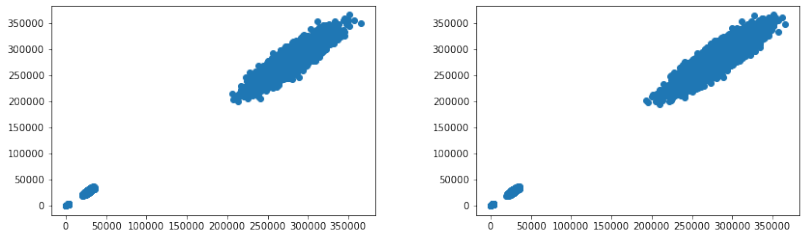
where W is given.

4 Estimation of travel demand values for multiple origin-destination pairs

Previously, we have shown that polynomial regression fits well to estimate the overall travel demand in an urban road network [19]. Indeed, if we consider the travel demand search task as an inverse traffic assignment problem, then the overall travel demand, i.e.,

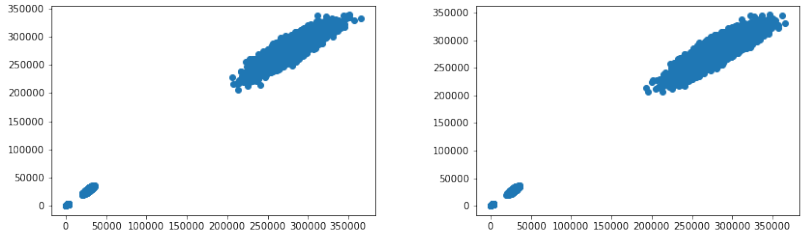
$$\sum_{w \in V \times V} F^w,$$

appears to be estimated via linear and polynomial regression at a high level of accuracy (figures 2, 3, and 4). Due to mapping $\chi(F)$, we were able to generate any size datasets to learn and test regression models, which allowed us to reconstruct overall travel demand by observing traffic. Firstly, we generated 400 000 pairs of the traffic assignment pattern and the overall demand value for the Sioux-Falls road network [8]. Then, we developed the linear regression model and visualized the obtained results as given in Fig. 2. Axis X reflects the real values of the overall demand, while axis Y reflects values predicted by the model. The closer the shape of the graph to $y = x$, the higher the quality of the model.



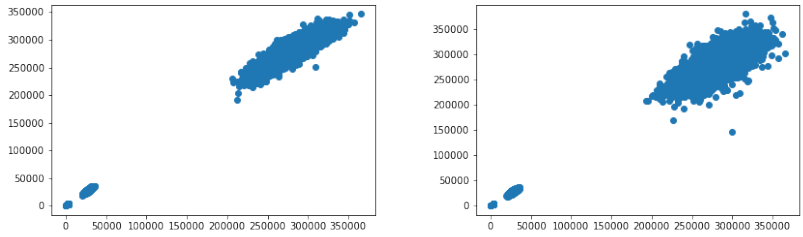
R^2: 0.9978288568823274 R^2: 0.9977746187284728

Fig. 2. Linear regression



R^2: 0.9982787603645912 R^2: 0.9981649141146739

Fig. 3. Polynomial regression 2 degree



R^2: 0.9980645257730227 R^2: 0.9965214104118396

Fig. 4. Polynomial regression 3 degree

Due to the graphs and the fact that the R^2 value is close to 1, we can conclude that the model works quite well, i.e., the regression predictions perfectly fit the data. The left picture demonstrates results for the model learned from 70% of the data and tested from 30% of the data, while the right picture demonstrates results for the model learned from only 20% of the data. Thus, the model does not require extra-large datasets to give a valuable response. The model appeared

to work even better for the second-degree polynomial regression model (fig. 3). Then, we developed the third-degree polynomial regression model (fig. 4). According to R^2 values, we concluded that the second-degree polynomial regression model demonstrates the best results. Eventually, we summarized our findings as follows:

- the second-degree polynomial regression model fits the best for the overall demand estimation by observing traffic;
- the polynomial regression model does not require large-scale learning datasets to work well.

Our current research further explores regression models in the context of travel demand estimation. However, instead of the overall demand value search, we now focus on the estimation of multiple demand values for several origin-destination pairs simultaneously. This time we again deal with the Sioux-Falls road network and generate 10 000 vectors of the traffic assignment patterns for 10 000 vectors of feasible demands. Our approach implies that we generate separate datasets for different patterns of origin-destination locations and the number of OD pairs. Table 2 demonstrates the values of the most important metrics for our multi-output regression model.

Table 2. Metrics for linear multi-output regression model.

Number of OD pairs	Input layer dimension	MSE	R^2	MAE	MAPE
5	76	$8.32e^{-24}$	1	$2.24e^{-12}$	$2.69e^{-15}$
24	76	$1.07e^{-24}$	1	$7.87e^{-13}$	$5.64e^{-15}$
50	76	0.17	0.99	0.16	0.0013
100	76	17368.99	0.67	101.48	2.28

Since the Sioux-Falls road network has 76 arcs, then the input layer of the model consists of 76 components. Indeed, for any feasible travel demand pattern, mapping $\chi(F)$ returns a value of flow for every road arc.

Table 3. Metrics for some other models.

Number of OD pairs	Input layer dimension	MSE	R^2	MAE	MAPE
Gradient boosting					
100	76	26936.54	0.51	133.23	37.99
Neural network					
100	76	18387.02	0.66	104.43	2.21

One can see that while the number of OD pairs is less than the number of arcs in a road network, the developed linear multi-output regression model works very well. However, once the number of OD pairs exceeds the number of arcs, the quality of prediction drastically degrades. Table 3 shows that other models face the same problem.

The equilibrium travel time value (7) seems to be an important characteristic of traffic congestion. We extract data on these values from the generated 10 000 vectors of traffic assignment patterns and put them in accordance with the 10 000 vectors of feasible demands. We taught different models to predict travel demands using equilibrium travel time values as the input layer. The most important metrics for different models are given in Table 4.

Table 4. Metrics for some models.

Number of OD pairs	Input layer dimension	MSE	R^2	MAE	MAPE
Linear multi-output regression model					
5	5	269234.95	0.95	389.25	0.65
Gradient boosting					
5	5	30628.13	0.99	97.53	0.18
Neural network					
5	5	341104.4	0.93	365.55	0.58

Gradient boosting (GB) appears to fit quite well for this task. We proceed with testing GB in cases with a larger number of OD pairs.

Table 5. Metrics for gradient boosting.

Number of OD pairs	Input layer dimension	MSE	R^2	MAE	MAPE
24	24	42850.28	0.81	152.47	1.34

We should mention that 552 is the maximum number of OD pairs for the Sioux-Falls road network. However, even for 24 OD pairs, the model demonstrates a deterioration of metrics values (table 5). As a next research step, we extract data on the numbers of actually used routes from the generated 10 000 vectors of traffic assignment patterns and put them in accordance with the 10 000 vectors

of feasible demands. We taught a linear multi-output regression model to predict travel demands using equilibrium travel time values and the number of actually used routes as the input layer. The most important metrics for our linear multi-output regression model are given in Table 6.

Table 6. Metrics for linear multi-output regression model.

Number of OD pairs	Input layer dimension	MSE	R^2	MAE	MAPE
5	10	335731.19	0.94	429.9	1.58

One can see that the results are even worse than those without data on the number of actually used routes. Perhaps additional data appears to be correlated traits that prevent the model from better learning (fig. 5).

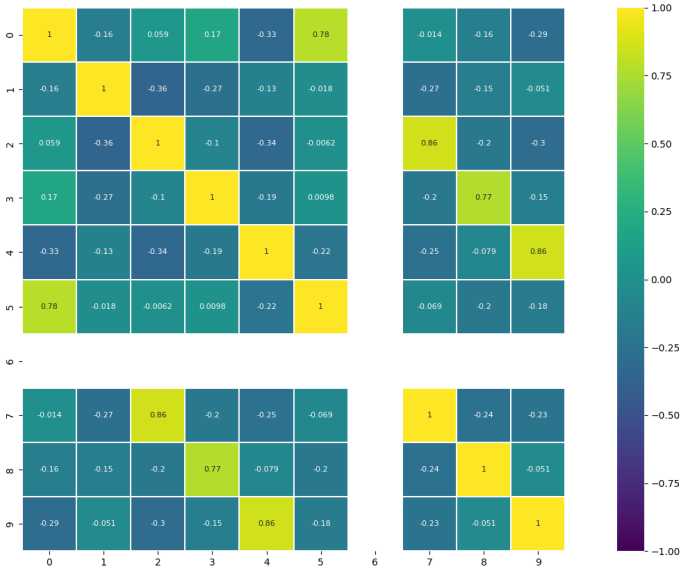


Fig. 5. Correlation matrix.

Therefore, our computational study leads to the conclusion that the best scenario for the implementation of such a tool as a multi-output regression for travel demand estimation is a smaller number of OD pairs compared to the number of arcs in a road network. Indeed, it is this scenario that appears to demonstrate the best model performance in terms of metrics. Moreover, when we tried to

increase the dimension of the input layer using additional information besides traffic congestion, we faced the problem of correlated traits. In other words, traffic congestion values seem to be the most informative data to estimate travel demands in the case of multiple OD pairs, but the number of these pairs should be less than the number of arcs in a road network.

5 Conclusion

The present paper was focused on the travel demand search task, formulated as an inverse traffic assignment problem. We developed multi-output regression models to estimate travel demand values for a road network with multiple origin-destination pairs. We used solutions to the non-linear optimization problem of equilibrium traffic assignment under different demand patterns to generate data for training and test sets. Our computational study led us to the conclusion that traffic congestion values were the most informative data to estimate travel demands in the case of multiple OD pairs, but the number of these pairs should be less than the number of arcs in a road network. Such implementation conditions provided the best performance of the multi-output regression model when estimating travel demand values.

References

1. Bar-Gera, H.: Primal method for determing the most likely route flows in large road network. Transport. Sci, **40(3)**, 269–286 (2006)
2. Bell, M. G., Shield, C. M., Busch, F., Kruse, C.: A stochastic user equilibrium path flow estimator. Transport. Res. Part C, **5(34)**, 197–210 (1997)
3. Beyer, H.-G., Schwefel, H.-P.: Evolution strategies. Natural Computing, **1**, 3–52 (2002)
4. Bierlaire, M.: The total demand scale: a new measure of quality for static and dynamic origin-destination trip tables. Transportation Research Part B, **36**, 837–850 (2002)
5. Brederode, L., Pel, A., Wismans, L., Rijksen, B., Hoogendoorn, S.: Travel demand matrix estimation for strategic road traffic assignment models with strict capacity constraints and residual queues. Transportation Research Part B, **167**, 1–31 (2023)
6. Fisher, F.: A Priori Information and Time Series Analysis. North-Holland Publishing Co. (1962)
7. Fisk, C.: On combining maximum entropy trip matrix estimation with user optimal assignment. Transport. Res. Part B, **22(1)**, 69–73 (1988)
8. https://github.com/bstabler/TransportationNetworks/tree/master/SiouxFalls
9. Isard, W.: Methods of Regional Analysis: An Introduction to Regional Science. John Wiley and Sons, Inc. (1960)
10. Krylatov, A., Raevskaya, A., Zakharov, V.: Travel Demand Estimation in Urban Road Networks as Inverse Traffic Assignment Problem. Transport and Telecommunication, **22(3)**, 287–300 (2021)
11. Krylatov, A.: Reduction of a minimization problem for a convex separable function with linear constraints to a fixed point problem. Journal of Applied and Industrial Mathematics, **12(1)**, 98–111 (2018)

12. Krylatov, A., Shirokolobova, A.: Projection approach versus gradient descent for network's flows assignment problem. Lecture Notes in Computer Science, **10556**, 345–350 (2017)
13. Krylatov, A., Shirokolobova, A., Zakharov, V.: OD-matrix estimation based on a dual formulation of traffic assignment problem. Informatica (Slovenia), **40(4)**, 393–398 (2016)
14. Kumarage, S., Yildirimoglu, M., Zheng, Z.: Demand and state estimation for perimeter control in large-scale urban networks. Transportation Research Part C, **153**, 104184 (2023)
15. Lundgren, J. T., Peterson, A.: A heuristic for the bilevel origin-destination matrix estimation problem. Transportation Research Part B, **42**, 339–354 (2008)
16. Makowski, G. G., Sinha, K. C.: A statistical procedure to analyze partial license plate numbers. Transpn. Res., **10**, 131–132 (1976)
17. Nguyen, S.: Estimating an OD matrix from network data: A network equilibrium approach. Publication 60, Centre de Recherche sur les Transports, Universitét de Motréal. (1977)
18. Patriksson, M.: The traffic assignment problem: models and methods. VSP, Utrecht (1994)
19. Raevskaya, A., Krylatov, A., Ageev, P., Kuznetsova, D.: Restriction Set Design for Travel Demand Values in an Urban Road Network. Lecture Notes in Networks and Systems, **502**, 110–120 (2022)
20. Sheffi, Y.: Urban transportation networks: equilibrium analysis with mathematical programming methods. Prentice-Hall, Inc, Englewood Cliffs (1985)
21. Shen, W., Wynter, L.: A new one-level convex optimization approach for estimating origin-destination demand. Transportation Research Part B, **46**, 1535–1555 (2012)
22. Sun, R., Fan, Y.: Stochastic OD demand estimation using stochastic programming. Transportation Research Part B, **183**, 102943 (2024)
23. Wei, C., Asakura, Y.: A bayesian approach to traffic estimation in stochastic user equilibrium networks. Transport. Res. Part C, **36**, 446–459 (2013)
24. Yang, H., Sasaki, T., Iida, Y., Asakura, Y.: Estimation of origin-destination matrices from link traffic counts on congested networks. Transportation Research Part B, **26(6)**, 417–434 (1992)

Approximate dynamic programming for inland empty container inventory management *

Sangmin Lee[1,2][0000−0001−8495−6779] and Trine Krogh Boomsma[2][0000−0001−5127−1888]

[1] Maersk, Esplanaden 50, 1263 Copenhagen K, Denmark sangmin.lee@maersk.com
[2] University of Copenhagen, Universitetsparken 5, 2100 Copenhagen Ø, Denmark trine@math.ku.dk

Abstract. Empty container repositioning (ECR) is crucial in handling global trade imbalances by managing the flow and storage of empty containers to effectively accommodate customer demands and returns. We formulate a Markov decision process that accounts for practical characteristics of ECR, including multiple transportation modes and lead times, and uncertainty and serial correlation in net container inflows. To determine a cost-minimising repositioning policy for real-life scenarios, we employ a stochastic approximate dynamic programming approach, integrated with statistical techniques, such as convex regression and Latin hyper cube sampling. We highlight the advantages of incorporating exogenous variables into the approximation model. A case study with historical daily data on empty container in- and out-flows demonstrates the effective control of on-hand inventory levels while optimising holding and leasing costs. Moreover, we quantify the benefits of leveraging all transportation modes, with cost reduction potentials of up to 26.05%. Finally, we evaluate the robustness of our algorithm under variations in key parameters.

Keywords: Empty container repositioning · Approximate dynamic programming · Inventory control

1 Introduction

Empty Container Repositioning (ECR) is critical to optimising container transport networks, ensuring efficient flow and storage of empty containers to effectively meet customer demands and manage global trade imbalances [8]. ECR encounters practical challenges, including the dynamic nature of freight transportation operations, uncertainties in container inflows and outflows, and diverse transportation modes for repositioning empty containers, each with unique costs, lead times and capacities [7]. To tackle these challenges, this paper adopts the stochastic approximate dynamic programming (ADP) approach [5], which inherently captures the sequential decision-making process and uncertainties over time while integrating statistical approximation techniques for scalability.

* Sangmin Lee acknowledges funding from Innovation Fund Denmark (grant number [1044-00022B]).

In the literature tackling the ECR problem at the regional level, [2] considers multiple ports and stochastic demand but does not account for multiple modes of transport and serial correlation in demand. On the other hand, [3] incorporates multiple inland container stations and multiple modes of transport, yet assuming zero lead time and deterministic demand. Our work extends a single-depot ECR problem presented in [8] to more generic case, incorporating multiple modes of transport with lead times and addressing stochastic and serially correlated net container inflows.

The main contributions of this paper are as follows: we formulate a Markov decision process (MDP) for managing inland empty container inventory, which considers multiple transportation modes and lead times, and addresses uncertainties and serial correlation in empty net container inflows; we also develop an ADP algorithm that exploits the structure of the MDP and can scale to real-life instances. Through a case study with historical daily data on empty container inflow and outflow, we examine the efficiency of our optimal policy estimate, and investigate the benefits of leveraging multiple transportation modes for container repositioning and incorporating net inflow information into our approximation model. We further assess convergence and sensitivity to critical parameters, such as the variance of white noise affecting net inflows and the unit cost of leasing containers.

2 Problem formulation

To formulate the MDP, consider a discrete, finite-horizon inventory system with a single-depot, managing a specific type of containers and accounting for lost sales[3]. A point in time is indexed by $t \in \{0, 1, \ldots, T\}$, where T represents the end of the time horizon, and time periods are denoted by $[t, t+1[$, including time point t but not $t+1$.

Decisions on repositioning empty containers are made at the discrete points in time $t = 0, 1, \ldots, T-1$. A repositioning-in order relocates empty containers from external locations, such as ports or other depots, into the internal depot, while a repositioning-out order does the opposite. Repositioning-in orders involve different transportation modes from the set $\mathcal{M}$. These modes are characterised by distinct lead times, unit ordering costs and capacities, and indexed by $m = 1, 2, \ldots, M$, where the modes are sorted according to increasing lead times. We assume that

$$l^{(m')} > l^{(m)} \text{ and } c^{(m')} < c^{(m)} \quad \forall m, m' \in \mathcal{M} : m' > m,$$

[3] Throughout the paper, we denote $x^+ = \max(x, 0)$ and $x^- = \max(-x, 0)$ for any real number $x \in \mathbb{R}$. We use lower-case bold letters ($\mathbf{x}, \mathbf{y}$, etc.) for column vectors and calligraphic letters ($\mathcal{X}$, $\mathcal{Y}$, etc.) for sets. However, the sets of real and integer numbers are denoted by $\mathbb{R}$ and $\mathbb{Z}$, and the sets of non-negative numbers by $\mathbb{R}_+$ and $\mathbb{Z}_+$. When referring to (exogenous) random variables, uppercase letters denote the variables themselves, while lowercase letters represent their realisations.

where $l^{(m)} \in \{0, 1, \ldots, T\}$, $c^{(m)} \in \mathbb{R}_+$ and $k^{(m)} \in \mathbb{Z}_+$ denote the lead time, unit ordering cost and capacity of transportation mode m, respectively. These assumptions indicate that transportation modes with shorter lead times provide greater flexibility in handling uncertainties in demand but have higher ordering costs than those with longer lead times. For simplicity, we assume that repositioning-out orders are facilitated by a specific transportation mode not in the set $\mathcal{M}$. Once repositioning-out orders are placed, the containers are promptly dispatched. The unit ordering cost and transportation capacity for repositioning the containers out of the depot are denoted by $c^{\text{out}} \in \mathbb{R}_+$ and $k^{\text{out}} \in \mathbb{Z}_+$, respectively. To render the problem nontrivial, we adopt the same assumptions as [8], i.e., that repositioning-in costs are lower than penalty cost for lost-sales and repositioning-out costs are lower than holding costs:

$$c^{(m)} < c^{\text{l}} \quad \forall m \in \mathcal{M} \text{ and } c^{\text{out}} < c^{\text{h}},$$

where c^{h} and c^{l} denote the unit holding and penalty cost, respectively.

In addition to endogenous repositioning decisions, the depot manages exogenous random inflow and outflow of empty containers. Exogenous inflow involves empty containers returns by customers, while exogenous outflow represents empty container customer demands. We make the following assumptions concerning the exogenous random information. Let $\{W_t : t = 0, 1, \ldots\}$ denote a discrete-time stochastic process on the state space $\mathcal{W}$ and with the Markov property, i.e., $P(W_{t+1} \in A | W_0, \ldots, W_t) = P(W_{t+1} \in A | W_t)$ for $A \subseteq \mathcal{W}$ and $t = 0, \ldots, T-1$. The net inflow of empty containers into the depot during period $[t, t+1[$ is represented by a discrete random variable $Z_{t+1} = h(W_{t+1}, t)$, where h is a deterministic function of W_{t+1} and t. For $Z_{t+1} = z_{t+1}$, a positive value $z_{t+1} > 0$ indicates a net inflow, while a negative value $z_{t+1} < 0$ presents a net outflow. We assume that the random variable W_{t+1} is realised at the end of the time period, just before time $t+1$, and so is Z_{t+1}. If the inventory level is insufficient to meet all customer demands, excess demand is satisfied by leasing additional containers from lessors. Thus, the unit penalty cost for lost-sales (c^{l}) reflects the unit leasing cost and is referred to this throughout the remainder of the paper.

For each $t \in \{0, \ldots, T-1\}$, the inventory system undergoes periodic review through the following sequence of events within time period $[t, t+1[$: (1) repositioning-in orders, placed in previous time points and due at time t, arrive; (2) the current state of the inventory system is observed; (3) repositioning decisions are made, given the state of the inventory system and the exogenous information available at time t, and repositioning-in and -out orders are placed; (4) repositioned empty containers are dispatched for the repositioning-out order; (5) the net inflow of empty containers during period $[t, t+1[$ is realised; (6) ordering, holding and leasing costs accrue; We note that the first five events occur in the beginning of the period (right after time t), while the last two events take place in the end of the period (just before time $t+1$). A summary of the sequence of events and the corresponding notation is provided in Table 1.

For each $t \in \{0, \ldots, T\}$, the so-called pre-decision state of the internal inventory system is represented by an $l^{(M)}$-vector $\mathbf{s}_t = (s_{0,t}, s_{1,t}, \ldots, s_{l^{(M)}-1,t})^\top$,

Table 1. The sequence of events in the periodic review of the inventory system within time period $[t, t+1[$.

Events	notation
(1) repositioning-in orders arrive	$x^{(1)}_{t-l^{(1)}}, \ldots, x^{(M)}_{t-l^{(M)}}$
(2) inventory state is observed	$s_{0,t} = [s^{\mathrm{x}}_{0,t-1} + z_t]^+ + \sum_m x^{(m)}_{t-l^{(m)}}$
(3) re-positioning orders are placed	$x^{\mathrm{out}}_t, x^{(1)}_t, \ldots, x^{(M)}_t$
(4) re-positioning out orders are dispatched	$s^{\mathrm{x}}_{0,t} = s_{0,t} - x^{\mathrm{out}}_t$
(5) net inflow is realised	$z_{t+1} = h(w_{t+1}, t)$
(6) one-period cost accrues	$C(s_{0,t}, w_{t+1}, \mathbf{x}_t)$

where $s_{0,t}$ denotes the inventory level of empty containers at time t following the arrival of repositioning-in orders that are due at time t but before the placement of new orders, and $s_{i,t}$, for $i = 1, 2, \ldots, l^{(M)} - 1$, denotes cumulative repositioning-in orders scheduled to arrive at time $t + i$. The set of pre-decision states is denoted by $\mathcal{S} = \mathcal{S}_0 \times \mathcal{S}_1 \times \cdots \times \mathcal{S}_{l^{(M)}-1}$, where:

$$\mathcal{S}_0 = \mathbb{Z}_+, \quad \mathcal{S}_i = \{0, 1, \ldots, \sum_{m=1}^{M} k^{(m)}\} \quad \forall i \in \{1, \ldots, l^{(1)} - 1\},$$

$$\mathcal{S}_i = \{0, 1, \ldots, \sum_{\tilde{m}=m}^{M} k^{(\tilde{m})}\} \quad \forall i \in \{l^{(m-1)}, \ldots, l^{(m)} - 1\},\ m \in \{2, \ldots, M\}.$$

The repositioning decisions made at time $t \in \{0, \ldots, T-1\}$ are denoted by an $(M+1)$-vector $\mathbf{x}_t = (x^{\mathrm{out}}_t, x^{(1)}_t, x^{(2)}_t, \ldots, x^{(M)}_t)^\top$, where x^{out}_t denotes the size (number of containers) of the repositioning-out order placed at time t, while $x^{(m)}_t$ represents the size of the repositioning-in order placed at time t for transportation mode m. The decision variables must satisfy the following constraints:

$$x^{\mathrm{out}}_t \le s_{0,t}, \quad x^{\mathrm{out}}_t \le k^{\mathrm{out}}, \quad x^{(m)}_t \le k^{(m)}, \qquad \forall m \in \mathcal{M}, \tag{1}$$

$$x^{\mathrm{out}}_t,\ x^{(m)}_t \in \mathbb{Z}_+, \qquad \forall m \in \mathcal{M}. \tag{2}$$

Constraints (1) ensure that the repositioning-out order is limited by the current inventory level and its transportation capacity, while feasible repositioning-in orders do not exceed the capacity of the corresponding transportation modes. Constraints (2) define domains for decision variables, enforcing them to be non-negative integers. Given $s_{0,t}$, the set of solutions satisfying the constraints (1) - (2) is denoted by $\mathcal{X}(s_{0,t})$. The set of all feasible solutions is denoted by $\mathcal{X} = \cup_{s \in \mathcal{S}_0} \mathcal{X}(s)$.

In addition to the pre-decision state, we define the post-decision state of the internal inventory system for each $t \in \{0, \ldots, T-1\}$. The post-decision states are denoted by an $(l^{(M)}+1)$-vector $\mathbf{s}^{\mathrm{x}}_t = (s^{\mathrm{x}}_{0,t}, s^{\mathrm{x}}_{1,t}, \ldots, s^{\mathrm{x}}_{l^{(M)},t})^\top$ and determined

by $\mathbf{s}_t^{\mathrm{x}} = g(\mathbf{s}_t, \mathbf{x}_t)$, where:

$$g_i(\mathbf{s}_t, \mathbf{x}_t) = \begin{cases} s_{i,t} - x_t^{\text{out}} & \text{if } i = 0, \\ s_{i,t} + x_t^{(m)} & \text{if } i = l^{(m)}, \quad m = 1, \dots, M-1, \\ x_t^{(M)} & \text{if } i = l^{(M)}, \\ s_{i,t} & \text{otherwise.} \end{cases}$$

The first component, $s_{0,t}^{\mathrm{x}}$, represents the inventory level of empty containers immediately after the dispatch of empty containers for the repositioning-out order but prior to the realisation of the net inflow z_{t+1}. The remaining components indicate outstanding repositioning-in orders, including those placed at time t. That is, for each $m = 1, \dots, M-1$, $s_{l^{(m)},t}$ includes the repositioning-in order for transportation mode m placed at time t, i.e., $x_t^{(m)}$, arriving at time $t + l^{(m)}$. Given $s_{0,t}$, the set of post-decision states is represented by $\mathcal{S}^{\mathrm{x}}(s_{0,t}) = \mathcal{S}_0^{\mathrm{x}}(s_{0,t}) \times \mathcal{S}_1^{\mathrm{x}} \times \cdots \times \mathcal{S}_{l^{(M)}}^{\mathrm{x}}$, where:

$$\mathcal{S}_0^{\mathrm{x}}(s_{0,t}) = \{0, 1, \dots, s_{0,t}\}, \quad \mathcal{S}_i^{\mathrm{x}} = \{0, 1, \dots, \sum_{m=1}^{M} k^{(m)}\} \quad \forall i \in \{1, \dots, l^{(1)}\},$$

$$\mathcal{S}_i^{\mathrm{x}} = \{0, 1, \dots, \sum_{\tilde{m}=m}^{M} k^{(\tilde{m})}\} \quad \forall i \in \{l^{(m-1)} - 1, \dots, l^{(m)}\},\ m \in \{2, \dots, M\}.$$

We denote the set of all possible post-decision states by $\mathcal{S}^{\mathrm{x}} = \cup_{s \in \mathcal{S}_0} \mathcal{S}^{\mathrm{x}}(s)$.

Given $\mathbf{s}_t^{\mathrm{x}}$ and w_{t+1}, the pre-decision state of the internal inventory system at time $t+1$ is determined by $\mathbf{s}_{t+1} = f(\mathbf{s}_t^{\mathrm{x}}, w_{t+1}, t)$, where:

$$f_i(\mathbf{s}_t^{\mathrm{x}}, w_{t+1}, t) = \begin{cases} [s_{i,t}^{\mathrm{x}} + z_{t+1}]^+ + s_{i+1,t}^{\mathrm{x}} & \text{if } i = 0, \\ s_{i+1,t}^{\mathrm{x}} & \text{otherwise,} \end{cases} \text{ with } z_{t+1} = h(w_{t+1}, t).$$

The term $[s_{i,t}^{\mathrm{x}} + z_{t+1}]^+ + s_{i+1,t}^{\mathrm{x}}$ represents the inventory level on hand, including the net inflow of empty containers into the depot during period $[t, t+1[$ and repositioning-in orders arriving at time $t+1$. Note that the pre-decision dynamics of the internal inventory system can be expressed by the composite function $\mathbf{s}_{t+1} = f(g(\mathbf{s}_t, \mathbf{x}_t), W_{t+1}, t)$.

At the end of each period $[t, t+1[$, the one-period cost is given by the function:

$$C(s_{0,t}, w_{t+1}, \mathbf{x}_t) = \mathbf{c}^\top \mathbf{x}_t + c^{\mathrm{h}}[s_{0,t} - x_t^{\text{out}} + z_{t+1}]^+ + c^{\mathrm{l}}[s_{0,t} - x_t^{\text{out}} + z_{t+1}]^- \quad \text{with } z_{t+1} = h(w_{t+1}, t),$$

where $\mathbf{c} = (c^{\text{out}}, c^{(1)}, c^{(2)}, \dots, c^{(M)})^\top$. The first term presents ordering costs and the second and third term represent holding and leasing costs, respectively.

The empty container inventory control problem consists of making repositioning decisions such as to minimize expected total costs during $[0, T[$:

$$\min_{\pi} \mathbb{E}_\pi \left[\sum_{t=0}^{T-1} C(s_{0,t}, W_{t+1}, \mathbf{x}_t) \;\middle|\; \mathbf{s}_0, w_0 \right] \tag{3}$$

$$\text{s.t.} \quad \mathbf{x}_t = \pi_t(\mathbf{s}_t, W_t),\ \mathbf{s}_{t+1} = f(g(\mathbf{s}_t, \mathbf{x}_t), W_{t+1}, t),\ \mathbf{x}_t \in \mathcal{X}(s_{0,t}),$$

where $\pi = (\pi_0, \pi_1, \ldots, \pi_{T-1})$ is a policy, i.e., π_t is a function that maps a pair of a pre-decision state and exogenous variable $(\mathbf{s}_t, w_t)$ to a repositioning decision $\mathbf{x}_t$. The subscript π of the expectation $\mathbb{E}_\pi[\cdot]$ indicates that decisions are determined by the policy π. It should be remarked that decisions $\mathbf{x}_t$ depend on the realisation w_t of W_t, but must be independent of the realizations of $W_t, \ldots, W_T$.

By the application of dynamic programming [6], the problem (3) can be solved sequentially, decomposing it into subproblems for each time period. For $t \in \{0, \ldots, T-1\}$, let $V_t(\mathbf{s}_t, w_t)$ denote the pre-decision value function, capturing the minimal expected total costs during period $[t, T[$, given $\mathbf{s}_t$, w_t. By [6], these value functions satisfy the Bellman optimality equations [1]:

$$V_t(\mathbf{s}_t, w_t) = \min_{\mathbf{x}_t \in \mathcal{X}(s_{0,t})} \mathbb{E}\big[C(s_{0,t}, W_{t+1}, \mathbf{x}_t) + V_{t+1}(f(g(\mathbf{s}_t, \mathbf{x}_t), W_{t+1}, t), W_{t+1}) | w_t\big] \\ \forall \mathbf{s}_t \in \mathcal{S},\ w_t \in \mathcal{W},\ t \in \{0, 1, \ldots, T-1\}, \tag{4}$$

where $V_T(\mathbf{s}_T, w_T) \equiv 0$ for all $\mathbf{s}_T \in \mathcal{S}$ and $w_T \in \mathcal{W}$. That is, the pre-decision value can be determined by the minimal expected current one-period cost during period $[t, t+1[$ and the future costs during $[t+1, T[$, given $\mathbf{s}_t$, w_t. Solving (4) for each $t \in \{0, 1, \ldots, T-1\}$, the optimal decisions, denoted by $\pi_t^*(\mathbf{s}_t, w_t)$, are:

$$\pi_t^*(\mathbf{s}_t, w_t) = \operatorname*{argmin}_{\mathbf{x}_t \in \mathcal{X}(s_{0,t})} \mathbb{E}\big[C(s_{0,t}, W_{t+1}, \mathbf{x}_t) + V_{t+1}(f(g(\mathbf{s}_t, \mathbf{x}_t), W_{t+1}, t), W_{t+1}) | w_t\big] \tag{5}$$

The Bellman equation can equivalently be expressed using post-decision states. For each $t \in \{0, 1, \ldots, T-1\}$, we define the post-decision value function $V_t^{\mathrm{x}}(\mathbf{s}_t^{\mathrm{x}}, w_t)$, capturing expected minimal total costs during period $[t+1, T[$, given $\mathbf{s}_t^{\mathrm{x}}$, w_t:

$$V_t^{\mathrm{x}}(\mathbf{s}_t^{\mathrm{x}}, w_t) = \mathbb{E}\big[V_{t+1}(f(\mathbf{s}_t^{\mathrm{x}}, W_{t+1}, t), W_{t+1}) | w_t\big]. \tag{6}$$

Using (6) and (4), the pre-decision value function can be expressed as:

$$V_t(\mathbf{s}_t, w_t) = \min_{\mathbf{x}_t \in \mathcal{X}(s_{0,t})} C^{\mathrm{e}}(s_{0,t}, w_t, \mathbf{x}_t) + V_t^{\mathrm{x}}(g(\mathbf{s}_t, \mathbf{x}_t), w_t), \tag{7}$$

where $C^{\mathrm{e}}(s_{0,t}, w_t, \mathbf{x}_t) = \mathbb{E}\left[C(s_{0,t}, W_{t+1}, \mathbf{x}_t) | w_t\right]$ denotes the expected value of the one-period cost, given $W_t = w_t$. Similarly, using (7) and (6), we obtain the following equations:

$$V_{t-1}^{\mathrm{x}}(\mathbf{s}_{t-1}^{\mathrm{x}}, w_{t-1}) = \mathbb{E}\left[\min_{\mathbf{x}_t \in \mathcal{X}(s_{0,t})} C^{\mathrm{e}}(s_{0,t}, W_t, \mathbf{x}_t) + V_t^{\mathrm{x}}(g(\mathbf{s}_t, \mathbf{x}_t), W_t) \big| w_{t-1}\right] \\ \text{s.t.} \quad \mathbf{s}_t = f(\mathbf{s}_{t-1}^{\mathrm{x}}, w_t, t-1), \tag{8}$$

where $V_{T-1}^{\mathrm{x}}(\mathbf{s}_{T-1}^{\mathrm{x}}, w_{T-1}) = 0$. Using the post-decision value functions, the optimal decisions are:

$$\pi_t^*(\mathbf{s}_t, w_t) = \operatorname*{argmin}_{\mathbf{x}_t \in \mathcal{X}(s_{0,t})} C^{\mathrm{e}}(s_{0,t}, w_t, \mathbf{x}_t) + V_t^{\mathrm{x}}(g(\mathbf{s}_t, \mathbf{x}_t), w_t). \tag{9}$$

Algorithm 1 Backward ADP algorithm

1: **for** $t = T-2, T-3, \ldots, 0$ **do**
2: **for** $n = 1, 2, \ldots, N$ **do**
3: sample $\mathbf{s}_t^{\mathrm{x},n}$ from $\mathcal{S}^{\mathrm{x}}$ and w_t^n from $\mathcal{W}$
4: **for** $j = 1, 2, \ldots, J$ **do**
5: sample w_{t+1}^j given w_t^n
6: solve the minimisation problem (11)
7: **end for**
8: compute the sample mean (10)
9: **end for**
10: estimate $\boldsymbol{\theta}_t$ by fitting $\hat{V}_t^{\mathrm{x}}(\mathbf{s}_t^{\mathrm{x},n}, w_t^n; \boldsymbol{\theta}_t)$ to $\{\hat{v}_t^{\mathrm{x},n}\}_{n=1,\ldots,N}$
11: **end for**

Using pre-decision value functions, the dynamic programming problem involves a minimisation of the expected value, (5), i.e., the solution of a stochastic optimisation problem. With post-decision value functions, however, it involves solving a deterministic minimisation problem (9), provided that the expected one-period cost C^{e} is known. For this reason, we focus on the post-decision value formulation (8).

3 Approximate dynamic programming

When dealing with large state and action spaces, classical dynamic programming methods, such as value iteration [6], become computationally intensive, as they entail storing values $V_t(\mathbf{s}_t, w_t)$ and solving the recursive equations (4) for all $\mathbf{s}_t \in \mathcal{S}$, $w_t \in \mathcal{W}$ and $t \in \{0, 1, \ldots, T-1\}$. Given a state, the action space is finite but grows significantly with the number of transportation modes ($|\mathcal{M}|$) and their capacities ($k^{(m)}$). As we do not assume a storage limit for the depot, however, the size of the pre- and post-decision state space of the internal inventory system is countable but infinite.

To address the curse-of-dimensionality in dynamic programming [5], we approximate the post-decision value function using a parameterised model, denoted by $\hat{V}_t^{\mathrm{x}}(\mathbf{s}_t^{\mathrm{x}}, w_t; \boldsymbol{\theta}_t)$, where $\boldsymbol{\theta}_t \in \mathbb{R}^d$ is a parameter vector that allows for access to function values for all $(\mathbf{s}_t^{\mathrm{x}}, w_t) \in \mathcal{S}^{\mathrm{x}} \times \mathcal{W}$.

The parameter vectors $\{\boldsymbol{\theta}_t\}_{t\in\{0,1,\ldots,T-2\}}$ are estimated using backward approximate dynamic programming [5], as outlined in Algorithm 1. In each iteration (time point) from $t = T-2$ to 0, we obtain N sample post-decision states of the internal inventory system ($\mathbf{s}_t^{\mathrm{x}}$) and the exogenous random variable (w_t) from their respective space (line 3). For each sample $(\mathbf{s}_t^{\mathrm{x},n}, w_t^n)$, indexed by n, we further sample J exogenous random variable w_{t+1}^j, indexed by j, given w_t^n (line 5).

For the post-decision state of the internal inventory system ($\mathbf{s}_t^{\mathrm{x}}$), we employ the Latin hypercube sampling (LHS) method [4], known for its efficiency in representing the sample space compared to the traditional random sampling in Monte Carlo studies. To avoid an infinite state space for sampling, we obtain upper bounds for the sets $\mathcal{S}_0^{\mathrm{x}}(s_{0,t})$, by generating $\tilde{N}$ simulation paths using a

random policy that selects random actions, and selecting the maximum of $\tilde{N}$ simulated $s^{\mathrm{x}}_{0,t}$ for each time point t. For realisations of the exogenous random variable (w_t), we sample from its stationary distribution.

For each sample $(\mathbf{s}_t^{\mathrm{x},n}, w_t^n)$, a bootstrapped estimate $\hat{v}_t^{\mathrm{x},n}$ of $V_t^{\mathrm{x}}(\mathbf{s}_t^{\mathrm{x},n}, w_t^n)$ is computed by solving J minimisation problems (11) (line 6) and computing the sample mean (10) (line 8):

$$\hat{v}_t^{\mathrm{x},n} = \frac{1}{J}\sum_{j=1}^{J} \hat{a}_t^{j,n} \quad \text{where} \tag{10}$$

$$\hat{a}_t^{j,n} = \min_{\mathbf{x}_{t+1}\in\mathcal{X}(s_{0,t+1})} C^{\mathrm{e}}(s_{0,t+1}, w_{t+1}^j, \mathbf{x}_{t+1}) + \hat{V}_{t+1}^{\mathrm{x}}(\mathbf{s}_{t+1}^{\mathrm{x}}, w_{t+1}^j; \boldsymbol{\theta}_{t+1}) \tag{11}$$

$$\text{s.t.} \quad \mathbf{s}_{t+1}^{\mathrm{x}} = g(\mathbf{s}_{t+1}, \mathbf{x}_{t+1}),\ \mathbf{s}_{t+1} = f(\mathbf{s}_t^{\mathrm{x},n}, w_{t+1}^j, t).$$

Indeed, each iteration requires solving $N \times J$ minimisation problems.

Finally, we estimate the parameter vector $\boldsymbol{\theta}_t$ by fitting the approximation model $\hat{V}_t^{\mathrm{x}}(\mathbf{s}_t^{\mathrm{x},n}, w_t^n; \boldsymbol{\theta}_t)$ to the N bootstraped estimates $\{\hat{v}_t^{\mathrm{x},n}\}_{n=1,2,\ldots,N}$, which involves solving the following problem:

$$\hat{\boldsymbol{\theta}}_t = \underset{\boldsymbol{\theta}_t}{\operatorname{argmin}} \frac{1}{N}\sum_{n=1}^{N} L(\hat{v}_t^{\mathrm{x},n}, \hat{V}_t^{\mathrm{x}}(\mathbf{s}_t^{\mathrm{x},n}, w_t^n; \boldsymbol{\theta}_t)), \tag{12}$$

where $\hat{\boldsymbol{\theta}}_t$ denotes the estimated parameter vector and L represents a loss function measuring how accurately the approximation model fits the data $\{\hat{v}_t^{\mathrm{x},n}\}_{n=1,2,\ldots,N}$. We employ the mean squared error as our chosen loss function, i.e.,

$$L(\hat{v}_t^{\mathrm{x},n}, \hat{V}_t^{\mathrm{x}}(\mathbf{s}_t^{\mathrm{x},n}, w_t^n; \boldsymbol{\theta}_t)) = (\hat{v}_t^{\mathrm{x},n} - \hat{V}_t^{\mathrm{x}}(\mathbf{s}_t^{\mathrm{x},n}, w_t^n; \boldsymbol{\theta}_t))^2.$$

Using the post-decision value function approximations, the optimal decisions are estimated by:

$$\pi_t^*(\mathbf{s}_t, w_t) \approx \hat{\pi}_t(\mathbf{s}_t, w_t) = \underset{\mathbf{x}_t\in\mathcal{X}(s_{0,t})}{\operatorname{argmin}} C^{\mathrm{e}}(s_{0,t}, w_t, \mathbf{x}_t) + \hat{V}_t^{\mathrm{x}}(g(\mathbf{s}_t, \mathbf{x}_t), w_t; \hat{\boldsymbol{\theta}}_t),$$

where $\hat{\pi}$ denotes the optimal policy estimate obtained by Algorithm 1.

4 Case study

To assess the effectiveness and performance of our proposed approach, we conduct a case study using historical data spanning from 1 January 2014, to 31 December 2019. This dataset comprises daily records of both outflow (customer demand) and inflow (returns from customers) of empty 40-foot dry containers for a specific depot located in a region experiencing a high deficit of empty containers due to greater export than import. In Section 4.1, we demonstrate the modelling of the exogenous random net inflow. Further algorithmic details, including the approximation of the expected one-period cost function C^{e} and the post-decision value functions $\hat{V}_t^{\mathrm{x}}$, are presented in Section 4.2.

4.1 Modelling the exogenous random net inflow

The stochastic process of net inflow $\{Z_t : t = 0, 1, \dots\}$ is determined by the model $Z_{t+1} = h(W_{t+1}, t) := \lfloor \eta_t + W_{t+1} \rfloor$, where η_t represents the sum of trend and seasonal components, W_{t+1} is a random component and $\lfloor x \rfloor$ denotes the largest integer less than or equal to $x \in \mathbb{R}$. For the exogenous stochastic process $\{W_t : t = 0, 1, \dots\}$, we use an auto-regressive model of order 1, i.e., $W_{t+1} = \phi W_t + \epsilon_{t+1}$, where ϕ is an auto-regressive parameter with $|\phi| < 1$ and $\epsilon_{t+1} \sim N(0, \sigma^2)$ is a Gaussian white noise with variance $\sigma^2 < \infty$. We assume that $\{\eta_t\}_{t\in\{0,1,\dots,T-1\}}$, ϕ and σ are given parameters.

To estimate the yearly trend component, we employ rolling moving averages, computing the average of values within a window centred around each day. This provides a smooth representation of the trend over time. For instance, to determine the trend value for 2 July 2019, we compute the average of values from 1 January 2019, to 31 December 2019. The yearly seasonal components are estimated by aggregating the detrended data across the same time points in each year. For example, the seasonal value for 2 July 2019 is determined by averaging detrended values for each 2 July across the years 2014 to 2019. The random component (W_t) represents the residual in the data after removing both the trend and seasonal components. Using ordinary least squares, we obtain parameter estimates of $\phi = 0.4660$ and $\sigma = 236.092$.

4.2 Approximation strategies

For the minimisation problem in (8) to be deterministic and convex, we approximate the expected one-period cost using a convex function by:

$$\hat{C}^{\mathrm{e}}(s_{0,t}, w_t, \mathbf{x}_t) = \mathbf{c}^\top \mathbf{x}_t + c^{\mathrm{h}} p(y) + c^{\mathrm{l}} p(-y), \tag{13}$$

where, considering the modelling of the exogenous random net inflow presented in Section 4.1,

$$y = -\frac{\tilde{\mu}}{\tilde{\sigma}}, \quad \tilde{\mu} = s_{0,t} - x_t^{\mathrm{out}} + \eta_t + \phi w_t - 0.5, \quad \tilde{\sigma} = \sqrt{\sigma^2 + \frac{1}{12}},$$
$$p(x) = -\tilde{\sigma} x \left(1 - \Phi\left(x\right)\right) + \tilde{\sigma}\varphi(x), \quad \varphi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}.$$

Here, $\Phi(\cdot)$ and $\varphi(x)$ denote the cumulative distribution function (cdf) and probability density function (pdf) of the standard Gaussian distribution, respectively.

For the post-decision value function approximation, we employ a convex quadratic function given by:

$$\hat{V}_t^{\mathrm{x}}(\mathbf{s}_t^{\mathrm{x}}, w_t; \boldsymbol{\theta}_t) = (\mathbf{s}_t^{\mathrm{x}}, w_t)^\top H(\mathbf{s}_t^{\mathrm{x}}, w_t) + \mathbf{q}^\top(\mathbf{s}_t^{\mathrm{x}}, w_t) + r,$$

where $\boldsymbol{\theta}_t$ consists of the columns of a symmetric positive semi-definite matrix $H \in \mathbb{R}^{(l^{(M)}+2)\times(l^{(M)}+2)}$, $\mathbf{q} \in \mathbb{R}^{l^{(M)}+2}$ and $r \in \mathbb{R}$, i.e., $d = (l^{(M)}+3)\times(l^{(M)}+2)+1$. Note that the minimisation problem in (8), with V_t^{x} and C^{e} replaced by $\hat{V}_t^{\mathrm{x}}$ and

$\hat{C}^{\text{re}}$, respectively, simplifies to a convex integer program (CIP) with upper and lower bounding constraints, since the transition function $g(\mathbf{s}_t, \mathbf{x}_t)$ is an affine transformation.

To reduce computation time, we further approximate the problem (11) by relaxing the integrality constraints in $\mathcal{X}(s_{0,t+1})$. This allows us to employ gradient-based methods for optimization.

5 Computational results

We fix the length of the time horizon T to be 14 days and select the period from 1 April 2019 to 14 April 2019 for trend and seasonal parameters η_t. The case study involves three transportation modes—truck, train and barge—available for repositioning-in orders, each mode indexed by 1, 2, and 3, respectively. All computational experiments discussed in this section are conducted on an `AMD EPYC 7763 64-Core Processor`.

To evaluate the performance of the optimal policy estimate $\hat{\pi}$, we generate scenarios, each representing a sample realisation of the initial pre-decision state $\mathbf{s}_0$ and the exogenous random variables w_t from $t = 0$ to T. The collection of generated scenarios, indexed by ω, is denoted by the set Ω. For the initial pre-decision state ($\mathbf{s}_0$), each component $s_{i,0}$ is sampled from its respective space $\mathcal{S}_i$, $i = 0, 1, \ldots, l^{(M)} - 1$, using the LHS method. The upper bound for $\mathcal{S}_0$ is set to 10^3. The initial exogenous random variable W_0 is sampled from its stationary distribution, while the subsequent variables $W_1, \ldots, W_T$ are generated using the AR(1) model presented in Section 4.1. The number of scenarios $|\Omega|$ is set to 10^4.

We measure the performance of the estimated optimal policy $\hat{\pi}$ by average total costs across all scenarios, denoted by $\rho(\hat{\pi})$:

$$\rho(\hat{\pi}) = \frac{1}{|\Omega|} \sum_{\omega=1}^{|\Omega|} \sum_{t=0}^{T-1} C(s_{0,t}, w_{t+1}(\omega), \hat{\pi}_t(\mathbf{s}_t, w_t(\omega))),$$

where $w_t(\omega)$ represents the realisation of the exogenous random variable W_t in scenario ω.

5.1 Convergence

We investigate the convergence of Algorithm 1 by varying the sampling sizes, i.e., N and J. For each combination of (N, J), we perform 10 replications of Algorithm 1, using different random seeds to sample $\{\mathbf{s}_t^{\text{x},n}\}_{n=1,\ldots,N}$ and $\{w_{t+1}^j\}_{j=1,\ldots,J}$. The performance of each replication is evaluated using the same set of scenarios Ω. The sample mean of $\rho(\hat{\pi})$ across 10 replications and its 95% confidence interval for each combination (N, J) are illustrated in Figure 1.

The optimal policy estimates $\hat{\pi}$ demonstrate convergence as both the number of samples of the post-decision state $\mathbf{s}_t^{\text{x},n}$ and the exogenous random variable w_{t+1}^j increases. With J fixed at 15, the average total cost decreases by 17% from $N = 25$ to $N = 750$. Likewise, with N fixed at 750, the cost decreases by 6% as

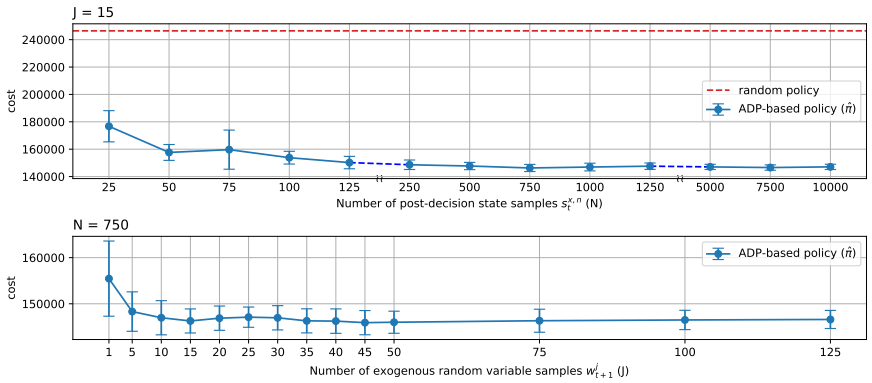


Fig. 1. Average total costs as a function of the number of samples.

J increases from 1 to 45. Beyond $N > 750$ and $J > 45$, the average total cost remains relatively steady, with variations within the range of 0.2% to 0.9% and 0.05% to 0.5%, respectively, which confirms the convergence. We designate the combination $(N, J) = (750, 45)$ as the baseline for subsequent analysis due to its effective and efficient performance.

5.2 Policy performance

To assess the performance of our baseline policy, we compare to a policy that randomly places repositioning-in orders at each point in time. The red dotted line of Figure 1 represents the average total costs of this random policy. Average total costs of the baseline policy estimate is, on average, 41% lower than that of the random policy, clearly demonstrating the ability of our algorithm to achieve cost efficiency.

We continue our analysis of the performance of the optimal policy estimate $\hat{\pi}$. The first panel of Figure 2 illustrates the on-hand inventory level $s_{0,t}$, the number of leased empty containers $[s_{0,t} - x_t^{\text{out}} + z_{t+1}]^-$ and the net inflow z_{t+1} for each time point $t = 0, \ldots, T-1$. The second panel depicts the repositioning-in and -out behaviour of the policy over the time horizon. These values are averaged across all scenarios in Ω.

Towards the very end of the time horizon, particularly at $t = 13$, the policy places relatively more repositioning-out orders, leaving only the necessary amount for optimising the one-period cost and repositioning out the rest. Evidently, during this phase, the policy has less incentive to maintain the on-hand inventory level as high as in preceding periods, since future customer demands are not considered. Note that the pre-decision state of the inventory system at the end of the time horizon does not affect the total cost, provided that $V_T \equiv 0$. While this assumption may not fully reflect reality, it is less of an issue, given that we frequently rerun the algorithm and update the policy, e.g., on a weekly basis. Therefore, our focus lies on understanding the behaviour of the policy during the initial and middle phases of the time horizon.

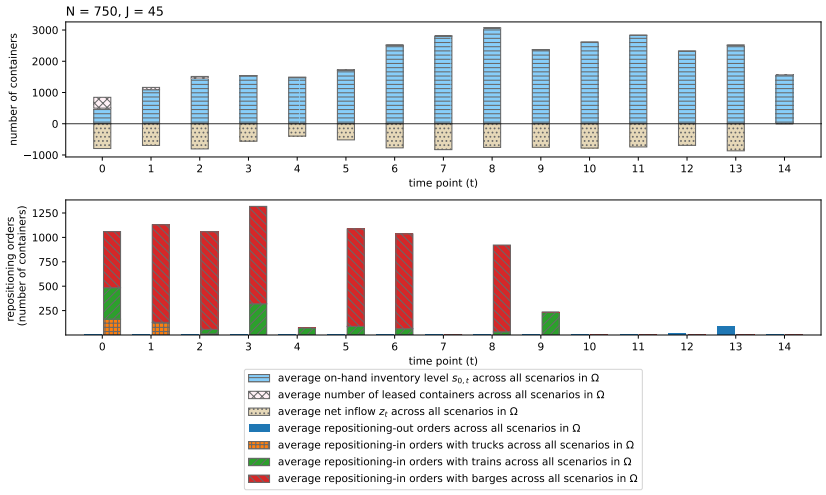


Fig. 2. Performance of the optimal policy estimate $\hat{\pi}$.

The inventory system experiences average net outflow (the negative of net inflow) throughout the entire time horizon, reflecting the serial correlation of exogenous random net flows. In response, average on-hand inventory level rises over time, reaching a level approximately 3-4 times higher than the average net outflow to avoid expensive leasing costs. Barges serve as the primary source for repositioning-in due to their low cost. Trucks, while the fastest but the most expensive, are rarely used except at the beginning of the time horizon ($t = 0$ and $t = 1$), when immediate replenishment of empty containers is necessary due to the low on-hand inventory level. Leasing is likewise necessary at the beginning of the time horizon.

We further investigate the impact of multiple transportation modes, using Figure 3. Each pair of bars in the figure represents the average values across all scenarios in Ω. More specifically, the bar on the left-hand-side indicates the average total costs of repositioning-in and -out, holding and leasing containers over the entire time horizon, while the right-hand-side bar illustrates the average total number of containers stored in the inventory and leased throughout the time horizon.

The first panel compares cases in which different transportation modes are accessible. Introducing either train or truck in addition to barge results in total cost reductions of 17.92% and 16.24%, respectively, compared to when only barge is available and the other two modes are unavailable. These reductions are primarily due to maintaining 70.62% and 29.06% more containers in the inventory and leasing 52.39% and 47.54% fewer containers, respectively. This indicates that the ordering capacity of barge alone is insufficient to maintain the on-hand inventory level as high as required. When all transportation modes are available, on the other hand, the total cost decreases by 26.05%, alongside storing 61.58% more containers in the inventory and leasing 61.39% fewer containers,

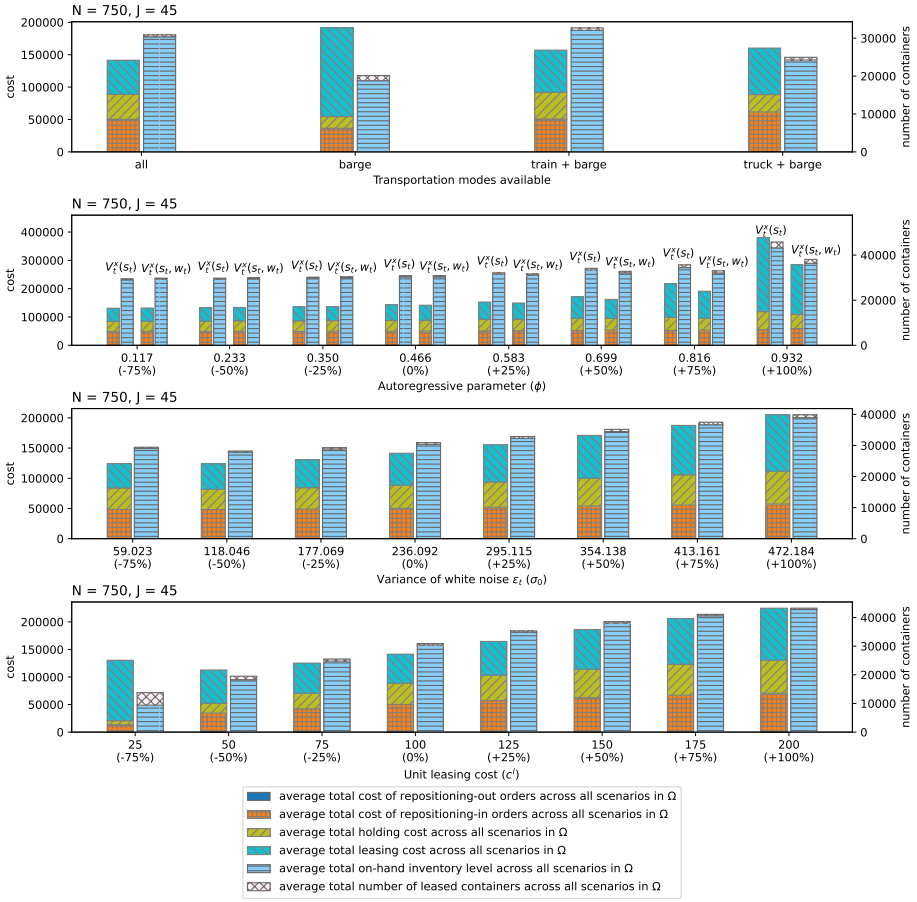


Fig. 3. Impact of available transportation modes (first panel) and sensitivity analysis to key parameters ϕ, σ and c^{l} (second, third and fourth panels, respectively).

compared to when only barge is available. This represents an even greater cost reduction than when only two modes are available, underscoring the significance of leveraging all transportation modes.

5.3 Sensitivity Analysis

As also depicted in Figure 3, we conduct sensitivity analysis on key parameters to examine their impact on the stability of the optimal policy estimate discussed in Section 5.2.

The second panel in Figure 3 demonstrates the impact of serial correlation for net container inflows, determined by the exogenous variables. As previously indicated, the higher the serial correlation of net inflows, the higher the costs. In fact, alternating between inflow and outflow will stabilise the inventory level

and avoid excessive holding and leasing. With serial correlations of the exogenous random variables below the baseline value of $\phi = 0.466$, the impact on the total cost is minimal, with differences ranging from 3.45% to 7.01%. For values above $\phi = 0.699$, however, cost differences become significant. As the auto-regression parameter ϕ rises from 0.466 to 0.699, 0.816 and 0.932, total costs increases by 14.57%, 34.94% and 100.97%, respectively, emphasising the importance of accounting for such correlations.

The panel also illustrates the effect of incorporating the exogenous variable w_t into the post-decision value function V_t^{x}. When the serial correlation of the exogenous random variables is less than or equal to 0.466, the policy estimate based on $V_t^{\mathrm{x}}(\mathbf{s}_t)$ is nearly the same as the one derived from $V_t^{\mathrm{x}}(\mathbf{s}_t, w_t)$, with differences ranging from 0.1% to 0.7%. When the serial correlation exceeds 0.466, however, the policy estimate incorporating the exogenous variable clearly outperforms the alternative policy estimate. As the auto-regression parameter ϕ increases from 0.583 to 0.932, the policy estimate based on $V_t^{\mathrm{x}}(\mathbf{s}_t, w_t)$ yields a reduction in total cost ranging from 2.3% to 25%. This emphasises the importance of incorporating the exogenous variable into the value function when the serial correlation is significant.

The third panel in Figure 3 illustrates how costs are affected by the variance of net container inflows, determined by the variance σ of the white noise ϵ_t, affecting the exogenous variable w_t. As the variance increases, the total costs tend to rise, primarily due to increased in the total leasing cost. For instance, a 100% increase in variance leads to a 45% rise in the total cost. More specifically, the total costs of repositioning-in and -out, holding and leasing containers increase by 14%, 78%, 42% and 77%, respectively. Although the total repositioning-out cost experiences the highest increase, it only makes up about 0.1% of the total cost. The total leasing cost, on the other hand, comprises 37% of the total cost, and thus, is the main driver behind the cost increase. This indicates that significant fluctuations in net inflow necessitate higher inventory flexibility, which is provided by leasing or repositioning out more containers.

The last panel depicts how changes in the unit leasing cost c^{l} affect the total cost. As the unit leasing cost rises, naturally, the total number of leased containers decreases, whereas the total on-hand inventory level increases. For instance, with a 100% increase in the unit leasing cost, the policy leases 11% fewer containers and maintains on-hand inventory levels 41% higher by placing more repositioning-in orders, resulting in a 41% increase in the total repositioning-in cost. Consequently, the total cost increases by 59%. When the unit leasing cost decreases by 75%, however, the policy leases many more containers, in fact 730% more, leading to 7.87% lower total cost compared to the baseline but 15.66% higher total cost compared to when the unit leasing cost decreases by 50%. Indeed, when the unit leasing cost approaches the unit repositioning-in cost, our algorithm exhibits limited learning capability. In practice, however, leasing costs typically far surpass repositioning-in costs, thereby validating the suitability of our approach for real-world applications.

6 Conclusion

This paper presents a dynamic and stochastic inventory management model for empty containers with multiple transportation modes, lead times, lost-sales and correlated exogenous variables, employing approximate dynamic programming with convex quadratic value functions. The proposed ADP algorithm converges with a modest sampling size and considerably outperforms a random policy. Furthermore, we demonstrate the importance of incorporating exogenous information into the value function, especially in scenarios with significant serial correlation.

Through sensitivity analysis, we demonstrate effective management of inventory levels and container leasing amidst varying factors. We highlight the benefits of leveraging both flexible yet costly transportation modes and inflexible yet economical options. For our case study, we obtain cost savings of up to 17.92% by introducing either train or trucks in addition to barge, and savings of 26.05% by allowing for all three modes of transportation. We show that it is less expensive to manage a container depot with low variance and serial correlation in exogenous container demand and returns. Real data, however, reveals significant variance and correlation. Moreover, increases in variance and correlation will substantially increase costs, illustrating the importance accurately modelling exogenous in- and out-flows to and from the depot. Finally, as leasing costs largely drive repositioning decisions, accurate estimates of such costs are crucial.

Future directions include exploring multiple depots and diverse distribution models for inflows and outflows, as well as investigating alternative value function approximations and/or closed-form solutions to the optimisation problems to enhance computational efficiency.

References

1. Bellman, R.: Dynamic programming and stochastic control processes. Information and control **1**(3), 228–239 (1958)
2. Cai, J., Li, Y., Yin, Y., Wang, X., Lalith, E., Jin, Z.: Optimization on the multi-period empty container repositioning problem in regional port cluster based upon inventory control strategies. Soft Computing **26**(14), 6715–6738 (2022)
3. Chi, M., Zhu, X., Wang, L., Zhang, Q., Liu, W.: Multi-period empty container repositioning approach for multimodal transportation. Measurement and Control p. 00202940231198133 (2023)
4. McKay, M.D., Beckman, R.J., Conover, W.J.: A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. Technometrics **42**(1), 55–61 (2000)
5. Powell, W.B.: Approximate Dynamic Programming: Solving the curses of dimensionality, vol. 703. John Wiley & Sons (2007)
6. Puterman, M.L.: Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons (2014)
7. Song, D.P.: Container logistics and maritime transport. Routledge (2021)
8. Song, D.P., Dong, J.: Modelling Empty Container Repositioning Logistics. Springer (2022)

Algorithm Switching for Multiobjective Predictions in Renewable Energy Markets

Zijun Li and Aswin Kannan

Institut für Mathematik, Humboldt-Universität zu Berlin, Berlin, Germany

Abstract. We study the multiobjective hyperparameter optimization problem that arises in classification and regression type settings in renewable energy markets. Examples of objectives include accuracy, computational time, and bias. Models can range from complex decision trees and neural networks to simpler KNNs (K-Nearest Neighbours). Multiobjective hyperparameter optimization has entailed deployment of both Bayesian and Direct-Search methods individually in the past. In this paper, we propose a switching framework that effectively uses both of these methods in an iterative fashion. As an additional contribution, we propose a warm-start based training for the machine learning models, which effectively deploys the benefits of Direct-Search methods. We compare our method with traditional Bayesian and Direct-Search methods by using multiple real-world datasets and machine learning models. We observe a clear-cut computational advantage in using our "combined method" and propose to extend this to general derivative-free regimes in the future.

Keywords: Classification · Regression · Multiobjective · Energy Markets.

1 Introduction

Multiobjective versions of the unit commitment problem [30] aim at the reduction of CO_2 emissions in addition to cost minimization and penetration of renewable power. These can be put forth in a two-level form as follows.

$$\begin{aligned} \{P_{out}\} \quad & \min_{s \in S(z)} G(s,z) = [g_1(s,z) \dots g_p(s,z)] \\ \{P_{in}\} \quad & \min_{\theta \in \Theta, z \in \mathcal{Z}(\theta)} F(\theta,z) = [f_1(\theta,z) \dots f_m(\theta,z)]\,. \end{aligned}$$

The unit commitment problem is defined by P_{out}, where g_i's denote objectives on costs or emissions. The variables s refer to decisions on generation levels and the set $S(z)$ encompasses the constraints on generation levels, ramp rates, and consumer demand. The machine learning problem P_{in}, aims to accurately estimate the generation levels. Here, θ refers to model hyperparameters, z refers to model parameters, and f_i's can refer to metrics like balanced accuracy, precision, recall, and mean bias error. Sets Θ and $\mathcal{Z}$ refer to constraints on hyperparameters and model parameters respectively.

Renewable energy planning predominantly focuses on solar and wind power based generators. Wind power forecasting is aimed at multi-class classification [24, 2], where wind-speeds fall under discrete buckets (say, $0-5$ mph, $5-20$ mph etc.). Solar energy based problems rely on PV-cells and have been viewed from a regression angle [29, 2]. Models ranging from Gradient Boosted Trees [27] and Neural Networks [2] to SVMs (Support Vector Machines) [28] and KNNs (K-Nearest-Neighbours) [17] have been studied in detail in both contexts. We note that all the above works have predominantly focused on single-objective optimization. For a complete survey, the reader is asked to refer to [24, 29]. Some works have focused on biobjective versions of the problem [14]. However, the focus in this case was on only some problem-specific metrics related to accuracy. Recent work [19] motivates the need for multiobjective learning based formulations in the context of renewable energy, bearing in mind all the aforementioned factors. Algorithms in lieu of general problems in multiobjective machine learning can be vaguely classified as follows.

- Hyperparameter Optimization (HPO) [4, 6]: Here, the focus is on tuning θ. The model parameters z^* are a consequence of training machine learning models on standard losses like cross entropy. Note that $G(z)$ refers to one such standard loss function (denoted below).

$$\min_{\theta \in \Theta} \; F(\theta, z^*(\theta)) = [f_1(\theta, z^*(\theta)), \ldots, f_m(\theta, z^*(\theta))], \quad z^*(\theta) = \underset{z \in Z(\theta)}{\arg\min}\, G(z).$$

- Model Parameter Optimization (MPO): MPO attempts to solve the following model training problem: $z^* = \underset{z \in Z(\theta^*)}{\arg\min}\, F(\theta^*, z)$. Here, θ^* is estimated prior to training by subject matter expertise. Their applicability is restricted due to practical structural and implementation aspects.
- Other schemes: Some works [20] have proposed fusion type schemes that deploy the best of both worlds (HPO and MPO). However, the applicability of fusion type schemes faces limitations similar to MPO. Multi-objective learning also has been studied from the angle of "multi-task learning" (MTL) [25], where the aim is to "simultaneously learn" multiple models in z (based on multiple data sources). This is different from our framework, where we intend to have "one model" that needs to be optimized across all objectives. Lexicographic type non-Pareto methods have also been considered in literature. But, their use is restricted to problems with stylized structure on objectives.

In this work, we consider the hyperparameter optimization (HPO) portion of the problem. In the context of HPO, Bayesian optimization (BO) methods have found great traction over the last two decades [4, 12, 15]. BO methods have been observed to be robust for smaller dimensional settings with smooth objective functions. Besides BO, search type methods have also been deployed considerably for HPO. Search methods can be two fold: either based out of Direct-Search (DS) [3, 5] or genetic algorithms [13]. While genetic algorithms have also proved to be very efficient, we focus on using DS, due to the latter's convergence guarantees. In this work, we propose a "combined" method that uses the strengths

of both BO and DS. Our method alternates between BO and DS by means of a switching mechanism that is dependent on objective descent. To incorporate multiple objectives, we use weights from a uniform mesh and solve a sequence of scalarized single objective optimization problems. Our key contributions are three fold.

1. *Combined method*: As stated, we intend to use the benefits of both BO and DS type methods.
   - Why alternate?: BO examines objectives in a global sense and can potentially lead to many iterations that do not yield in descent. On the other hand, DS is very beneficial in local examination and has shown significant promise in discrete settings like ours, where there are fewer variables (dimension less than 10). Additionally, the nature of the spanning directions in the case of DS helps with warm-starting the process of model training. However, DS can stagnate at locally optimal solutions. This necessitates the reverse switching to a global optimization technique like BO.
   - We compare our combined method against state-of-the-art BO and DS type solvers. We use three widely used machine learning models, each belonging to a different class for our studies: XGBoost (decision trees), ResNets (Neural Networks), and KNNs (K-Nearest-Neighbours - unsupervised model). We use a comprehensive set of metrics and consider both regression and classification problems.
2. Warm Starting: Deploying our problem structure, we use a number of low-budget evaluations, which helps in improving our computational efficiency.
3. Practical Data: We obtained real weather data from [23] across two locations in Germany, namely Berlin and Stuttgart. The related target generational data was obtained from [22] for the firms, 50hertz and Transnet BW (in both locations). In summary, we built four real-world datasets, which can be readily used for benchmarking purposes.

The rest of the paper is organized into four sections. Section 2 discusses the preliminaries related to multiobjective learning. Section 3 proposes our algorithm. Section 4 tests our algorithm and compares it with other methods on real datasets. We conclude in Section 5.

2 Preliminaries

Prior to beginning our discussion, for clarity, we define our notation in table 1. We formally restate the problem in (1) for clarity.

$$\begin{aligned} &\min_{\theta\in\Theta} \quad F(\theta) = [f_1(\theta), \ldots, f_m(\theta)], \quad f_i(\theta) = f_i(y^{pred}, y^{true}), \quad \text{where} \\ &z_\theta^* = \underset{z\in\mathcal{Z}(\theta)}{\arg\min}\, G(z, y^{true}),\ y^{pred} = M(z_\theta^*). \end{aligned} \tag{1}$$

Here, the individual functions are evaluated as a consequence of model training on standard losses G. We also note that the above formulation extends to several

N	No. of data samples
t	Index/time-stamp of a datapoint
x_t	Data feature vector at time t
y_t^{true}	True label at time t
y_t^{pred}	Predicted labels at time t
M	Machine Learning model
G	Standard loss function
m	No. of objectives
f_i	i^{th} objective function
n	Problem dimension
θ	Model hyperparameters
z	Model parameters

Table 1: Notation.

generic multiobjective machine learning problems. The hyperparameters θ define the architecture of the machine learning model, eg: no. of leaves/tree depth in decision trees, no. of layers/neurons in neural networks, and no. of clusters in KNNs. Model parameters z refer to the individual weights that arise in the model M. Say, in the case of neural networks, these refer to the weights associated with each neuron. Prior to delving into algorithmic aspects, we first look at some fundamentals of such multiobjective formulations in renewable energy predictions.

2.1 Metrics

The metrics can be classified into three different forms as follows. For comprehensive details, the reader is asked to refer to [19].

1. Accuracy type indicators (classification and regression): Examples include False Positive Rate error (FPR_{err}), Balanced Accuracy, False Negative Rate eror (FNR_{err}), F1-score, R-square, and Mean Square Error (MSE). Here, FPR (error) is a classification type metric, while MSE is used in regression.
2. Bias type indicators: Examples include fairness type metrics like Demographic Parity Difference (DPD) and elementary metrics like Mean Bias Error (MBE). Metrics like MBE are common in regression, while DPD type metrics are found usually in classification regimes.
3. Model specific: Examples include blackbox objectives like computational time and closed-form objectives like sparsity.

$$\text{FPR}_{\text{err}}(.) = \frac{\sum_{t=1}^{N}(1-y_t^{true})(y_t^{pred}(\theta,z))}{\sum_{t=1}^{N}(1-y_t^{true})}, \ \text{MBE}(.) = \frac{1}{N}\left|\sum_{t=1}^{N}\left(y_t^{pred}(\theta,z) - y_t^{true}\right)\right|.$$

2.2 Algorithmic Fundamentals

A majority of the methods that have been studied in literature deal with estimating the Pareto frontier of the problem of interest. A Pareto frontier refers

to the set of non-dominated solutions that solve the multiobjective problem. Non-dominated points can be formally defined as follows.

Definition 1. *A point θ^a is said to dominate point θ^b iff $f_j(\theta^a) < f_j(\theta^b)$ for at least one index j and $f_i(\theta^a) \leq f_i(\theta^b)$, $\forall\ i \neq j$. That is,*

$$\theta^a \succcurlyeq \theta^b \ \ \textit{iff} \ \ f_j(\theta^a) < f_j(\theta^b) \ \ \textit{and} \ \ f_i(\theta^a) \leq f_i(\theta^b).$$

Noting that non-Pareto methods are mostly application specific, we restrict our focus to Pareto methods and classify them as follows.

Scalarization [20, 11]: Here, a series of scalarized problems are solved for different choices of weights ω: $\min_{\theta\in\Theta} \sum_{i=1}^{m} \omega_i f_i(\theta)$. Note that $\sum_{i=1}^{m} \omega_i = 1$. We also note that nonlinear choices of weights [11] can also be deployed.

Epsilon constraints [21]: Here, combinations of objectives are posed as constraints and the following is solved for different sequences of ϵ.

$$\min_{\theta\in\Theta} \ \ f_1(\theta), \ \ \text{subject to:} \ \ f_2(\theta) \leq \epsilon_2, \ldots, f_m(\theta) \leq \epsilon_m.$$

Say, if objectives $\{f_i\}_{i=1}^{m}$ are normalized between 0 to 1, then, ϵ is varied from 0 to 1 element wise and the process is repeated. For the next set, f_2 is moved to the objective and the next sets are continued likewise for the other objectives.

Search type methods [5, 13]: For a current iterate θ^k, stepsize β, and spanning direction set $\mathcal{D}^k$, a non-dominated point $y \in \Theta^k$ (in lieu of definition 1) is attempted to be estimated in the search space Θ^k. Note that Θ^k is defined as follows: $\Theta^k = \{\theta | \theta = \theta^k + \beta d^k, \ d^k \in \mathcal{D}^k \subseteq \mathcal{D}\}$.

2.3 Hyperparameter Learning

Recently, the hyperparameter optimization problem has been studied significantly from a multiobjective angle [4, 5, 12]. For completeness, we summarize the basics of both BO and DS type schemes as follows.

Bayesian schemes [15, 4]: Here, the objective function f_i is assumed to follow a Gaussian Process: $f_i(\theta) \sim \mathcal{N}(\mu_i(\theta), \sigma_i^2(\theta, \theta))$. These functions μ and σ take the following form and are iteratively updated based on evaluations of f_i.

$$\sigma^2(\theta,\theta) = \begin{pmatrix} k(\theta^1,\theta^1), \ldots, k(\theta^1,\theta^n) \\ \vdots, \quad \ddots, \quad \vdots \\ k(\theta^n,\theta^1), \ldots, k(\theta^n,\theta^n) \end{pmatrix}, \ k(\theta^j,\theta^k) = \gamma e^{-\beta\|\theta^j - \theta^k\|^2}, \text{ and } \mu(\theta) = \mu.$$

Note that $k(\theta,\theta^{'})$ refers to kernel functions. The problem hence boils down to the estimation of scalar parameters, μ, γ, and β. At a high level, given $t = 0$ and samples $\mathcal{R}^0 = \left\{\theta^j, f(\theta^j)\right\}_{j=1}^{N}$, BO can be summarized as follows. For further details, the reader is asked to refer to [26, 15].

1. Obtain an estimate of μ and σ based on $\mathcal{R}^t$ using Gaussian regression.

2. Set $t = t + 1$. Solve the following problem with the acquisition function α: $\min_{\theta \in \Theta} \ \alpha(\mu(\theta), \sigma(\theta))$. Note that the acquisition function α is an approximation or an indicator of the objective function f_i.
3. Based on the solution θ^{t+1} to the above, evaluate the function $f(\theta^{t+1})$. Update $\mathcal{R}^{t+1} = \mathcal{R}^t \cup \left\{\theta^{t+1}, f(\theta^{t+1})\right\}$. Examine termination criteria based on $f(\theta^{t+1})$.

Direct-Search [5, 3]: Given $k = 0$, $\Delta^0 = \delta^0 = 1$, θ^0, a set of spanning directions $\mathcal{D}$, and a subset $\mathcal{A}^0$, a simple version is stated below.

1. Perform local search around "ALL" evaluated points indexed by j: $f(\psi^k) < f(\theta^k), \quad \psi^k = \theta^j + \Delta^k d^k, \ \ \forall j, \ \ \forall d^k \in \mathcal{A}^k$.
2. If successful, set $\theta^{k+1} = \psi^k$. Increase Δ^k.
3. Else, perform the following step and decrease Δ^k: $f(\psi^k) < f(\theta^k), \quad \psi^k = \theta^k + \delta^k d, \forall d \in \mathcal{D}$.
4. If successful, update θ^{k+1} and increase δ^k. If step 3 is not successful, decrease δ^k. Set $k = k + 1$.

3 Algorithms

In this section, we propose a novel algorithm that sequentially alternates between DS and BO based on certain prefixed criteria. To retain our global optimization framework, we initiate our method with BO. When BO becomes expensive and fails in improving the objective significantly, our framework switches to DS. On the other hand, we switch from DS to BO, when the algorithm has spent sufficiently long in DS (for a set of evaluations) "and" if DS does not yield descent directions. We further note the following important points prior to stating our formal version of the algorithm.

- *Weights:* To solve the multiobjective optimization (MOO) problem (1), we solve a sequence of single objective problems, each parameterized by a combination of weights [20]: $\min_{\theta \in \Theta} \ \sum_{i=1}^{m} w_i f_i(\theta)$. These weights are generated from a uniform distribution, where $w \in \mathbb{W}^{\text{uniform}}$ and $\sum_{i=1}^{m} \omega_i = 1$. While there are several ways to solve the MOO, we resort to weights, given the established robustness of this framework in literature. At this point, we specifically note that the intention of our work is to show *only* the benefit of the combined method over BO or DS. We do not aim to compare the performance of weighted schemes over other possible schemes of multiobjective optimization.
- Our proposition is very different from direct search methods that use Bayesian fundamentals to define the scope of search directions [1]. Our work on the contrary uses search directions to enhance Bayesian optimization.

3.1 Combined method

For each weighted combination, we attempt to solve the scalarized single objective problem to complete optimality. Prior to formally stating the alternating

θ_{init}	Initial hyperparameter configuration
τ^{overall}	Total computational time budget
ω_i	Weight for objective i
τ^{BO}	Time budget for single iteration using BO
τ^{search}	Time budget for single iteration using DS
τ_k	Time taken for training the model at iteration k

Table 2: Notation for the algorithmic parameters.

method, we define the algorithmic parameters in table 2. Our algorithm alternates between BO and DS based on two switching criteria defined as follows.

Switching from BO to DS: BO in contrast to DS can lead to new evaluations significantly different from the current iterate. Warm-starting model training in an iteration of BO is therefore difficult. Benefits of "exploration" (new points) in BO are usually less useful in comparison to "exploitation" (investigation with current points) when the model training times are high and when there is no observed descent. This is put forth by our following switching condition.

$$f(\theta^k) > f(\theta^{k-1}), \;\; \tau_k > \tau^{\text{BO}}.$$

Switching from DS to BO: Search directions generated by DS usually are incremental in one or two elements of θ. DS also operates within a smaller search radius (in comparison to BO). In the context of our problem, this implies that the hyperparameters do not differ much between successive iterations. This naturally aids with warm-starting the model training process, thereby helping with cheaper evaluations. With the aim of using this to the maximum, we do not demand descent from search in one evaluation. Instead, we switch from DS to BO, only if the search fails to improve upon the objective over a set of evaluations. The switching condition is stated as follows.

$$f(\theta^k) > f(\theta^{k-1}), \text{ and } \textstyle\sum_{l=0}^{L} \tau_{k,l} > \tau^{\text{search}}.$$

Note that the integer L is not determined prior to running the algorithm and this is dependent on training times at each evaluation. Formally, the combined method is presented in Algorithm 1. The switching criteria are formally defined in Algorithm 2. Note that we use the notation for $\mathcal{R}^k$, $\mathcal{D}^k$, and $\mathcal{A}^k$ as defined in section 2. For completeness, we specifically note that $\tau^{c,\text{search}}$ refers to the total time across multiple evaluations of search and this is set to zero when the algorithm switches from BO to DS.

Convergence Aspects: For our problem statement, we do not have a formal proof of convergence yet and we intend to focus on the same as part of our future work. It can be observed that by the design of our algorithm, if unsuccessful, the time spent by DS is finite. This intuition helps forming the basis behind the theoretical aspects.

Warm Starts: Noting that hyperparameters are mostly discrete, using simple coordinate directions for search is a great alternative. For instance, let us consider the case of training with XGBoost and two hyperparameters, θ_1 and

Algorithm 1 Multi-objective optimization using BO and DS

Input: $\theta_{\text{init}}, \tau^{\text{overall}}, \omega \in W^{\text{uniform}}, \tau^{\text{BO}}, \tau^{\text{search}}$

Output: $\left\{\theta^k\right\}_{\forall k}$ - final repository (trajectory) of points for the problem $\hat{f}(\theta, \omega) = \sum_{i=1}^{m} \omega_i f_i(\theta)$.

Initialize:
$k = 1$ (Number of iterations) and $\theta^0 = \theta_{\text{init}}$
Set $\mathcal{R}^0 = \left\{\theta^0, \hat{f}(\theta^0)\right\}$
current_method = "BO" (Start with BO)
$\tau = 0$ (Current computation time)
while $\tau < \tau^{\text{overall}}$ **do**
  if current_method == "BO" **then**
    Obtain the acquisition function α^k with samples $\mathcal{R}^k$
    $\theta^k \leftarrow BO(\theta^{k-1})$
    Evaluate $\hat{f}^k = \sum_{i=1}^{m} \omega_i f_i(\theta^k)$ and obtain the corresponding computational time τ_k^{BO}
    $\tau \leftarrow \tau + \tau_k^{\text{BO}}$
  else if current_method == "$search$" **then**
    $\theta^{k,l} \leftarrow search(\theta^{k-1})$
    Evaluate $\hat{f}^{k,l} = \sum_{i=1}^{m} \omega_i f_i(\theta^{k,l})$ and obtain the corresponding computational time $\tau_{k,l}^{\text{search}}$
    $\tau \leftarrow \tau + \tau_{k,l}^{\text{search}}$
    $\tau^{c,\text{search}} \leftarrow \tau^{c,\text{search}} + \tau_{k,l}^{\text{search}}$
    $l = l + 1$
  end if
  Use the switching criteria in Algorithm 2
  Update $\mathcal{R}^k = \mathcal{R}^{k-1} \cup \left\{\theta^k, \hat{f}(\theta^k)\right\}$ for BO.
  Update $\mathcal{A}^k$ and $\mathcal{D}^k$ similarly for search
  $k \leftarrow k + 1$
end while
Return all evaluated points for $\left\{\theta^k\right\}_{\forall k}$.

θ_2. Let the hyperparameters respectively refer to the number of trees and tree depth. Say, at iteration k, we have $\theta_1^k = 10$ and $\theta_2^k = 5$. Suppose, we use the coordinate direction $\begin{pmatrix}1 & 0\end{pmatrix}^T$. Then, for the $(k+1)^{\text{th}}$ iteration, we have $\theta_1^{k+1} = 11$ and $\theta_2^{k+1} = 5$. This implies that we have one additional tree/round for model training. In this case, for the initial solution to the model training process, we simply use the weights from the previous iteration and set the weights of the new tree to be 0. That is, $z^{0,k+1} = \begin{pmatrix}(z^{*,k})^T & 0\end{pmatrix}^T$. This immensely reduces the model training time.

Hypervolume Indicator: Since the methods of interest work on different classes of objectives, we deploy the widely accepted hypervolume indicator [10] as our yardstick for all comparisons. Post normalization of objectives between 0 and 1 (0 is best and 1 is the worst with respect to minimization), we use the unit vector as a reference point for hypervolume computation.

4 Numerics

This section analyzes the performance of our combined method, Bayesian optimization (BO), and Mesh Adaptive Direct Search (MADS). For our experiments, we used three machine learning models, namely, ResNet [16], XGBoost [8], and K-Nearest Neighbors (KNN) [9]. Details on our use cases are stated in the

Algorithm 2 Switching Criteria

Input: current_method, $\hat{f}^{k-1}(\theta^{k-1}), \hat{f}^{k}(\theta^{k}), \tau_k^{\text{BO}}, \tau^{c,\text{search}}$
Output: current_method.

if current_method == "*search*" **then**
  if $\hat{f}^{k,l}(\theta^{k,l}) - \hat{f}^{k-1}(\theta^{k-1}) < 0$ **then**
    $\theta^{k} = \theta^{k,l}$
    Initialize search counter $\tau^{c,\text{search}} = 0$ and $l = 0$
  end if
  if $(\tau^{c,\text{search}} > \tau^{\text{search}})$ AND $\hat{f}^{k-1}(\theta^{k-1}) - \hat{f}^{k}(\theta^{k}) < 0$ **then**
    $\theta^{k} = \theta^{k-1}$
    current_method = "*BO*"
  end if
else if current_method == "*BO*" **then**
  if $(\tau_k^{\text{BO}} > \tau^{\text{BO}})$ AND $\hat{f}^{k-1}(\theta^{k-1}) - \hat{f}^{k}(\theta^{k}) < 0$ **then**
    current_method = "*search*"
    Initialize search counter $\tau^{c,\text{search}} = 0$ and $l = 0$
  end if
end if
Return updated current_method.

forthcoming paragraph. Our experiments were performed on an NVIDIA RTX 3080, with an Intel Core i5-12600K processor and a memory (RAM) of 32GB 4800MT/s DDR5.

Data: We used four real world datasets for this study. The datasets are based out of two locations, Berlin and Stuttgart in Germany. For each location, we further studied two types of data: solar energy and wind energy. We adopted a similar formulation style as with a host of previous works in literature [24, 29]. We studied a regression system for solar energy data and presented a multi-class classification type framework for wind energy data. Data processing involved two steps. First, we obtained feature data from [23]. Feature information represents structured data regarding weather factors like temperature, humidity, and wind-speed. There were twenty three input features in all our datasets. Second, this feature information was integrated with the actual energy generation (target variable). The generational data was obtained from [22]. The data covers the entire time period between January 1st, 2018, to December 31st, 2019. Our implementation (codes + data) is publicly available [1]. For the solar regression task, we refined the dataset by excluding time periods when solar power is not available (e.g., nighttime hours, 10 p.m. to 5 a.m.). For wind energy multi-class classification, we standardized all features (excluding the target variable) using Z-Scores. We divided the actual wind power generation data into several nearly equal sized classes. We chose five classes for wind-energy based problems to ensure that the data is reasonably balanced (choice of lower or higher numbers lead to highly imbalanced data). Our methodology can easily be extended to any number of classes. Each solar dataset finally contained about $36,000$ data points and each wind energy dataset consisted of roughly $70,000$ data points. All the datasets were split into training, test, and validation sets using an $80\% : 10\% : 10\%$ ratio.

[1] The codes are available at https://scm.cms.hu-berlin.de/aswinkannan1987/lion.

Evaluation metrics: We considered three types of metrics, namely, accuracy, bias, and complexity. In the context of the solar regression problem, the metrics employed include mean squared error (MSE), mean bias error (MBE), and model complexity. For the wind energy multi-class classification, the chosen metrics were accuracy error, demographic parity difference (DPD), and model complexity. We elaborate on the model complexity portion in the forthcoming portions of this section. Given a dataset with H features and N samples, we define y^{true} and y^{pred} as the true and predicted labels respectively. Let $y = \left\{y^{true}, y^{pred}\right\}$ and $yd = y^{true} - y^{pred}$. Then, mean squared error (MSE), mean bias error (MBE), and accuracy error (ACE) can be calculated as follows.

$$MSE(y) = \frac{1}{N}\sum_{i=1}^{N}(yd_i)^2,\ MBE(y) = \frac{1}{N}\left|\sum_{i=1}^{N} yd_i\right|,\ ACE(y) = 1 - \frac{1}{N}\sum_{i=1}^{N}|yd_i|\,.$$

Let $a_i \in \{0,1\}$ be a fairness feature of the i-th data sample. Then, the index sets of $(a_i)_{i=1}^{N}$ can be defined as follows.

$$A_0 := \{a_i = 0 \text{ for } i \in \{1,...,N\}\},\quad A_1 := \{a_i = 1 \text{ for } i \in \{1,...,N\}\}.$$

Here, A_0 and A_1 contain N_0 and N_1 samples respectively, and $N = N_0 + N_1$. We used the metrics, demographic parity difference (DPD) and symmetric distance error (SDE) to assess bias along the lines of previous literature [20, 19, 7].

$$DPD(y) = \left|\frac{\sum_{i\in A_0} y_i^{pred}}{N_0} - \frac{\sum_{i\in A_1} y_i^{pred}}{N_1}\right|,\quad SDE = \frac{1}{H}\sum_{c=1}^{H}|\Delta FNR(c) - \Delta FPR(c)|\,.$$

Here, c is the c-th feature, while $\Delta FNR(c)$ and $\Delta FPR(c)$ represent the false negative rate (FNR) and false positive rate (FPR) of the binary feature c.

Model Complexity and Summary: The KNN is a simple model. Hence, we considered only the first two metrics and excluded model complexity for our study. In ResNet type models, **WSpar** represents the proportion of zero or nearly zero weights. We chose the weight density, "**WDens** $= 1 -$ **WSpar**" as our complexity metric (that we intend to minimize). The goal is to simplify the network by reducing the number of active connections. For the XGBoost model, we chose a similar metric, **average depth (ADep)** to measure complexity (deeper tree implies a complex model). Table 3 summarizes the metrics used by each model for different problems.

Table 3: Metrics to minimize in different models.

Problem type	Model	Metrics
Solar regression	ResNet	MSE, MBE, WDens
	XGBoost	MSE, MBE, ADep
	KNN	MSE, MBE
Wind classification	ResNet	ACE, DPD, WDens
	XGBoost	ACE, DPD, ADep
	KNN	ACE, SDE

Table 4: Hyperparameters used for optimization

Model	Hyperparameters	lb	ub
ResNet	no. of residual blocks	1	15
	no. of neurons	8	128
	dropout rate	0.01	0.5
XGBoost	no. of estimators	1	50
	max depth	2	20
	max leaves	1	20
	min child weight	0.1	5
	gamma	0	0.5
KNN	no. of neighbors	1	20
	ϕ_m	1	4

Selected solvers and Results: As mentioned earlier, we used the "hypervolume" as the indicator of performance for all our studies. When using weights, the trajectory of iterates were aggregated across all combinations of weights and the hypervolume was computed with these final set of points. For the weighted version of Bayesian optimization (BO), we used a standard Surrogate Model Toolbox (SMT) solver, known as Efficient Global Optimization (EGO) [18]. For the weighted version of search, we used a weighted version of MADS [3]. We note that both these solvers are tailored for single objective optimization. We also tested the performance with solvers specifically designed for multiobjective optimization. In the case of BO, USeMO [4] (Uncertainty-aware Search framework for optimizing Multiple Objectives) was deployed. Similarly for DS, a newer multiobjective solver, namely DMulti-MADS [5] was deployed. For the combined method, the time thresholds were set to $\{\tau^{BO} = 2, \tau^{search} = 1\}$, $\{\tau^{BO} = 1, \tau^{search} = 0.5\}$, and $\{\tau^{BO} = 12, \tau^{search} = 6\}$ respectively for the ResNet model, the XGBoost model, and the KNN model. We note that this difference arises due to different complexities and training times across models. For bi-objective problems, we explored five uniform weights: $(w_i)_{i=1}^2 \in \{0.25i, 1 - 0.25i\}$, where $i \in \{0, 1, 2, 3, 4\}$. For tri-objective problems, the weight selection expanded to more combinations as follows: $w^T = \begin{pmatrix} w_1 & w_2 & w_3 \end{pmatrix} = 1/3 \begin{pmatrix} i_1 & i_2 & i_3 \end{pmatrix}$, where, $\{i_1, i_2, i_3\} \in \{0, 1, 2, 3\}$ and $i_1 + i_2 + i_3 = 1$. The summary of the hyperparameters used, and their corresponding lower bounds (lb) and upper bounds (ub) are listed in Table 4. Note that ϕ_m denotes the Minkowski metric's power parameter. The results from our computation are shown in Figures 1 and 2. The results are individually discussed in the forthcoming paragraphs.

Berlin solar regression: We can see that our combined method consistently performs better than the other algorithms across all models. While weighted MADS and weighted BO approaches do improve over time, their progress is slower and tends to plateau. An important observation is that with ResNet and XGBoost models, the hypervolume improvement with the combined method is significantly greater than that of the weighted BO. However, in the case of the KNN model, the hypervolume improvement of the combined method is modest and about 10% higher than that of the weighted BO. This can be attributed to the simplicity of the KNN model with only two hyperparameters and the possible attainment of the best level of efficiency.

Stuttgart solar regression: As earlier, it is observed that the combined method achieves better results than the weighted MADS and weighted BO methods for all models. The ResNet model paired with the combined method once again demonstrates the most promising outcomes. These findings suggest the combined method's adaptability and efficiency in solving regression problems.

Berlin wind multi-class classification: It can be noticed that both the combined method and the weighted MADS demonstrate competitive performance. They achieved similar hypervolumes in a limited time. The weighted MADS shows strong initial performance in the ResNet model, while the combined method catches up as time progresses. Although the weighted BO shows a significant improvement in hypervolume, it does not reach as high a hypervol-

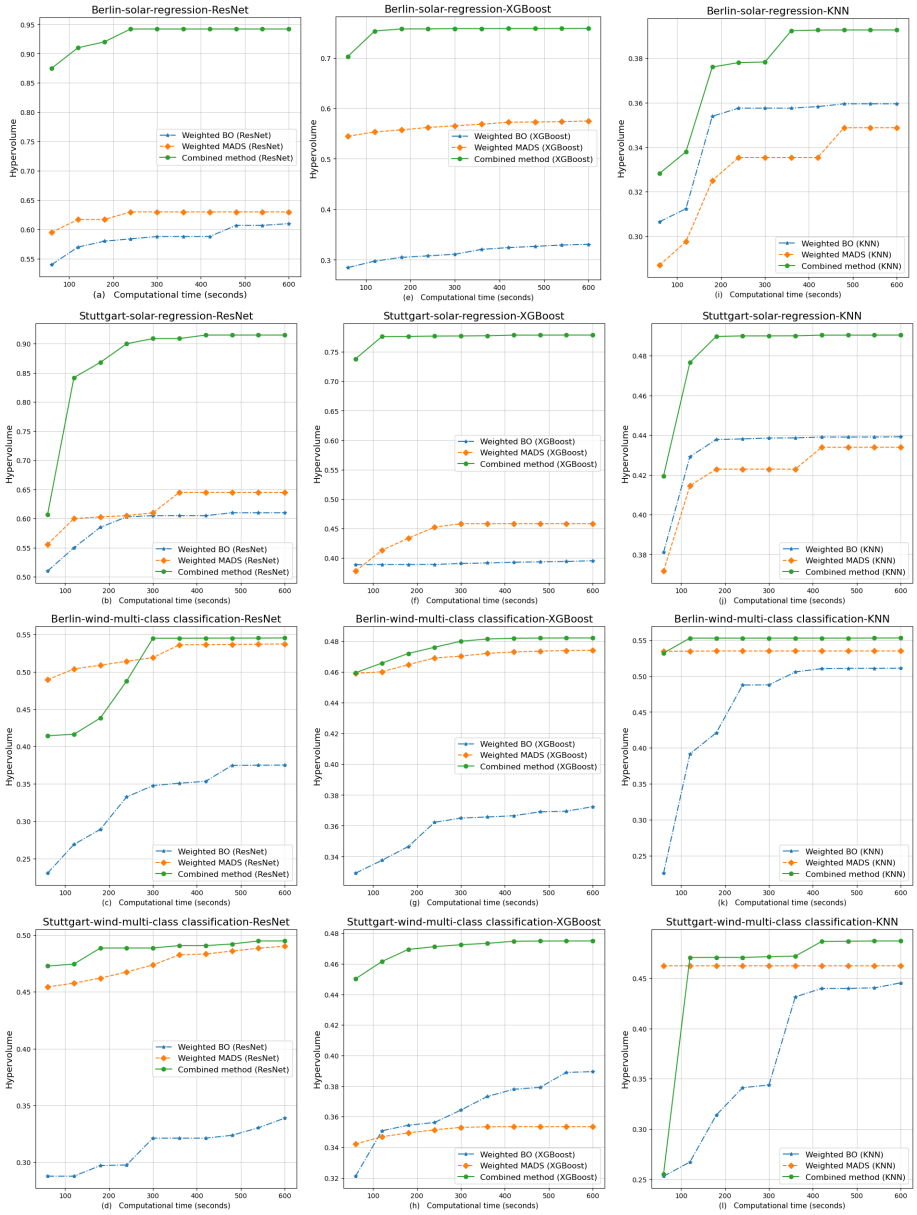


Fig. 1: Evaluating the performance of different algorithms on ResNet, XGBoost, and KNN models.

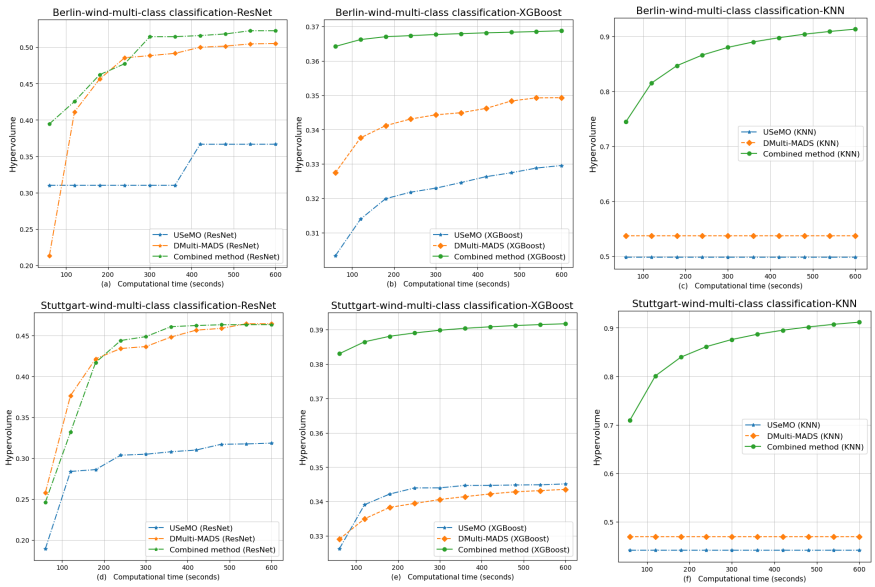


Fig. 2: Evaluating the performance of various algorithms in solving wind multi-class classification using MOO solvers.

ume as the other two methods. While weighted MADS offers initial advantages, the combined method can be inferred to be effective overall.

Stuttgart wind multi-class classification: The combined method achieves high hypervolume quickly in different models. The weighted MADS shows good performance in comparison to the combined method, particularly in the ResNet and KNN models. This indicates that weighted MADS could be an alternative if sufficient computational time is available. The weighted BO shows a notable increase in hypervolume, especially with the KNN model. Yet, it is still lower than the hypervolume of the combined method.

Multiobjective Solvers vs. Combined Method: We also evaluate the performance of our combined method with state-of-the-art multiobjective solvers, USeMO and DMulti-MADS. In this case, we used the "hypervolume" as our optimization objective instead of $\hat{f}$ to check for descent and switching. Additionally, we imposed at least a five-percent improvement in hypervolume as our descent condition. As seen in Figure 2, the combined method consistently and rapidly achieves better performance than USeMO and DMulti-MADS across various models and datasets. DMulti-MADS occasionally matches the performance of the combined method, particularly with the ResNet model in Berlin. USeMO generally improves slightly over time, which indicates that it might be less suitable for complex models. There is potential for enhancing the performance of the combined method further by appropriately investigating this descent criterion.

5 Conclusion

This paper introduces a combined method for multiobjective hyperparameter optimization that uses both Bayesian Optimization (BO) and Direct-Search (DS). Additionally, we propose a warm-start to significantly reduce the model training time. Our focus is on regression and multi-class classification problems for multiple machine learning models. We assess BO, DS, and our combined method using four real-world datasets. The results demonstrate that our method outperforms the other two approaches, particularly in scenarios with limited computational resources. Future research will explore extending this framework to more general derivative-free optimization problems.

Acknowledgments. This research was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – The Berlin Mathematics Research Center MATH+ (EXC-2046/1, project ID: 390685689).

References

1. Acerbi, L., Ji, W.: Practical bayesian optimization for model fitting with bayesian adaptive direct search. Proceedings of the Neural Information Processing Systems (2017)
2. Alkhayat, G., Mehmood, R.: A review and taxonomy of wind and solar energy forecasting methods based on deep learning. Energy and AI **4**, 100060 (03 2021)
3. Audet, C., Dennis, J.E.: Mesh adaptive direct search algorithms for constrained optimization. SIAM Journal on Optimization **17**(1), 188–217 (2006)
4. Belakaria, S., Deshwal, A., Jayakodi, N.K., Doppa, J.R.: Uncertainty-aware search framework for multi-objective bayesian optimization. AAAI Conference on Artificial Intelligence **34**(06) (2020)
5. Bigeon, J., Le Digabel, S., Salomon, L.: Dmulti-mads: mesh adaptive direct multisearch for bound-constrained blackbox multiobjective optimization. Computational Optimization and Applications **79** (06 2021)
6. Binder, M., Moosbauer, J., Thomas, J., Bischl, B.: Multi-objective hyperparameter tuning and feature selection using filter ensembles. Genetic and Evolutionary Computation Conference p. 471–479 (2020)
7. Blakeney, C., Atkinson, G., Huish, N., Yan, Y., Metsis, V., Zong, Z.: Measuring bias and fairness in multiclass classification. In: IEEE International Conference on Networking, Architecture and Storage (NAS). pp. 1–6 (2022)
8. Chen, T., Guestrin, C.: Xgboost: A scalable tree boosting system. International Conference on Knowledge Discovery and Data Mining pp. 785–794 (2016)
9. Cover, T., Hart, P.: Nearest neighbor pattern classification. IEEE transactions on information theory **13**(1), 21–27 (1967)
10. Custódio, A.L., Emmerich, M., Madeira, J.: Recent developments in derivative-free multiobjective optimisation. Computational Technology Reviews **5**, 1–30 (2012)
11. Das, I., Dennis, J.: A closer look at drawbacks of minimizing weighted sums of objectives for pareto set generation in multicriteria optimization problems. Structural Optimization **14**, 63–69 (01 1997)
12. Daulton, S., Balandat, M., Bakshy, E.: Hypervolume knowledge gradient: A lookahead approach for multi-objective Bayesian optimization with partial information. In: Proceedings of the 40th International Conference on Machine Learning. vol. 202, pp. 7167–7204 (2023)

13. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Transactions on Evolutionary Computation **6**, 182–197 (2002)
14. Galván, I., Valls, J., Cervantes, A., Aler, R.: Multi-objective evolutionary optimization of prediction intervals for solar energy forecasting with neural networks. Information Sciences **418**, 363–382 (08 2017)
15. Garnett, R.: Bayesian Optimization. Cambridge University Press (2023)
16. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
17. Huang, J., Perry, M.: A semi-empirical approach using gradient boosting and K-Nearest Neighbors regression for GEFCom2014 probabilistic solar power forecasting. International Journal of Forecasting **32**(3), 1081–1086 (2016)
18. Jones, D.R., Schonlau, M., Welch, W.J.: Efficient global optimization of expensive black-box functions. Journal of Global optimization **13**, 455–492 (1998)
19. Kannan, A.: Benefits of multiobjective learning in solar energy prediction. 2nd Annual AAAI Workshop on AI to Accelerate Science and Engineering (2023)
20. Kannan, A., Choudhury, A.R., Saxena, V., Raje, S.M., Ram, P., Verma, A., Sabharwal, Y.: Hyperaspo: Fusion of model and hyper parameter optimization for multi-objective machine learning. Proceedings of the IEEE International Conference on Big Data (2021)
21. Mavrotas, G.: Effective implementation of the epsilon-constraint method in multi-objective mathematical programming problems. Applied Mathematics and Computation **213**(2), 455–465 (2009)
22. Netztransparenz: Renewable energies and levies, website. https://www.netztransparenz.de/de-de/Erneuerbare-Energien-und-Umlagen. (2023)
23. NREL: System advisor model version 2022.11.21 (sam 2022.11.21) website. pv cost data. national renewable energy laboratory. golden, co. https://sam.nrel.gov/photovoltaic/pv-cost-component.html. (2022)
24. Pérez-Ortiz, M., Jiménez-Fernández, S., Gutiérrez, P.A., Alexandre, E., Hervás-Martínez, C., Salcedo-Sanz, S.: A review of classification problems and algorithms in renewable energy applications. Energies **9**(8), 1–27 (2016)
25. Sener, O., Koltun, V.: Multi-task learning as multi-objective optimization. Proceedings of the Neural Information Processing Systems (2018)
26. Shahriari, B., Swersky, K., Wang, Z., Adams, R.P., de Freitas, N.: Taking the human out of the loop: A review of bayesian optimization. Proceedings of the IEEE **104**(1), 148–175 (2016)
27. Singh, U., Rizwan, M., Alaraj, M., Alsaidan, I.: A machine learning-based gradient boosting regression approach for wind power production forecasting: A step towards smart grid environments. Energies **14**(16), 1–21 (2021)
28. Theocharides, S., Theristis, M., Makrides, G., Kynigos, M., Spanias, C., Georghiou, G.E.: Comparative analysis of machine learning models for day-ahead photovoltaic power production forecasting. Energies **14**(4) (2021)
29. Yang, B., Zhu, T., Cao, P., Guo, Z., Zeng, C., Li, D., Chen, Y., Ye, H., Shao, R., Shu, H., Yu, T.: Classification and summarization of solar irradiance and power forecasting methods: A thorough review. CSEE Journal of Power and Energy Systems pp. 1–19 (2021)
30. Yang, D., Zhou, X., Yang, Z., Guo, Y., Niu, Q.: Low carbon multi-objective unit commitment integrating renewable generations. IEEE Access **8**, 207768–207778 (2020)

ClassBO: Bayesian Optimization for Heterogeneous Functions

Mohit Malu[1,3], Giulia Pedrielli[2], Gautam Dasarathy[1], and Andreas Spanias[1,3]

[1] School of ECEE, Arizona State University, Tempe AZ 85281, USA
[2] SCAI, Arizona State University, Tempe AZ 85281, USA
[3] SenSIP Center
{mmalu,giulia.pedrielli,gautamd,spanias}@asu.edu

Abstract. Bayesian Optimization (BO) frameworks typically assume the function to be optimized is stationary (homogeneous) over the domain. However, in many real-world applications, we often deal with functions that present a rate of variation across the input space. In this paper, we optimize functions where a finite set of homogeneous functions defined over partitions of the input space can represent the heterogeneity. The disconnected partitions that can be characterized by the same function are said to be in the same class, and evaluating the function at input returns the minimum distance to a boundary of the contiguous class (partition). The ClassGP modeling framework, previously developed to model for such heterogenous functions along with a novel ClassUCB acquisition function and partition sampling strategy, is used to introduce a novel tree-based optimization framework dubbed as ClassBO (Class Bayesian Optimization). We demonstrate the superior performance of ClassBO against other methods via empirical evaluations.

Keywords: Bayesian Optimization · Gaussian process · Black-box Optimization · Heterogeneous function · Non-stationary function

1 Introduction

Bayesian optimization (BO) has emerged as a powerful sequential optimization approach for non-convex expensive to evaluate black-box functions [1,3]. The standard BO typically assumes the function to be optimized is stationary (homogeneous) over the domain, which allows using a single covariance kernel function with constant hyperparameters over the entire domain to model the function accurately. However, in many emerging applications such as machine learning, neural networks, and cyber-physical systems, the function to be optimized is heterogeneous, and the standard BO framework cannot accurately model these functions, leading to inadequate optimization performance. Often, heterogeneous functions can be characterized by locally stationary and globally non-stationary functions, calling for more sophisticated optimization techniques that utilize the underlying structure to model and optimize these functions.

Many approaches have been proposed to extend BO for heterogeneous function optimization. Bayesian treed Gaussian Process (GP) in [5] and TuRBO in [2] use a collection of locally stationary GP's to model and optimize heterogeneous functions. Various methods that use non-stationary kernels have been proposed [8,4]. Methods

in [9,7] warp the input space to a new space where the function is stationary and the standard BO framework can be applied. Compared to these methods, our approach utilizes the underlying structure, which allows the sharing of information across non-contiguous partitions if they share the same function, i.e., they are in the same class [6]. Specifically, we adopt the modeling architecture presented in [6] to develop a novel tree-based optimization algorithm.

Our contributions include: (i) A novel ClassBO framework to optimize heterogeneous functions; (ii) A set of novel ClassBO acquisition functions; (iii) Empirical analysis of ClassBO with different acquisition functions and compare it against other optimization techniques.

2 Notations and Problem Setup

We want to find the optimum of a heterogeneous function $f : \mathcal{X} \subseteq \mathbb{R}^d \rightarrow \mathbb{R}$ where the heterogeneity can be represented by a finite set of homogeneous functions g_j's defined over the axis-aligned partitions of the input space. Further, multiple partitions associated with the same function belong to the same class. For many engineering systems, the structure of the non-stationarity is known or can be evaluated. Hence, the observation model is such that evaluating the function at any point $\mathbf{x}$ reveals function evaluation (y), the class label (z), and the minimum distance to a boundary of partition (w). The optimization problem is formally given as follows:

$$\underset{\mathbf{x}\in\mathcal{X}}{\arg\max}\, f(\mathbf{x}) = \underset{\mathbf{x}\in\mathcal{X}}{\arg\max} \left(\sum_{j=1}^{p} \mathbb{1}\{\mathbf{x} \in \mathcal{X}_j\} g_j(\mathbf{x}) \right) \tag{1}$$

3 The ClassBO Algorithm

The ClassBO algorithm has three key components - (i) Bayesian statistical model: the statistical modeling framework, ClassGP, introduced in [6] is used to model heterogeneous functions by formulating a distribution over the space of objectives and computing the posterior conditioned on the observed samples; (ii) Acquisition functions: A set of novel ClassBO acquisition functions described in Section3 are used to navigate the input space efficiently; (iii) Partition sampling strategy: A novel sampling strategy to learn the partitions of the input space accurately.

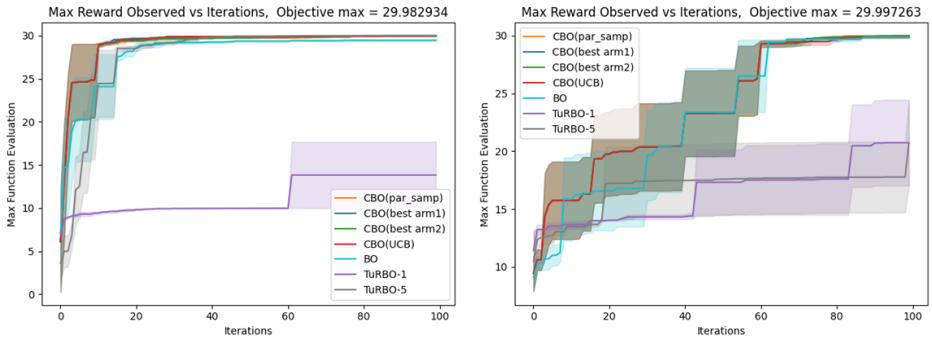
ClassUCB Acquisition Functions: BO algorithms use the acquisition function to decide where to sample in the next iteration, as the acquisition function can quantify the potential of finding an objective maximum at any given point in the input space. In this work, we formulate a set of novel acquisition functions for the ClassBO algorithm that uses mean and uncertainty estimates of the posterior distribution of the functions in each partition. One of the ClassUCB acquisition functions is given as follows:

$$\text{(CBO - UCB)} \quad \mathbf{x}_t = \underset{\mathbf{x}\in\mathcal{X}_j, j\in[p]}{\arg\max} \left(\mu_{j,t_j}(\mathbf{x}) + \beta_{t_j}^{1/2} \sigma_{j,t_j}(\mathbf{x}) \right)$$

Here, for the j^{th} partition learned using tailored CART algorithm of ClassGP, μ_{j,t_j}, σ_{j,t_j} represent the posterior mean and variance respectively of the function being modeled in the given partition after t_j iterations, β_{t_j} is a parameter that controls the trade-off between exploration and exploitation, and t is the current iteration of the algorithm. The ClassUCB acquisition function forms a set of points that maximize the UCB in each partition $j \in [p]$ and selects the point with maximum UCB from the set as the next sampling point.

Partition Sampling Strategy: ClassBO performs poorly when the partitions of the input space learned by the tailored CART algorithm are inaccurate or not complete. To resolve this, we run a proposed novel partition sampling strategy before using ClassUCB acquisition functions for sampling. The partition sampling strategy is a bottom-up approach applied to each partition learned by the tailored CART algorithm applied to initial samples. This approach constructs an axis-aligned hyper-rectangle for every sampled point of the length twice the minimum distance from the boundary (w) with the sampled point at the center and uniformly samples from the region not covered by the hyper-rectangles. This approach guarantees all the partitions of input space are learned accurately. The pseudo-code for the algorithm is given as follows:

Step 1: For each sampled point *construct* an axis-aligned hyper-rectangle of length twice the minimum distance from the boundary with the sampled point at the center.
Step 2: *Merge* hyper-rectangles that overlap or are sufficiently close within a given partition to form a larger hyper-rectangle.
Step 3: Uniformly *sample* from the regions that is not covered by the hyper-rectangles.
Step 4: *Stop* - If the final merged hyper-rectangle covers the entire partition else *repeat*.



(a) $p = 4$ (b) $p = 16$.

Fig. 1. Maximum observed reward vs iterations to compare of performance of ClassBO with ClassUCB acquisition functions and other baselines for fixed parameters.{number of classes $(k = 2)$, initial samples $(n = 5)$, iterations $(T = 100)$}

4 Simulation Results

Simulations were performed over the function constructed using standard optimization test functions. We plot the maximum observed reward (averaged over multiple runs) vs iterations to compare the performance. Following parameters are initialized for each simulation: dimension $(d = 2)$, initial samples $(n = 5)$, number of partitions $(p = 4, 16)$, number of classes $(k = 2)$, number of iterations $(T = 100)$, and for a fixed set of initialized parameters 5 independent simulation runs for each framework are performed for comparison. The results in Fig.1(a) shows that ClassBO outperforms BO and TuRBO algorithms. However, as the number of partitions increase, the partitions sampling strategy ends up sampling regions of lower interest to learn the partitions accurately, in turn leading to slower convergence to the optima as observed in Fig.1(b).

5 Conclusions and Future Work

In this paper, we propose a new tree-based ClassBO framework that uses ClassGP modeling technique for heterogeneous functions with access to class information. We also introduce a set of novel ClassUCB acquisition functions and compare the performance of ClassBO against other baselines. Additionally, we establish that the performance of ClassBO is heavily dependent on the accuracy of learning the partition that contains the maximum of the objective function. For future work, improving the sampling strategy by incorporating uncertainties pertaining to learned partitions instead of deterministic partition sampling strategy, scaling to higher dimensions, and theoretical analysis of the algorithm are promising avenues to explore.

References

1. Archetti, F., Candelieri, A.: Bayesian optimization and data science, vol. 849. Springer (2019)
2. Eriksson, D., Pearce, M., Gardner, J., Turner, R.D., Poloczek, M.: Scalable global optimization via local bayesian optimization. Advances in neural information processing systems **32** (2019)
3. Frazier, P.I.: A tutorial on bayesian optimization. arXiv preprint arXiv:1807.02811 (2018)
4. Gibbs, M.N.: Bayesian Gaussian processes for regression and classification. Ph.D. thesis, Citeseer (1998)
5. Gramacy, R.B., Lee, H.K.H.: Bayesian treed gaussian process models with an application to computer modeling. Journal of the American Statistical Association **103**(483), 1119–1130 (2008)
6. Malu, M., Pedrielli, G., Dasarathy, G., Spanias, A.: Class GP: Gaussian process modeling for heterogeneous functions. In: International Conference on Learning and Intelligent Optimization. pp. 408–423. Springer (2023)
7. Marmin, S., Ginsbourger, D., Baccou, J., Liandrat, J.: Warped gaussian processes and derivative-based sequential designs for functions with heterogeneous variations. SIAM/ASA Journal on Uncertainty Quantification **6**(3), 991–1018 (2018)
8. Paciorek, C.J., Schervish, M.J.: Spatial modelling using a new class of nonstationary covariance functions. Environmetrics: The official journal of the International Environmetrics Society **17**(5), 483–506 (2006)
9. Schmidt, A.M., O'Hagan, A.: Bayesian inference for non-stationary spatial covariance structure via spatial deformations. Journal of the Royal Statistical Society: Series B (Statistical Methodology) **65**(3), 743–758 (2003)

How evolutionary algorithms consume energy depending on the language and its level

Juan J. Merelo-Guervós[1][0000−0002−1385−9741] and Mario García-Valdez[2][0000−0002−2593−1114]

[1] Department of Computer Engineering, Automatics and Robotics, University of Granada, Granada, Spain
[2] Department of Graduate Studies, National Technological Institute of Mexico, Tijuana, Mexico
jmerelo@ugr.es, mario@tectijuana.edu.mx

Abstract. Making evolutionary algorithms greener implies tackling implementation issues from different angles. Practitioners need to focus on those that can be more easily leveraged, such as the choice of the language that is going to be used; high-level (interpreted), mid-level (based on multi-platform virtual machines), and low-level (native) languages will need power in different ways, and choosing one or the other will have an impact on energy consumption. We will be looking at the implementation of key evolutionary algorithm functions, in three languages at three different levels: the high-level JavaScript, the mid-level Kotlin, which runs on the Java Virtual Machine, and the low-level Zig. Looking beyond the obvious, as the lower the level, the less energy consumption should be expected, we will try to have a more holistic view of the implementation of the algorithms in order to extract best practices regarding its green implementation.

Keywords: Green computing, metaheuristics, JavaScript, energy-aware computing, evolutionary algorithms, zig, Kotlin

1 Introduction

The interest in greener computing has grown in the last decades. In part this has been fueled by the Millennium Development Goals[3] that advocate for a lower carbon footprint in human activity, but also due to the fact that performance improvements brought by faster hardware are not coming with the same speed as they used to [19].

There are, however, no universal solutions, so we need to focus on specific field; the general field of artificial intelligence has received a lot of attention lately; we are, however, interested in evolutionary algorithms [3], which, broadly speaking, fall within that field, but, unlike it, is not targeted by specific hardware processors relying on general CPU power (and its consumption). In the general sense, there are many different ways to lower the carbon footprint of computing

[3] Described in https://www.un.org/millenniumgoals/bkgd.shtml, for instance

workloads, however: as Hidalgo et al. affirm [7], there are four different levels for the evaluation and eventual improvement of energy consumption in AI (including, as in this paper, metaheuristics): just above the hardware level there is the programming language level, which includes not only the language itself, but also libraries and compilers or interpreters (inclusively named the *toolchain*. The latter was the topic of [15], where our focus was to work on different JavaScript interpreters in order to discover which one used system energy more efficiently.

However, there are many user cases in which the language used by all or part of the metaheuristic implementation can also be chosen. A simple choice of language will greatly impact energy consumption, as shown in [17] or [12]. In [16], we looked at the performance of different languages when implementing an evolutionary algorithm; in this paper, we will focus on the language level, before delving into specific language characteristics. Languages can be classified along many different axes: compiled vs. interpreted (or compiled natively vs. complied to bytecode, as is the case with Java and Kotlin), low vs. high-level, functional vs. procedural vs. object-oriented, for instance. In most cases, it is not a dichotomy, but a continuum; most interpreted languages, for instance, use Just-In-Time (JIT) compilers before actually running a script; most languages are also, nowadays, multi-paradigm, using procedural and functional as well as object-oriented features. Interpreted vs. compiled, low-level vs. high-level are still useful distinctions, however, from the point of view of performance as well as what interests us most in this line of research, energy consumption.

Thus, for this paper, we will work with languages that occupy different positions along those two axes. We will work with a high-level language, JavaScript, using the fastest implementation available, `bun`; this is also an interpreted language, a mid-level (that is, high-level from the point of view of language design, low-level in the sense that it is compiled to bytecode), compiled language, Kotlin, that compiles to bytecode of the Java virtual machine (JVM) and a low-level language, Zig, a relatively new language that is still not reached production, but is however used for `bun` itself[4]. All these languages have toolchains that are free software and thus can be used without any kind of limitation.

These languages can also be divided along two different axes that impact on their energy consumption:

- Memory management: it is done automatically in high-level languages, like JavaScript and Kotlin[5], while it involves a series of choices in the case of Zig. Inasmuch as allocating and releasing memory consumes energy, there is going to be a difference between having a runtime or interpreter make heuristic choices, or the developers making those decisions themselves.
- Compilation: JavaScript is interpreted, Kotlin is compiled to bytecode and then interpreted by the JVM, and Zig is compiled to native code, but this also implies what happens with optimization: high-level languages take all

[4] Please note that there are no languages that are considered low-level and interpreted, although there are minimalist interpreted languages like Lua [9]

[5] Which, for the purposes of this paper, is considered mid-level, since it compiles to bytecode

decisions, while Zig will allow you to choose between different optimization levels; it also means that a different kind of overhead will be incurred when running a program: JavaScript will load the interpreter in memory, and then parse and run the program; Kotlin will actually use the Java virtual machine to run, and it will be loaded and then the bytecode parsed and run; finally, Zig creates executables, which will have some overhead due to standard linked libraries.

In general, this implies that although *a priori* we can assume that a low-level language, being *closer to the iron*, will consume fewer resources, the fact that high- or mid-level languages apply heuristics and best practices to those decisions might eventually mean that, on the default case, the balance might be tipped towards high-level languages; this is going to be the main focus of this paper.

Making comparisons of energy spent by different implementations needs, first, a methodology to make choices that are comparable across all three platforms; then what exactly is going to be measured needs to be established, and how to make those measurements so that they make differences stand out. As in previous papers, we will focus on two critical evolutionary algorithm functions [1]: the fitness function, as well as *genetic* operators. However, we will use a heavier fitness function in this case: Hierarchical If and only IF (HIFF) [21] for independent measurement, as well as combine mutation and crossover when solving the OneMax optimization problem. In general, we are going for combined, or more complex, operations in this paper so that we can implicitly evaluate more features for every language, such as function calling or argument passing, which will have a different implementation, and thus energy overhead, in every language.

The rest of the paper is organized as follows: next, we present the state of the art in software engineering and its analysis of energy consumption of different software platforms, to be followed by the methodology we are going to apply in this paper, mainly concerned with evolutionary algorithms in 3. Then we will present the results of the experiments in 4, and finally, we will discuss the results and draw some conclusions in 5.

2 State of the art

The "shade of green" of different languages has been repeatedly examined in the literature, at least since it actually became a concern in software engineering [18], very recently, indeed. This concern has been systematized in what are called the GREENER principles [11], which try to provide a foundation for Environmentally Sustainable Computer Science (ESCS), and are placed at different functional and pragmatic levels, from governance to education (that would be the last E). In this paper, we are mainly concerned with two other Es, estimation and "Energy and embodied impact," as well as the second R, Research. These principles seek to monitor and then minimize the energy needs of computations; the Research principle, in turn, encourages investigating those topics. We will

call these principles EER, for short. The GREENER principles try to kick-start a feedback loop that allows researchers, and then developers in turn, to keep making decisions so that the same wallclock performance can be obtained with a lesser amount of energy.

The enunciation of these principles is very recent; however, they have been applied to different areas, in different forms in the case of evolutionary algorithms, these have been studied for some time [20] following the EER principles' point of view; however, it is interesting to note what these studies have focused in: population size [4], hardware platform [20], the kind of algorithm [6] or the specific interpreter chosen to run the algorithm written in a specific language, JavaScript [15]. Although in this last case, in specific situations, energy savings of 90% can be achieved, there are other possible choices that, in principle, might have a more significant impact without sacrificing performance (as would be the case when working with different hardware platforms).

One of the ways of applying these EER principles is researching the programming language where we are going to implement the workload itself. A comprehensive and recent study [17], that measures energy consumption on a general workload (the so-called CLBG corpus) shows that low-level languages like C, Rust or C++ are consistently coming on top. Mid-level languages like Java are placed 5th on the overall energy ranking, with almost double energy consumption. Interpreted languages like JavaScript (probably using the mainstream interpreter, node.js) are placed 17th, spending energy at four times the rate of C. The last language in the ranking, Perl, spends two orders of magnitude more than the baseline; Python is next to last, with a less significant difference. Choosing the right platform is not everything, however, [12] shows that the same algorithm implemented using different data structures might yield different levels of consumption; a similar problem is approached in [2], that shows that using the visitor pattern has a big impact in the energy profile of any program; however, the achieved reductions are very different depending on the language: while there is a slight reduction in the case of Java, the reduction achieved is dramatic in the case of C++. This probably shows that when we are comparing platforms we need to include the evaluation of higher-level components, that is, we need to go beyond the analysis of single functions.

In this paper, however, we will focus on the choice of programming languages based on how much energy they consume when applying the main operators of evolutionary algorithms. We also consider the role of higher-level language elements in its energy consumption. How we will be doing this work is presented next.

3 Methodology

An energy profiling methodology will start with selecting a tool to perform measurements, then present the workload that is going to be used, proceed to the different systems that are going to be measured, and end with the specific

versions of the instruments and systems to be used; after this, the circumstances under which the measurement is going to take place need to be presented.

The measuring instrument was already chosen in a previous paper [15]. In general, there are system-wide as well as per-process measuring tools. We settled for pinpoint [10], a command line tool under active development that takes per-process measures that are more accurate than other system-wide tools; it is also able to work with different sensor APIs across different operating systems, giving us an uniform tool to measure energy consumption.

In that paper, we selected a single integer arithmetic fitness function, MAX-ONES, and the crossover operation. However, in this paper we are taking a wider view of the problem so we will try to work with fundamental building blocks, but with ones that involve a bigger range of runtime system of the language. We will thus use two functions:

- The Hierarchical If and only IF (HIFF) [21] is an integer arithmetic function that works recursively, reducing a Boolean string of 0 and 1s by splitting it and applying the function to the halves; the final result will depend on the organization of 0s and 1s in the initial string, but will anyway involve many recursive function calls that will have to make use of the heap.
- The other function will apply crossover to randomly selected pairs of binary strings, followed by mutation to every member of the resulting pair, and evaluate the ONEMAX function on that result. These are essentially five function calls, not as complicated as before, but it is a fundamental operation in evolutionary algorithms and will essentially test the ability to access random elements in a string, as well as the creation of new ones (or the cloning of them, depending on the implementation)

We will use a workload of the same size as in the previous paper, 40K chromosomes, with chromosomes of different sizes: 512, 1024, and 2048 bits. This is a difference with respect to the previous paper, where we did not use 512 bits, using 4096 instead. We consider that the bigger size is not as realistic, and is relatively unlikely to be found in real problems; besides, the amount of time needed to compute HIFF was extremely high for the high-level JavaScript, so we decided to drop it so that experiments can take place in a relatively reasonable amount of time.

The languages have been chosen using a specific criterion:

- JavaScript is, as shown in the state of the art, one of the languages that consume the least energy as proved in [17]; in our previous paper, we have also found that using the bun interpreter can results in savings as high as 90% [15]. It is a high-level, object-oriented language that is, indeed, quite popular; popularity is the main criterion we have used to choose languages used. It has been preferred over other languages that might have a higher popularity in metaheuristics, such as Python, since the above-mentioned paper includes it as one of the languages that consume the most energy in a general payload; metaheuristics need not be an exception. At any rate, this paper is about choosing among kinds of languages, so any result that is

obtained might be extended to languages of the same kind; at the same kind, we propose a methodology for choosing the right system for metaheuristics, so more precise measurements would have to be performed on whatever alternative to JavaScript we want to introduce. The implementation of the above-mentioned function is open source and hosted at https://github.com/JJ/energy-ga-icsoft2023 with a free license. The actual code is the same used in the above mentioned paper.

- Kotlin is a language that compiles to the JVM, and in that sense, it would be comparable to Java in terms of energy consumption. However, data structures and other runtime characteristics might differ, as well as the workload used to compare it with other languages, so what we obtain in terms of ranking might be different from what [17] shows. As we mentioned above, and on a first approximation, results obtained here might be extended to other languages such as Java, Scala, or Clojure that also target the JVM. Kotlin has not been the target of any popular EA library as far as we can tell, although it is mentioned in this context in works such as [8]. The implementation that has been used is also included in the same repository and has been adapted from the one used in [13,14]. In that study, Kotlin was one of the fastest, even faster than Java in one of the versions. Being the main language used to create Android apps, its popularity makes it meet the main criterion used to choose languages in this paper.
- Finally, we use Zig as the low-level language instead of the more popular in EA circles C++, or, in general terms, Rust. We chose it mainly because the interpreter we use for JS is written using it; so we should expect less energy consumption when we lower the level of abstraction. On the other hand, it would be a totally new implementation of evolutionary algorithms, so this work could serve to introduce the language (and its possibilities) to practitioners. Again, the implementation is hosted in the same repository as the others used for this paper. This language cannot exactly be said to be as popular as the other two in this study; however, since it is the language used to implement the JavaScript interpreter `bun`, it was precisely the lowest level of the chosen high-level language, so we were interested to see how that would translate to differences in energy consumption. Another criterion used to choose this specific one is that, being an emergent language, there are no studies that we know of that work out its energy consumption.

All experiments for this paper have been carried out in a Linux machine `5.15.0-94-generic #104 20.04.1-Ubuntu SMP` using AMD Ryzen 9 3950X 16-Core Processor. These are the versions used for every tool and language:

- `pinpoint` does not have a version, but it has been compiled from commit `1578db07b1ee30318966d7a2097ee1bb219a9dc8`, October 26 2023.
- `bun` uses version 1.0.7.[6]

[6] Please note that at the time of writing this, many other versions have been published; we have used this one to be able to compare with previous papers. These same papers show that its performance and energy consumption have important improvements,

- zig uses version 0.11.0. This version, released by August 3, 2023, is the last one at the time of writing this paper.
- Kotlin version string is `1.9.22-release-704`; this includes the JVM version, OpenJDK 11.0.21.

All programs are run through a Perl script that captures and processes output, generating CSV data files that are committed to this repository, and available under a free license. Instructions to compile and run it are also included in the repository; in general, all that is needed to reproduce the results of this paper. All the code is automatically tested and tagged so that the exact version used in this paper can be retrieved.

Implementing an algorithm will always imply some choices in the specifics used to program it. In general, we opted for the default implementation (as suggested by tutorials), but explicitly, these are the decisions we took:

- `bun` uses the same implementation as previously used, namely, mutable strings, to represent the chromosome.
- zig, being a low-level language, needs a bigger set of choices. We have used `DefaultPrng` as random number generator, `page_allocator` as allocator, and arrays of byte-size integers (`u8`) for the chromosomes. This is the default implementation of integers in the language. The executable has been generated with default options too (`optimize` for optimization).
- Kotlin uses again the same implementation as [13], a `BooleanArray` for every chromosome.

All results shown below are averages for 15 runs for every configuration.

Please note that no sort of energy-wise optimizations have been performed on these implementations, since the main users of this work would be students and scientists with no deep knowledge of specific programming languages who want to create energy-efficient implementations of their algorithms. Every one of the languages has different levers that could be used to optimize energy, but that is not the focus of this paper. At any rate, if there is a close call in the results, the reader should take it into account and again measure energy consumption for specific workloads.

Previously, the experiment runner eliminated the baseline energy consumption by subtracting the average consumption of an idle process during the average time the experiments took; however, that implied that the energy (and time) measured included *generation* of the chromosomes, as well as the operations themselves.

In this paper, we will factor out that time, as well as the energy spent. Since the operation of generating chromosomes is made just once at the beginning of the run in evolutionary algorithms and is thus not very significant for evolutionary algorithm as a whole[7], subtracting it from the experiment run time will

so in case of close calls, we would recommend retaking measurements at the time of running your experiments

[7] We are not saying here that it is not an energy-consuming operation, even more so here where we are generating *all* chromosomes we are using in a single loop; however,

allow us to focus on the added energy consumption of the operations themselves; the results published in the next section will then subtract the average time and energy consumption of this operation, which is shown in Figure 1.

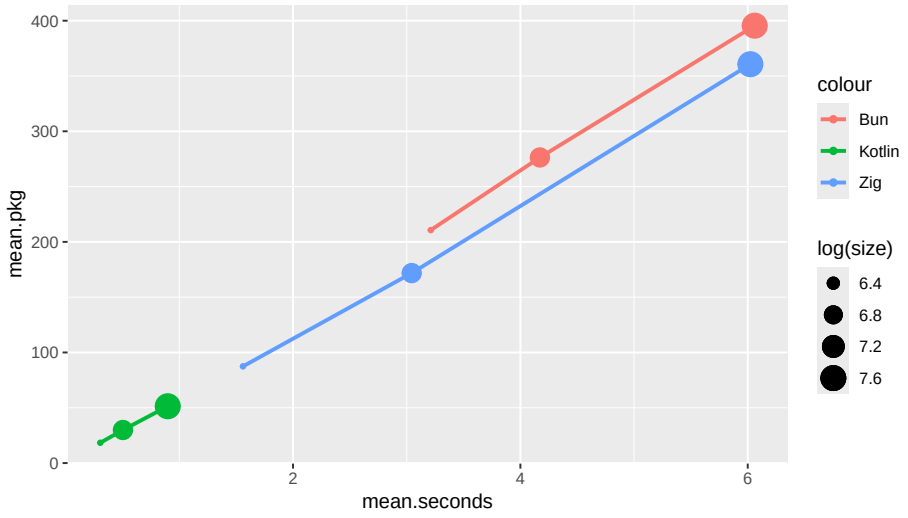


Fig. 1. Average running time and PKG (CPUs and memory) energy consumption generating 40K chromosomes for the three languages (represented with different colors); dot size is proportional to the logarithm of the chromosome size.

This Figure 1 shows average energy consumed (PKG, by CPU and memory `mean.pkg`, GPU use is negligible for this problem) vs. time taken (`mean.seconds`), already allows us at least an initial comparison of the orders of magnitude of the difference we should expect. Clearly, Kotlin is the fastest and also the one that consumes the least energy; it is followed by `zig`, although time as well as energy consumption grow very fast with the size of the chromosomes (here represented logarithmically as the dot size). `bun` is the slowest, as well as the one that consumes the most energy, although we should note that for the bigger size (2048 bits) `zig` takes almost the same amount of time, although with a lower consumption of energy[8].

while the initial population is generated only once in an EA, every other operation used here is going to be repeated every generation; thus, in terms of either time of energy consumption, the proportion employed by chromosome generation will be one vs. number of generations needed to find the solution

[8] `zig` will greatly vary the amount of energy needed depending on relatively irrelevant choices such as using constant or mutable pointers; in a previous experiment run, included in the repository, that used mutable pointers, energy consumption was almost 10% higher.

We will proceed to the experiments on an evolutionary algorithm workload, which will use this as a baseline.

4 Results

The first experiment will perform 40K crossover + mutation + onemax operations on chromosomes of size 512, 1024 and 2048.

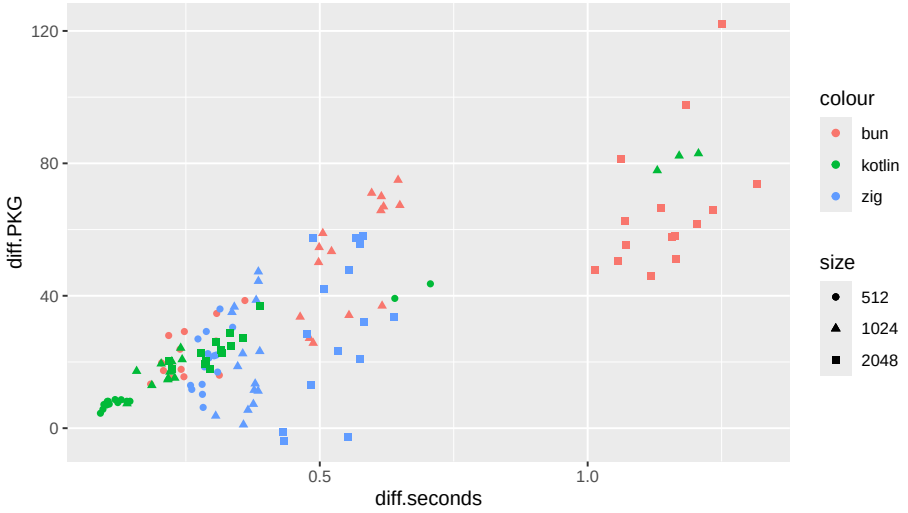


Fig. 2. Running time and PKG energy consumption processing 40K chromosomes via crossover, mutation and ONEMAX for the three languages (represented with different colors); dot shape represents the chromosome size.

Figure 2, shows the difference in running time and PKG energy consumption for the three languages examined. We have opted to show the results for every experiment in a single plot, to reveal trends as well as some quirks that might be explained more by language idiosyncrasies than by experimental errors. For instance, we can see that there are several cases where the Kotlin experimentation takes a long time, even for 1024 bits and 512 bits. One of the issues that the JVM has is garbage recollection; if there is a garbage recollection cycle it can clearly impact performance. The fact that it has hit a percentage of the experiments reveals that specific behaviors need to be taken into account and, to the extent that it is possible, mitigated.

Another quirk is the lower-than-0 energy consumption for zig, simply revealing that the difference in energy consumption is so low that it falls below the average for the generation of the chromosomes. zig's energy consumption is never higher than 60 Joules, anyway.

The trends need to be pondered, too. In general, bun is going to take longer and consume more energy than the rest for every size; Kotlin is the fastest and boasts the lower consumption of energy overall for smaller sizes. zig, on the other hand, is faster than bun, and its speed is quite consistent; not so for consumption of energy, which can go from very low, to even less than Kotlin in some experiments (and virtually zero), to relatively high, with a maximum closer to the average for bun.

Table 1. Average operations per Joule in the combined operations experiment for the three languages.

Language	Size	PKG average	PKG SD	Ops/Joule average
bun	512	22.26	7.58	1796.68
bun	1024	52.72	17.12	758.79
bun	2048	66.51	20.57	601.40
zig	512	20.01	8.34	1998.67
zig	1024	21.35	15.52	1873.42
zig	2048	30.82	22.36	1298.06
kotlin	512	12.00	12.02	3334.26
kotlin	1024	29.67	26.88	1348.22
kotlin	2048	23.71	5.10	1686.72

What we see in Table 1 is a complicated scenario, where Kotlin has a very high number of operations per Joule, with a power consumption as low as 12 Joules for the smallest size, although the trend is slightly different for the middle size of 1024 bits, where zig obtains the lowest energy consumption and operations per Joule, mainly due to the fact that the standard deviation for Kotlin, probably due to garbage recollection operations, is very high. Remarkably, the energy consumption for Kotlin can be as low as $1/3$ of what bun requires; zig requires half (although they can be very close for the smallest value, where Kotlin excels).

We need to check the speed and consumption of the selected fitness function, HIFF, for the three languages, so we can have a more complete picture of the energy profile for the three languages. This is a heavy-weight function, involving function calls in the order of one thousand. Even if it is going to take much more time than generating the chromosomes, we will still subtract the time needed for that operation to compare different things uniformly.

The operation for these functions needed some tweaking, namely, conversion to string in the case of Kotlin and to constant string in the case of zig. This adds what is essentially a copying operation to the fitness itself, but we decided to do that in order to have HIFF defined in the most similar way possible (working with strings of 0s and 1s).

The results for time and energy consumption are represented in Figure 3. We can already observe that the consumption is more than one order of magnitude higher than before, and since the number of function calls is dependent on the

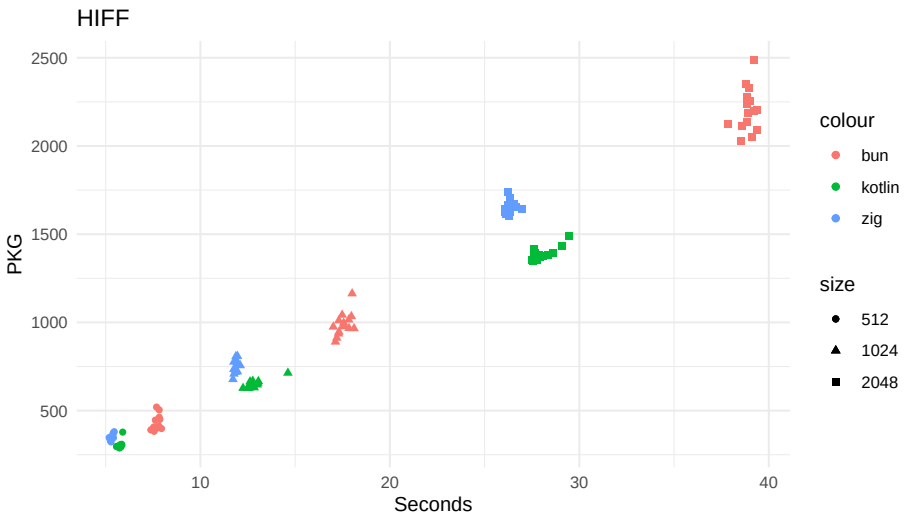


Fig. 3. Running time and PKG energy consumption computing the HIFF fitness function for 40K chromosomes for the three languages (represented with different colors); dot shape represents the chromosome size.

chromosome size, it grows more or less linearly with size. We can also observe that, as usual, bun is the slowest and the one that consumes the most energy.

However, the situation with zig and Kotlin is different. Kotlin is in most cases slightly slower, but also more energy-saving, thus more *green* than zig. We represent a boxplot comparing them in Figure 4.

Differences are significant for all three languages, for all sizes involved; in the case of the biggest size, there is indeed a considerable difference, with Kotlin spending less than 1500 Joules on average.

Table 2. Average operations per Joule in the HIFF experiment for the three languages.

Language	Size	PKG average	PKG SD	Ops/Joule average
bun	512	425.91	41.91	93.92
bun	1024	987.69	64.54	40.50
bun	2048	2204.06	123.40	18.15
zig	512	349.66	15.60	114.40
zig	1024	747.59	37.19	53.51
zig	2048	1651.50	35.18	24.22
kotlin	512	305.29	20.93	131.02
kotlin	1024	648.83	22.88	61.65
kotlin	2048	1386.58	38.43	28.85

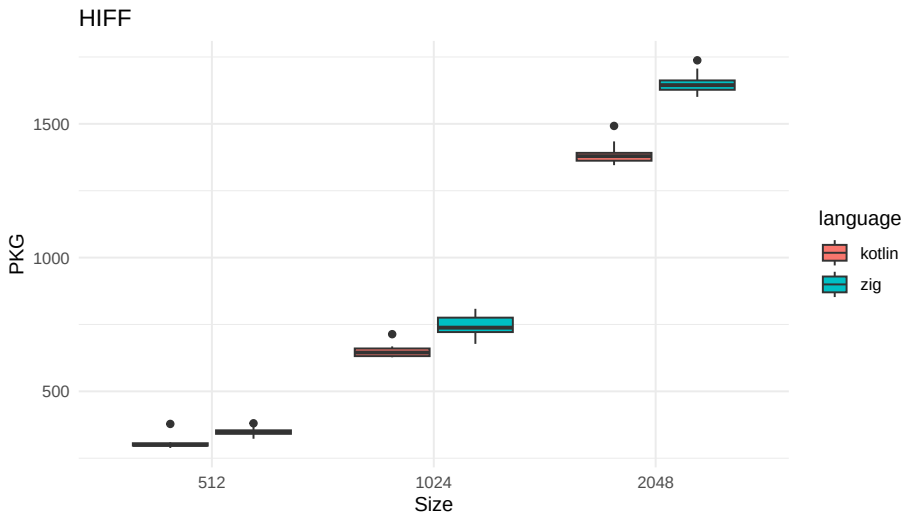


Fig. 4. Boxplot of the energy consumption of the HIFF fitness function for 40K chromosomes for Kotlin and zig

What we see in Table 2 is the number of operations per Joule all three languages are able to perform.

5 Discussion, conclusion and future work

In this paper, we have performed a series of experiments running an evolutionary algorithm workload using three different languages with different *levels* (different ways of running the –possibly compiled– code), which translate to different ways of running the algorithms. Focusing on a set of operations allows us to create specific energy profiles for the algorithm implementations, as well as the languages. The methodology followed enables to benchmark the energy consumption of the different languages, as well as to differentiate them; it does not consume an excessive amount of time. Showing the results as operations per Joule focuses the the attention on how "green" every language is, by showing precisely how many operations can be performed with a certain amount of energy.

In this work, we compared the performance regarding power consumption of three languages, JavaScript (using the bun high-performance interpreter), Kotlin (using a free JVM), and zig, when implementing certain important operations in evolutionary algorithms. We have used two workloads, one with a combination of genetic operators and the OneMax fitness function (which would be a lightweight, although widely used, combination), and another with the HIFF fitness function, which is not only heavier in terms of operations, but also recursive, thus involving specific language overheads.

In all cases (that include also the generation of the chromosomes measured, used as baseline) we have found that the combination of Kotlin and the JVM is the most energy-efficient language (as well as the faster, although that was not the main focus of the experiment), followed by zig, and finally JavaScript using bun as interpreter. The difference is bigger for core-only operations, where Kotlin can be twice as fast as bun, than in other operations that involve the overhead of function calls, like HIFF, where the difference is much smaller, around 5% for zig and Kotlin in the chromosomes with the biggest size. The fact that zig can be slightly faster than Kotlin in this last case is not relevant from the point of view of the energy efficiency; Kotlin still needs less energy even if it takes a bit longer; you can trade off easily energy efficiency for speed in this case, since the average difference is just a few percentage points.

The relatively small difference between Kotlin and zig might imply that with some engineering and optimizations, zig energy efficiency might be on a par with Kotlin. Low-level languages have many different ways of optimizing its design and tooling, and we did not create the zig program at an expert level. However, this level is consistent with the use case we are giving the application, those of a scientist acting as a developer, not an expert developer of system software (which is the most common use case for zig). We cannot then affirm that Kotlin is the most efficient across the board, but we can definitely propose it as the *greener* technology for the specific case of evolutionary algorithms that do not have an extensive amount of GPU use; this would include mainly tests for new operators or, in general, new methodologies for evolutionary algorithms that use common fitness functions.

The interesting thing about modern experimental setup is that different languages can be easily mixed in a single application. As we can see in the experiments above, energy expenses of fitness functions can be twice as high as the rest of the operations; using *green* languages exclusively in this case, leaving the rest of the application to easier-to-use interpreted languages will not have a big impact in the energy profile, while simplifying the development as well as the integration within current frameworks such as DEAP [5].

As future lines of work, we plan to extend this study to other languages and workloads, especially those that use different kind of virtual machines like Haskell or Elixir. On the algorithmic side, another research venue is to analyze the energy consumption of languages implementing different concurrency models, implemented either by language constructs or through specialized virtual machines. We also plan to analyze the energy consumption of different hardware platforms (including virtualized cloud platforms), and to develop a tool to help researchers and developers to choose the most energy-efficient language for their needs. Different fitness functions will also present a different energy profile, so including them in the investigation might help us have a more complete dashboard to be able to minimize resource consumption of EA implementations. Fitness function that employ floating-point arithmetic would need to use the GPU, with a totally different energy profile, so that will need to be taken into account.

An investigation focused on zig would also help understand how low-level languages can be made *greener* and to what extent energy efficiency can be improved.

Acknowledgements

This work is supported by the Ministerio español de Economía y Competitividad (Spanish Ministry of Competitivity and Economy) under project PID2020-115570GB-C22 (DemocratAI::UGR).

We are also very grateful to the zig community, without which programming these operations in that language would have been impossible.

Data availability

The source of this paper as well as the data and whole writing history is available from https://github.com/JJ/energy-ga-icsoft-2023 under a GPL license.

References

1. Abdelhafez, A., Alba, E., Luque, G.: A component-based study of energy consumption for sequential and parallel genetic algorithms. The Journal of Supercomputing **75**, 6194–6219 (2019)
2. Connolly Bree, D., Ó Cinnéide, M.: Energy efficiency of the visitor pattern: contrasting Java and C++ implementations. Empirical Software Engineering **28**(6), 145 (2023)
3. Corne, D., Lones, M.A.: Evolutionary Algorithms, p. 1–22. Springer International Publishing (2018). https://doi.org/10.1007/978-3-319-07153-4_27-1, http://dx.doi.org/10.1007/978-3-319-07153-4_27-1
4. Díaz-Álvarez, J., Castillo, P.A., Fernandez de Vega, F., Chávez, F., Alvarado, J.: Population size influence on the energy consumption of genetic programming. Measurement and Control **55**(1-2), 102–115 (2022)
5. Fortin, F.A., De Rainville, F.M., Gardner, M.A.G., Parizeau, M., Gagné, C.: DEAP: Evolutionary algorithms made easy. The Journal of Machine Learning Research **13**(1), 2171–2175 (2012)
6. Garg, K., Jindal, C., Kumar, S., Juneja, S.: Analyzing and rating greenness of nature-inspired algorithms. In: 6th International Conference on Innovative Computing and Communication (ICICC 2023) (2023)
7. Hidalgo, I., Fenández-de_Vega, F., Ceberio, J., Garnica, O., Velasco, J.M., Cortés, J.C., Villanueva, R., Díaz, J.: Sustainable artificial intelligence systems: An energy efficiency approach (2023)
8. Holló-Szabó, Á., Albert, I., Botzheim, J.: Statistical racing crossover based genetic algorithm for vehicle routing problem. In: 2021 IEEE 21st International Symposium on Computational Intelligence and Informatics (CINTI). pp. 000267–000272. IEEE (2021)
9. Ierusalimschy, R., de Figueiredo, L.H., Celes, W.: The evolution of Lua. In: Proceedings of the third ACM SIGPLAN conference on History of programming languages. pp. 2–1 (2007)

10. Köhler, S., Herzog, B., Hönig, T., Wenzel, L., Plauth, M., Nolte, J., Polze, A., Schröder-Preikschat, W.: Pinpoint the Joules: Unifying runtime-support for energy measurements on heterogeneous systems. In: 2020 IEEE/ACM International Workshop on Runtime and Operating Systems for Supercomputers (ROSS). pp. 31–40 (2020). https://doi.org/10.1109/ROSS51935.2020.00009
11. Lannelongue, L., Aronson, H.E.G., Bateman, A., Birney, E., Caplan, T., Juckes, M., McEntyre, J., Morris, A.D., Reilly, G., Inouye, M.: Greener principles for environmentally sustainable computational science. Nature Computational Science **3**(6), 514–521 (2023)
12. Lima, L.G., Soares-Neto, F., Lieuthier, P., Castor, F., Melfe, G., Fernandes, J.P.: Haskell in green land: Analyzing the energy behavior of a purely functional language. In: 2016 IEEE 23rd international conference on Software Analysis, Evolution, and Reengineering (SANER). vol. 1, pp. 517–528. IEEE (2016)
13. Merelo-Guervós, J.J., Blancas-Alvarez, I., Castillo, P.A., Romero, G., García-Sánchez, P., Rivas, V.M., García-Valdez, M., Hernández-Águila, A., Román, M.: Ranking the performance of compiled and interpreted languages in genetic algorithms. In: Proceedings of the International Conference on Evolutionary Computation Theory and Applications, Porto, Portugal. vol. 11, pp. 164–170 (2016)
14. Merelo-Guervós, J.J., Blancas-Álvarez, I., Castillo, P.A., Romero, G., García-Sánchez, P., Rivas, V.M., García-Valdez, M., Hernández-Águila, A., Román, M.: Ranking programming languages for evolutionary algorithm operations. In: Applications of Evolutionary Computation: 20th European Conference, EvoApplications 2017, Amsterdam, The Netherlands, April 19-21, 2017, Proceedings, Part I 20. pp. 689–704. Springer (2017)
15. Merelo-Guervós, J.J., García-Valdez, M., Castillo, P.A.: An analysis of energy consumption of JavaScript interpreters with evolutionary algorithm workloads. In: Fill, H., Mayo, F.J.D., van Sinderen, M., Maciaszek, L.A. (eds.) Proceedings of the 18th International Conference on Software Technologies, ICSOFT 2023, Rome, Italy, July 10-12, 2023. pp. 175–184. SCITEPRESS (2023). https://doi.org/10.5220/0012128100003538, https://doi.org/10.5220/0012128100003538
16. Merelo-Guervós, J.J., Romero, G., García-Arenas, M., Castillo, P.A., Mora, A.M., Jiménez-Laredo, J.L.: Implementation matters: Programming best practices for evolutionary algorithms. In: Cabestany, J., Rojas, I., Caparrós, G.J. (eds.) IWANN (2). Lecture Notes in Computer Science, vol. 6692, pp. 333–340. Springer (2011)
17. Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J.P., Saraiva, J.: Ranking programming languages by energy efficiency. Science of Computer Programming **205**, 102609 (2021). https://doi.org/https://doi.org/10.1016/j.scico.2021.102609, https://www.sciencedirect.com/science/article/pii/S0167642321000022
18. Pinto, G., Castor, F.: Energy efficiency: a new concern for application software developers. Communications of the ACM **60**(12), 68–75 (2017)
19. Theis, T.N., Wong, H.S.P.: The end of Moore's Law: A new beginning for information technology. Computing in science & engineering **19**(2), 41–50 (2017)
20. de Vega, F.F., Chávez, F., Díaz, J., García, J.A., Castillo, P.A., Merelo, J.J., Cotta, C.: A cross-platform assessment of energy consumption in evolutionary algorithms. In: Handl, J., Hart, E., Lewis, P.R., López-Ibáñez, M., Ochoa, G., Paechter, B. (eds.) Parallel Problem Solving from Nature – PPSN XIV. pp. 548–557. Springer International Publishing, Cham (2016)
21. Watson, R.A., Hornby, G.S., Pollack, J.B.: Modeling building-block interdependency. In: Parallel Problem Solving from Nature—PPSN V: 5th International Con-

ference Amsterdam, The Netherlands September 27–30, 1998 Proceedings 5. pp. 97–106. Springer (1998)

Minimizing evolutionary algorithms energy consumption in the low-level language Zig

Juan J. Merelo-Guervós[1][0000−0002−1385−9741]

Department of Computer Engineering, Automatics and Robotics, University of Granada, Granada, Spain

Abstract. Managing energy resources in scientific computing implies awareness of a wide range of software engineering techniques that, when applied, are able to minimize the energy footprint of experiments. In the case of evolutionary computation, we are talking about a specific workload that includes the generation of chromosomes and operations that change parts of them or access and operate on them to obtain a fitness value. In a low-level language such as Zig, we will show how different choices will affect the energy consumption of an experiment.

Keywords: Green computing, metaheuristics, energy-aware computing, evolutionary algorithms, zig

One of the concerns in modern evolutionary computing is reducing the amount of energy spent in experiments, trying to make nature-inspired computing more nature-friendly [6]. This involves developing a methodology to measure energy spent, as well as identifying the EC operations that consume the most energy. In [4] we settled on a language and OS-independent set of tools, but also followed [1] in choosing the set of operations under measure: mutation, crossover and a simple fitness evaluation, ONEMAX.

In [4] the main factor under study was the different interpreters used in a high-level language, JavaScript. In the case of low-level languages like zig, a language that emphasizes safety and maintainability [2], there is a single compiler, but there are several choices to be made, even if the defaults should provide enough performance and energy efficiency. Yet, in general, developers, and even more so scientific ones, are generally unaware of the energy impact of their algorithm implementations [5], not to mention techniques available for their reduction [3].

In this paper we will work on a generic evolutionary algorithm workload, and see what the impact of different choices will have on its energy consumption. With this, we will try to find some best practices that will help practitioners implement evolutionary algorithms in zig or other low-level languages.

The experiment setup will match the one used in [4], using the same tools for energy profiling (pinpoint) as well as Perl scripts to run the experiments and process the results. All experiments for this paper have been carried out in a Linux machine `5.15.0-94-generic #104 20.04.1-Ubuntu SMP` using AMD Ryzen 9 3950X 16-Core Processor. These are the versions used for every tool

and language, with zig version 0.11.0, released by August 3, 2023, which is the last stable one at the time of writing this paper. The Perl scripts generate CSV files that are then processed and plotted using R embedded in the source code of this paper. All code, data and source for this paper are available at https://github.com/JJ/energy-ga-icsoft2023 under a free license.

There are several units whose consumption can be measured using pinpoint via the RAPL interface; since the use of GPU is negligible in these examples, only memory and CPUs will be measured. Together, they are called the *package* (alongside with caches and memory controllers); this is usually represented by the acronym PKG.

We will be examining choices in three different areas

- By default, zig adds debug information to the resulting binary, without performing any kind of optimization. We will test the impact of using the ReleaseFast option when building binaries.
- The first version used strings for representing chromosomes. We will test arrays of Boolean values instead, which is a primitive type too.
- Unlike other languages, zig provides different memory allocators, which can be chosen by the developer. By default, a page allocator is used, but there is the possibility of using a fixed buffer size allocator.

We will first generate 40000 chromosomes of size 512, 1024 and 2048, and measure the energy consumption and running time of this operation; every combination is run 15 times. Not all combinations of the three techniques above could be tested, we show the results in Figure 1, along with the baseline that was compiled using default settings, character strings and page allocator. The other choices tested are: Built with ReleaseFast option (tagged "ReleaseFast"), built in the same way and using a Fixed Buffer Allocator ("ReleaseFastFBA"), built with default options, using Boolean arrays as well as the Fixed Buffer Allocator ("Boolean")[1].

This Figure 1 shows the dramatic change in energy consumption (as well as performance) that can be obtained just by changing compiling and programming options. Using a fixed buffer alongside the ReleaseFast option implies a 95% reduction in the energy consumed[2]. Even with the default compilation options, applying changes in the allocator and the data structure used reduces by 50% energy used. The impact on running time, although not the focus of this paper, is also significant.

We will now run an experiment that, after generating the 40K chromosomes, will perform crossover + mutation + onemax operations on chromosomes of size 512, 1024 and 2048 [3]. In this case, the combination of fixed buffer allocator plus ReleaseFast has been skipped, since beyond the initial generation of

[1] This combination proved difficult in practice, crashing the program for size 2048; it did not work with the ReleaseFast option either

[2] We should take into account that generating chromosomes uses allocation heavily, so this is the kind of operation that would be the most impacted

[3] As indicated above, we could not make the Boolean bitstring along with fixed buffer allocation work for the biggest size

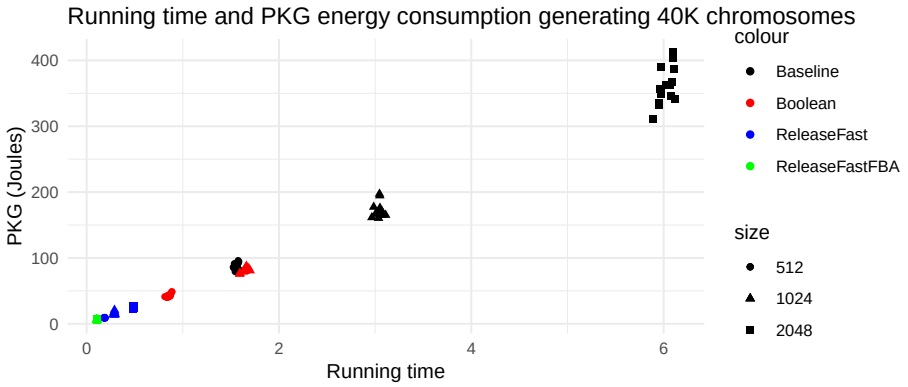


Fig. 1. Average running time and PKG energy consumption generating 40K chromosomes for the different techniques used (represented with different colors); dot shape represents the chromosome size.

chromosomes, there is barely any allocation happening; in fact, just two temporary bitstrings during the crossover operation. Comparisons for the other two combinations plus baseline are shown in Figure 2.

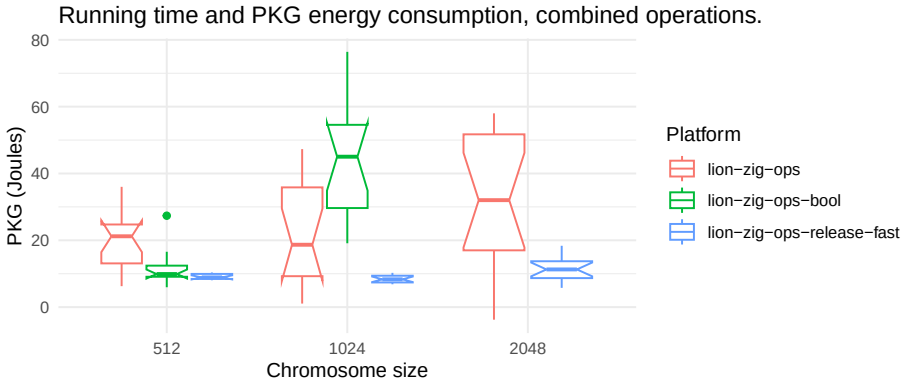


Fig. 2. Boxplot of PKG energy consumption processing 40K chromosomes via crossover, mutation and ONEMAX for different combinations of optimization techniques in Zig

This case is remarkably different to the one shown in Figure 1. For starters, the energy consumption is very low, one order of magnitude lower; regular genetic operations do not spend a lot of energy indeed, with most consumption focused on fitness functions. Even so, the fast compilation spends half the energy of the other combinations. Remarkably, using a different data structure does not always

imply a better energy profile; it does so only for the smaller size, 512 bits (which is anyway closer to usual chromosome sizes).

In this paper we have used different techniques that potentially could reduce the amount of energy spent by a genetic algorithm implemented using the low-level language zig: testing different data structures, allocation policies and compilation options. The most dramatic reduction is achieved with the *fast* compilation policy, but additional improvements can be obtained by using adequate data structures and allocation policies. By using best practices in this area, zig implementations can indeed be considered the *greener* ones among other high-level or Java Virtual Machine based languages. it remains as future work, however, how applying similar techniques when available will impact the energy consumption of other languages.

Acknowledgements and data availability

This work is supported by the Ministerio español de Economía y Competitividad (Spanish Ministry of Competitivity and Economy) under project PID2020-115570GB-C22 (DemocratAI::UGR). We are also very grateful to the zig community for helpful tips. Source and data available from https://github.com/JJ/energy-ga-icsoft-2023 under a GPL license.

References

1. Abdelhafez, A., Alba, E., Luque, G.: A component-based study of energy consumption for sequential and parallel genetic algorithms. The Journal of Supercomputing **75**, 6194–6219 (2019)
2. Friesen, A.: Designing programming languages for writing maintainable software (2023), https://digitalcommons.unl.edu/cgi/viewcontent.cgi?article=1625&context=honorstheses
3. Longo, M., Rodriguez, A.V., Mateos Diaz, C.M., Zunino Suarez, A.O.: Reducing energy usage in resource-intensive java-based scientific applications via micro-benchmark based code refactorings (2019), http://hdl.handle.net/11336/121006
4. Merelo-Guervós, J.J., García-Valdez, M., Castillo, P.A.: An analysis of energy consumption of JavaScript interpreters with evolutionary algorithm workloads. In: Fill, H., Mayo, F.J.D., van Sinderen, M., Maciaszek, L.A. (eds.) Proceedings of the 18th International Conference on Software Technologies, ICSOFT 2023, Rome, Italy, July 10-12, 2023. pp. 175–184. SCITEPRESS (2023). https://doi.org/10.5220/0012128100003538, https://doi.org/10.5220/0012128100003538
5. Pang, C., Hindle, A., Adams, B., Hassan, A.E.: What do programmers know about software energy consumption? IEEE Software **33**(3), 83–89 (2016). https://doi.org/10.1109/MS.2015.83
6. Fernández de Vega, F., Díaz, J., García, J.Á., Chávez, F., Alvarado, J.: Looking for energy efficient genetic algorithms. In: Idoumghar, L., Legrand, P., Liefooghe, A., Lutton, E., Monmarché, N., Schoenauer, M. (eds.) Artificial Evolution. pp. 96–109. Springer International Publishing, Cham (2020)

CLS-Luigi: Analytics Pipeline Synthesis

Anne Meyer[1][0000−0001−6380−1348], Hadi Kutabi[1⋆][0000−0002−6023−7742], Jan Bessai[2][0000−0002−8506−0808], and Daniel Scholtyssek[3⋆][0009−0004−0258−5767]

[1] Information Management in Engineering, Karlsruhe Institute of Technology, Germany {anne.meyer,hadi.kutabi}@kit.edu
[2] Statistisches Bundesamt, Germany jan.bessai@destatis.de
[3] Corporate Logistics, TU Dortmund University, Germany daniel.scholtyssek@tu-dortmund.de

Abstract. We present CLS-Luigi, a framework for synthesizing analytics pipelines that enable prediction and decision-making. Analytics pipelines typically consist of a number of diverse steps ranging from simple preprocessing to informed machine learning and optimization algorithms. Implementing pipelines and selecting algorithms is time-consuming and the performance of selected algorithms is interdependent. CLS-Luigi improves the implementation process by adapting and using a well-established synthesis framework, Combinatory Logic Synthesizer (CLS), to automatically generate pipeline variants based on a repository of typed components that are implemented in Python. Luigi, a pipeline framework developed by Spotify, executes pipeline variants efficiently and allows intermediate results to be shared among similar pipelines to optimize resource utilization. We demonstrate the simplicity, expressive power, and run-time gains of CLS-Luigi through two examples: First, we show that CLS-Luigi has the modeling capabilities to automatically generate machine learning pipelines that cover all algorithms included in AutoSklearn, and delivers high-quality results. Second, we go beyond the capabilities of AutoML by facilitating the consistent synthesis and execution of decision pipelines covering different optimization paradigms. Our framework is easily accessible, open-source, and compatible with most existing tools and Python libraries used in data analytics.

Keywords: Data-driven decision-making · Pipeline synthesis · Machine learning · AutoML

1 Introduction

The goal of data analytics is to apply algorithms to data in order to make good decisions [4]. In predictive analytics, data is used to predict future trends, or outcomes based on patterns identified in historical data using machine learning (ML). Prescriptive analytics applications automate decision-making by applying optimization methods that range from simple rules to complex mathematical

⋆ This work was partly funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 502552827;276879186.

approaches. Due to the complexity of real-world applications, the predictive or decision-making process is typically broken down into discrete tasks. Each task is handled by a component. Linked components form pipelines [11].

ML pipelines include algorithms for data cleaning, feature preprocessing and selection, and one or several predictive models. In addition, models tailored to specific problems by incorporating prior knowledge (also referred to as informed ML) are becoming increasingly important [25].

Decision-making in real-world is often subject to uncertainty. The simplest pattern for dealing with uncertainty is the predict-then-optimize paradigm [5]: In the first step, a ML pipeline provides point estimates for uncertain parameters. In the second step, an optimization approach solves the (nominal) optimization problem using the estimates as input.

For most tasks in predictive and prescriptive analytics, there exist different algorithms, often with large parameter spaces. Moreover, the performance of one algorithm depends on the performance of an upstream algorithm. Thus, the design space is large, and building analytics pipelines is a complex, time-consuming task.

Automated machine learning (AutoML) frameworks, such as AutoSklearn [6], significantly advance the automatic creation and optimization of ML pipelines but lack extensibility, struggling to integrate informed ML models or decision-making algorithms. Pipeline frameworks, on the other hand, accommodate a broad range of computations with data input and output but do not automate pipeline generation.

Our vision with CLS-Luigi is to provide a framework that automatically generates analytics pipelines based on algorithmic repositories tailored to recurring problems in different domains. The contribution of this paper is twofold: (1) We delineate the core concepts and components of CLS-Luigi and illustrate its application through a small coding example. (2) We show the simplicity, the expressive power, and the run time gains of CLS-Luigi by two examples: First, we model and automatically generate classification pipelines based on the repository of algorithms used in AutoSklearn. Second, we model and generate pipelines for shortest path optimization with uncertain cost, facilitating a consistent comparison of two optimization paradigms.

The paper is organized as follows: We start with a brief review of the related literature in Section 2, followed by the introduction of our framework in Section 3. Sections 4 and 5 detail the ML and the decision pipeline example, respectively. We conclude in Section 6 by discussing strengths, limitations, and future research directions. All presented examples are available at https://github.com/cls-python/cls-luigi_paper.

2 Related Work

Data pipelines are collections of interconnected tasks or actions that transform data in a specific order to yield specific results [9]. They can be modeled as

directed acyclic graphs, where nodes represent transformations and arrows represent data flow (and dependencies). Pipeline frameworks such as Luigi [13], Apache Airflow [1], and Metaflow [17] have been developed to address various issues, e.g., dependencies and error handling, scalability, portability, and deployment [26]. Although newer frameworks are technically more versatile, Luigi is still popular because of its ease of learning and negligible setup effort.

The field of AutoML involves generating and optimizing ML pipelines. A good overview of the field is given in [27,10]. AutoML frameworks build pipelines that usually include steps for data cleaning and preprocessing, feature selection, and model training [27]. The automation of pipeline generation is abundantly present in AutoML: AutoSklearn [6] uses Bayesian optimization to search a hierarchical search space consisting of ML algorithms and their hyperparameters, and sets the structure of resulting pipelines using a template [27]. AlphaD3M [12] employs a Monte Carlo Tree Search (MCTS) with a neural network to select pipeline steps incrementally, while relying on context-free grammar that describes the pipeline structure and permitted algorithms. DSWIAZRD [28] also uses MCTS, yet selects algorithms based on the meta-features of the intermediate datasets. AutoML frameworks usually use a pre-defined set of algorithms and are not easily adaptable to domain-specific problems. In [23], the authors show this issue and present the AutoML framework ML-Plan-RUL, which is tailored to estimating the remaining useful life of machine components. Moreover, the AutoML Toolkit (AMLTK) [2] framework addresses this exact issue and offers components for modeling, generating, and optimizing ML pipelines based on a template. Yet, it only handles algorithms with the Scikit-Learn [18] interface, since it compiles the resulting pipelines into Scikit-Learn pipeline objects.

Decision-making in real-world is typically complex and often subject to uncertainty. Predict-then-optimize is a straightforward and very common pattern in analytics practice [5]. Alternatives to managing uncertainty include stochastic optimization, which models uncertainty as a distribution during the optimization process, and robust optimization, which accounts for uncertainty by considering minimum and maximum values [19]. Recent approaches integrate the optimization step into learning. An example is SPO+ [5]. To ease the comparison between optimization paradigms, PyEPO [22], a library that implements some of these methods, has been developed, yet pipeline construction remains manual.

Software synthesis is a well-researched field, which by the late 1970s already had become too vast to cover within one paper [16]. Approximately 45 years later, it is impossible to keep up with or even summarize all published software synthesis approaches, which is why we focus our discussion on the approach we use as a basis for CLS-Luigi. The CLS (Combinatory Logic Synthesizer) framework is a well-established tool for component-oriented synthesis based on finite combinatory logic with intersection types [20], which is flexible enough to support multiple languages [3]. It has proven to be useful in a variety of engineering applications, such as, for example, robotics [21], manufacturing [15], scheduling [14], and planning [7]. CLS uses a type-based specification for components and enumerates all possible well-typed solutions for a given target type. This is

similar to functionalities of dependency injection tools allowing to automatically inject instances of a given type into code, but CLS can enumerate all possible candidates instead of enforcing the uniqueness of injection targets. The framework has a Python implementation, making it compatible with many of the analytics pipeline frameworks and libraries. Our work goes beyond prior uses of CLS by focusing on general analytics pipelines instead of a fixed narrow field of engineering problems and providing a full integration into Luigi data pipelines.

3 CLS-Luigi Framework

The CLS-Luigi framework is best introduced by a minimal practical example. We want to run four separate ML pipelines using two scaling algorithms and two classifiers. As shown in Figure 1(A), we first load the well-known Iris dataset from Scikit-Learn [18], then scale the features (using *RobustScaling* and *MinMaxScaling*), and finally train a classifier (*DecisionTree* or *RandomForest*).

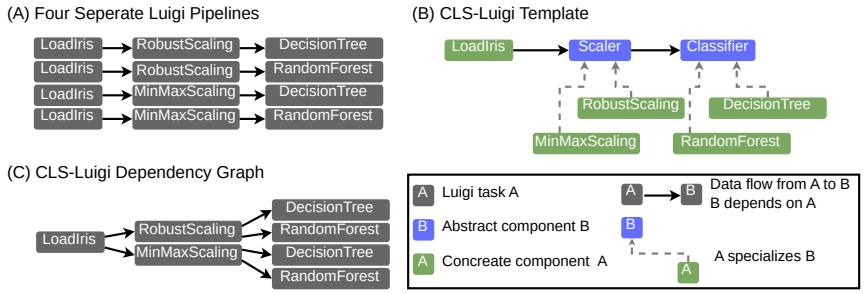


Fig. 1. Visualization of the minimal example.

Instead of explicitly modeling all pipelines separately in Luigi, we define the CLS-Luigi template in Figure 1(B). The *LoadIris* task is defined as a concrete component, while the *Scaler* and *Classifier* are defined as abstract components. *RobustScaling* and *MinMaxScaling* are concrete components that specialize the *Scaler* component, and hence, they are valid substitutions. Similarly, the *Classifier* component is specialized using *DecisionTree* and *RandomForest* components. This specialization mechanism is realized using class inheritance in Python. The implementation of pipelines with CLS-Luigi is very similar to the implementation with Luigi: Classes of task components inherit from both `luigi.Task` and `luigiCombinator`. They implement the methods `requires()` (to specify dependencies), `output()` (to specify a resource written when the task is done), and `run()` (to actually perform the task). In what follows, we present the code blocks for implementing and executing our minimal example:

(1) Define *LoadIris* Component This component downloads the dataset in the `run()` method and saves it using the path specified in `output()`. Since this component has no requirements, `requires()` is not implemented.

```
import luigi
from cls_luigi.inhabitation_task import LuigiCombinator

class LoadIris(luigi.Task, LuigiCombinator):

    def output(self):
        return luigi.LocalTarget("iris.csv")

    def run(self):
        from sklearn.datasets import load_iris
        ds = load_iris(as_frame=True).frame
        ds.to_csv(self.output().path, index=False)
```

(2) Define Abstract *Scaler* Component We implement an abstract component (placeholder) for scaling algorithms and make sure that the class variable `abstract` is set to `True`. To include LoadIris as a dependency, we (1) set the class variable `iris` as a `ClsParameter` to the type returned when constructing *LoadIris* and (2) return an instance of it in the `requires()` method. As shown under `run()`, this component loads the saved data from the *LoadIris* (using method `input()` which is automatically provided by Luigi based on `requires()`), and scales the features using the scaler provided by `get_scaler()`. The scaled features and labels are saved with the path specified in `output()`. The output file names are annotated with the task ID to ensure unique output file names.

```
from cls_luigi.inhabitation_task import ClsParameter
import pandas as pd

class Scaler(luigi.Task, LuigiCombinator):
    abstract = True # placeholder
    iris = ClsParameter(tpe=LoadIris.return_type())

    def requires(self):
        return self.iris() # from line 6

    def output(self):
        return luigi.LocalTarget(f"scaled_iris_{self.task_id}.csv")

    def run(self):
        # load dataset from previous component
        ds = pd.read_csv(self.input().path)
        feats = ds.drop(columns="target", axis=1)

        # transform features and save
        scaler = self.get_scaler()
        ds[feats.columns] = scaler.fit_transform(feats)
        ds.to_csv(self.output().path, index=False)

    def get_scaler(self):
        return NotImplementedError
```

(3) Define Concrete *Scaler* Components The two concrete scaling components are created by inheriting from *Scaler* and setting the class variable `abstract` to `False`. Then we implement the `get_scaler()` method. Task *RobustScaling* is similarly implemented, and we omit its presentation for brevity.

```
class MinMaxScaling(Scaler):
    abstract = False

    def get_scaler(self):
        from sklearn.preprocessing import MinMaxScaler
        return MinMaxScaler()
```

(4) Define Abstract & Concrete *Classifier* Components Now we implement an abstract component for classification that requires the *Scaler*. *DecisionTree* and *RandomForest* are the corresponding specializations. The structure here is the same as for the implementation of the scaling components.

```
class Classifier(luigi.Task, LuigiCombinator):
    abstract = True  # placeholder
    scaled_ds = ClsParameter(tpe=Scaler.return_type())

    def requires(self):
        return self.scaled_ds()

    def output(self):
        return luigi.LocalTarget(f"y_pred-{self.task_id}.csv")

    def run(self):
        # load data from previous component
        ds = pd.read_csv(self.input().path)
        X = ds.drop(columns="target", axis=1)
        y = ds["target"]

        # fit classifier
        clf = self.get_classifier()
        clf.fit(X, y)

        # predict and save predictions
        y_pred = pd.DataFrame(data=clf.predict(X), columns=["y_pred"])
        y_pred.to_csv(self.output().path, index=False)

    def get_classifier(self):
        return NotImplementedError

class DecisionTree(Classifier):
    abstract = False

    def get_classifier(self):
        from sklearn.tree import DecisionTreeClassifier
        return DecisionTreeClassifier()
```

(5) Synthesize and Run Pipelines First, a repository of all task components is built and the target is set to the type used for constructing the final component, namely *Classifier*. The framework will automatically (using meta-class-based reflection available in Python) construct this repository and its subtype structure, so we can use it as a basis for the finite combinatory logic of CLS. Then, we use CLS to build a finite representation (tree-grammar) based on the target (method `inhabit`), which is the way to synthesize all feasible pipelines. The number of pipelines can be infinite. In the infinite case, we explore the smallest 10 of them by number of task components. In the finite case we explore all pipelines. In our particular example, four pipelines are possible which we then hand over to the Luigi scheduler for execution. Figure 1(A) shows all of them, while (C) shows the actual dependency graph as it gets executed. Luigi runs the *LoadIris* task and the *Scaler* tasks only once with successors sharing their results.

```
from cls.fcl import FiniteCombinatoryLogic
from cls.subtypes import Subtypes
from cls_luigi.inhabitation_task import RepoMeta

# build component repository and set target type
repository = RepoMeta.repository
target = Classifier.return_type()

# build tree-grammar and synthesize pipelines
fcl = FiniteCombinatoryLogic(repository, Subtypes(RepoMeta.subtypes))
results = fcl.inhabit(target)

# restrict number of pipelines to 10 if infinite
num_pipes = results.size() # returns -1 if infinite
num_pipes = 10 if num_pipes == -1 else num_pipes
pipes = [t() for t in results.evaluated[0:num_pipes]]

# handover pipelines to luigi scheduler
luigi.build(pipes, local_scheduler=True)
```

The repository can be easily extended by adding more scaling and classification algorithms that conform with the Scikit-Learn's [18] API. In Section 4, we show an example where extensions change the pipeline structure.

4 AutoML Example

In this section, we show how to apply CLS-Luigi to systematically generate ML pipelines based on the algorithmic repository of the AutoML framework AutoSklearn. AutoSklearn uses a template that sets the structure of all resulting pipelines and uses Bayesian optimization to select algorithms and their hyperparameters from the template's search space. For binary classification problems with numerical features, AutoSklearn's pipelines have the following steps: imputation, scaling, feature preprocessing (including dimensionality reduction, feature generation and selection), and classification. While imputation is compulsory in AutoSklearn, even when no missing values are present, scaling and feature preprocessing can be omitted by replacing them with dummy steps.

As depicted in Figure 2, we constructed a pipeline template in CLS-Luigi that imitates the pipeline structure of AutoSklearn and implemented the same algorithm choices for each step. The pipeline template starts by loading the dataset. For simplicity, we limit our example to CSV files. Hence, we implement the concrete component *LoadCSVData*. In this step, the features and targets are also split into disjoint training and validation subsets. Next, missing values of features are imputed using the *SimpleImputer* a specialized *NumericalImputer*. Then, features can be scaled using one out of six specialized *Scaler* components, or they are passed using *NoScaler*. Afterward, one out of 12 specialized *FeaturePreprocessor* components transforms the features, or the features are passed on to the next task using the *NoFeaturePreprocessor*. Note that the *NumericalImputer* and *Scaler* components require only the features, while *FeaturePreprocessor* requires both features and targets as some algorithms require also the target values. Finally, the last component is a *Classifier* and it requires processed features and target values. This template results in 1456 unique pipelines (1 x 1 x 7 x 13 x 16). It is easy to extend this pipeline template with a second specialized *NumericalImputer* (e.g. IterativeImputer from Scikit-Learn [18]). Doing so increases the number of pipelines to 2912.

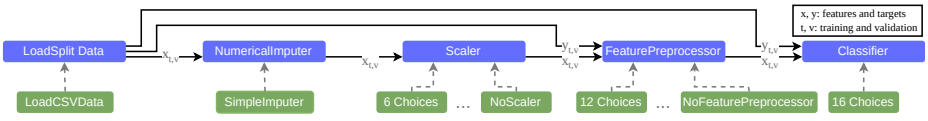


Fig. 2. Visualization of the pipeline template that imitates AutoSklearn's pipelines. Refer to Figure 1 for the meaning of box colors and arrow types.

4.1 Enhancing AutoSklearn Pipeline Modeling

The pipeline templates, both in CLS-Luigi and AutoSklearn, include redundant steps (*NoScaler*, *NoFeaturePreprocessing*) that only "emulate" structural variance to generate pipelines with a variable number of components. However, CLS-Luigi can express structural variance with specialization hierarchies and without introducing redundant dummy components in solutions.

Recall that *Classifier* can be proceeded by a *FeaturePreprocessor*, *Scaler*, or a *SimpleImputer*. A *FeaturePreprocessor*, if contained in a pipeline, can be proceeded by a *Scaler*, a *NumericalImputer*, or both. As shown in Figure 3, this dependency can be abstracted away by implementing a new abstract component, namely the *DataPreprocessor*, which manages the dependency of the *Classifier*. As the *FeaturePreprocessor*, the *Scaler*, and *NumericalImputer* specialize this *DataPreprocessor* all three are valid proceeding components of the *Classifier*. Analogously, the *NumericalPreprocessor* manages the dependency of the *FeaturePreprocessor*, and as such, the valid proceeding components are the specializations *Scaler* and *NumericalImputer*. Now, all pipelines will include either (1) a *FeaturePreprocessor*, (2) a *NumericalImputer*, or (3) a *Scaler*. Finally, to guarantee that all pipelines include a specialized *NumericalImputer*, we set the *Scaler* component to depend on the *NumericalImputer*. As in the previous section, this pipeline template generates a total of 1456 unique pipelines. However, some of these pipelines have fewer steps (3 or 4 instead of 5) omitting the dummy tasks. Therefore, this modeling variant allows for the incorporation of structural variance into pipeline templates.

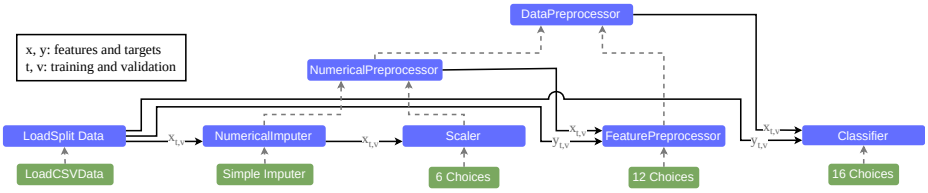


Fig. 3. Visualization of the enhanced pipeline template that resembles AutoSklearn's pipelines. Refer to Figure 1 for the meaning of box colors and arrow types.

4.2 Assessing Run Times & Prediction Quality

We evaluate the enhanced modeling approach of CLS-Luigi in terms of performance scores and computational efficiency, comparing it to AutoSklearn.

Dataset Selection and Computational Setup We use 15 datasets from OpenML [24] covering various domains with different numbers of instances and columns. For the brevity of presentation of models and results, all selected

datasets contain solely numeric features. This selection is inspired by the dataset selection in the original AutoSklearn paper [6] and is shown in Table 1. The left-hand side of this table shows the names of the selected datasets and their corresponding number of rows and features. The column "in AutoSklearn?" shows whether a dataset is part of its meta-learning datasets.

To allocate a run time limit for AutoSklearn, we executed CLS-Luigi and recorded the total elapsed time for all pipelines per dataset. Then, we allocated double the elapsed time to AutoSklearn. To select the best-performing pipeline, we apply the same procedure as AutoSklearn (holdout subset): We used the original training and testing indices from OpenML for a first split, and split the training dataset further into training and validation subsets in CLS-Luigi. We selected the pipeline with the highest accuracy score on the validation dataset.

Since we do not build ensembles in our CLS-Luigi template, we disabled AutoSklearn's ensemble building mechanism for a fair comparison. Moreover, in AutoSklearn, certain component combinations are forbidden. We accounted for that by filtering the synthesized pipelines such that they are permitted in AutoSklearn. To avoid long-running pipelines we restricted the run time of task components in CLS-Luigi to 100 seconds, while AutoSklearn uses 10% of the total allowed time for the individual entire pipeline execution by default. Note that all pipeline components in CLS-Luigi are executed using default hyperparameters from AutoSklearn while AutoSklearn performs hyperparameter optimization. This example was conducted using an Intel(R) Xeon(R) w5-2445 x86_64 CPU with two NVIDIA RTX 6000 Ada Generation GPUs and 128GB memory. Since CLS-Luigi and AutoSklearn require different Python versions, we used Python v3.11 with CLS-Luigi v0.1.1 and Python v3.8 with AutoSklearn v0.15.0 in separate virtual environments.

Accuracy Scores & Number of Evaluated Pipelines We report the accuracy score of the best-performing pipeline on the test dataset, and the total number of evaluated pipelines in Table 1. CLS-Luigi achieved higher accuracy scores on 11 out of 15 datasets, while AutoSklearn achieved a higher accuracy score on two datasets. For the remaining datasets, the achieved accuracy scores for both frameworks were equal. For all aforementioned datasets, CLS-Luigi evaluated 1183 pipelines, while AutoSklearn evaluated significantly fewer pipelines across the vast majority of the datasets. This includes both successful and failed pipelines for both frameworks.

Computational Efficiency In Table 2, we show the total run time and our estimate for time savings using the caching mechanism of Luigi per dataset. As stated earlier, identical sub-graphs between pipelines in CLS-Luigi are cached and executed only once. To estimate the saved time by caching, we first computed the geometric mean of run seconds for each concrete task across all successful pipelines per dataset, then multiplied it by the number of occurrences in all successfully evaluated pipelines. Finally, we summed this value for all components of the same type (e.g. scaling algorithms).

Table 1. Dataset information, number of executed pipelines, and test accuracy of the best pipeline. Higher accuracy per dataset is underlined.

Dataset				# Pipelines		Test Accuracy	
Name	# Rows	# Columns	In AutoSklearn?	CLS-Luigi	AutoSklearn	CLS-Luigi	AutoSklearn
higgs	98050	28	-	1183	120	0.731	**0.733**
numerai28.6	96320	21	-	1183	233	**0.515**	0.512
mozilla4	15545	5	X	1183	379	**0.963**	0.950
eeg-eye-state	14980	14	X	1183	192	**0.977**	0.963
bank-marketing	10578	7	-	1183	219	0.824	**0.831**
phoneme	5404	5	-	1183	142	**0.915**	0.885
sylvine	5124	20	-	1183	205	**0.947**	0.945
wilt	4839	5	X	1183	227	**0.983**	0.981
spambase	4601	57	X	1183	71	0.939	0.939
madelon	2600	500	X	1183	746	**0.915**	0.896
ozone-level-8hr	2534	72	X	1183	163	**0.949**	0.941
kc1	2109	21	-	1183	116	**0.891**	0.863
steel-plates-fault	1941	33	X	1183	221	**1.0**	0.995
pc4	1458	37	X	1183	191	0.911	0.911
qsar-biodeg	1055	41	X	1183	109	**0.849**	0.811

Table 2. Total run seconds per pipeline, the estimated time savings per component, and the number of failed and timed-out pipelines in CLS-Luigi.

Dataset	# Pipelines		Total Run	Estimated Saved Seconds for			% Saved
	Failed	Timed-out	Seconds	Imputer	Scaler	Feature Preproc.	Run Time
higgs	0	135	18074.67	42.33	132.52	1237.42	7.25
numerai28.6	0	127	16687.88	25.61	84.88	1019.82	6.34
mozilla4	4	1	789.12	11.65	12.16	925.24	54.6
eeg-eye-state	34	1	1043.03	9.82	24.03	726.42	42.16
bank-marketing	0	0	440.93	10.8	12.2	285.47	41.16
phoneme	0	0	207.02	7.8	17.86	80.21	33.84
sylvine	4	1	464.59	12.71	42.59	114.86	26.81
wilt	34	1	245.15	10.02	15.84	60.74	26.10
spambase	0	0	435.39	14.94	87.54	155.07	37.17
madelon	19	33	4916.87	28.17	313.55	1742.27	29.77
ozone-level-8hr	19	1	540.99	9.97	49.18	140.65	26.97
kc1	0	0	136.2	9.18	17.53	48.22	35.49
steel-plates-fault	9	2	343.02	9.51	20.7	56.58	20.19
pc4	4	1	238.04	9.27	18.78	51.49	25.04
qsar-biodeg	0	0	132.1	8.86	17.66	64.19	40.71
Geometric Mean			**696.37**	**12.79**	**33.60**	**204.70**	**26.51**

The total run time for all pipelines across all datasets ranges from 132 to 18074 seconds, with an average time of 696 seconds. Our estimation reports the time savings between 7 to 42 seconds, and 12 to 313 seconds for imputer and scaler components respectively. For feature preprocessor components, the time savings are significantly higher and are between 48 and 1742 seconds. This is to be expected, since feature preprocessing algorithms (such as principle component analysis and feature selection using classification models) may require more time in comparison to scaling and imputation algorithms. Apart from the dataset

higgs and **numerai28.6**, the caching mechanism has (approximately) saved between 20% to 54% in run seconds. As for the datasets **higgs** and **numerai28.6**, our investigations show that many pipeline tasks reached their timeout limit of 100 seconds. Thus, the total run time was significantly prolonged and the caching benefits were lessened. Apart from reaching the timeout limit, some CLS-Luigi pipelines fail due to (1) algorithms transforming inputs into infinity values, (2) algorithms being not compatible with negative inputs, or (3) feature selection algorithms resulting in no "no selected" features. These issues also affect some regions of the AutoSklearn's search space. Lastly, since the same pipelines are executed for all datasets, pipelines are synthesized only once and saved. The synthesis procedure lasted less than 2 seconds, and loading the pipelines for reuse lasted less than 0.5 seconds.

5 Decision Pipeline Example

In this section, we use the shortest path problem with uncertain cost to illustrate the capabilities of CLS-Luigi. Our modeling is inspired by the experiments from the original PyEPO paper [22] and can re-use a large part of its implementation of several algorithms. Also, we report on the realized computational benefits of CLS-Luigi's caching mechanism. We proceed to show how our model can use a single pipeline template for multiple solution approaches.

The objective of the shortest path problem, considered in [22], is to find the cost-minimal path in a grid from the northwest to the southeast corner. We implement a pipeline template that includes two different pipeline structures. Each structure corresponds to a different approach to solving the shortest path problem with uncertain cost: a two-stage approach (predict-then-optimize) and an end-to-end learning approach (smart predict-then-optimize). As depicted in Figure 4, the first component in the template handles generating features and cost vectors synthetically and splitting them into disjoint training and testing subsets. The generation of data is parameterized (as in PyEPO) using the grid size, the number of features, the number of instances, the polynomial degree, the half-noise width, and a seed value for reproducibility. Using the generated costs from the previous task, optimal solutions and objective values are computed by [8]. Next, using a specialized *SolutionApproach*, a predictive model is trained to predict the cost of the testing data. We specialize the *EndToEndLearning* and the *TwoStage* solution approaches. We further specialize an *EndToEndLearning* using the *SPOPlus* (with PyEPO's implementation) that requires the training dataset as well as its optimal solutions and objective values. As for the *TwoStage* approach, we specialize four *MultiOutputRegression* models. Note that the pipelines using multi-output regression models are structurally different, only requiring the synthetic dataset and not the optimal solutions and objective values. Following PyEPO, we implement *RandomForest* and *LinearRegression* models. However, given the simplicity of extending pipeline templates in CLS-Luigi, we also added a *LightGBM* Model with default hyperparameters and another *LightGBM* model with linear tree (method to use a linear model at leaves).

After training and predicting costs, *GenerateSolutionsForPredictedCosts* computes solutions and objective values for the predicted costs via Gurobi. Lastly, the pipeline template ends with the *Evaluation* component for computing the normalized regret metric, which is defined (informally) as the difference between the cost incurred by the decision of an algorithm and the cost of the optimal decision. In contrast to AutoML frameworks, such pipelines integrate algorithms from different fields, namely prediction and optimization.

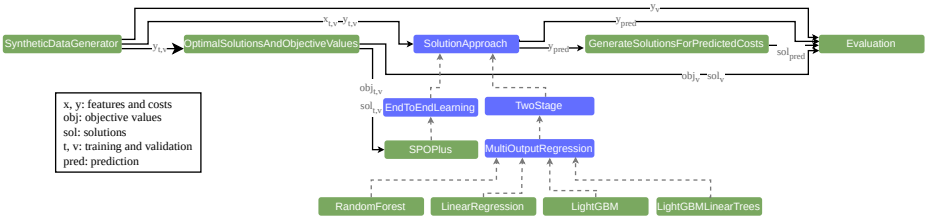


Fig. 4. Visualization of pipeline template for solving the shortest path problem. Refer to Figure 1 for the meaning of box colors and arrow types.

5.1 Assessing Run Times

To assess the run times of the resulting pipelines, we execute them using different parameter combinations and present our results below.

Computational Setup As stated, feature and cost vectors are generated at the beginning of the pipelines and are generated for a 5x5 grid. To replicate the experiment of PyEPO, we executed the five resulting pipelines using all combinations of the parameters training dataset size (100, 1000, 5000), polynomial degree (1, 2, 4, 6), and half-width noise (0.0, 0.5). To account for the validation dataset, an additional 1000 rows are always generated. Finally, every combination is executed 10 times using unique seed values, which results in a total of 240 parameter combinations and 1200 pipeline executions. To run this example, we used the same hardware and software versions from Section 4. In terms of the regret metric, we were able to reproduce the PyEPO results with an expected average rank of our extensions. Hence, we only discuss computational efficiency.

Computational Efficiency Recall that our pipeline template generates 5 pipelines, each with 5 components, which share the first two components, namely *SyntheticDataGenerator* and *OptimalSolutionsAndObjectiveValues*. Thus, for a given parameter combination, CLS-Luigi runs these two components only once and caches their output. This achieves a 32% (17 out of 25 components) decrease in the number of executed components per parameter combination. In Table 3

Table 3. Total run seconds and estimated saved seconds per parameter combination. Each row shows the minimum and maximum values for an aggregate of 10 runs.

Training Dataset size	Total Run Seconds	Estimated Saved Seconds for Generating		% Saved Run Time
		Synthetic Data	Optimal Solution	
100	98-100	0.35-0.41	23-24	19
1000	272-296	0.6-0.71	41-42	12-13
5000	1027-1183	1.69-2.03	124-125	9-11
Geometric Mean	**314.48**	**0.77**	**49.86**	**13.79**

we show the run time of pipelines per parameter combination and the total saved time for generating the data and computing the optimal solutions and objective values. Note that the values are very similar for all runs with the same training sizes, as such we only show the minimum and maximum values. Here, the total run time and the estimated saved seconds per component are summed for all 10 seeds. The estimated saved seconds per component and seed are computed by multiplying the run time of the respective component by 4 (since this component runs once for all 5 pipelines). It is observable that the total run time for all pipelines is very similar for the same training dataset size. Moreover, *SyntheticDataGenerator* component takes considerably less time in comparison to *OptimalSolutionsAndObjectiveValues*. In summary, the caching mechanism (approximately) reduces the total run time by 19% for datasets with 100 training rows. Similarly, for datasets with 1000 and 5000 training rows, the run time is reduced by 12-13% and 9-11% respectively. Finally, using the geometric mean, we estimate an average reduction in run time by 13.79%.

6 Conclusion and Outlook

We introduced CLS-Luigi, a framework for modeling and synthesis of analytics pipelines based on a repository of algorithms. CLS-Luigi stands out for Python developers because of its simplicity, allowing pipelines to be modeled within code instead of separately in another language. It leverages the pipeline execution strengths of Luigi, which is easy to set up and offers good abstractions for constructing analytics pipelines out of the box. Our framework combines Luigi with CLS to synthesize pipelines systematically. Unlike conventional AutoML frameworks, which are limited in scope and extendability, CLS-Luigi goes beyond the state-of-the-art AutoML frameworks by automating the synthesis of pipelines that include both ML and optimization algorithms. It achieves this while integrating popular analytics tools like Scikit-Learn and Gurobi. Our framework also showcases the expressiveness to accommodate multiple optimization paradigms within a single pipeline template, allowing for a consistent evaluation. Extending a pipeline template with new algorithms comes at a negligible cost and incorporating structural variance in pipelines is also covered and can be explicitly modeled. Our framework also offers additional modeling capabilities that have not been discussed so far. Yet, there exists a trade-off between the expressiveness and simplicity of models. CLS-Luigi caches outputs and avoids redundant

execution of mutual pipeline tasks. The benefits of this are shown in an example implementation that imitates AutoSklearn's pipelines. CLS-Luigi outperformed AutoSklearn in both accuracy scores and number of executed pipelines across a majority of the selected datasets.

There exists some limitation to CLS-Luigi that we are addressing in future work: Our brute-force enumeration approach runs into scalability issues for large datasets, despite caching outputs. Additionally, hyperparameters are still set manually by users. Also, we depend on Luigi's simple pipeline scheduling implementation, and executing multiple pipelines with multiple workers is yet to be tested as well as hardware parallelism.

In our next steps, we want to integrate a Monte Carlo Tree Search to select pipeline components based on CLS-Luigi's tree grammar and incorporate a hyperparameter optimization. To facilitate implementation we want to develop and integrate a visualization for pipeline templates, dependency graphs, and individual pipelines. Finally, we intend to apply our framework to more realistic ML and optimization use cases and enhance its modeling capabilities, for example, to describe data semantically.

References

1. Ariflow: Website https://airflow.apache.org/ (2011–2024), accessed: 2024-02-14
2. Bergman, E., Feurer, M., Bahram, A., Balef, A.R., Purucker, L., Segel, S., Lindauer, M., Hutter, F., Eggensperger, K.: AMLTK: A Modular Automl Toolkit in Python. Under Review for the Journal of Open Source Software (2023)
3. Bessai, J.: A type-theoretic framework for software component synthesis. Ph.D. thesis, Technical University of Dortmund, Germany (2019). https://doi.org/10.17877/DE290R-20320
4. Davenport, T.H., et al.: Competing on analytics. Harvard business review **84**(1), 98 (2006)
5. Elmachtoub, A.N., Grigas, P.: Smart "predict, then optimize". Management Science **68**(1), 9–26 (2022). https://doi.org/10.1287/mnsc.2020.3922
6. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: Advances in neural information processing systems. pp. 2962–2970 (2015)
7. Graefenstein, J., Winkels, J., Lenz, L., Weist, K., Krebil, K., Gralla, M.: A hybrid approach of modular planning - synchronizing factory and building planning by using component based synthesis. In: HICSS (2020). https://doi.org/10.24251/HICSS.2020.806
8. Gurobi Optimization LLC: Gurobi optimizer reference manual (2023), https://www.gurobi.com, accessed: 2024-03-06
9. Harenslak, B.P., de Ruiter, J.: Data Pipelines with Apache Airflow. Simon and Schuster (2021)
10. He, X., Zhao, K., Chu, X.: Automl: A survey of the state-of-the-art. Knowledge-Based Systems **212**, 106622 (2021). https://doi.org/10.1016/j.knosys.2020.106622
11. Lawrence, N.D.: Data science and digital systems: The 3ds of machine learning systems design. arXiv preprint arXiv:1903.11241 (2019). https://doi.org/10.48550/arXiv.1903.11241

12. Lopez, R., Lourenco, R., Rampin, R., Castelo, S., Santos, A., Ono, J., Silva, C., Freire, J.: Alphad3m: An open-source automl library for multiple ml tasks. In: AutoML Conference 2023 (ABCD Track) (2023)
13. Luigi: Website https://luigi.readthedocs.io (2011–2024), accessed: 2024-02-02
14. Mäckel, D., Winkels, J., Schumacher, C.: Synthesis of scheduling heuristics by composition and recombination. In: OLA (2021). https://doi.org/10.1007/978-3-030-85672-4_21
15. Mages, A., Mieth, C., Hetzler, J., Kallat, F., Rehof, J., Riest, C., Schäfer, T.: Automatic component-based synthesis of user-configured manufacturing simulation models. In: WSC (2022). https://doi.org/10.1109/WSC57314.2022.10015425
16. Manna, Z., Waldinger, R.J.: Synthesis: Dreams - programs. IEEE Trans. Software Eng. **5**(4), 294–328 (1979). https://doi.org/10.1109/TSE.1979.234198
17. Metaflow: Website https://metaflow.org/ (2011-2024), accessed: 2024-02-14
18. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
19. Powell, W.B.: A unified framework for stochastic optimization. European Journal of Operational Research **275**(3), 795–821 (2019). https://doi.org/10.1016/j.ejor.2018.07.014
20. Rehof, J., Urzyczyn, P.: Finite combinatory logic with intersection types. In: TLCA (2011). https://doi.org/10.1007/978-3-642-21691-6_15
21. Schäfer, T., Bessai, J., Chaumet, C., Rehof, J., Riest, C.: Design space exploration for sampling-based motion planning programs with combinatory logic synthesis. In: WAFR (2022). https://doi.org/10.1007/978-3-031-21090-7_3
22. Tang, B., Khalil, E.B.: Pyepo: A pytorch-based end-to-end predict-then-optimize library for linear and integer programming. arXiv preprint arXiv:2206.14234 (2022). https://doi.org/10.48550/arXiv.2206.14234
23. Tornede, T., Tornede, A., Wever, M., Mohr, F., Hüllermeier, E.: Automl for predictive maintenance: One tool to rul them all. In: Gama, J., Pashami, S., Bifet, A., Sayed-Mouchawe, M., Fröning, H., Pernkopf, F., Schiele, G., Blott, M. (eds.) IoT Streams for Data-Driven Predictive Maintenance and IoT, Edge, and Mobile for Embedded Machine Learning. pp. 106–118. Springer Intl. Pub. (2020)
24. Vanschoren, J., Van Rijn, J.N., Bischl, B., Torgo, L.: Openml: networked science in machine learning. ACM SIGKDD Explorations Newsletter **15**(2), 49–60 (2014). https://doi.org/10.1145/2641190.2641198
25. Von Rueden, L., Mayer, S., Beckh, K., Georgiev, B., Giesselbach, S., Heese, R., Kirsch, B., Pfrommer, J., Pick, A., Ramamurthy, R., et al.: Informed machine learning–a taxonomy and survey of integrating prior knowledge into learning systems. IEEE TKDE **35**(1), 614–633 (2021). https://doi.org/10.1109/TKDE.2021.3079836
26. Wratten, L., Wilm, A., Göke, J.: Reproducible, scalable, and shareable analysis pipelines with bioinformatics workflow managers. Nature methods **18**(10), 1161–1168 (2021). https://doi.org/10.1038/s41592-021-01254-9
27. Zöller, M.A., Huber, M.F.: Benchmark and survey of automated machine learning frameworks. Journal of artificial intelligence research **70**, 409–472 (2021). https://doi.org/10.1613/jair.1.11854
28. Zöller, M.A., Nguyen, T.D., Huber, M.F.: Incremental search space construction for machine learning pipeline synthesis. In: Advances in Intelligent Data Analysis XIX. pp. 103–115. Springer (2021). https://doi.org/10.1007/978-3-030-74251-5_9

A R2 based Multi-objective Reinforcement Learning Algorthim

Sofia Magdalena Borrel Miller and Carlos Ignacio Hernández Castellanos

Universidad Nacional Autónoma de México, México, sofiamborrelm@gmail.com, carlos.hernandez@iimas.unam.mx

Abstract. In many real-world problems, one faces the problem of having to make decisions considering several conflicting objectives. In such problems, the solution is a set of policies rather than a single one. This leads to multi-objective reinforcement learning problems (MORL) which have not received much attention until recent years by the reinforcement learning community.
In this work, we propose coupling the R2 indicator with Pareto Q-learning. The R2 indicator has been successfully used for multi-objective optimization problems making it a good candidate for MORL. We tested our approach on several problems from MO Gymnasium and compared it with HB-MORL, which uses the hypervolume indicator. Our preliminary results show that the novel algorithm obtains competitive results and that could be an interesting alternative when dealing with MORL problems.

Keywords: Multi-objective reinforcement learning, Performance indicators, R2 indicator

1 Introduction

Reinforcement Learning (RL) [4] is one of the branches of machine learning. RL focuses on training agents to act in an environment by learning a policy to maximize cumulative reward. Typically, this is modeled by a Markov decision process (MDP). MDPs are characterized by states (S), actions (A), transition functions ($T : S \times A \times S \to [0, 1]$), a discount factor ($\gamma \in \{0, 1\}$), and reward functions ($R : S \times A \times S \to \mathbb{R}$). Multi-objective reinforcement learning (MORL) [5] extends RL to problems where we need to consider reward functions ($R : S \times A \times S \to R^k$) where k is the number of rewards. Note that in this kind of problems, the solution is a set of policies rather than a single solution.

To tackle this kind of problems, different approaches have been proposed in the literature. One of the most common is to solve a scalarized version of the problem (for instance using a weighted sum approach) [8]. Other approaches include using the hypervolume indicator to find a set of policies [7]. The scalarized approaches have the advantage that it is possible to use all the algorithms of single-objective RL but require that the decision-maker defines his/her preferences. While approaches such as the hypervolume avoids the need to define

preferences before the search. However, the hypervolume indicator has a high computational cost as the number of objectives increases. Due to space limitation, we refer the interested reader to [3] for further details on MORL.

In light of this, we propose to use the R2 indicator [2]. This indicator can be seen as a compromise between previous approaches. On one hand, it can approximate the solution set (or be adjusted with the decision-maker preferences) and it has a polynomial complexity. This indicator has been successfully used in several algorithms for multi-objective optimization (e.g. [6]).

The rest of the document is structured as follows: Section 2 describes the proposed algorithm, Section 3 shows the comparison to HB-MORL on several academic test functions from literature, and finally 4 concludes the work and describes several future research directions.

2 R2 Based Multi-objective Reinforcement Learning Algorithm

In this section, we introduce our proposed algorithm. The algorithm adapts [7] to use the R2 indicator instead of hypervolume. Algorithm 1 shows the R2-based MORL algorithm which is based on Q-learning with an ϵ-greedy policy. The main difference occurs in line 8, where the action is selected based on the contribution to the R2 indicator. Further, Algorithm 2 shows the action selection where the contribution to the R2 indicator is used as proposed in [6].

Algorithm 1 R2-based MORL

1: Initialize $Q(s, a, o)$ arbitrarily
2: **for** each episode t **do**
3: Initialize state $s, l = \{\}$
4: **repeat**
5: Choose action a from s using a policy derived from the Q
6: Take action a and observe state s', reward vector $r \in \mathbb{R}^k$
7: Add o to l
8: $max_{a'} \leftarrow gR2BAS(s', l')$
9: **for** objective o **do**
10: $Q(s, a, o) \leftarrow Q(s, a, o) + \alpha[r(s, a, o) + \gamma Q(s', \max_(a'), o) - Q(s, a, o)]$
11: **end for**
12: $s \leftarrow s'$
13: **until** s is terminal
14: **end for**

3 Empirical Preliminary Results

In this section, we show the empirical results of the novel algorithm and compare it with HB-MORL. For this purpose, we use MO Gymnasium [1]. This frame-

Algorithm 2 Greedy R2-based Action Selection ($gR2AS(s, l)$)

1: $contributions \leftarrow \emptyset$
2: **for all** $a_i \in A$ of state s **do**
3: $o \leftarrow \{Q(s, a_i, o_1), \ldots, Q(s, a_i, o_k)\}$
4: $r2c \leftarrow r2Contribution(l + o)$
5: Append $r2c$ to contributions
6: **end for**
7: **return** argmax_a *contributions*

work provides a standardized platform for comparing different multi-objective reinforcement learning algorithms through various test environments: **Tree Fruits (TF):** Balances maximizing fruit collection with minimizing time/distance.**Resource Gathering (RG):** Aims to optimize the amount and diversity of resources within time limits. **Deep Sea Treasure (DST):** Focuses on efficiently seeking valuable underwater treasures. **Mountain Car (MC):** Targets reaching the hilltop using momentum with an emphasis on efficiency, like reducing energy or time.**Four Room (FR):** Tests agent navigation and goal achievement in a complex, multi-room layout. For both algorithms, we performed 20 independent experiments and were performed on a Google Compute Engine server utilizing Python 3. The data in Table 1 shows that R2-MORL outperforms HB-MORL in all five environments. Figure 3 confirms these preliminary results, indicating higher hypervolume values for the proposed approach.

Table 1. Environment, Parameters (Objective, and Episodes, Reference Point (R), Ideal Point (I)), and mean HV results of both algorithms.

Env.	k	Episodes	Reference point(R)	I	R2-MORL	HB-MORL
T.F.	6	200	(-5,...,-5)	(10,...,10)	1.67e+06	1.18e+06
R.G.	3	200	(-10,-10,-10)	(0,1,1)	1,210	67
D.S.T	2	200	(0,-25)	(124,21)	156.85	84.25
M.C.	3	100	(-110,-100,-50)	(-1,0,0)	3.65e+06	2.46e+06
F. R.	3	100	(-110,-110,-110)	(100,100,100)	1.46e+06	1.37e+06

4 Conclusions, Future Work

In this work, we have proposed to use the R2 indicator in the context of MORL. We compared the approach to HB-MORL on several academic environments from two to six objectives with competitive results. The preliminary results show a promising path for further studying the R2 indicator in the RL context. In future work, we plan to explore different indicators and algorithms, focusing on adaptability and scalability concerning the number of objectives. Finally, we would like to apply the approach to real-world applications.

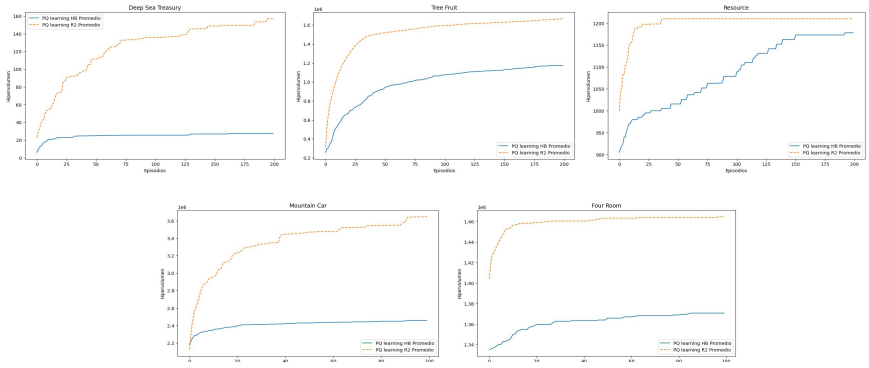


Fig. 1. Median hypervolume of both algorithms on the selected environments.

5 Acknowledgements

The first and second authors acknowledge the funding from CONAHCyT's scholarship and project PAPIIT no. IA102923 respectively.

References

1. Felten, F., Alegre, L.N., Nowé, A., Bazzan, A.L.C., Talbi, E.G., Danoy, G., Silva, B.C.da.: A toolkit for reliable benchmarking and research in multi-objective reinforcement learning. In: NeurIPS 2023 (2023)
2. Hansen, M.P., Jaszkiewicz, A.: Evaluating the quality of approximations to the non-dominated set. IMM, Dept. of Mathematical Modelling, TU of Denmark (1994)
3. Hayes, C.F., Rădulescu, R., Bargiacchi, E., Källström, J., Macfarlane, M., Reymond, M., Verstraeten, T., Zintgraf, L.M., Dazeley, R., Heintz, F., et al.: A practical guide to multi-objective reinforcement learning and planning. Autonomous Agents and Multi-Agent Systems **36**(1), 26 (2022)
4. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: A survey. Journal of artificial intelligence research **4**, 237–285 (1996)
5. Liu, C., Xu, X., Hu, D.: Multiobjective reinforcement learning: A comprehensive overview. IEEE Transactions on Systems, Man, and Cybernetics: Systems **45**(3), 385–398 (2014)
6. T., H., Wagner, T., Brockhoff, D.: R2-emoa: Focused multiobjective search using r2-indicator-based selection. In: Learning and Intelligent Optimization: 7th International Conference, LION 7, Catania, Italy, January 7-11, 2013, Revised Selected Papers 7. pp. 70–74. Springer (2013)
7. V. Moffaert, K., D., M.M., Nowé, A.: Hypervolume-based multi-objective reinforcement learning. In: Evolutionary Multi-Criterion Optimization: 7th International Conference, EMO 2013, Sheffield, UK, March 19-22, 2013. Proceedings 7. pp. 352–366. Springer (2013)
8. Van Moffaert, K., Nowé, A.: Multi-objective reinforcement learning using sets of pareto dominating policies. The Journal of Machine Learning Research **15**(1), 3483–3512 (2014)

Robust Airline Fleet and Crew Scheduling: A Matheuristic Approach

Abtin Nourmohammadzadeh[0000−0003−0383−0379] and Stefan Voß[0000−0003−1296−4221]

Institute of Information Systems, University of Hamburg, Hamburg, Germany
{abtin.nourmohammadzadeh,stefan.voss}@uni-hamburg.de

Abstract. Fleet and crew scheduling belong to the significant problems in the airline industry. Especially after the Covid-19 pandemic, robustness issues become even more important in this realm. This asks for advanced solution methods and appropriate modeling approaches. To support this avenue, a mixed-integer mathematical model is presented together with a matheuristic approach consisting of a hybridization of particle swarm optimization (PSO), simulated annealing (SA) and mathematical programming of sub-problems. It is shown that the proposed matheuristic is a promising approach deserving further consideration.

Keywords: Robust Airline Fleet and Crew Scheduling · Matheuristics · Robustness.

1 Motivation

Optimization of scarce and expensive resources is a vital task in the aviation industry. In the extremely competitive environment and under huge difficulties such as the Covid-19 pandemic, airline companies have to find practical cost-reduction methods in order to survive as well as avoid large increases in their ticket prices. A considerable proportion of airline expenses is related to the management of its fleet and crew. Hence, optimal or close-to-optimal scheduling of these resources can considerably help airlines to raise their benefits.

Although airline fleet and crew scheduling are considered as two separate problems in many works, they are actually very inter-related. This is due to the fact that each aircraft type or trip (flight) needs its own specialized crew members. In other words, each pilot or cockpit member cannot direct all the aircraft types of an airline or each flight attendant cannot be used for all trips because of her/his travel requirements or some crew members are better to be used for special trips because of their nationalities or languages they speak. The fleet assignment is done based on the passenger demands of the flights and the capacities as well as facilities of the available aircraft. Simultaneously, the crew members have to be assigned to flights regarding the aircraft and destinations.

The maximum working hours of the aircraft and crew, and their minimum rest time afterwards must be respected in any schedule. Note that the implications of crew rostering on airline operations are addressed, e.g., in [12].

Uncertainty in the inputs is a challenging factor in such a problem, which we should deal with in real-world settings. Robust optimization can be regarded as an appropriate technique to take non-deterministic inputs into account. This method is used in some works like [16] for the airline fleet assignment, [19] for the airline crew scheduling and in [5] for an integrated airline fleet and crew scheduling problem (AFCSP). The idea in [5] relates to incorporating robustness into the problem by making some important assumptions regarding the problem settings. In that paper, "to avoid the curse of dimensionality, crew connections rather than explicit crew pairings or duties are modeled to represent the crew-scheduling problem within the integrated model." To achieve robustness, they utilize the idea of purity as it was also considered by other authors. For instance, fleet purity ensures that the number of fleet types serving a given "station" does not exceed a specified limit. In addition, station purity may be included into a model by limiting the number of fleet types and crew bases allowed to serve each airport. As another related work, [15] presents algorithms for airline crew scheduling under uncertainty (though not in an integrated fashion).

A good literature review on airline crew scheduling is provided by [4]. Mathematical modeling can be regarded as a practical method if the problem or its sub-problems are tractable by standard solvers. [13] is an attempt in mathematical modeling of a specific version of the AFCSP.

In this work, we develop a novel comprehensive mathematical model for the AFCSP based on real conditions and regulations. Due to the computational complexity of solving the entire model, a practical hybrid (matheuristic) approach consisting of the particle swarm optimization (PSO) [9], simulated annealing (SA) [18] and exact solutions of the sub-problems is devised. Such methods are called matheuristics due to their integrated use of both mathematical programming and metaheuristics.[1] Uncertainties are considered for the availability periods of the aircraft and crew members, and handled by a robust optimization approach. The performance of our method is tested on a set of real-based generated instances by being compared with a state-of-the-art method from the literature.

In summary, we present a modeling approach which extends the previous ones by considering more real elements and propose a suitable solution method-

[1] For more details about metaheuristics and matheuristics, see [8] and [10], respectively.

ology which is capable of locating solutions of better qualities compared to other alternative solution methods in tolerable computational times.

The organization of the rest of this paper is as follows: Section 2 describes the focused problem, its importance and necessities. Consequently, Section 3 presents the developed mathematical model. Our solution approach is explained in Section 4. Section 5 is dedicated to the presentation of the results and related comparisons. Finally, Section 6 draws the overall conclusions of this work and shows some avenues for future research.

2 Problem statement

In the AFCSP, there are a set of aircraft and a set of crew members which have to be assigned to a set of flights. The aircraft are all initially at the airline hub, whereas all the crew members can have their own hub, which is their residence place. The goal is to minimize the total costs of using as well as transferring the aircraft and crew. The required cockpit and other board members have to be allocated to flights regarding the chosen aircraft. If an aircraft or a crew member is not or has not arrived by a previously scheduled flight at the origin airport of its/his/her next flight, they should be transferred there and this incurs extra costs. A very important task is to respect the maximum consecutive usage duration of aircraft before returning to the airline hub for the regular service and the maximum working hours of the crew before taking a short and long rest. While the short rest can be at any airport, the crew must return to their hub for taking the long rest.

A big challenge in this problem is that some aircraft and crew members may fail to work within some time slots, which were deemed as their working periods before. Therefore, there are several possible scenarios in this regard and it is very important to provide schedules which are good enough considering all such scenarios.

3 Mathematical Modeling

In the AFCSP, we have a set of aircraft A, a set of crew members C, a set of airports L, and a set of flights F, which must be done by available aircraft and crew. C is divided into two sets of cockpit members C_1 such as pilots and other board members C_2 like flight attendants. The necessary numbers of cockpit and other board members on aircraft a are $NCCN_a$ and $NOCN_a$. The set of crew members who can be assigned to flight f regarding their travel requirements or language skills (in case of flight attendants, for example) are shown by C_f, while

the set of cockpit members who have the expertise to fly aircraft a are denoted by $C_{1,a}$. Each crew member has a hub airport related to his/her residence location h_c. Flights have each an origin o_f, a departure time dep_f, a destination d_f, and an arrival time at destination arr_f. The cost of operating flight f by aircraft a is $aoc_{a,f}$, which is calculated based on the flying cost of a per unit of time, the flight duration of f, empty seats or lost demand according the estimated number of passengers. Nevertheless, the cost of using crew member c on flight f, $cac_{c,f}$, is based on his/her salary per unit of time, the flight duration, etc. The costs of transferring aircraft a and crew member c from airport l_1 to l_2 without being used for any flights are shown by EMA_{a,l_1,l_2} and EMC_{a,l_1,l_2}, respectively, and the required transfer times are $rtta_{a,l_1,l_2}$ and $rttc_{a,l_1,l_2}$, respectively. Each crew member has a short resting duration of at least MCS_c and a long resting duration of at least MCL_c time units at his/her hub. The short and long resting durations have to be scheduled before having worked for $MWCTS_c$ and $MWCTL_c$ consecutive time units, respectively. However, the aircraft must have a minimum service period of MA_a at the airline hub $AH \in L$ after having been used for $MWAT_a$ consecutive hours.

The main decision variables of our model are as follows: $x_{a,f}$ and $y_{c,f}$ which are binary variables equal to one if aircraft a and crew member c are assigned to flight f. $IA_{f_1,f_2,a}$ and $IC_{f_1,f_2,c}$ are other binary variables, which are true if flight f_2 is directly operated after f_1 by aircraft a and crew member c. We have binary location values $LA_{a,l,t}$, $LC_{c,l,t}$ which are true in case that a and c are at airport l at time t, respectively. $MA_{a,l_1,l_2,t}$ and $MC_{c,l_1,l_2,t}$ are binary variables equal to 1 if the transfers of a and c from l_1 to l_2 happen at time t. $WAT_{a,t}$ and $WCT_{c,t}$ are binary variables, which are 1 if the aircraft/crew is working at time t.

The notations and formulations of the developed holistic mathematical model for the AFCSP are presented as follows:

Notations

Inputs

T: the planning horizon;
A: the set of aircraft;
C: the set of crew members;
C_1: the set of cockpit crew members;
C_2: the set of other board crew members;
C_f: the set of crew members who can be assigned to flight f;
$CC_{1,a}$: the set of cockpit members who can be assigned to aircraft a;
L: the set of airports;
h_c: the hub of crew c;
H: the airline hub;
$reat_{a,f}$: the time required to make aircraft a ready for flight f;

$rect_{c,f}$: the time that crew member c requires to get ready for flight f;
F: the set of flights;
o_f: the origin of flight f;
dep_f: the departure time of flight f;
d_f: the destination of flight f;
arr_f: the arrival time of flight f;
$aoc_{a,f}$: the cost of operating flight f by aircraft a;
$coc_{c,f}$: the cost of using crew member c in flight f;
$MWAT_a$: the maximum consecutive working hours of aircraft a;
MAS_a: the minimum service time of flight a;
$MWCTS_c$: the maximum consecutive working hours of crew member c;
MCS_c: the minimum resting time of crew member c;
$MWCTL_c$: the maximum consecutive working hours of crew member c before returning to his/her hub and taking a long rest;
MCL_c: the minimum duration of the long resting time of crew member c at his/her hub.
EMA_{a,l_1,l_2}: the cost of empty movement of aircraft a from airport l_1 to l_2 without any scheduled flight;
EMC_{c,l_1,l_2}: the cost of transferring crew member c from airport l_1 to l_2 without working on any scheduled flight;
$rtta_{a,l_1,l_2}$: the required time to move aircraft a from airport l_1 to l_2;
$rttc_{a,l_1,l_2}$: the required time to transfer crew member a from airport l_1 to l_2.

Binary variables
$x_{a,f}$: =1 if aircraft a is assigned to flight f;
$y_{c,f}$: =1 if crew member c is assigned to flight f;
$IA_{f_1,f_2,a}$: =1 if flight f_2 is directly operated after f_1 by aircraft a;
$IC_{f_1,f_2,c}$: =1 if crew member c is directly assigned to flight f_2 after f_1;
$LA_{a,l,t}$: =1 if aircraft a is at airport l at time t;
$LC_{c,l,t}$:=1 if crew member c is at airport l at time t;
$MA_{a,l_1,l_2,t}$:=1 if aircraft a is moved without operating any flight from airport l_1 to l_2 at time t;
$WAT_{a,t}$:=1 if aircraft a is used for a flight at time t;
$WCT_{c,t}$:=1 if crew member c is working at time t;
$AR_{a,t}$=1 if a resting duration of aircraft a for the regular service begins at time t;
$SCR_{c,t}$:=1 if crew c takes a short rest beginning at time t;
$LCR_{c,t}$:=1 if crew c takes a long rest beginning at time t.

$$Min \quad Z = \sum_a \sum_f aoc_{a,f} x_{a,f} + \sum_c \sum_f coc_{c,f} y_{c,f} + \sum_a \sum_{l_1} \sum_{l_2} \sum_t MA_{a,l_1,l_2,t} EMA_{a,l_1,l_2} + \sum_c \sum_{l_1} \sum_{l_2} \sum_t MC_{c,l_1,l_2,t} EMC_{c,l_1,l_2} \quad (1)$$

$$\sum_a x_{a,f} = 1 \qquad \forall f \in F \quad (2)$$

$$x_{a,f} \sum_{c \in CC_{1,a}} y_{c,f} = NCCN_a \qquad \forall f \in F, \forall a \in A \quad (3)$$

$$x_{a,f} \sum_{c \in C_2} y_{c,f} = NOCN_a \qquad f \in F, \forall a \in A \tag{4}$$

$$2IA_{f_1,f_2,a} \leq x_{a,f_1} + x_{a,f_2} \qquad \forall f_1, f_2 \in F, \forall a \in A \tag{5}$$

$$IA_{f_1,f_2,a} \leq 1 - \sum_{\substack{f \in F, \\ arr_{f_1} \leq dep_f \leq dep_{f_2}}} x_{a,f} + |F|(1 - IA_{f_1,f_2,a}) \qquad f, f_1, f_2 \in F, \forall a \in A \tag{6}$$

$$IA_{f_1,f_2,a} \geq x_{a,f_1} x_{a,f_2} (1 - \sum_{\substack{f \in F, \\ arr_{f_1} \leq dep_f \leq dep_{f_2}}} x_{a,f}) \qquad f_1, f_2 \in F, \forall a \in A \tag{7}$$

$$2IC_{f_1,f_2,a} \leq y_{c,f_1} + y_{c,f_2} \qquad \forall f_1, f_2 \in F, \forall c \in C \tag{8}$$

$$IC_{f_1,f_2,a} \leq 1 - \sum_{\substack{f \in F, \\ arr_{f_1} \leq dep_f \leq dep_{f_2}}} y_{c,f} + |F|(1 - IA_{f_1,f_2,a}) \qquad f, f_1, f_2 \in F, \forall c \in C \tag{9}$$

$$IC_{f_1,f_2,a} \geq y_{a,f_1} y_{a,f_2} (1 - \sum_{\substack{f \in F, \\ arr_{f_1} \leq dep_f \leq dep_{f_2}}} y_{a,f}) \qquad f_1, f_2 \in F, \forall c \in C \tag{10}$$

$$x_{a,f} \leq LA_{a,o_f,dep_f - reat_{a,f}} \qquad \forall a \in A, f \in F \tag{11}$$

$$y_{c,f} \leq LC_{c,o_f,dep_f - rect_{a,f}} \qquad \forall a \in A, f \in F \tag{12}$$

$$\sum_{t=arr_{f_1}}^{dep_{f_2}} LA_{a,d_{f_1},t} \geq (dep_{f_2} - arr_{f_1}) IA_{f_1,f_2,a} \qquad \forall f_1, f_2 \in F, D_{f_1} = O_{f_2} \tag{13}$$

$$\sum_{t=arr_{f_1}}^{dep_{f_2}} LC_{c,d_{f_1},t} \geq (dep_{f_2} - arr_{f_1}) IC_{f_1,f_2,c} \qquad \forall f_1, f_2 \in F, D_{f_1} = O_{f_2} \tag{14}$$

$$IA_{f_1,f_2,a} \leq \sum_{t=arr_{f_1}}^{dep_{f_2} - reat_{a,f_2} - rtta_{a,d_{f_1},o_{f_2}}} MA_{a,d_{f_1},o_{f_2},t} \qquad \forall f_1, f_2 \in F, d_{f_1} \neq o_{f_2} \tag{15}$$

$$IC_{f_1,f_2,a} \leq \sum_{t=arr_{f_1}}^{dep_{f_2} - rect_{a,f_2} - rttc_{a,d_{f_1},o_{f_2}}} MC_{a,d_{f_1},o_{f_2},t} \qquad \forall f_1, f_2 \in F, d_{f_1} \neq o_{f_2} \tag{16}$$

$$\sum_{t=tt+rtta_{a,d_{f_1},o_{f_2}}}^{dep_{f_2}} LA_{a,d_{f_1},t} \geq (dep_{f_2} - tt - rtta_{a,d_{f_1},o_{f_2}}) \times IA_{f_1,f_2,a} MA_{a,d_{f_1},o_{f_2},tt} \quad f_1, f_2 \in F, d_{f_1} \neq o_{f_2}, arr_{f_1} \leq tt \leq dep_{f_2} - reat_{a,f_2} \tag{17}$$

$$\sum_{t=tt+rttc_{c,d_{f_1},o_{f_2}}}^{dep_{f_2}} LC_{c,d_{f_1},t} \geq (dep_{f_2} - tt - rttc_{c,d_{f_1},o_{f_2}}) \times IA_{f_1,f_2,c} MC_{c,d_{f_1},o_{f_2},tt} \quad f_1, f_2 \in F, d_{f_1} \neq o_{f_2}, arr_{f_1} \leq tt \leq dep_{f_2} - rect_{a,f_2} \tag{18}$$

$$(arr_f - d_f - reat_{a,f}) x_{a,f} \leq \sum_{t=def_f - reat_{a,f}} WAT_{a,t} \quad \forall a \in A, \forall f \in F \tag{19}$$

$$(arr_f - d_f - rect_{c,f}) y_{c,f} \leq \sum_{t=def_f - rect_{a,f}} WCT_{c,t} \quad \forall c \in C, \forall f \in F \tag{20}$$

$$WAT_{a,t} \geq \sum_{\substack{f, \\ dep_f - reat_{a,f} \leq t \leq arr_f}} x_{a,f} \qquad a \in A, t \in T \tag{21}$$

$$WCT_{c,t} \geq \sum_{\substack{f, \\ dep_f - rect_{c,f} \leq t \leq arr_f}} y_{c,f} \qquad c \in C, t \in T \tag{22}$$

$$SR_{a,t} \geq 1 - \sum_{tt=t}^{t+rt} WAT_{a,tt} \qquad t \in T, MAS_a \leq rt \tag{23}$$

$$SCR_{c,t} \geq 1 - \sum_{tt=t}^{t+rt} WCT_{a,tt} \qquad t \in T, MCS_c \leq rt \leq MCL_c - 1 \tag{24}$$

$$LCR_{c,t} \geq 1 - \sum_{tt=t}^{t+rt} WCT_{a,tt} \qquad t \in T, MCL_c - 1 \leq rt \tag{25}$$

$$AR_{a,t_1} + AR_{a,t_2} - 2 - |T|(2 - AR_{a,t_1} - AR_{a,t_2}) \leq MWAT_a - (t_2 - t_1) \quad \forall a \in A, \forall t_1, t_2 \in T \tag{26}$$

$$SCR_{c,t_1} + SCR_{c,t_2} - 2 - |T|(2 - SCR_{c,t_1} - SCR_{c,t_2}) \leq MWCTS_c - (t_2 - t_1) \quad \forall a \in A, \forall t_1, t_2 \in T \tag{27}$$

$$SLR_{c,t_1} + SLR_{c,t_2} - 2 - |T|(2 - SLR_{c,t_1} - SLR_{c,t_2}) \leq MWCTS_c - (t_2 - t_1) \quad \forall a \in A, \forall t_1, t_2 \in T \tag{28}$$

$$LC_{c,h_c,0} = 1 \qquad \forall c \in C \tag{29}$$

$$LC_{c,h_c,t} = SLR_{c,t} \qquad \forall c \in C, \forall t \in T \tag{30}$$

$$LA_{a,H,0} = 1 \qquad \forall a \in A \tag{31}$$

$$LA_{a,H,t} = AR_{a,t} \qquad \forall c \in C, \forall t \in T \tag{32}$$

$$\sum_{l_1} MA_{a,l_1,l_2,t} \leq LA_{a,l_2,t+rtta_{rtta_{a,l_1,l_2}}} \quad \forall a \in A, \forall l_2 \in L, \forall t \in T \quad (33)$$

$$\sum_{l_1} MC_{a,l_1,l_2,t} \leq LC_{c,l_2,t+rtta_{rttc_{c,l_1,l_2}}} \quad \forall c \in C, \forall l_2 \in L, \forall t \in T \quad (34)$$

$$\sum_{l} LA_{a,l,t} \leq 1 \qquad \forall a \in A, \forall t \in T \quad (35)$$

$$\sum_{l} LC_{c,l,t} \leq 1 \qquad \forall c \in C, \forall t \in T \quad (36)$$

(1) is the objective function. The first two terms calculate the operating costs of the aircraft and crew, whereas the two next terms add the costs of transferring aircraft and crew without scheduled flights. (2) implies the assignment of exactly one aircraft to each flight. We assign the required number of cockpit and other crew members to flights by (3) and (4). Note that these constraints are nonlinear. Constraints (5-10) set the $IA_{f_1,f_2,a}$ and $IC_{f_1,f_2,c}$ equal to 1 only if the flights are performed consecutively by the same aircraft and crew. The aircraft and crew can be assigned if they are available at the corresponding airport and time. This is applied by constraints 11 and 12, respectively.

Updating the locations of aircraft and crew members if they are used for two consecutive flights that the destination of the first and origin of the second are the same airport is done by (13-14). (15-16) set the location if the airports of flights are different. (17-18) are related to transferring aircraft and crew. If an aircraft or crew member is assigned to a flight, the flight duration is considered as its/his/her working hours. This is implied by constraints (19-20). Constraints (21-22) ensure that there is no assignment at the resting time of aircraft/crew. The resting time of aircraft is guaranteed by (23), while the short and long resting time of crew are ensured by (24) and (25), respectively. (26-28) are for respecting the maximum consecutive working time of aircraft and crew before taking the resting times. (29) and (30) express that crew members are at their hubs at the beginning of the planning horizon and also at the long resting time. (31-32) set the airline hub (H) as the initial and service location of aircraft. Changing the locations of aircraft and crew after transferring them to a new airport is implemented by (33-34). Finally, (35-36) enforce that each aircraft and crew member can be at most at one airport at a specific time.

Variable declarations are formalized as indicated in the notation above and not formalized for brevity.

The model and subsequent optimization strategy are transformed into their robust versions, whereby multiple scenarios are taken into account for the uncertain parameters. Every solution derived from the model is associated with a particular objective value based on each scenario. The robust objective value for each solution represents the most unfavorable outcome, considering all potential scenarios. Consequently, we can ascertain that there are no inferior results with respect to that particular solution. This aligns with the definition of robustness as outlined in [1]. Within our robust optimization framework, our aim is to identify the solution linked to the least unfavorable value across all scenarios. In other words, from a mathematical perspective, with reference to Equation (1) in the model:

$$Min_{scen \in Scen} \ (Max \ Z(scen))$$

where $Scen$ denotes the set of conceivable scenarios concerning the uncertain parameters, $scen$ denotes an individual scenario within this set, and $Z(scen)$ signifies the objective value corresponding to $scen$.

4 Approach

In this work, it is assumed that aircraft and crew members can be unavailable during some random periods. So different availability scenarios are considered and the problem is solved based on all the scenarios and the worst objective value of them and its corresponding solution are regarded as the robust objective value and robust solution.

Due to the high computational complexity, a matheuristic algorithm is designed to find good quality solutions in (almost) real time. This algorithm consists of a main PSO framework where in each iteration apart from the movements of the particles (solutions) towards their best experienced positions and the global best position of all particles so far, some solutions are chosen probabilistically according to their quality to be improved using the SA concept by neighbourhood searches, and also some other solutions are similarly selected to be improved through solving the mathematical model related to a random tractable part of them by the Baron solver [14] (which is able to handle nonlinearity). The size of this part is a parameter in this algorithm, which is tuned at the beginning. This mathematical optimization process in done based on the concept of the partial optimization metaheuristic under special intensification conditions (POPMUSIC) [17]. The termination condition of this matheuristic is exceeding a number of consecutive iterations without any improvement in the best solution of the population. The pseudocode of our proposed algorithm is shown in Algorithm 1.

Algorithm 1: The proposed matheuristic

Data: Problem inputs
Result: A feasible solution expected to be of good quality
Set the parameters of the algorithm.
Generate $nPop$ random candidate solutions as the initial particles.
Evaluate the population.
$It = 0$.
Initialize a variable called BO, which keeps the best objective value found so far.
while $It \leq MCUI$ **do**
    Move the particles based on their new speed.
    Evaluate the particles.
    Choose $\lceil PSA \times nPop \rceil$ candidates from the population by the roulette wheel selection.
    Improve the selected candidates by neighborhood searches based on SA.
    Evaluate the SA results, and then merge them with the previous population.
    Sort the total population based on the objective values.
    Choose $\lceil PMB \times nPop \rceil$ from the best solutions and $\lceil PMO \times nPop \rceil$ based on the roulette wheel selection from other solutions and input them in a set called $PSET$.
    for *Each solution* $spm \in PSET$ **do**
        Decompose spm in K parts, i.e., the set of parts is $H = \{part_1, ..., part_K\}$
        Set $P = \emptyset$
        while $P \neq \{part_1, ..., part_K\}$ **do**
            Select a seed part $s_{seed} \in H$ and $s_{seed} \neq P$ at random
            Build a sub-problem R composed of s_{seed} and its nearest D parts (corresponding to area A)
            Optimize R through solving the mathematical formulation by the Baron solver
            if R *has been improved* **then**
                Update solution S
                $P \leftarrow \emptyset$
            else
                Include s_{seed} in P
            end
        end
    end
    Merge the POPMUSIC results with the population.
    Sort the population based on the objective value and choose the first $nPop$ best of them as the new population.
    Set $BO =$ Objective value of the first solution of the population.
    if BO *is improved in comparison to its previous value* **then**
        $It = 0$
    else
        $It = It + 1$
    end
end
Report BO.

A feasible solution is encoded as a string, where each element corresponds to one variable and contains a real value between 0 and 1. These values can be easily converted to the equivalent values of the variables. The constraints are respected by generating values of the variables one by one according to the alphabetical order in their feasible interval. These intervals are calculated based on the amounts of the already filled variables. The initial population is generated at random.

The hyper-parameter tuning of the algorithm is done by the response surface method (RSM) ([2]).

5 Results

Our algorithm undergoes testing on various test instances constructed using data from an airline with a hub situated in the Middle East. These instances involve varying sizes of aircraft sets, with corresponding adjustments in crew members and flights directly correlating to the number of considered aircraft. The range of aircraft varies from 10 to 100, while the number of flights spans from 400 to 4000. The crew size and their location distributions align with the considered fleet and flights. Five types of aircraft are under consideration, with the proportion of each type reflecting their real presence in the airline's fleet. Planning is projected over a one-month horizon, with aircraft and crew resting times conforming to established regulations [11], [3]. Other elements of the scenario are derived from the airline's available data sheets.

Random unavailable periods are considered for 2% of the aircraft and 5% of the crew. For each instance, 10 random scenarios of uncertainty are generated. In each scenario, the aircraft and crew subjected to unavailability are chosen at random, and they are considered to be unavailable within a random contiguous period with a duration equal to 0.1 of the planning horizon.

A state-of-the-art metaheuristic solution method, specifically the vibration damping optimization [13], is selected from existing literature, and it has demonstrated successful application in addressing a variant of the AFCSP. Furthermore, in order to evaluate the impact of various components within our matheuristic approach, we conduct it once without mathematical optimization and another time with a sole mathematical optimization excluding any metaheuristic operator is examined. Since the outcomes of metaheuristics and matheuristics can vary in each execution due to their probabilistic characteristics, we perform five runs for each instance and base our analysis on the average results obtained from these runs.

The average robust results obtained by our matheuristic (M), our algorithm without mathematical programming improvements (H), the vibration damping optimization used in [13] (S) re-implemented by us and also the pure mathematical programming (MP) are shown in terms of the objective value and execution time in Fig. 1 and Fig. 2. It should also be noted that (MP) is not able to reach the optimality and terminate within the considered one hour time limit in any case and even does not find a feasible solution for instances with 40 and more aircraft. Therefore, the objective values are not shown for the instances having 40 aircraft and more, and no execution times are depicted for MP.

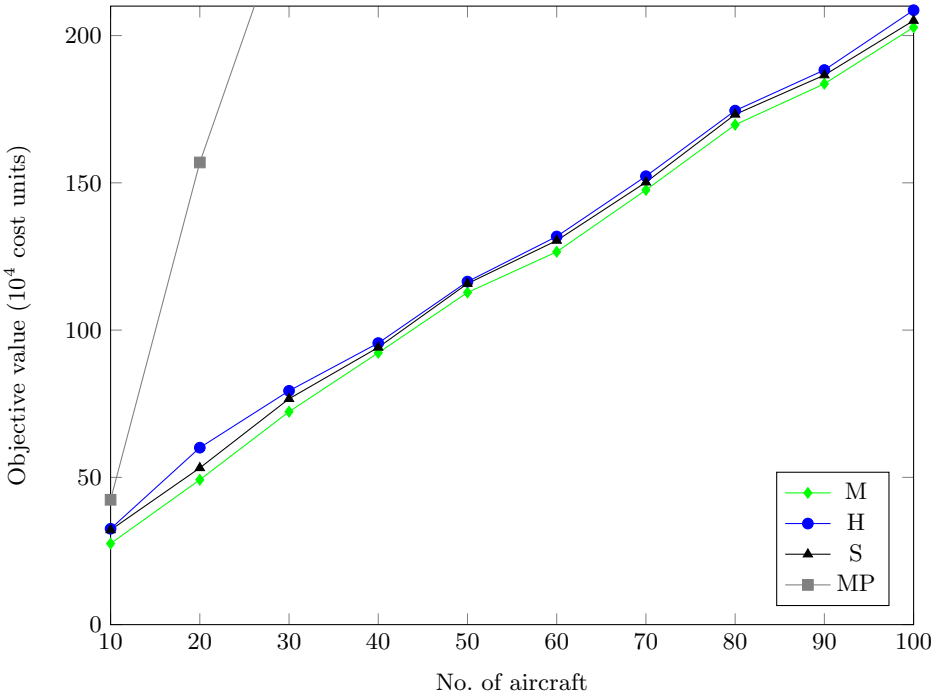


Fig. 1. Comparison of the objective value of the methods

It is evident that our algorithm consistently delivers superior results compared to others. On the other hand, its average execution times are still tolerable although it is slower than the methods H and S due to the mathematical optimization embedded in it.

In Fig. 1, the lines denoted as M, S, and H, specifically the green, black, and blue lines, may exhibit a close proximity to one another. Nevertheless, notable disparities exist among the outcomes yielded by these three methodologies.

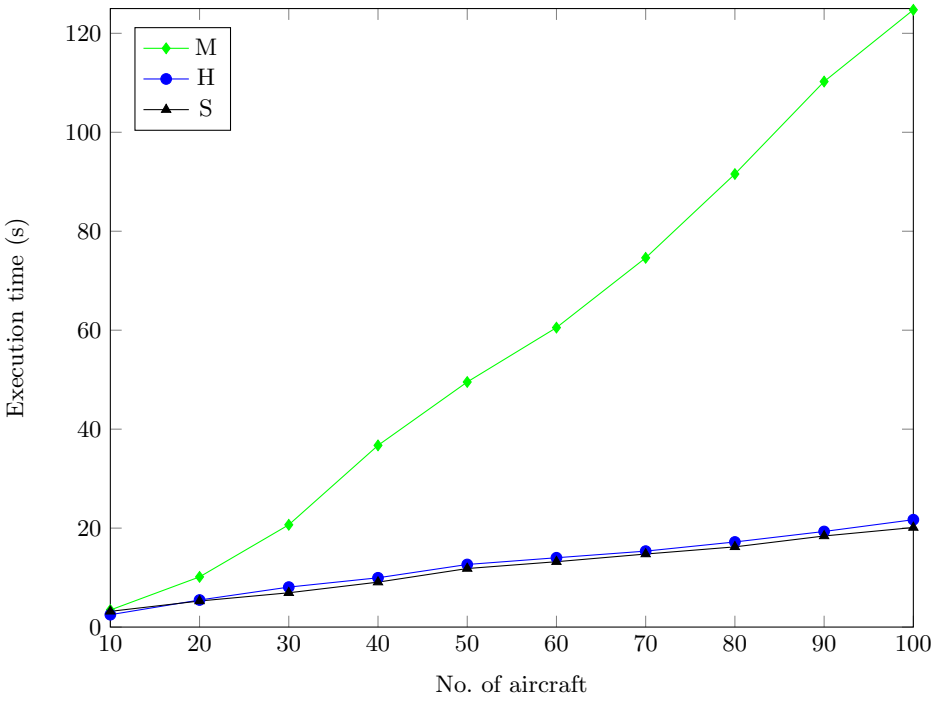


Fig. 2. Comparison of the execution time of the methods

Table 1. All results of all instances (shown by the No. of aircraft) obtained by our algorithm M vs. vibration damping S; the shown results are multiplied by 10^4

	M	S		M	S		M	S		M	S		M	S
	26.14	32.78		71.80	75.61		111.62	114.50		146.79	149.56		182.82	185.75
	26.14	33.92		72.03	76.03		112.88	115.45		147.10	150.32		184.35	186.11
10	27.83	30.47	30	72.62	76.75	50	112.75	115.45	70	147.25	150.27	90	183.80	186.28
	27.97	34.56		72.75	77.22		113.30	116.30		147.90	150.32		183.18	187.11
	27.30	30.47		72.75	77.39		112.75	116.30		148.71	151.13		184.35	187.75
	48.25	52.46		91.75	93.56		125.75	129.69		168.38	172.53		201.83	204.32
	48.55	52.74		92.75	93.94		126.25	130.11		169.07	173.05		202.17	204.32
20	49.37	53.38	40	92.58	94.21	60	126.50	130.50	80	169.07	173.12	100	202.74	205.45
	50.30	53.38		92.91	94.52		127.15	131.15		170.12	174.20		202.74	206.25
	51.43	54.54		93.21	95.02		127.35	131.45		171.36	174.20		204.02	206.96

To illustrate, in order to elucidate the supremacy of our matheuristic over the most competitive approach S, the objective values derived from all executions (runs) on all instances related to these methodologies are delineated in Table 1. Evidently, the outcomes generated by our approach (M) surpass those of the

vibration damping technique (S), underscoring a substantial advantage in the performance of our approach over its nearest competitor.

To investigate whether a statistically significant difference exists between the means of the objective values generated by our algorithm (M) and its rival, vibration damping optimization (S), we carried out the Wilcoxon signed-rank test [20]. This test offers the benefit of being non-parametric and not requiring the assumption of normal data distribution within each group. The resulting p-value is below 0.00001, strongly rejecting the null hypothesis of the equality of mean values of outcomes associated with the two groups.

6 Conclusions

The conducted computational experiments show that the proposed matheuristic is able to provide results of better quality as compared with other methods. In addition, mathematical programming plays an important role in its success because without this operator the results are considerably deteriorated. Nonetheless, this increases the solution time but the required computational times are still acceptable. So, we developed a solution approach which can also outperform a state-of-the-art method already applied to a version of the AFCSP.

Future research may incorporate further real-world requirements like delay propagation and other issues that have already been investigated in public transport, as, e.g., exemplified in [6,7]. Using our matheuristic for larger instances including more details and enhancing its abilities are also among our future research directions.

References

1. Ben-Tal, A., Nemirovski, A.: Robust solutions of uncertain linear programs. Operations Research Letters **25**(1), 1–13 (1999). https://doi.org/10.1016/s0167-6377(99)00016-4
2. Box, G.E.P., Draper, N.R.: Response Surfaces, Mixtures, and Ridge Analyses, 2nd Edition. Wiley-Interscience (2007)
3. Chen, B.: Maintenance and repair management and optimization of aircraft base on working hour standard. Applied Mechanics and Materials **511–512**, 783–786 (2014). https://doi.org/10.4028/www.scientific.net/amm.511-512.783
4. Deveci, M., Demirel, N.Ç.: A survey of the literature on airline crew scheduling. Engineering Applications of Artificial Intelligence **74**, 54–69 (2018). https://doi.org/10.1016/j.engappai.2018.05.008
5. Gao, C., Johnson, E., Smith, B.: Integrated airline fleet and crew robust planning. Transportation Science **43**, 2–16 (2009). https://doi.org/10.1287/trsc.1080.0257
6. Ge, L., Kliewer, N., Nourmohammadzadeh, A., Voß, S., Xie, L.: Revisiting the richness of integrated vehicle and crew scheduling. Public Transport (2022). https://doi.org/10.1007/s12469-022-00292-6

7. Ge, L., Nourmohammadzadeh, A., Voß, S., Xie, L.: Robust optimization for integrated vehicle and crew scheduling based on uncertainty in the main inputs. In: The Fifth Data Science Meets Optimisation Workshop at IJCAI-22. Vienna (2022), https://sites.google.com/view/ijcai2022dso/, last accessed 30 April 2023
8. Gendreau, M., Potvin, J.Y.: Handbook of Metaheuristics, International Series in Operations Research & Management Science, vol. 272. Springer Nature, Cham (2018). https://doi.org/10.1007/978-3-319-91086-4
9. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of ICNN'95 - International Conference on Neural Networks. pp. 1942–1948 vol.4 (1995). https://doi.org/10.1109/ICNN.1995.488968
10. Maniezzo, V., Stützle, T., Voß, S. (eds.): Matheuristics: Hybridizing Metaheuristics and Mathematical Programming, Annals of Information Systems, vol. 10. Springer, Dordrecht (2009). https://doi.org/10.1007/978-1-4419-1306-7
11. Missoni, E., Nikolić, N., Missoni, I.: Civil aviation rules on crew flight time, flight duty, and rest: Comparison of 10 ICAO member states. Aviation, Space, and Environmental Medicine **80**(2), 135–138 (2009). https://doi.org/10.3357/asem.1960.2009
12. Novak, A., Badanik, B., Brezonakova, A., Lusiak, T.: Implications of crew rostering on airline operations. Transportation Research Procedia **44**, 2–7 (2020). https://doi.org/10.1016/j.trpro.2020.02.001
13. Rashidi Komijan, A., Tavakkoli-Moghaddam, R., Dalil, S.A.: A mathematical model for an integrated airline fleet assignment and crew scheduling problem solved by vibration damping optimization. Scientia Iranica **28**(2), 970–984 (2021). https://doi.org/10.24200/sci.2019.51516.2230
14. Sahinidis, N.: Baron: A general purpose global optimization software package. Journal of Global Optimization **8**, 201–205 (1996). https://doi.org/10.1007/BF00138693
15. Schaefer, A.J., Johnson, E.L., Kleywegt, A.J., Nemhauser, G.L.: Airline crew scheduling under uncertainty. Transportation Science **39**(3), 340–348 (2005). https://doi.org/10.1287/trsc.1040.0091
16. Smith, B.C., Johnson, E.L.: Robust airline fleet assignment: Imposing station purity using station decomposition. Transportation Science **40**(4), 497–516 (2006). https://doi.org/10.1287/trsc.1060.0153
17. Taillard, É.D., Voß, S.: POPMUSIC — partial optimization metaheuristic under special intensification conditions. In: Ribeiro, C., Hansen, P. (eds.) Essays and Surveys in Metaheuristics, pp. 613–629. Springer, Boston (2002). https://doi.org/10.1007/978-1-4615-1507-4_27
18. Van Laarhoven, P.J.M., Aarts, E.H.L.: Simulated annealing: theory and applications. Kluwer, Dordrecht (1987)
19. Wen, X., Ma, H.L., Chung, S.H., Khan, W.A.: Robust airline crew scheduling with flight flying time variability. Transportation Research Part E: Logistics and Transportation Review **144**, 102132 (2020). https://doi.org/10.1016/j.tre.2020.102132
20. Wilcoxon, F.: Individual comparisons by ranking methods. Biometrics Bulletin **1**(6), 80 (1945). https://doi.org/10.2307/3001968

Measuring Social Mood on Economy During Covid Times: A BiLSTM Neural Network Approach

Mauro Bruno, Elena Catanese, Francesco Ortame, and Francesco Pugliese

Italian National Institute of Statistics

Abstract. Supervised machine learning algorithms, while popular for sentiment analysis, face limitations tied to the quantity and quality of the training data, especially in the presence of biases or insufficient data. In this study, we try to address these issues investigating the impact of Covid-19 on the Social Mood on Economy Index (SMEI), through a bidirectional long-short term memory (BiLSTM) classifier, computing a daily sentiment index for Italian tweets with economic content throughout 2020 (comprising over 11 million Tweets). We show that training the model with labeled Covid-related Tweets improves the classifier accuracy, while quantitative analysis reveals a sharper decrease in the level of the index during the first lockdown period. Additionally, we explore the effects of different training and tuning procedures, including training set balancing and one-step and two-step approach-es with fine-tuning, on the dynamics of the index. We conclude that, when the size of the training data is small compared to the corpus, it is preferable to perform a one-step training procedure, while balancing the training sets generally improves the model accuracy.

Keywords: Sentiment Analysis · Deep Learning · Twitter (X) · Word Embedding

1 Introduction

In recent years, there has been a significant surge in the use of social media platforms for consuming news, expressing emotions, and engaging in discussions. The interest in using social media to measure public sentiment has been on the rise during the last decade. National Statistical Offices (NSOs) have explored the use of social media messages to develop specialized sentiment indices for specific domains. One of the earliest examples of adopting Twitter data to perform sentiment analysis tasks in NSOs is a work by the National Statistical Office of Mexico (INEGI) [1]. Our goal is to assess the collective mood of Italian people on a specific subset or aspect of life, enabling frequent evaluations, potentially on a daily basis. This paper introduces a novel supervised approach to calculating a Daily Sentiment Index, specifically focusing on the public mood regarding the Italian economy – the Social Mood on Economy Index (SMEI) [2]. Adopting an unsupervised method since 2018, the Italian National Institute of Statistics

(Istat) initially relied on lexicons due to the absence of extensive labeled Italian tweet datasets. However, with the increasing availability of such datasets, our investigation shifted towards advanced supervised deep learning algorithms for sentiment classification. Our approach incorporates a Bidirectional Long Short-Term Memory (BiLSTM) neural network as a binary classifier. We employ a hybrid approach, which involves two steps: 1) unsupervised training of a word embedding model on an unlabeled cor-pus of Italian tweets from the SMEI, 2) supervised training to generate the sentiment scoring model. The Covid-19 outbreak prompted us to re-train the network, incorporating a Covid-related labeled dataset for improved accuracy. We also examined the impact of splitting the training into two steps, resulting in four scenarios: the original model and the re-trained model, both in one and two-step (fine-tuning) frameworks. Additionally, we investigated the effect of balancing training data on accuracy across all scenarios. This study aims to assess the network's effectiveness in detecting the drop in social mood related to Covid-19 during the lockdown period. The paper follows the following structure: in Section 2, we elaborate on Embed-ding and LSTM methods. Section 3 delves into the different scenarios, details the dataset employed in the simulations and delineates the training settings. Finally, Section 4 analyzes the dynamics of the resulting daily social mood indices, and Section 5 encapsulates the conclusions drawn from our study.

2 Methods

In this section, we briefly outline the methodology behind our implemented models. We will provide an overview of word embeddings and LSTM networks, with a specific emphasis on bidirectional LSTM networks.

2.1 Word Embeddings

Word vector spaces have become a pivotal tool in unsupervised machine learning, enabling the extraction of word representations from extensive unstructured data [3]. These representations capture both syntactic and semantic patterns, aligning with the distributional hypothesis: "You shall know a word a word by the company it keeps" [4]. This principle posits that words with similar meanings tend to appear in similar contexts. Word Embeddings (WE) are techniques that map textual tokens, such as words, into a dense and low-dimensional vector space, built from extensive unlabeled corpora, where each token is correlated with other tokens in its context. Methods for generating this mapping include neural networks, dimensionality reduction on the word co-occurrence matrix, and probabilistic models. To capture nuanced semantic relationships and enhance the overall contextual understanding of the texts in our corpus, a FastText [5] word embedding model was employed as the foundational embedding matrix for our Bidirectional LSTM sentiment classifier. FastText, an extension of traditional word embeddings algorithms (such as Word2Vec [6]), excels in representing subword ("n-grams") information, therefore it allows handling morphologically rich

languages and capturing the meaning of out-of-vocabulary words, the latter being a fundamental feature of Twitter (X) conversations. Key hyper-parameters are:

- *Embedding space dimension.* The vector space size to which the words of the corpus are mapped;
- *Window size.* The width of the sliding window defining the amount of context to consider;
- *Iterations.* How many times the weights of the neural network are updated during training;
- *Learning model.* The approach used to train the neural network, either CBOW (continuous bag-of-words) or Skip-gram.

Other factors affect the performance of a word embedding model, like the size of the corpus, as bigger corpora tend to lead to better performance, and the quality of the corpus, as noisy, fragmented and poorly curated texts generally produce lower quality embedding spaces.

2.2 Long Short-Term Memory Networks

Our classification model employs Long Short-Term Memory (LSTM) networks [7], a type of Recurrent Neural Networks (RNN). LSTMs excel at processing sequential data, making them highly effective for tasks that require understanding temporal dependencies. When dealing with textual data, this translates to a better understanding of longer contexts. In general, a LSTM memory cell comprises four integral components: an input gate, an output gate, a forget gate, and a self-recurrent neuron (Fig. 1).

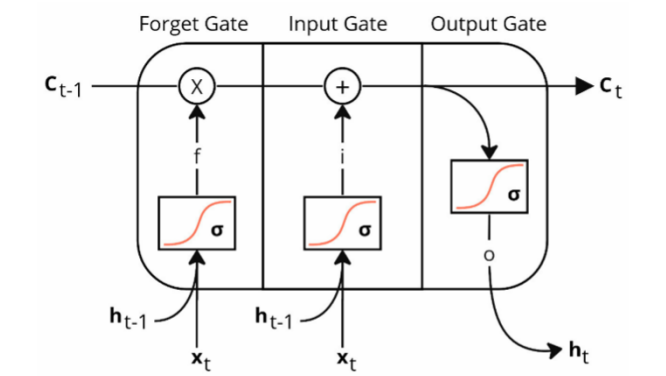


Fig. 1. Simplified scheme of an LSTM cell.

These gates collectively manage the flow of information within the memory cell. The input gate controls whether incoming signals can modify the state of

the memory cell, while the output gate regulates whether the memory cell can influence other neighboring cells. Additionally, the forget gate has the capability to decide whether to retain or discard information from the existing state. LSTM layers consist of a series of interconnected memory cells, typically activated by hyperbolic tangents, which control the information flow through the system of gates. These gates operate using sigmoid functions, defined as follows:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

This function maps any real-valued number to a value between 0 and 1, where 0 signifies a complete inhibition or blockage of information, while a value of 1 indicates unrestricted passage, allowing the gates to selectively open or close based on the input data. In particular:

$$i_t = \sigma\left(w_i[h_{(t-1)}, x_t] + b_i\right) \tag{2}$$

$$f_t = \sigma\left(w_f[h_{(t-1)}, x_t] + b_f\right) \tag{3}$$

$$o_f = \sigma\left(w_o[h_{(t-1)}, x_t] + b_o\right) \tag{4}$$

where i_t, f_t, and o_t represent the input, forget and output gates respectively, σ represents the sigmoid function, w_x is the weight for the respective $gate_x$ neurons, $h_{(t-1)}$ is the output of the previous LSTM block, x_t is the input of the current LSTM block and b_x represents the bias for the respective $gate_x$.

In addition, LSTM networks make use of hyperbolic tangent activation functions, defined as follows:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{5}$$

Similarly to the sigmoid function, the hyperbolic tangent function maps any real value to a number between -1 and +1, but unlike the sigmoid function, it centers the output around 0, allowing for both positive and negative activations. This property ensures that information can flow in both directions through the network's gates and memory cells, allowing capturing both positive and negative temporal dependencies in the data.
Specifically, the hyperbolic tangent function is employed in two key areas:

- *Memory cell computation.* The output of the forget gate, multiplied by the previous cell state, and the candidate memory cell state (generated using a tanh function to combine input and previous hidden state) are added together, effectively controlling the information retained in the memory cell;
- *Hidden state generation.* The final step involves applying the tanh function to the memory cell state modulated by the output gate. This generates the final hidden state, which captures the relevant information from the current in-put and previous states, ready to be passed onto the next LSTM block in the sequence.

The choice of an LSTM-based architecture for text classification is motivated by the inherently short nature of tweets. While LSTMs have been noted for struggling with longer sequences, the issue is less pronounced in the context of brief textual content. In contrast to more recent algorithms such as Transformers, LSTMs still show some advantages in scenarios with limited labeled datasets.

2.3 Bidirectional LSTM Architecture

While traditional LSTM networks excel at analyzing input sequences in a forward direction, they might miss important time dependencies present in later data. This is particularly true with textual data, where later information is just as important as earlier information in understanding context. That is where Bidirectional LSTM (BiLSTM) models [8] come in handy. Our chosen BiLSTM model incorporates this powerful architecture to leverage contextual information from both the past and the future within the text.

The first stage involves *embedding* the input texts into numerical vectors using a pre-trained embedding matrix. This matrix encodes semantic relationships between words, allowing the model to understand their meaning. These embeddings are then fed into two separate *LSTM layers*, processing the sequence in both the forward and backward directions simultaneously. Each LSTM layer consists of several memory cell with gates regulating information flow. These gates, governed by sigmoid and hyperbolic tangent functions, control what information is retained, forgotten, and ultimately passed onto the next cell. By processing the text in both directions, BiLSTMs gain access to richer contextual information, allowing for a more informed prediction about the overall sentiment of a sentence. Our BiLSTM model employs two stacked BiLSTM layers, further enriching its ability to extract complex features from the textual data. By stacking layers, the model learns increasingly abstract representations of the input at each level. Finally, a *Global Max Pooling* layer captures the most significant activation across the sequence, summarizing the learned information. This condensed representation is then fed into a single neuron with a sigmoid activation, ultimately predicting the binary classification outcome.

3 Training Settings

In this section, we analyze the structure of our training data and of the different train-ing scenarios for the BiLSTM neural network, with a focus on the distinctive influ-ence of Covid-related expressions.

3.1 Training Set Composition

The training set is made of four distinct labeled datasets, each focused on different features that contribute to the model's understanding. The datasets are the following:

- *Sentipolc.* A widely used Italian labeled dataset for sentiment analysis [9];

- *Happy parents.* Another Italian labeled dataset contributing several sentiment expressions related to parenting behavior and experiences [10];
- *Feel-it.* The only dataset that explicitly contains tweets related to the Covid pandemic [11];
- *Istat.* A dataset annotated directly by the Italian National Institute of Statistics.

The dataset labels are distributed as follows:

Table 1. Training Datasets

	Dataset							
	Sentipolc		*Happy parents*		*Feel-it*		*Istat*	
Label	**pos**	**neg**	**pos**	**neg**	**pos**	**neg**	**pos**	**neg**
N	1914	3242	641	671	724	1299	363	374
Total	5156		1312		2023		737	
	9228							

The neutral sentiment class, where present, was dropped to build a binary classifier.
In the case of unbalanced data, the final training set contains 9228 Tweets, while in the case of balanced data (under-sampling), the final training set contains 7284 Tweets. The training sets that were the most affected by the under-sampling were Sentipolc and Feel-it. The latter contained 157 Tweets related to the Covid-19 pan-demic before the under-sampling and 104 Tweets after.

3.2 Training Scenarios

The training scenarios are strategically designed to explore the impact of different training set compositions on sentiment predictions. There are two primary categories: scenarios without fine-tuning (*scenario 1* and *scenario 2*) and scenarios with fine-tuning (*scenario 3* and *scenario 4*).

Table 2. Training Scenarios

Scenario	Training	Fine-tuning
Scenario 1	Sentipolc, Happy parents, Istat, Feel-it	-
Scenario 2	Sentipolc, Happy parents, Istat	-
Scenario 3	Sentipolc, Happy parents, Istat	Feel-it
Scenario 4	Sentipolc, Happy parents	Istat

In detail (Table 2), scenario 1 incorporates all available datasets, providing a general understanding of sentiment across various domains, including Covid-related expressions. Scenario 2 omits the Feel-it dataset, focusing on general sentiment patterns without considering Covid-related expressions. In scenario 3, the first stage of the training process covers general sentiment patterns, while the fine-tuning stage specifically adapts the model to the nuances of Covid-related sentiments. In scenario 4, the first stage focuses on general sentiments, while the second stage fine-tunes the model to the sentiment expressions captured by the Istat dataset. It is important to note that the *test set* remains consistent across all training scenarios, encompassing tweets from each training dataset. The extraction process employs stratification to preserve the original proportions of the datasets. In total, the test set constitutes 16% of the original training set.

3.3 Common Training Settings

The training process across all scenarios adheres to a set of consistent parameters. In particular, *data balancing* involves under-sampling to achieve equilibrium between positive and negative labels due to a significant prevalence of negative-labeled tweets. Models have been trained both with unbalanced and balanced training sets for every scenario. The *validation set* constitutes 20% of the scenario-specific training dataset and it is used to validate the training process.

Embedding Model. A common FastText-based word embedding model is fed to the BiLSTM network to ensure shared natural language understanding across all scenarios. The training set-tings with which the embedding model is trained are shown in Table 3.

Table 3. Embedding Model Specifications

Embedding Model					
Algorithm	Vector Size	Window	Min. Count	SG	Epochs
FastText	200	8	10	0	25

The *vector size* parameter represents the embedding space, i.e. the dimensionality of the word vectors produced by the model. In this case, each word is represented as a 200-dimensional array in the embedding space. The *window* parameter defines the maximum distance between the current and predicted word within a sentence. If set to 8, the model considers 8 words to the left and 8 words to the right of the target word during training. The *min. count* parameter refers to the threshold set for the minimum number of times a word needs to occur in the corpus to be included in the vocabulary. If the *sg* parameter is set to 0, a CBOW (continuous bag-of-words) approach is specified where the model predicts context words given a target word.

BiLSTM Model. A BiLSTM network with a common architecture was implemented across all training scenarios. The specifications for the model are shown in Table 4.

Table 4. BiLSTM Model Specifications

BiLSTM				
Layer 1 Cells	Layer 2 Cells	Dropout Rate	Trainable	Max. Length
64	64	0.5	True	280

Layer 1 cells and *layer 2 cells* denote the number of memory cells or units in the first and second layers of the BiLSTM neural network. The *dropout rate* is a regulariza-tion technique employed to prevent the model from overfitting on the training set dropping out a fraction of inputs during training. The *trainable* parameter indicates whether the embedding layer can be fine-tuned during the training process. If so, the embedding layer's weights are updated during training, allowing the model to learn and adjust word representations.

Training Loop. Concerning the training loop, the *NAdam* optimizer is employed with a starting *learning rate* of $1e^{-3}$, incorporating dynamic learning rate reduction on plateau to mitigate overfitting. The *binary crossentropy* loss function is evalued to carry out the gradient descent procedure. A fixed *batch size* of 32 is set for every training scenario. The model undergoes training for 15 *epochs* in both the first and second stages, when admissible.

Fine Tuning. In scenarios 3 and 4, the base model undergoes fine-tuning on the Feel-it and Istat datasets, respectively. The settings for this second step mirror those of the initial training, with the exception that all layers of the model, except the last one, are fro-zen. This precaution is taken to prevent the weights of the model from being overly influenced by the dataset on which it undergoes fine-tuning. By freezing all layers, we aim to retain the knowledge captured during the initial training while adapting the model to the specific characteristics of each target dataset.

This results in eight distinct models, corresponding to the four scenarios in both balanced and unbalanced training set configurations.

3.4 Corpus of Prediction

The sentiment prediction is carried out on a corpus of approximately 11.2 million Tweets spanning from January 1, 2020 to December 31, 2020. Notably, the word embedding model utilized for the BiLSTM network is trained on this same corpus, ensuring alignment between the training and prediction phases. The reference period includes the burst of the Covid pandemic, we focus particularly

on understanding the impact of including Covid-related texts in the training process.

3.5 The Index

The predicted sentiment of each Tweet is a number between 0 and 1. If the resulting prediction is higher than 0.5, the tweet is considered positive, if it is lower than 0.5, it is considered negative. The daily index is built in the following way:

$$I_t = \frac{N_{POS} - N_{NEG}}{N_{POS} + N_{NEG}} \tag{6}$$

where N_{POS} is the number of tweets that have been classified as positive for day t, while N_{NEG} is the number of tweets that have been classified as negative.

4 Results

In this section we discuss aspects related to the model performance and the trends of the predicted indices across the different training scenarios.

4.1 Model Performance

The model's performance has been assessed using both accuracy and F1-scores, providing robust measures that account for the unbalanced nature of the dataset. The results for the different scenarios refer to metrics computed on the test set (common across all scenarios) and are shown in Table 5.

Table 5. Model Performance

Model Performance					
		One-step		*Two-step*	
Metric	*Training Set*	Scenario 1	Scenario 2	Scenario 3	Scenario 4
Accuracy	Balanced	0,80	0,80	0,82	0,76
	Unbalanced	0,81	0,79	0,80	0,77
F1-score	Balanced	0,80	0,81	**0,82**	0,79
	Unbalanced	0,76	0,74	0,75	0,74

Balanced datasets were employed to enhance the classifier's performance, a trend observed across all scenarios except for scenario 4 two-step, where accuracy exhibits a slight decrease. However, the F1-score consistently improves. Notably, the optimal model performance is achieved when trained on a balanced dataset in scenario 3. In this case, the first step utilizes three general datasets (Sentipolc,

Happy Parents, Istat), and the subsequent second step fine-tunes the model using the dataset that includes Covid-related expressions (Feel-it).
In terms of performance, there is a marginal decrease in all scenarios, whether balanced or not, during the second step. The exception to this trend is the most performant model, obtained in scenario 3 balanced.

4.2 Predictions

The indexes obtained using the predicted sentiment in the different training scenarios are depicted in Figure 2 and Figure 3. The grey area in both figures represents the first lockdown period in Italy, spanning from March 9, 2020 to May 18, 2020. This period is of particular interest due to its significant impact on dynamics.

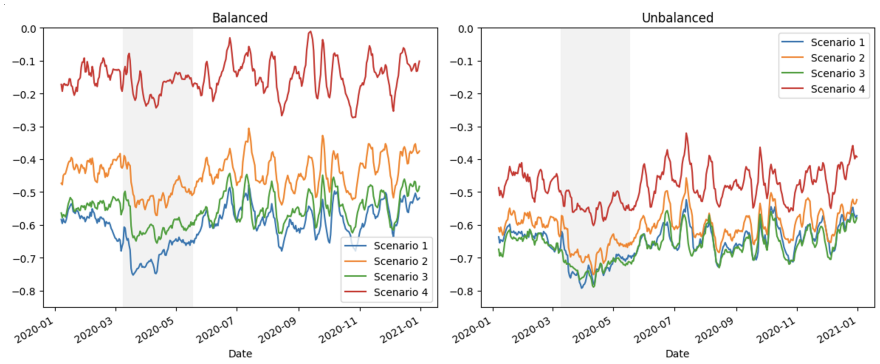


Fig. 2. Predictions – balanced vs unbalanced training sets for all the different scenarios (7-day moving average, left window)

In Figure 2, the 7-day moving averages (MA) for the four scenarios trained with both balanced and unbalanced datasets are presented. Notably, scenario 4 exhibits the highest variation in the mean value between balanced and unbalanced (0.33), while scenario 1 shows the lowest variation (0.05). Scenario 3 unbalanced attains the lowest mean value (-0.66), followed by scenario 1 unbalanced (0.65). However, the most substantial decline during the lockdown period is observed in scenario 1 balanced (Figure 3).
The influence of Feel-it is most noticeable in balanced and one-step settings, with the highest downward trend recorded in the fine-tuning setting with the unbalanced training set. In both balanced and unbalanced frameworks, scenarios 1 and 3, which include the Feel-it dataset in their training process, consistently register the lowest values. These findings align with the correlation matrices.

The correlation matrices for the level of the 7-day moving average series and the day-over-day variations of the 30-day moving average series are presented in

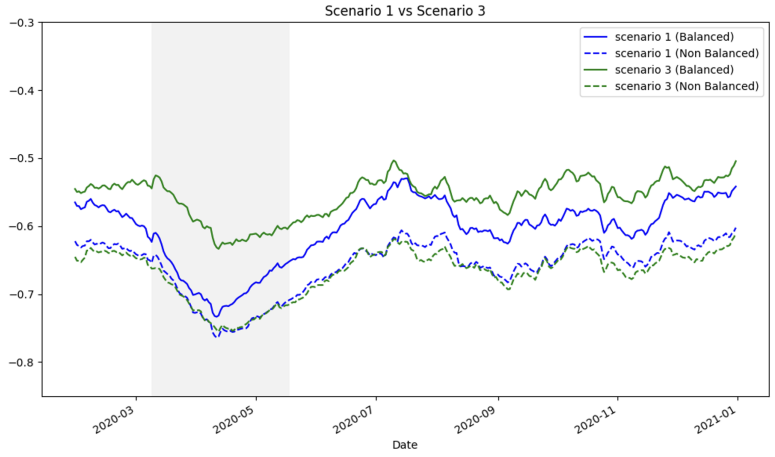


Fig. 3. Predictions – Highlight of the differences in predictions between scenario 1 and scenario 3, both with balanced and unbalanced training sets (30-day moving average, left window)

Table 6. Correlation Matrix of Predictions (7-day moving average)

Correlation Matrix									
		Balanced				Unbalanced			
		S1	S2	S3	S4	S1	S2	S3	S4
Balanced	S1	1	0,85	0,82	**0,66**	**0,94**	0,90	0,91	0,84
	S2		1	0,90	0,76	0,89	0,92	0,87	0,91
	S3			1	0,77	0,86	0,89	0,90	0,84
	S4				1	**0,64**	0,70	0,67	0,75
Unbalanced	S1					1	**0,93**	0,92	0,86
	S2						1	0,92	0,89
	S3							1	0,88
	S4								1

Tables 6 and 7. Notably, the maximum correlations exhibit coherence between both matrices. The highest correlation values are observed between scenario 1 balanced vs scenario 1 unbalanced (0.94 for 7-day MA, 0.91 for 30-day d-o-d variation) and scenario 1 unbalanced vs scenario 2 unbalanced (0.93 for 7-day MA, 0.91 for 30-day d-o-d variation).

Table 7. Correlation Matrix of the d-o-d variation of the 30-day moving average

Correlation Matrix									
		Balanced				Unbalanced			
		S1	S2	S3	S4	S1	S2	S3	S4
Balanced	S1	1	0,85	0,82	**0,75**	**0,91**	0,88	0,83	0,79
	S2		1	0,87	0,82	0,88	0,86	0,79	0,85
	S3			1	**0,76**	0,83	0,79	0,87	0,80
	S4				1	0,79	0,85	0,80	0,82
Unbalanced	S1					1	**0,91**	0,83	0,80
	S2						1	0,84	0,83
	S3							1	0,82
	S4								1

The lowest correlations are recorded for scenario 4 balanced, particularly against scenario 1 balanced (0.66 for 7-day MA, 0.75 for 30-day d-o-d variation). Specifically, in the case of the 7-day MA, the lowest correlation is against scenario 1 balanced (0.66), while for the 30-day d-o-d variations, it is against scenario 3 balanced. Notably, scenario 1 balanced shows the highest correlation with scenario 1 unbalanced, followed by scenario 2 unbalanced.
In general, fine-tuning the weights of the last layer appears not to produce a significant effect on the dynamics. For example, scenario 3 balanced, which is associated to the highest F1-score, exhibits a high correlation with its unbalanced counterpart and scenario 2 balanced, with which it shares the first-step of training.

Regarding the impact of balancing on dynamics, in the case in which we incorporate Feel-it, results show that balancing benefits predictions. Conversely, when Feel-it is not included, the unbalanced setting leads to more coherent dynamics. This may stem from the fact that we analyze a period characterized by general negativity. While Feel-it explicitly introduces Covid references, in the other cases, the negativity is better learnt in the unbalanced setting.

5 Conclusion

In this paper we extensively analyzed the impact of balanced and unbalanced datasets, as well as different learning procedures, particularly focusing on the

effects of fine-tuning. When datasets do not largely differ from the first to the second step of training, a single-step approach is more advisable. While balanced datasets generally yield higher accuracy, the observed effects on dynamics are not clear enough to draw generalized conclusions. Additionally, we incorporated a recent labeled dataset (Feel-It) explicitly containing references related to the 2020 pandemic, such as terms like "Covid-19" and "lockdown". Notably, the highest accuracy and the most consistent dynamics are recorded in learning scenarios that include Feel-It. However, even scenarios that do not incorporate Feel-It still exhibit noteworthy breakdowns during the first lockdown. This is likely attributed to the nature of the re-trained embedding model, encompassing the entirety of 2020 and thus Covid-related expressions. Consequently, the model can infer the meaning of such expressions during sentiment classification, mitigating the lack of explicit references. Except for the lockdown period, dynamics among different indices remain relatively consistent, up to a level shift. As expected, when analyzing tens of millions of tweets and comparing them to less than ten thousand labeled texts, fairly correlated dynamics are observed.

In this work, we highlighted the different dynamics achieved by varying training settings of the networks, even though all networks showed comparable levels of accuracy and F1-scores. Unlike most published works in the field, we aimed to evaluate the consistency of the achieved dynamics, analyzing results through our domain-based knowledge.

In conclusion, we recommend pre-training an embedding model on the corpus to be predicted and use it as an input layer for the network. Additionally, we advise incorporating more specialized labeled datasets to enhance the network's nuanced semantic understanding of the domain of interest. For instance, as a future improvement, we could draw a sample from the corpus of prediction and label it. Moreover, as we gather larger labeled training sets, we plan to train more advanced architectures, e.g. Encoder-only Transformers, in order to achieve more accurate results.

References

1. Instituto Nacional de Estadística y Geografía. (2018). Estado de ánimo de los tuiteros en los Estados Unidos Mexicanos: documento metodológico, 2da. Edición, México INEGI
2. Catanese, Elena, et al. "Natural language processing in official statistics: The social mood on economy index experience." Statistical Journal of the IAOS 38 (2022): 1451-1459. https://doi.org/10.3233/SJI-220062
3. Mikolov, Tomáš, Wen-tau Yih, and Geoffrey Zweig. "Linguistic regularities in continuous space word representations." Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies. 2013.
4. Widdowson, Henry. "Jr firth, 1957, papers in linguistics 1934–51." International Journal of Applied Linguistics 17.3 (2007): 402-413.
5. Bojanowski, Piotr, et al. "Enriching word vectors with subword information." Transactions of the association for computational linguistics 5 (2017): 135-146.

6. Mikolov, Tomas, et al. "Efficient estimation of word representations in vector space." arXiv preprint arXiv:1301.3781 (2013).
7. Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.
8. Schuster, Mike, and Kuldip K. Paliwal. "Bidirectional recurrent neural networks." IEEE transactions on Signal Processing 45.11 (1997): 2673-2681.
9. Basile, Pierpaolo, and Nicole Novielli. "Uniba at evalita 2014-sentipolc task predicting tweet sentiment polarity combining micro-blogging, lexicon and semantic features." UNIBA at EVALITA 2014-SENTIPOLC Task Predicting tweet sentiment po-larity combining micro-blogging, lexicon and semantic features (2014): 58-63.
10. Mencarini, Letizia, et al. "Happy parents' tweets." Demographic Research 40 (2019): 693-724.
11. Bianchi, Federico, Debora Nozza, and Dirk Hovy. "FEEL-IT: Emotion and sentiment classification for the Italian language." Proceedings of the eleventh workshop on computational approaches to subjectivity, sentiment and social media analysis. As-sociation for Computational Linguistics, 2021.

Deep Learning for the Classification of Ports in Maritime Transport Statistics via AIS Data

A. Pappagallo, F. Ortame, G. Massacci, F. Sisti, F. Pugliese

Italian National Institute of Statistics

Abstract. Maritime data surveillance has significantly grown in the last decade, notably through technologies like AIS (Automatic Identification System). Predicting vessel positions is crucial for various maritime applications, with a focus on trajectory forecasting. This involves anticipating vessel direction and future locations, which is vital for tasks such as search and rescue, traffic management, and pollution monitoring. Despite the abundance of AIS data available, predicting vessel paths remains challenging. AIS technologies, which use ship transponders, assist vessel traffic services (VTS) and serve as the foundation for tasks such as trajectory prediction and the classification of the arrival port given a route, namely 'Port Classification'. AIS data includes essential vessel information, such as latitude, longitude, speed, and identity, transmitted via VHF signals. Deep learning models are considered state-of-the-art for analyzing AIS data. This study focuses on implementing a 'Port Classifier', evaluating several models including Conv1D, MLP and LSTM on AIS trajectories labelled through heuristic algorithms, and promising results are achieved, with Conv1D showing superiority in port classification tasks. Additionally, we conducted an Exploratory Data Analysis (EDA) to better understand the data. Our findings contribute to enhance maritime data analysis, and demonstrate potential applications for Official Statistics.

Keywords: AIS, Maritime Transport Statistics, Official Statistics, Machine Learning, Deep Learning, Artificial Intelligence

1 Introduction

In recent years, the surveillance of maritime data has gained significant traction. The increase in maritime activities over the past few decades has sparked considerable concerns surrounding Maritime Surveillance (MS) and Maritime Situational Awareness (MSA). In this context, applications leveraging AIS (Automatic Identification System) data have become particularly relevant.

AIS is an automated tracking system that uses transponders on ships to support vessel traffic services (VTS). AIS signals incorporate real-time vessel kinetic information [1], such as current latitude (LAT) and longitude (LON) coordinates, Speed Over Ground (SOG), Course Over Ground (COG), and IMO (International Maritime Organization) - a unique identifier for vessels - among other parameters. Transmitted through

VHF radio signals, AIS data plays a crucial role in navigation safety, maritime security, vessel traffic management, and commercial endeavors [2, 3].

Most studies focus on predicting vessel trajectories. Anticipating the course of ships and estimating their approximate future locations, ranging from several hours to several tens of hours ahead, is crucial for various MS and MSA applications. These applications include search and rescue operations [4], traffic management [5], route planning [6], mitigating port congestion [7], and monitoring pollution levels [8]. Although the AIS provides a wealth of information, forecasting vessel paths using AIS data remains a challenging task [9]. Due to the complex nature and diverse patterns of motion data [10], modern machine learning and deep learning techniques gained popularity [11].

The trajectory prediction task is complemented by emerging initiatives such as 'Routes Missing Values Management' and 'Port Classification'. In this work, we focus on the 'Port Classification' task, hence we aim to assign an arrival port to a given route, i.e. a sequence of AIS signals. Port Classification is especially interesting for Official Statistical Purposes, where information about the vessel movements, connections between ports and the structure seaborn trade is crucial. So far, these type of analyses can be carried out with ad-hoc surveys conducted at National level, which have been very costly and time-consuming.

To achieve this goal, we define the following pipeline: we build a 'Heuristic Algorithm' (HA) that automatically and deterministically assigns an arrival port to a route when certain conditions are met. Then, we train a selection of Deep Learning models on the data labelled by the HA with the aim of validating and expanding it, by labelling routes that do not meet the conditions for HA classification. The Deep Learning step is essential to account for spatial and long-term dependencies in the time-series of AIS signals.

In accordance to the literature, we carry out a model selection process, comparatively evaluating the performance of each model. In particular, we explore the use of a Multi-Layer Perceptron (MLP), a Long Short-Term Memory (LSTM) network and a 1-Dimensional Convolutional (Conv1D) network. We initially treat AIS trajectories as 2-dimensional images with 4 channels (LAT, LON, SOG, COG) and subsequently add two channels to the signal (departure port, H3 encoding index 8[1]) to improve the accuracy of the results.

Results suggest that Conv1D outperforms LSTM, as the spatial correlations captured by the convolutional layer seem to be more effective than the capability of the recurrent neural networks of 'remembering' the previous states and inputs of the series. Moreover, Deep Learning algorithms obtain promising results on the test set, mitigating the overfitting issue observed with traditional machine learning models.

The innovation brought by this work relates to the fact that a small fraction of studies deal with Port Classification compared to Trajectory Prediction with AIS data. The Port

[1] H3 is a geospatial indexing system, developed by Uber Technologies [12], that approximates the GPS coordinates using a hexagonal tessellation of the earth's surface. The H3 index identifies the hexagon containing the ship's coordinates, with the hexagon size depending on the resolution chosen (e.g. for the H3 encoding index 8 the considered resolution is 8)

Classification task has the advantage of not requiring the filling or imputation of missing AIS signals by immediately determining the port of arrival, making it highly performant. At the same time, the task is challenging as it requires the models to infer a series of intermediate steps during the training stage. Despite the possibility of noise or non-stationarity, the models were able to extract valuable internal representations of data patterns and exhibit stable metrics. Our best model (Conv1D) achieved an F1-score of 0.51 (0.84) on the test set when considering 4-channels signals and 0.67 (0.89 accuracy) when considering the 6-channels signals, across a heavily unbalanced training set of 115 classes (ports).

These results demonstrate significant breakthroughs in Official Statistics, as shown in the Results section of this work. Furthermore, mitigating the overfitting effect in Deep Learning models (compared to Traditional Machine Learning approaches) suggests potential for better generalization to other test sets once this statistical system enters the production stage.

1.1 Related Works

Lately, in the domain of maritime data analysis, the use of deep learning architectures gained significant traction, particularly in the forecasting and classification of vessel trajectories derived from AIS messages [13]. In summary, prior research extensively explored the applications of Multi-Layer Perceptron (MLPs), 1-Dimensional Convolutional Neural Networks (Conv1Ds) and Long-Short Term Memories (LSTMs). These Deep Neural Networks (DNNs) in AIS messages trajectories forecasting and classification, demonstrated their efficacy in modeling complex spatio-temporal patterns inherent in maritime data. Multilayer Perceptrons (MLPs) deeply explored AIS Data applications due to their capability to capture complex nonlinear relationships in data. Previous work [14] demonstrated the effectiveness of MLPs in predicting vessel trajectories with high accuracy, attributing their success to the model's ability to learn intricate patterns from historical AIS data. By leveraging MLPs, they showcased the ability to capture complex nonlinear relationships inherent in maritime data, enabling accurate trajectory forecasts over varying time intervals. The main downside of MLPs is the dense pattern of synaptic connections which represents an obstacle for the training algorithms (stochastic gradient descent) to create the complex internal representations able to capture the complex spatial correlations within the vessels trajectories' AIS messages. On the other hand, very complex spatial correlations can be extracted by Convolutional Neural Networks (CNNs). CNNs were applied to AIS trajectory analysis, exploiting their spatial hierarchies to extract informative features from sequential data. For instance, [15] adopted 1D convolutions (Conv1D) to efficiently process AIS messages for trajectory classification tasks, achieving notable performance gains compared to traditional methods. Their work highlighted the spatial hierarchies present in maritime data and illustrated how Conv1D can effectively extract informative features, leading to enhanced classification performance compared to conventional methods. Instead, LSTM networks suddenly emerged as a prominent choice for sequential data modeling, particularly in AIS trajectory forecasting. In another work, researchers demonstrated the superiority of LSTM networks in capturing the long-term temporal

dependencies within AIS messages [16], leading to accurate trajectory predictions even in dynamic maritime environments. Temporal dependencies play a crucial role in AIS trajectory forecasting and the previously mentioned study emphasized the importance of memory cells in LSTM networks, which enable the model to retain contextual information over extended time periods, thereby improving trajectory forecasting accuracy.

2 Methods

A Multilayer Perceptron (MLP) is an artificial neural network (ANN) model, extending Rosenblatt's original Perceptron from 1950 [17]. Unlike the Perceptron, MLPs include hidden layers between input and output layers. Hidden layers process information, acting as the network's computational engine. Designing an optimal MLP architecture involves selecting layer count, neuron count, and connections, known as the architecture problem. Training an MLP involves adjusting synaptic weights to minimize errors (typically using backpropagation) comprising forward and backward propagation phases. On the other hand, Long Short-Term Memory (LSTM) [18] is a specialized type of Recurrent Neural Network (RNN) addressing long-term dependency challenges. Unlike traditional RNNs, which struggle with long-term temporal dependencies, LSTMs retain information over extended durations [19]. LSTMs feature four interconnected layers, diverging from RNNs' single layer, allowing them to capture long-term dependencies crucial for various tasks. Finally, Convolutional Neural Networks (CNNs) [20], initially for computer vision, have shown strong performance in text classification. CNNs utilize convolution and pooling layers with 1D filters, recommended for text classification. These layers extract features independently of position, generating feature maps and reducing dimensionality. Unlike MLPs, CNNs do not necessarily require fully connected layers but may incorporate them for classification purposes. The Conv1D layer applies a convolution kernel along a single dimension yielding output tensors.

3 Results

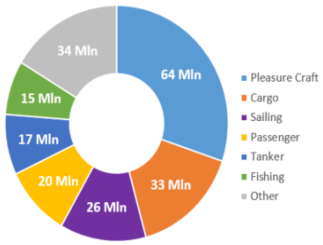
3.1 Dataset

The Deep Learning models were evaluated using records of ship voyages built from the global collection of live and historical AIS data supplied by the UN Global Platform (UNGP) [21]. The platform is based on Hadoop [22] and Spark [23] frameworks, where the first one was used to store data in a distributed file system and the second one was used for the data processing phase. Data is divided into small parts called 'blocks' and distributed among several nodes of a computer network. Each node (called 'worker') can process a part of the data. These frameworks are designed to work together with this data representation model, exploiting the parallelization of processing across different workers to speed up executions. In particular, Hadoop uses a data model called ‘Hadoop Distributed File System’ (HDFS) [24], which is a distributed file system that

stores data in a user-specified size (256MB, in this paper). Blocks are also replicated across multiple nodes to ensure availability and fault tolerance. Spark maps these HDFS blocks into a proprietary data representation model called 'Resilient Distributed Dataset' (RDD) [25], allowing Spark workers to process data in parallel. According with this architecture, the used dataset contains regular observations of all kinds of ships, with a gross tonnage greater than 300 tons, traveling all around the world from December 1, 2018. The time distance between two observations of the same ship is approximately 10 minutes. We used AIS data from April 1, 2023, to April 30, 2023, for an amount of 1,929,200 AIS observations. Besides AIS, we also used the Lloyd's Register of Ships [26], which is the largest maritime database in the world. It is updated monthly and contains comprehensive information about every ship. Finally, we incorporated a dataset of worldwide ports. For each port, the dataset specifies the center of gravity coordinates of the port area, the port name, and its UNLocode (United Nations Code for Trade and Transport Location).

3.2 Exploratory Data Analysis (EDA)

This section discusses the quality of the dataset adopted in the study, with a focus on synthesizing information within a Big Data system. AIS signals from January 2023 to December 2023 were examined to assess the stability of the AIS system over time, by exploiting a different time interval from the trained model. To offer an objective overview of AIS signals, the analysis considers messages transmitted near Italian maritime territory, rather than exclusively those docking at Italian ports. The first aspect analyzed concerns the quantity of signals provided by the system, examining monthly trends in general information, including the number of rows, the number of distinct identifiers, and other auxiliary information relevant to the study. The main findings indicate approximately 17.7 million AIS messages per month, with a rising trend during the summer months and a consistent increase in the number of IMO identifiers, particularly towards the end of the study period. Additionally, the average speed is notably low, indicating a tendency towards stationary ships. Notably, there is a variety of ship types, each with distinct characteristics primarily determined by the length of their voyages. Figure 1 illustrates the distribution of AIS signals for the main types.



Month	Pleasure Craft	Cargo	Sailing	Passenger	Tanker	Fishing
2023-01	6%	8%	5%	6%	8%	7%
2023-02	5%	7%	4%	5%	8%	7%
2023-03	6%	7%	6%	6%	8%	8%
2023-04	7%	8%	7%	8%	8%	8%
2023-05	9%	8%	9%	10%	8%	9%
2023-06	10%	9%	11%	10%	8%	10%
2023-07	12%	11%	13%	12%	10%	10%
2023-08	11%	9%	12%	11%	9%	8%
2023-09	11%	9%	11%	11%	9%	9%
2023-10	9%	9%	9%	9%	9%	8%
2023-11	7%	8%	7%	6%	8%	8%
2023-12	6%	8%	6%	6%	8%	8%

Fig. 1. Visualization of the AIS signals

There are different types of ships, each with specific purposes and travel patterns. Pleasure craft ships account for 30% of the annual AIS messages, followed by cargo ships at 16%. Fishing vessels and sailing ships each account for 12%, while passenger and tanker ships represent 10% and 8%, respectively. This data highlights the differences in the types of ships and their respective industries. This analysis demonstrates the varied nature of ship categories in terms of their operations. The study period shows the percentage distribution of each top category. Seasonal trends are evident, with the Passenger category being more prevalent in the summer months. Tanker ships exhibit a relatively consistent presence with slight peaks during midsummer and at the end of the year. Activity is generally reduced during the winter period, particularly in February, across all types. Subsequently, a signal throughput analysis of the messages is conducted to assess the behaviour of the AIS system and identify noteworthy aspects. The examined data is checked to verify whether two consecutive signals have a maximum latency of 10 minutes in an ideal system. It is important to note that the distance between signals is only measured if they differ by a maximum of one hour. This is because transmission systems can be deactivated and reactivated by ships at any time, such as during docking, which could invalidate the proposed analysis. Figure 2 presents a table that shows the percentage distribution of messages for the major types of ships related to our AIS messages, with a particular focus on certain months.

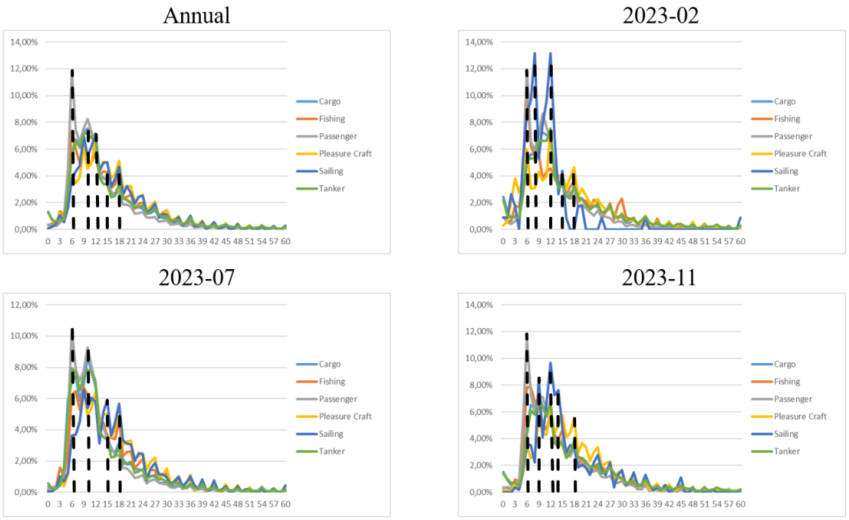


Fig. 2. Chart illustrating the percentage distribution of messages for major types

The annual distribution shows that message latency is primarily concentrated in the range of 6 to 12, with a significant bottleneck of messages taking over 20 minutes. This indicates that the system is generally stable. However, there may be occasional missing

messages due to various factors, such as antenna transmission issues or receiving system saturation [27].

3.3 Data Preparation

After collecting AIS data, we generated a dataset of ships voyages' AIS observations, which we used to train the Deep Learning (DL) models. A voyage is defined by the ship (i.e., by the ship's IMO), the departure port and date, and the arrival port and date, as illustrated in Figure 3.

IMO	VESSEL TYPE	DEPARTURE PORT	ARRIVAL PORT	DEPARTURE DATE	ARRIVAL DATE
8401561	Cargo	ITBRI (Bari)	ITRAN (Ravenna)	04/09/2023	05/09/2023
9483712	Passenger	ITGOA (Genova)	ILOLB (Olbia)	05/09/2023	06/09/2023

Fig.3. Dataset of ships' voyages

A departure and an arrival are two consecutive visit of the same ship. We will refer to a visit of a ship in a port as 'port call'. A port call in AIS data is a record where speed is zero and the position falls inside a port area. The 'Heuristic Algorithm' (HA) used to generate the dataset of voyages consists in the following steps:

1. Load ports dataset (i.e. dataset of geographical coordinates of each port).
2. Load Ships Register [26].
3. Select a time period.
4. Filter AIS data, by the time period and by the ships that have visited an Italian port at least once.
5. Join the filtered AIS data with the Ships Register, in order to get the gross tonnage and ship type attributes.
6. Furthermore, filter AIS data by ships exceeding 300 tonnes (the ones we are interested in).
7. Detect and manage significant outages in AIS data, as shown in the next Section 3.4.
8. Select port calls from AIS data, namely records where the speed is 0, to obtain a dataset of all the ships' voyages (see Fig.3).
9. Join this dataset with the AIS observations, collected at steps 4-6, to create a ships' voyages dataset. Each record of the dataset contains:
    a. AIS observations, chronologically ordered, from departure date and position, to arrival date and position of a voyage.
    b. A label (the 'class'), corresponding to the arrival port UNLocode.

The resulting voyages' dataset computed from the HA algorithm was used to train DL models. For this purpose, the dataset was divided into three distinct sets, i.e. training, validation and test set. Note that some voyages in the dataset may not have been labelled by the HA with the arrival port, due to the noise or lack of AIS signals, detected in step 7 of the HA. These voyages were not included in the training set. Once the training

stage was completed, we firstly adopted the DL models to validate the labelling generated by the HA. On the other hand, the trained DL models were used to classify (by the arrival port UNLocode) the routes that could not be labelled by the HA.

3.4 Detection of outages in AIS data

As we stated within the Introduction section, the objective of this research is also developing a Deep Learning algorithm capable of predicting vessel docking events in scenarios where AIS signals are unavailable for extended durations (signal outages). A critical preliminary task is the identification of such signal outages, which needs ad-hoc data preparation. We first sorted the data by the timestamps t_i of recorded vessel positions, this allowed the introduction of three key variables for our analysis:

- Δt_i which is the time interval between two successive timestamps at the i-th recorded location of the vessel, which helps in identifying periods of signal absence.
- Δs_i which is the distance between the consecutive locations of the vessels at timestamps t_i and t_{i+1} (computed as the geodesic between the two points on the earth's surface, this measurement employs the Haversine formula to approximate the earth's curvature).
- $\bar{v}_i = \frac{\Delta s_i}{\Delta t_i}$ which is the computed average velocity of a vessel during Δt_i.

We stressed the difference between the computed average speed $\bar{v}_i$ from the 'speed over ground' (sog) variable available in AIS data. The 'sog' represents the vessel's instantaneous speed at a specific timestamp, whereas our computed average speed offers insight into the vessel's pace over the entire interval Δt_i. In order to identify docking events during AIS signal outages, our algorithm implements a focused heuristic based on average speed computation. Initially, we isolate segments with extended time intervals (say $\Delta t_i > 60\,'$), indicative of signal absence, during which we are blind to the ship activity. Within these segments, we pinpoint 'suspect' intervals that could potentially include docking events of which we are not aware, thereby filtering out intervals where docking is highly unlikely. Under typical conditions, a vessel's movement during the outage interval Δt_i involves traversing the geodesic distance between two points, resulting in a standard average velocity for that segment. However, if a vessel docks during Δt_i, its actual journey will cover a significantly greater distance to dock and then return to the next recorded location. This means the straight-line distance Δs_i would be notably less than the actual navigated distance, leading to a ratio $\frac{\Delta s_i}{\Delta t_i}$ (average computed speed) lower than expected for normal travel. Our heuristic then flags a travel segment as a potential docking event if it meets two criteria:

1. The time interval between signals Δt_i exceeds a maximum threshold: $\Delta t_i > \Delta t_{Max}$
2. The average speed $\bar{v}_i$ is below a minimum threshold: $\bar{v}_i < v_{min}$

The above-mentioned parameters depend on how stringent one wants to be in searching for possible docking events during outages; we use $\Delta t_{Max} \simeq 60'$, while the minimum velocity v_{min} is chosen as a low percentile (around 25-th percentile) of the average velocity distribution, this is usually around $v_{min} \simeq 10\ knots$. It's worth mentioning

that a flagged travel segment is not necessarily (or likely) a possible docking event, on the other hand a non-flagged travel segment (ship travelling at usual average velocity) it's highly unlikely to hide a docking (since it physically would not have time to do so).

3.5 Pre-processing and Hyper-parameters

We trained each model for 50 epochs, observing that extending training led to overfitting. The dataset comprised 28,140 routes from April 2023, segmented into training, validation and test sets. Each set was labelled through Heuristic Algorithms and domain knowledge. The test set included 10% of the routes, randomly selected, while the validation set, extracted prior to training, constituted 20% of the remaining data. Both sets were processed in batches of 32. A fixed learning rate of 0.1 was maintained using a "reduce-on-plateau" strategy to avoid local minima. The best model from each epoch was retained for inference to assess metrics and perform final classifications, utilizing Stochastic Gradient Descent as the solver.

Determining the optimal number of neurons in a ML model isn't straightforward, lacking clear engineering principles. For MLP models, halving neuron numbers per stage can reduce input dimensionality, while CNNs advise doubling kernel counts. Lengthy routes with over 3000 AIS signals were found to degrade model accuracy due to high dimensionality, so they were truncated to the last 300 messages. The signals were normalized in the [0, 1] interval, and sequences were zero-padded for uniform input sizes. Adding "Departure Port" and "H3 encoding index 8" as covariates to the original quadruple (LAT, LON, SOG, COG) improved performance, resulting in each AIS signal comprising six variables, totaling 1,800 inputs per model. Hence, The MLP comprises two hidden layers with 900 and 450 neurons respectively, followed by a softmax output layer for the 116 ports in the dataset. The Conv1D network features four "lenet-like" Conv1D layers: the first two with 32 kernels and the last two with 64 kernels, all with a 3x3 receptive field. This network includes an additional MLP with a 256-neuron flatten layer and a softmax output layer. The LSTM model includes one layer with 128 cells, optimized for fixed input sequence windows to prevent overfitting from sliding windows and overlapping sequences due to limited data. Each model ends with a softmax layer corresponding to the port classes and uses the RELU activation function throughout. Initial experiments showed poor F1 scores due to missing departure ports in the dataset, leading to their exclusion. This adjustment ensured that only routes with defined departure and arrival ports were used, significantly improving F1 scores on the test set as detailed in the Deep Learning section.

3.6 Deep Learning

We conducted model selection to determine the best-performing model among MLP, Conv1D, and LSTM based on training and validation sets (Figure 4). The Conv1D Network showed the highest accuracy on the validation set, possibly due to its ability to detect spatial correlations. However, LSTM demonstrated superior generalization without overfitting, contrasting with MLP and Conv1D, which showed signs of overfitting in later epochs.

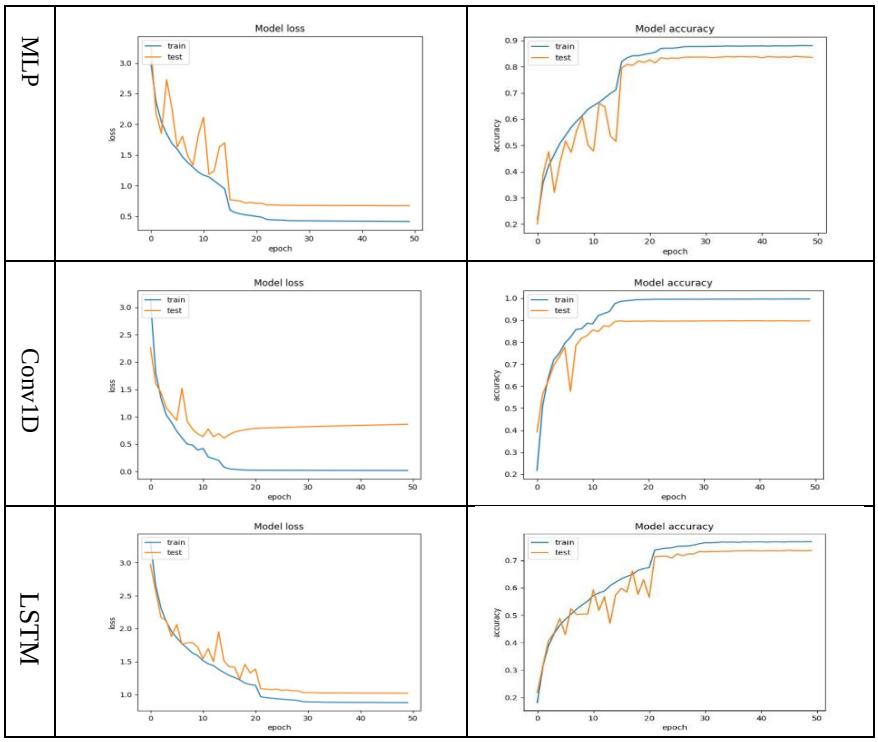


Fig. 4. Training Curves of Loss and Accuracy in MLP, Conv1D and LTSM models.

The overfitting issue is especially pronounced with Conv1D, evident by significant divergence in the curves around the 13th epoch. Despite this, we chose to preserve the best Conv1D model as a candidate for generating official maritime transport statistics. In contrast, the MLP lacks LSTM's advantage of no overfitting and doesn't match the performance of the best Conv1D model.

Due to a heavy imbalance in the dataset, accuracy alone isn't fit for evaluation; the F1-Score [28] is better suited, capturing model performance under strong imbalance. Our model selection, detailed in Table 1, considers metrics calculated on both training and test sets. Consequently, we proceeded with Conv1D for further analysis. Initially, with routes containing missing departure ports, the best model achieved an F1-Score of 0.51, but after trimming these routes, the metric improved to 0.669, deemed satisfactory for robust modeling before application to unlabeled routes with gaps. Regarding the per-class scores, the 25th percentile F1-Score is 0.50 and the 75th percentile F1-Score is 0.92, across 93 classes (ports) in the test set.

Table 1. Model performance

Model / Metric	Train Acc.	Train F1	Test Acc.	Test F1
MLP	0.872	0.629	0.825	0.543
Conv1D	0.976	0.901	0.893	0.669
LSTM	0.762	0.394	0.729	0.392

The hypothesis behind Conv1D's superiority is that in AIS data, spatial correlations (well captured by CNNs) may outweigh long-term temporal dependencies (well captured by LSTM). We counted the ports classified by both Heuristics and DL across all data (training and test sets).
Figure 5 illustrates the 10 highest positive and negative residuals comparing Heuristics and Deep Learning statistics.

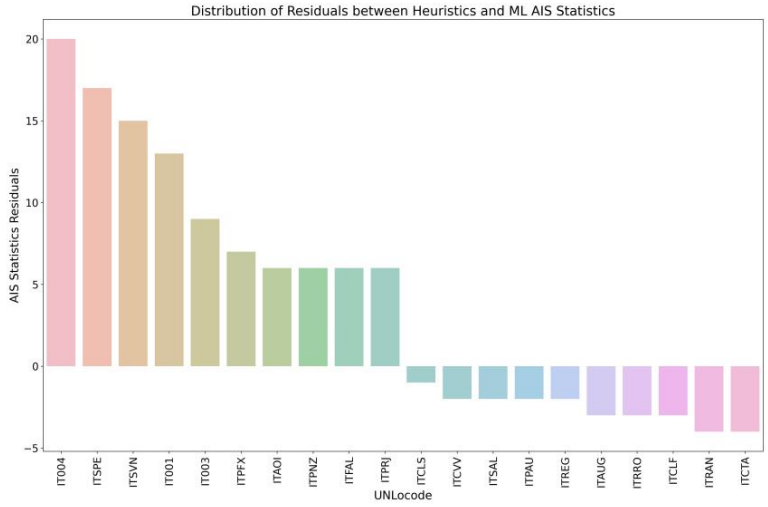


Fig. 5. Top 10 highest and lowest residuals between Heuristics and Deep Learning

Only two ports, ITGOA and ITTPS, out of 106, exhibit notable deviations between Heuristics and Deep Learning outcomes, while the majority display minimal residuals. This underscores the accuracy of the heuristic-based labeling procedure, as evidenced by the relatively high F1-Score achieved by Deep Learning, indicating consistency in the training set. Figure 6 illustrates a set of correctly labeled ports. Additionally, in certain instances, as depicted in Figure 7, Deep Learning seems to refine the heuristic-based labeling algorithm.

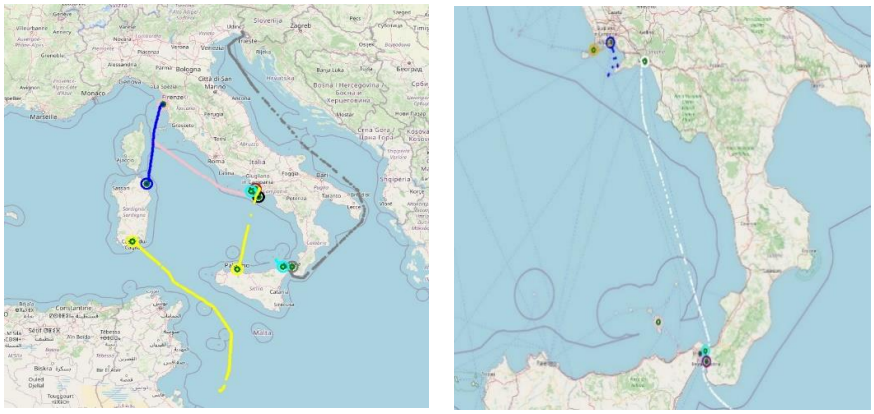


Fig. 6. True Port (internal green circle) and Predicted Port (external circle). The Predicted Port and the route have the same color.

It's hypothesized that Deep Learning models, while not perfect replicas of the Training Set, can provide alternative outcomes by leveraging historical route data. This enables them to correct label predictions using external information beyond the Training Set. Residual analysis highlights significant disparities in a few ports between heuristic and Deep Learning predictions, likely due to Deep Learning's corrections.

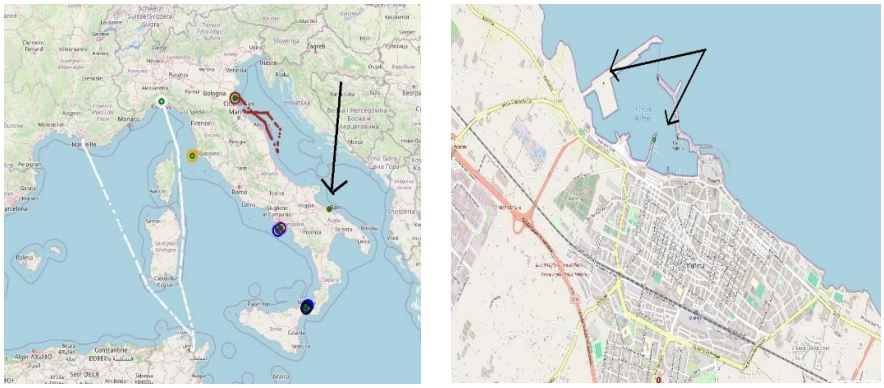


Fig. 7. Depiction of the corrective power of Deep Learning with regards to the Training Set.

Subsequently, we employed the pre-trained Deep Learning model to predict all unlabeled routes identified during docking events. Figure 8A shows the model's proficiency in labeling well-formed trajectories similar to those in the Training Set. However, Figures 8B and 8C demonstrate Deep Learning's key advantage in labeling trajectories with significant time gaps, including those originating from docking event

identifications. By leveraging trajectory history, Deep Learning can determine the most probable port of arrival, improving estimates made by heuristic procedures. This significantly enhances the accuracy of official statistics, especially in cases prone to detection errors, such as port arrivals affected by AIS signal gaps. Figure 8D illustrates a more extreme scenario, where the last AIS signals are far from the coast, suggesting a different port than initially estimated. Nonetheless, the Deep Learning model can discern directional changes, reducing ambiguity and improving accuracy, even in challenging situations.

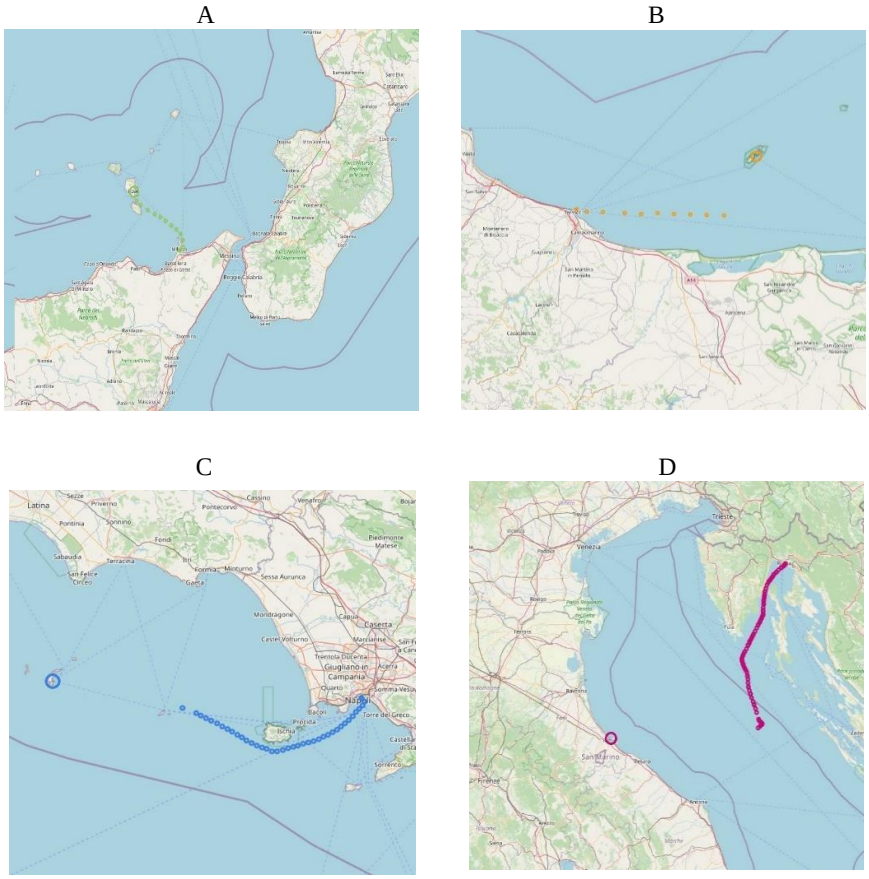


Fig. 8. Representation of all the routes with gaps that are correctly labelled by the best Deep Learning model.

4 Conclusions

This study highlights the impact of Artificial Intelligence (AI), specifically Deep Learning, on Official Statistical production. Deep Learning can analyze complex data and uncover patterns that traditional methods may miss. It can also classify segments of paths and trajectories that cannot be labeled using traditional methods. Deep Learning can assign a port to trajectories even with missing segments, improving Official Statistics estimation. It can also correct labels for well-formed routes. The study suggests training models over a full year to capture different situations and patterns. Future work will involve retraining the best model on labeled data from a full year and inferring labels for previous years. The latest data (from 2023) will be used as a labeling model for other years, since routes are not expected to change significantly. Additionally, introducing random artificial gaps in training data (while preserving port labels) can help determine if models can adapt to their presence. In the future, advanced deep learning models like Transformers might be tested for route labeling and reconstruction, as they have shown better performance in recent studies [29].

References

1. I. Varlamis, K. Tserpes, and C. Sardianos, “Detecting Search and Rescue Missions from AIS Data,” in 2018 IEEE 34th International Conference on Data Engineering Workshops (ICDEW), Apr. 2018, pp. 60–65, iSSN: 2473-3490.
2. T. Salzmann, B. Ivanovic, P. Chakravarty, and M. Pavone, “Trajectron++: Dynamical-ly-Feasible Trajectory Forecasting With Heterogeneous Data,” arXiv:2001.03093 [cs], Jan. 2021, arXiv: 2001.03093.
3. B. Murray and L. P. Perera, “An AIS-based deep learning framework for regional ship behavior prediction,” Reliability Engineering & System Safety, p. 107819, May 2021.
4. Ou, Ziqiang, and Jianjun Zhu. "AIS database powered by GIS technology for maritime safety and security." The Journal of Navigation 61.4 (2008): 655-665.
5. De Cubber, Geert, et al. "Distributed coverage optimisation for a fleet of unmanned maritime systems." ACTA IMEKO 10.3 (2021): 36-43.
6. Tu, Enmei, et al. "Exploiting AIS data for intelligent maritime navigation: A compre-hensive survey from data to methodology." IEEE Transactions on Intelligent Transpor-tation Systems 19.5 (2017): 1559-1582.
7. Mou, Jun Min, Cees Van der Tak, and Han Ligteringen. "Study on collision avoidance in busy waterways by using AIS data." Ocean Engineering 37.5-6 (2010): 483-490.
8. Soldi, Giovanni, et al. "Space-based global maritime surveillance. Part II: Artificial in-telligence and data fusion techniques." IEEE Aerospace and Electronic Systems Maga-zine 36.9 (2021): 30-42.
9. B. Ristic, B. La Scala, M. Morelande, and N. Gordon, “Statistical analysis of motion patterns in AIS Data: Anomaly detection and motion prediction,” in 2008 11th Interna-tional Conference on Information Fusion, Jun. 2008, pp. 1–7.
10. S. Capobianco, L. M. Millefiori, N. Forti, P. Braca, and P. Willett, “Deep Learning Methods for Vessel Trajectory Prediction based on Recurrent Neural Networks,” IEEE Transactions

on Aerospace and Electronic Systems, pp. 1–1, 2021, conference Name: IEEE Transactions on Aerospace and Electronic Systems.
11. A. Gupta, J. Johnson, L. Fei-Fei, S. Savarese, and A. Alahi, “Social GAN: Socially Acceptable Trajectories With Generative Adversarial Networks,” 2018, pp. 2255–2264.
12. H3 (Hexagonal hierarchical geospatial indexing system), https://h3geo.org/
13. C. Wang, H. Ren, and H. Li, “Vessel trajectory prediction based on AIS data and bidi-rectional GRU,” in 2020 International Conference on Computer Vision, Image and Deep Learning (CVIDL), Jul. 2020, pp. 260–264.
14. Næss, Patrick André. Investigation of multivariate freight rate prediction using ma-chine learning and ais data. MS thesis. NTNU, 2018.
15. Düz, Bülent, and Erwin van Iperen. "Ship trajectory prediction using encoder–decoder-based deep learning models." Journal of Location Based Services (2024): 1-21.
16. Gao, D. W., Zhu, Y. S., Zhang, J. F., He, Y. K., Yan, K., & Yan, B. R. (2021). A novel MP-LSTM method for ship trajectory prediction based on AIS data. Ocean Enginee-ring, 228, 108956.
17. Frank Rosenblatt. The perceptron: A theory of statistical separability in cognitive sys-tems. United States Department of Commerce, 1958.
18. Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computa-tion, 9(8), 1735-1780.
19. Olah, Christopher. "Understanding lstm networks." (2015).
20. Yoon Kim. Convolutional neural networks for sentence classification. arXiv preprint arXiv:1408.5882, 2014.
21. UN Global Platform, https://unstats.un.org/bigdata/un-global-platform.cshtml
22. White, Tom. Hadoop: The definitive guide. " O'Reilly Media, Inc.", 2012.
23. Salloum, Salman, et al. "Big data analytics on Apache Spark." International Journal of Data Science and Analytics 1 (2016): 145-164.
24. Shvachko, Konstantin, et al. "The hadoop distributed file system." 2010 IEEE 26th symposium on mass storage systems and technologies (MSST). Ieee, 2010.
25. Zaharia, Matei, et al. "Resilient distributed datasets: A Fault-Tolerant abstraction for In-Memory cluster computing." 9th USENIX symposium on networked systems design and implementation (NSDI 12). 2012.
26. Lloyd’s Register, https://www.lr.org
27. Lv, Taizhi, Peiyi Tang, and Juan Zhang. "A Real-Time AIS Data Cleaning and Indica-tor Analysis Algorithm Based on Stream Computing." Scientific Programming 2023 (2023).
28. Jeni, László A., Jeffrey F. Cohn, and Fernando De La Torre. "Facing imbalanced data--recommendations for the use of performance metrics." 2013 Humaine association con-ference on affective computing and intelligent interaction. IEEE, 2013.
29. Nguyen, Duong, and Ronan Fablet. "TrAISformer-A generative transformer for AIS trajectory prediction." arXiv preprint arXiv:2109.03958 (2021).

Exploitation Strategies in Conditional Markov Chain Search: A case study on the three-index assignment problem

Sahil Patel[1] and Daniel Karapetyan[1]

Computer Science, University of Nottingham, Nottingham NG8 1BB, UK,
sahil.j.patel@gmail.com, daniel.karapetyan@nottingham.ac.uk

Abstract. The Conditional Markov Chain Search (CMCS) is a framework for automated design of metaheuristics for discrete combinatorial optimisation problems. Given a set of algorithmic components such as hill climbers and mutations, CMCS decides in which order to apply those components. The decisions are dictated by the CMCS configuration that can be learnt offline. CMCS does not have an acceptance criterion; any moves are accepted by the framework. As a result, it is particularly good in exploration but is not as good at exploitation. In this study, we explore several extensions of the framework to improve its exploitation abilities. To perform a computational study, we applied the framework to the three-index assignment problem. The results of our experiments showed that a two-stage CMCS is indeed superior to a single-stage CMCS.

Keywords: Conditional Markov Chain Search (CMCS) · three-index assignment problem · axial index assignment problem · automated algorithm design · automated meta-heuristic design · combinatorial optimisation.

1 Introduction

The Conditional Markov Chain Search (CMCS) is a framework based on a highly-configurable meta-heuristic suitable for automated design of combinatorial optimisation algorithms. We will say that a *CMCS configuration* is a specific metaheuristic, as opposed to the generic CMCS framework. A CMCS configuration comprises a set of components such as hill climbers and mutations, and a control mechanism that decides in which order those components are applied. To build a CMCS configuration, the user needs to provide a component pool, an objective function and a set of test instances. Using this information, a *configurator* searches for a high-performance CMCS configuration.

A CMCS configuration is a single-point meta-heuristic based on multiple components treated as black boxes. Each component is a subroutine that takes a solution and modifies it according to the internal logic. Some components such as hill climbers may aim at improving solutions whereas other components such as mutations may perform random modifications; the framework generally treats

them the same. The control mechanism of a CMCS configuration is described by a set of numeric parameters thus enabling automated generation of CMCS configurations; by tuning these parameters, one can find the 'optimal' control mechanism [7].

A CMCS configuration performs as follows. It takes as an input the initial solution (usually produced by some construction heuristic, e.g. random solution) and then applies to it one of the components at each iteration. Any modification by any component is always 'accepted', i.e. there is no backtracking. CMCS records whether the component improved the solution or not. The choice of the next component depends only on which component was used in the current iteration and whether it improved the solution; thus the sequence of applied components is a Markov chain. The control mechanism can be defined by two transition matrices: one for the case when the solution was improved (M^{succ}) and another one for the case when the solution was not improved (M^{fail}). Transitions may be deterministic (if there is exactly one non-zero value in each row of each matrix) or probabilistic. Since CMCS may worsen the solution, it keeps track of the best solution found during the search and at the end returns that solution. The termination criterion is based on the user-defined time budget [7].

An important aspect in an optimisation heuristic is how it combines exploration and exploitation. By *exploration* we mean the ability to find diverse solutions and by *exploitation* we mean the ability to improve upon found solutions. If a heuristic does not do enough exploration, it converges prematurely. If it does not do enough exploitation, it misses good solutions even if it succeeds in finding the regions where they are. By configuring CMCS, one can achieve an optimal balance between exploration and exploitation, however this balance will mainly be static, as CMCS is trained offline and has limited online adaptiveness capabilities. In this paper, we study if adjusting the balance between exploration and exploitation dynamically could benefit the performance of CMCS configurations.

Specifically, we propose and study two extensions of CMCS that enable dynamic adjustment of the exploration/exploitation balance. Note that the original CMCS does not have any mechanisms that would allow adaptation based on the quality of the current solution. We explore two approaches to implement such adaptation: one triggered by finding a new best-found-so-far solution (such solutions, obviously, tend to be good), and another one based on the elapsed time, as the solution quality tends to improve throughout the solution process.

As a case study, we use the Three-Index Assignment Problem (AP3). AP3 is the three-dimensional extension of the job Assignment Problem. While the Assignment Problem is known to by polynomially solvable, AP3 is NP-hard.

Applications of AP3 include military troop assignment, minimising idle time in a rolling mill, scheduling capital investments, satellite coverage and cost optimisation [12], production of printed circuit boards [2], and scheduling a teaching practice [4].

The contributions of the paper are as follows. In all previous applications of CMCS [9,7,11], only one strategy of exploration and exploitation has been

applied. Hence, to address this gap, we propose two new strategies and compare all three strategies in a computational study. To do so, we also design a CMCS configurator and a pool of AP3 components.

The remainder of this paper is structured as follows. In Section 2, the different CMCS strategies are detailed, followed by the CMCS configurator (Section 3), discussion of AP3 and the component pool (Sections 4 and 5, respectively), experimental evaluation (Section 6) and conclusions and future work (Section 7).

2 CMCS Exploitation Strategies

As we mentioned earlier, in each iteration, CMCS selects one component and applies it to the current solution. The selection of the component depends on exactly two inputs: what was the previously applied component and whether it improved the solution. This means that the control mechanism cannot respond to the absolute quality of the current solution; it only 'knows' if the solution was improved or not in the last iteration. As a result, it cannot adjust its exploitation strength when a particularly good solution is found. In other words, when it finds a good solution, it may keep exploring without an attempt to improve this solution.

In this section, we describe three versions of CMCS that address the exploitation in different ways. Strategy A stands for the original CMCS, strategy B stands for a new extension that changes its behaviour when a new best-found-so-far solution is obtained, to intensify exploitation of particularly good solutions. Strategy C stands for a new extension that splits the time budget into two phases, where the first phase is expected to be focused on exploration while the second phase is focused on exploitation.

2.1 CMCS$^{\mathbf{A}}$: Strategy A

Strategy A is the core CMCS algorithm [7]. Its behaviour makes it very good at exploration of the solution space but it may not fully exploit good solutions before proceeding to further exploration.

The details of Strategy A are shown in Algorithm 1. It takes the following as the input:

- Ordered set $\mathcal{H}$ of components, called *component pool*. Each component is an algorithm that takes a solution and the instance data as parameters and returns an updated solution.
- Transition matrices M^{succ} and M^{fail} of size $|\mathcal{H}| \times |\mathcal{H}|$. Each row of each of M^{succ} and M^{fail} adds up to 1, as these are transition probabilities.
- Instance data $\mathcal{I}$. The instance data is only used for calculating the objective function.
- Objective function $f(S, \mathcal{I})$, where S is the solution and $\mathcal{I}$ is the instance data.
- Time budget T.

Algorithm 1: $\mathrm{CMCS}^{\mathrm{A}}(S,T)$: Strategy A

```
1  S* ← S_0; S ← S_0; f* ← f(S_0, I); f_prev ← f*; h ← 1;
2  while elapsed-time < T do
3      S ← H_h(S, I);
4      f_cur ← f(S, I);
5      if f_cur < f_prev then
6          h ← RouletteWheel(M^succ_{h,1}, M^succ_{h,2}, ..., M^succ_{h,|H|});
7          if f_cur < f* then
8              S* ← S;
9              f* ← f_cur;
10     else
11         h ← RouletteWheel(M^fail_{h,1}, M^fail_{h,2}, ..., M^fail_{h,|H|});
12     f_prev ← f_cur;
13 return S*;
```

The output of the algorithm is the best solution found during the search.

Here $RouletteWheel(p_1, p_2, \ldots, p_n)$ is a function that returns integer i between 1 and n with probability p_i.

2.2 CMCS$^{\mathrm{B}}$: Strategy B

Strategy B is an extension of Strategy A. In order to actively exploit good solutions, it incorporates the variable neighbourhood descent (VND) algorithm (a deterministic local search heuristic that explores a number of neighbourhood structures [3]) that consists of a subset of the hill climbers from the component pool. Given an ordered set of hill-climbers $\{\mathcal{HC}_1, \mathcal{HC}_2, \ldots, \mathcal{HC}_n\}$, VND applies $\mathcal{HC}_1$ to the solution as long as it improves the solution. If $\mathcal{HC}_1$ fails to improve the solution, $\mathcal{HC}_2$ is applied. If $\mathcal{HC}_2$ improves the solution, VND gets back to $\mathcal{HC}_1$; otherwise it proceeds to $\mathcal{HC}_3$, etc. If all the hill climbers fail to improve the solution, VND terminates. For details, see Algorithm 3.

In essence, Strategy B sacrifices some time to apply the VND algorithm to the best-found-so-far solution each time the algorithm finds a new best-found-so-far solution. As a result, best-found-so-far solutions are guaranteed to be local minima with respect to the hill climbers included in the VND.

Typically, a metaheuristic improves the best-found-so-far solutions particularly frequently at the beginning of its run. Applying VND each time is unnecessary; it is unlikely that a solution found at the early stages of the search will be better than solutions found later. Thus, we do not apply VND until $0.5T$. For details, see Algorithm 2.

2.3 CMCS$^{\mathrm{C}}$: Strategy C

Strategy C splits the time budget into two phases, where the first phase is intended mainly for exploration whereas the second phase is intended mainly for

Algorithm 2: CMCS$^{\mathrm{B}}$: Strategy B

```
1  S* ← S_0; S ← S_0; f* ← f(S_0, I); f_prev ← f*; h ← 1;
2  f*_polished ← ∞; f*_best ← ∞;
3  vnd-applied ← 0;
4  while elapsed-time < T do
5      S ← H_h(S, I);
6      f_cur ← f(S, I);
7      if f_cur < f_prev then
8          h ← RouletteWheel(M^succ_{h,1}, M^succ_{h,2}, ..., M^succ_{h,|H|});
9          if f_cur < f* then
10             S* ← S;
11             f* ← f_cur;
12             vnd-applied ← 0;
13     else
14         h ← RouletteWheel(M^fail_{h,1}, M^fail_{h,2}, ..., M^fail_{h,|H|});
15     if elapsed-time ≥ 0.5T and vnd-applied = 0 then
16         S*_polished ← VND(S*);
17         f*_polished ← f(S*_polished);
18         if f*_polished < f*_best then
19             S*_best ← S*_polished;
20         vnd-applied ← 1;
21     f_prev ← f_cur;
22 return S*_best;
```

exploitation. Both phases are implemented as CMCS$^{\mathrm{A}}$ although the configurations can be different; we call them 'sub-configuration 1' and 'sub-configuration 2', respectively. For details see Algorithm 4.

Please note that the roles of sub-configuration 1 and 2 are arbitrary; it is up to the configurator to choose what sub-configurations work best within CMCS$^{\mathrm{C}}$. It is our expectation that sub-configuration 1 is more likely to prioritise exploration whereas sub-configuration 2 is more likely to prioritise exploitation.

3 CMCS Configurator

This section details the CMCS configurator, i.e. the algorithm that produces a CMCS configuration given a training set of problem instances and a component pool.

Each variation of CMCS requires a slightly different configurator but the core algorithm is the same. The configurator for CMCS$^{\mathrm{A}}$ performs as follows. It takes as input a time budget T of a single CMCS run, a training set of problem instances and a component pool. It also takes the number of components that should be included in the configuration. It then generates all 'meaningful' subsets

Algorithm 3: Variable Neighbourhood Descent: VND(S^*)

```
1  i ← 1;
2  while elapsed-time < T do
3      S_new ← HC_i(S*, I);
4      f_new ← f(S_new, I);
5      if f_new < f* then
6          S* ← S_new;
7          f* ← f_new;
8          i ← 1;
9      else
10         if i = n then
11             return S*;
12         i ← i + 1;
13 return S*;
```

Algorithm 4: CMCS$^C(S, T)$: Strategy C

```
1  S* ← CMCS^A(S_0, 0.8T) executed with configuration 1;
2  S* ← CMCS^A(S*, 0.2T) executed with configuration 2;
3  return S*;
```

of components of the given size. A subset is meaningful if it includes at least one hill climber and at least one mutation.

Then, for each meaningful subset of components, the configurator searches for a good combination of the transition matrices. For that, it employs a simple population-based algorithm that performs as follows:

1. Produce a population of 50 configurations with the given set of components and random deterministic transition matrices. A transition matrix is deterministic if every row contains only one non-zero element.
2. Choose the best configuration in the population. If it is better than the best-observed-so-far configuration observed so far, update the best-observed-so-far configuration.
3. Produce a new population: 25 'children' of the best configuration in the previous population and 25 'children' of the best-observed-so-far configuration. Each child configuration is obtained from the parent configuration by applying a mutation. The configuration mutations are described in Section 3.1.
4. Repeat steps 2–4 for as long as the training time budget allows.
5. Return the best-observed-so-far configuration.

Each evaluation of a configuration consists of running CMCS with the given configuration on a set of benchmark instances. The average objective value is used as the measure of the configuration quality.

Once optimised configurations are produced for each meaningful subset of components, each of them is tested on an evaluation benchmark set; the one with the highest score is returned.

To configure CMCS$^{\mathrm{B}}$, we employ exactly the same algorithm as above. The VND is predetermined by the user so only the set of components and the transition matrices need to be optimised.

A configuration of CMCS^{C} consists of two independent CMCS sub-configurations. Hence, we need to choose two meaningful subsets of components: one for sub-configuration 1 and another one for sub-configuration 2. For each combination, we then optimise the two pairs of transition matrices. Specifically, we run the above algorithm to optimise the transition matrices for sub-configuration 1 with sub-configuration 2 disabled. Then we fix the matrices in sub-configuration 1 and optimise sub-configuration 2 matrices. Then we fix sub-configuration 2 matrices and optimise again sub-configuration 1 matrices. Once optimised matrices are produced for each pair of meaningful subsets of components, each of them is tested on an evaluation benchmark set and the one with the highest score is returned.

3.1 Configuration mutations

To produce a 'child' configuration from a 'parent' configuration, the algorithm selects two mutation operators and then applies the first one to M^{succ} and the second one to M^{fail}. The mutation operators are chosen uniformly at random from a set of available operators, and the choices for the two matrices are completely independent.

Below we describe the configuration matrix mutation operators.

Swap Rows. This mutation uniformly at random chooses two rows of the matrix and swaps them.

Shuffle Row. This mutation uniformly at random chooses a row in the matrix and a number n, $0 \leq n \leq |\mathcal{H}|$. It proceeds by making n swaps between randomly chosen elements in the chosen row.

Minimum change. This mutation uniformly at random chooses a row and two elements in that row. It then increments the value of the first element and decrements the value of the second element. If the values cannot be increased/decreased, no changes are made.

Ruin and Recreate. This mutation produces a new deterministic random matrix. Specifically, it assigns 1 to a randomly selected element in each row and 0 to the rest of the elements.

Void. This mutation does not make any changes to the matrix. It was introduced to allow changes that affect only one matrix.

In this research, we discretised the elements of the transition matrices. Each element could only take values $0, 1/|\mathcal{H}|, 2/|\mathcal{H}|, \ldots, 1$. Each row needs to add up to 1. This restriction is upheld by the random generation and optimisation algorithms developed for these matrices. Due to time constraints, the use of packages for parameter optimisation such as irace [10] could not be explored, and thus no evaluation of the effectiveness of the developed matrix optimisation algorithm has been done.

4 Case study: The three-index assignment problem (AP3)

The three-index assignment problem (AP3) was first mentioned in [12]. It is an NP-hard combinatorial optimisation problem with many applications.

It is described in [5] as follows: "Given $n \in \mathbb{N}$, three sets $I = J = K = \{1, \ldots, n\}$, and associated costs $c(i, j, k) \in \mathbb{R}$ for all ordered triples $(i, j, k) \in I \times J \times K$. A feasible solution for the AP3 is a set of n triples, such that each pair of triples (i_1, j_1, k_1), (i_2, j_2, k_2) has different entries in every component, i.e., $i_1 \neq i_2$, $j_1 \neq j_2$, and $k_1 \neq k_2$. The aim of the AP3 is to find a feasible solution S with minimal costs $\sum_{(i,j,k)\in S} c(i, j, k)$."

We used three families of AP3 instances for training, validation and testing of each CMCS strategy, all sourced from [8]:

- **Random instances:** each cost $c(i, j, k)$ is assigned a uniformly distributed random weight from $\{1, 2, \ldots, 100\}$.
- **Clique instances:** at first, produce a complete tri-partite graph with partites I, J and K and random edge weights in $\{1, 2, \ldots, 100\}$. Then $c(i, j, k)$ is the sum of the weights of the edges (i, j), (j, k) and (i, k).
- **Square Root instances:** at first, produce a complete tri-partite graph with partites I, J and K and random edge weights in $\{1, 2, \ldots, 100\}$. Then $c(i, j, k)$ is the square root of the sum of squares of the weights of the edges (i, j), (j, k) and (i, k).

The training dataset consisted of 12 newly generated instances: 4 instances of size 40 of each type. All 12 runs were executed in parallel to utilise 12 logical cores of the machine. The validation dataset consisted of 90 newly generated instances: 10 instances of each type and size (40, 70 or 100).

The test dataset was used to compare configured algorithms and is the same as that of [8] and can be found at `http://www.cs.nott.ac.uk/~pszdk/?page=publications&key=Karapetyan2011b`. It includes 10 instances of each size (40, 70, and 100) and instance type (Random, Clique, and Square Root).

4.1 Data structures

This section details the data structures used to hold the information of the problem instances and solutions.

The cost function $c(i, j, k)$ is represented by a three-dimensional integer array of size $n \times n \times n$.

A solution is represented by an array of size n of tuples of size 2; the index of the tuple represents i, whereas the values in the tuple are j and k. At all times, the solution is maintained feasible, i.e. $j_1 \neq j_2$ and $k_1 \neq k_2$ for every pair of tuples $(j_1, k_1), (j_2, k_2)$.

While we store solutions as arrays of 2-tuples, we will represent them as sets of 3-tuples in the remainder of the paper.

5 AP3 components

This section describes the components that make up the component pool used as input to the CMCS configurator. These components are algorithms that manipulate an AP3 solution. Some of them were sourced from the AP3 and CMCS literature, while others are newly designed.

5.1 Neighbourhoods

To describe the AP3 mutations and hill climbers, it is convenient to first introduce a few neighbourhoods.

Swap: the neighbourhood includes all the solutions that can be obtained from S by selecting $(i_1, j_1, k_1) \neq (i_2, j_2, k_2) \in S$ and swapping two values: i_1 and i_2, or j_1 and j_2, or k_1 and k_2.

Shuffle: the neighbourhood includes all the solutions that can be obtained from S by selecting $(i_1, j_1, k_1) \neq (i_2, j_2, k_2) \neq (i_3, j_3, k_3) \in S$ and shuffling i_1, i_2 and i_3, or j_1, j_2 and j_3, or k_1, k_2 and k_3.

Hungarian: the neighbourhood includes all the solutions that can be obtained by applying a permutation to the values in one of the three dimensions. For example, given a solution (i_1, j_1, k_1), (i_2, j_2, k_2), ..., (i_n, j_n, k_n), a permutation π applied to the first dimension will produce a solution $(\pi(i_1), j_1, k_1)$, $(\pi(i_2), j_2, k_2)$, ..., $(\pi(i_n), j_n, k_n)$.

5.2 Mutations

Here we discuss AP3 mutations, i.e., components that make random changes to the solution, usually applied to escape a local minimum by worsening its quality.

Random Swap: randomly selects a solution from the Swap neighbourhood [9].

Shuffle Three: new mutation that randomly selects a solution from the Shuffle neighbourhood.

Worst Swap: new mutation that selects the worst solution from the Swap neighbourhood, or returns the original solution if the current solution is a worst one in the Swap neighbourhood.

First Worsen: is a new mutation that is similar to Worst Swap except that it accepts the first worsening move.

Observe that Random Swap and Shuffle Three mutations are standard stochastic mutations whereas Worst Swap and First Worsen systematically explore a neighbourhood and choose the worst or first worsening move, respectively.

5.3 Hill Climbers

The following components are AP3 hill climbers, i.e., components that attempt to improve the solution.

First Swap: is a new hill climber that explores the Swap neighbourhood and returns the first improving solution, or returns the original solution if no better solution is found.

Best Swap: explores the entire Swap neighbourhood and returns the best solution in it [9].

Hungarian(d): explores the Hungarian neighbourhood for dimension d and returns the best solution in it. Note that the exploration can be reduced to the Assignment Problem and solved in polynomial time by the Hungarian method [6]. Also note that the size of the neighbourhood is exponential. Thus, the Hungarian(d) hill climber is a so-called very large neighbourhood search method. We used the implementation of the Hungarian method from [1].

Min-Dimension Hungarian: explores the Hungarian neighbourhood for all three dimensions and returns the best solution found [6].

All-Dimension Hungarian: applies Hungarian(1), then Hungarian(2), then Hungarian(3), and then repeats this process until no further improvements can be found [6].

Random Dimension Hungarian: a new hill climber that chooses the value of $d \in \{1, 2, 3\}$ randomly and applies Hungarian(d).

Table 1: Generated configurations.

Strategy	$\|\mathcal{H}\|$	# component sets	Generation time, min
A	2	12	48
A	3	54	216
B	2	12	48
B	3	54	216
C	2	132	792
C	3	2 862	17 172

6 Experimental Evaluation

In this section, we compare the three exploitation strategies on our case study problem, AP3. Note that our aim is not to obtain a state-of-the-art algorithm for a specific optimisation problem; indeed, the performance of CMCS greatly depends on the component library. Our aim is to test whether some variation of CMCS has the potential to deliver better performance compared to the standard CMCS.

The algorithms for this study were implemented in Java and executed on Acer Predator Triton (6 cores and 12 logical processors) with an Intel Core i7-10750H 2.6GHz processor with 16 GB RAM under Windows 11. At most six experiments were run in parallel during evaluation (one per physical core). CMCS and components, however, did not use concurrency.

6.1 CMCS generation parameters

The time budget allocated for solving an instance while training was set to 1000 ms. According to our experiments, this time was sufficient to model the long-term behaviour of the CMCS configurations. The time allocated for optimising the transition matrices of each CMCS configuration was 4 minutes for Strategies A and B and 2 minutes for each training stage of Strategy C.

Table 1 shows the list of configurations that we generated in this study. One can see that the generation of Stategy C configurations is significantly more expensive compared to Strategies A and B. Also, the number of configuration components has a great effect on the generation time. Nevertheless, the generation of a configuration in most of the experiments took no more than a few hours which is well within the time budget for most of the applications and significantly faster than the standard human-based design process.

6.2 Evaluation results

The performance of the obtained configurations is compared in Figure 1. The solid lines correspond to the 2-component configurations whereas the dashed lines correspond to the 3-component configurations.

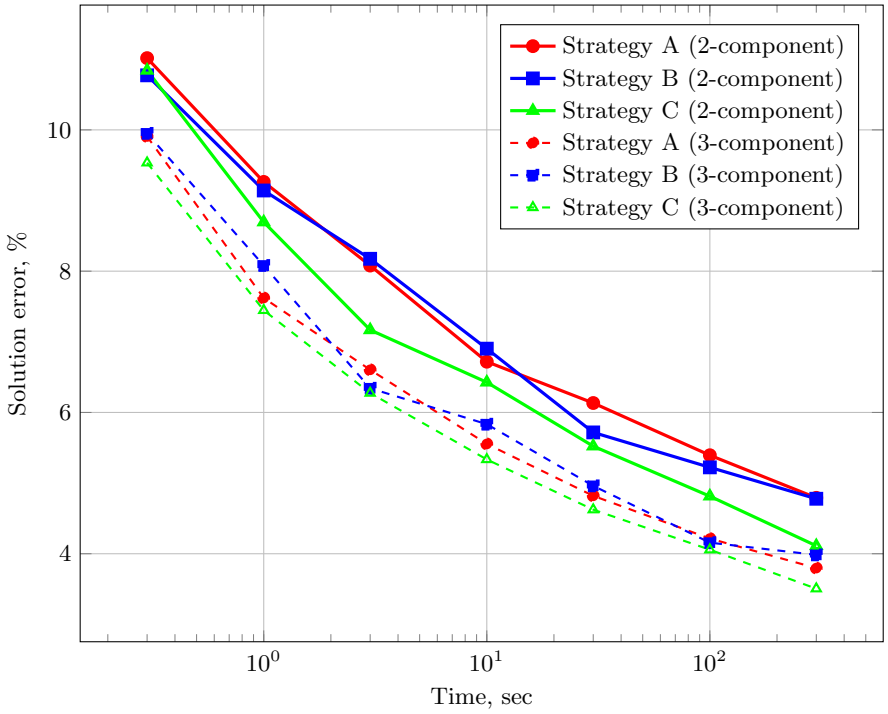


Fig. 1: Evaluation of the generated configurations. The graph shows how the solution error changes throughout the run of CMCS. The solution error is averaged across all the instances in the test set.

Among the two-component configurations, Strategy C is a clear winner for every time budget. Strategy B performs similarly to Strategy A.

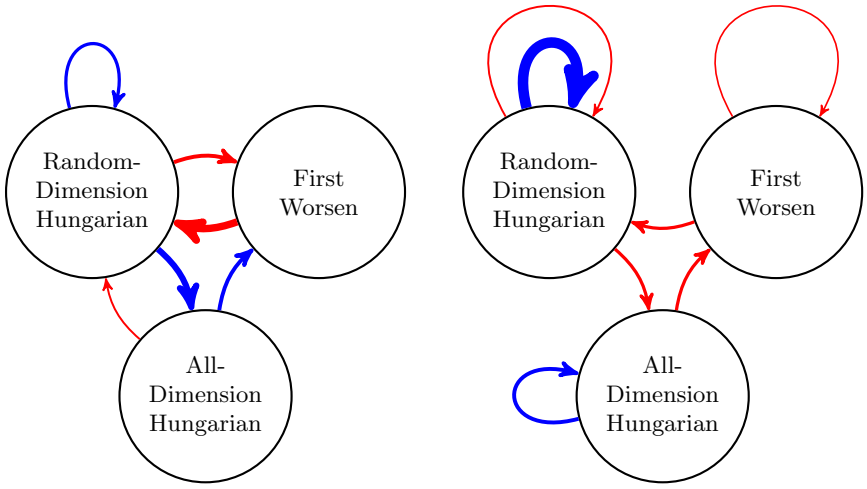
As expected, moving from 2-component to 3-component configuration improved the solution quality for each strategy. Otherwise, the results are similar for the 3-component configurations: Strategy C is a clear winner while Strategies A and B perform similarly.

The similarity in performance of Strategies A and B most likely indicates a bad choice of the VND stage. Indeed, our configuration method does not optimise the selection and the sequence of components in VND. Possibly, the algorithm spends too much or too little time on the VND phase.

Nevertheless, the success of Strategy C proves that CMCS benefits from the addition of an exploitation stage. This is despite the fact that the generation time for each sub-configuration in Strategy C was twice smaller than the training time of configurations in Strategies A and B.

Figure 2 shows the 3-component Strategy C configuration generated by our method. Both sub-configurations have the same component subsets but the transitions are different: sub-configuration 1 uses the mutation more often, which

matches our expectations that it would prioritise exploration. However, there are a few signs that the configuration generation process is not sufficiently robust. For example, sub-configuration 1 includes a transition from All-Dimension Hungarian to Random-Dimension Hungarian even though All-Dimension Hungarian is guaranteed to find a local minimum in the Hungarian neighbourhood, hence making a subsequent application of the Random-Dimension Hungarian hill-climber useless. A better configuration optimisation process would eliminate such a transition.



(a) Sub-configuration 1 (b) Sub-configuration 2

Fig. 2: Generated 3-component Strategy C configuration (the best configuration found in this study). The blue arcs correspond to transitions following improvement of the solution, while the red arcs correspond to the transitions following the solution not being improved). The thickness of each arc is proportionate to the frequency of the corresponding transition.

7 Conclusions

In this paper, we studied several approaches to intensify exploitation in CMCS. Specifically, we designed two new CMCS strategies (Strategies B and C) and compared them to the original strategy (Strategy A). Our experiments confirm that CMCS benefits from an exploitation mechanism; Strategy C has clearly outperformed Strategy A. (Strategy B demonstrated performance similar to that of Strategy A – possibly indicating some ineffective design choices.) On the other hand, generation of a Strategy C configuration is computationally expensive compared to Strategies A and B.

In future, we would like to improve the generation algorithms to study the performance of larger Strategy C configurations and compete with the more powerful AP3 metaheuristics from the literature. We would also like to test our new CMCS strategies on other combinatorial optimisation problems to confirm our findings and assess the effects of various parameters introduced in Strategies B and C: the ordering of components within VND in Strategy B, the time allocated to sub-configuration 1 in Strategy C, etc.

References

1. Bhojasia, M.: Java program to implement the Hungarian algorithm for bipartite matching (2022), `https://www.sanfoundry.com/java-program-implement-hungarian-algorithm-bipartite-matching`
2. Crama, Y., Spieksma, F.C.: Approximation algorithms for three-dimensional assignment problems with triangle inequalities. European Journal of Operational Research **60**(3), 273–279 (1992)
3. Duarte, A., Sánchez-Oro, J., Mladenović, N., Todosijević, R.: Variable neighborhood descent. Handbook of Heuristics p. 341–367 (2018)
4. Frieze, A.M., Yadegar, J.: An algorithm for solving 3-dimensional assignment problems with application to scheduling a teaching practice. Journal of the Operational Research Society **32**(11), 989–995 (1981)
5. Fügenschuh, A., Höfler, B.: Parametrized GRASP heuristics for three-index assignment. Evolutionary Computation in Combinatorial Optimization p. 61–72 (2006)
6. Gabrovšek, B., Novak, T., Povh, J., Rupnik Poklukar, D., Žerovnik, J.: Multiple hungarian method for k-assignment problem. Mathematics p. 2050 (2020)
7. Karapetyan, D., Goldengorin, B.: Conditional markov chain search for the simple plant location problem improves upper bounds on twelve Körkel-Ghosh instances. Optimization Problems in Graph Theory p. 123–147 (2018)
8. Karapetyan, D., Gutin, G.: A new approach to population sizing for memetic algorithms: A case study for the multidimensional assignment problem. Evolutionary Computation **19**(3), 345–371 (2011)
9. Karapetyan, D., Punnen, A.P., Parkes, A.J.: Markov chain methods for the bipartite boolean quadratic programming problem. European Journal of Operational Research **260**(2), 494–506 (2017)
10. López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., Stützle, T.: The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives **3**, 43–58 (2016)
11. Nalivajevs, O., Karapetyan, D.: Conditional Markov Chain Search for the generalised travelling salesman problem for warehouse order picking. 2019 11th Computer Science and Electronic Engineering (CEEC) (2019)
12. Pierskalla, W.P.: Letter to the editor—the multidimensional assignment problem. Operations Research **16**(2), 422–431 (1968)

Multi-objective Stochastic Optimization with AI Predictions on Management of Battery Energy Storage Systems

Behzad Pirouz and Francesca Guerriero

Department of Mechanical, Energy and Management Engineering, University of Calabria, Rende, Italy,
behzad.pirouz@unical.it & francesca.guerriero@unical.it

Abstract. Stochastic multi-objective optimization problems have broad applications due to various uncertainties in real-world problems. Despite the significant opportunities renewable energies create, the high uncertainty of wind speed and solar radiation is challenging for energy production units. These uncertainty effects can be investigated through stochastic multi-objective optimization programming [(see [1] and [2])]. On the other hand, due to the uncertainty in the output power of systems based on renewable energy, energy storage capacity is needed to increase the reliability of the service. Multi-objective stochastic optimization models have been proposed to reduce this uncertainty and storage capacity and determine the optimal timing of charging and discharging [(see [3] and [4])]. Artificial Intelligence (AI) techniques can significantly increase the accuracy of energy production forecasting using historical data, especially by training related data such as energy production data, weather patterns, and other relevant variables. Also, these techniques can predict sudden fluctuations in energy consumption demand caused by unforeseen conditions or events [5]. This study discusses a multi-objective stochastic optimization model for managing battery energy storage systems that can handle the uncertainties of renewable energies. We have also used AI techniques to predict data related to energy consumption and energy production from renewable resources in our model.

Keywords: Multi-objective Stochastic Optimization · Renewable Energy · Artificial Intelligence · Prediction Techniques · Battery Energy Storage.

Acknowledge: We acknowledge financial support from: PNRR MUR project PE0000013-FAIR.

References

1. Shaterabadi, M. and Jirdehi, M.A. : Multi-objective stochastic programming energy management for integrated INVELOX turbines in microgrids: A new type of turbines. Renewable Energy, 145, pp.2754-2769. (2020). https://doi.org/10.1016/j.renene.2019.08.002

2. Abdulnasser, G., Ali, A., Shaaban, M.F. and Mohamed, E.E. : Stochastic multi-objectives optimal scheduling of energy hubs with responsive demands in smart microgrids. Journal of Energy Storage, 55, p.105536. (2022). https://doi.org/10.1016/j.est.2022.105536
3. Lasemi, M.A., Arabkoohsar, A. and Hajizadeh, A. : Stochastic multi-objective scheduling of a wind farm integrated with high-temperature heat and power storage in energy market. International Journal of Electrical Power & Energy Systems, 132, p.107194. (2021). https://doi.org/10.1016/j.ijepes.2021.107194
4. Afzali, P., Rashidinejad, M., Abdollahi, A., Salehizadeh, M.R. and Farahmand, H. : A stochastic multi-objective model for energy efficiency and renewable resource planning in energy communities: A sustainably cost-effective trade-off. IET Renewable Power Generation, 17(5), pp.1078-1091. (2023). https://doi.org/10.1049/rpg2.12662
5. Rojek, I., Mikołajewski, D., Mroziński, A. and Macko, M. : Machine Learning-and Artificial Intelligence-Derived Prediction for Home Smart Energy Systems with PV Installation and Battery Energy Storage. Energies, 16(18), p.6613. (2023).

Efficient Line Search Method Based on Regression and Uncertainty Quantification

Tomislav Prusina and Sören Laue

Universität Hamburg, Germany
{tomislav.prusina, soeren.laue}@uni-hamburg.de

Abstract. Unconstrained optimization problems are typically solved using iterative methods, which often depend on line search techniques to determine optimal step lengths in each iteration. This paper introduces a novel line search approach. Traditional line search methods, aimed at determining optimal step lengths, often discard valuable data from the search process and focus on refining step length intervals. This paper proposes a more efficient method using Bayesian optimization, which utilizes all available data points, i.e., function values and gradients, to guide the search towards a potential global minimum. This new approach more effectively explores the search space, leading to better solution quality. It is also easy to implement and integrate into existing frameworks. Tested on the challenging CUTEst test set, it demonstrates superior performance compared to existing state-of-the-art methods, solving more problems to optimality with equivalent resource usage.

Keywords: Nonlinear Optimization · Line Search · Regression · Uncertainty Quantification · Bayesian Optimization.

1 Introduction

Optimization plays a central role in solving many real-world problems, from engineering and operations research to training machine learning models [3,4,12]. The selection of parameters and strategies significantly influences the effectiveness of algorithms designed to tackle such optimization problems. A crucial aspect of these algorithms, which are often iterative methods, is the identification of appropriate step lengths, a process known as line search, which is fundamental to the efficiency of the optimization algorithm.

Over the years, many line search methods have been proposed, each to ensure a step length that facilitates considerable progress in each iteration. Often, these methods try to satisfy the strong Wolfe conditions, ensuring a sufficient decrease in the function value and a reduction in the directional derivative. This ensures that the algorithm converges to a point close to a local minimum of the function along the search ray.

Traditionally, line search methods achieve this by maintaining and iteratively refining an interval with upper and lower bounds for the step length. This

'zooming-in' process is designed to pinpoint a suitable step length that guarantees sufficient progress. However, in this process, only the upper and lower bounds are kept and updated, and the information, i.e., function value and gradient information from all other query points visited during this line search, is discarded. Also, other regions of the line search interval might become more interesting, but they are never considered.

This paper addresses this shortfall. We propose a novel approach based on Bayesian optimization that harnesses all available information – the function value and gradient at all points queried so far – to identify a potential global minimum of the function along the line search ray. Bayesian optimization provides a principled framework for efficiently exploring the search space by constructing a probabilistic model of the objective function. By iteratively refining this model based on observed function values, Bayesian optimization guides the search toward promising regions, thus facilitating faster convergence and improved solution quality.

Our approach is simple, lightweight, and can be seamlessly implemented with a few lines of Python code, making it accessible and easily integrated into existing frameworks. We test our approach on the CUTEst test set [5] that contains a number of challenging unconstrained optimization problems. On these problems, our method outperforms other state-of-the-art approaches. Specifically, it enables a solver to solve more problems to optimality, while using an equivalent number of function evaluations and iterations.

2 Related Work

Many line search methods have been proposed. The seminal work by Armijo [1] introduced the Armijo rule, one of the first inexact line search methods. Building on this, Wolfe [16] formulated the weak and strong Wolfe conditions, crucial for balancing computational efficiency with sufficient descent and curvature criteria. Barzilai and Borwein [2] proposed a line search approach using gradient information, which proved effective in large-scale optimization. The work of Moré and Thuente [10]presented a line search method based on cubic interpolation, improving accuracy and efficiency in various applications. It is used in the quasi-Newton solver L-BFGS-B [18] which is also part of the Scipy library [13]. Additionally, Hager and Zhang [6] introduced an approach to refine step size estimation, enhancing performance in nonlinear optimization problems. Wächter and Biegler [14] introduced a filter line search strategy for primal-dual interior-point methods for large-scale nonlinear problems where traditional line search methods may struggle. It is part of the Ipopt solver [15], which is especially efficient in handling complex and large-scale optimization problems. All these line search methods have in common that they narrow down an interval containing a good step length that ensures adequate progress. Only the upper and lower bounds of this interval are maintained and updated, while the data, specifically function value and gradient information from other points assessed in the line

search, is disregarded. Additionally, other potentially relevant areas within the line search interval are not explored.

On the other hand, Bayesian optimization has emerged as a prominent method in global optimization, particularly for optimizing expensive-to-evaluate functions. The pioneering work by Močkus [9] laid the foundation for this technique in the one-dimensional case, focusing on efficiency in the face of limited evaluation opportunities. The integration of Gaussian processes in Bayesian optimization, as explored by Rasmussen and Williams [11], provided a robust framework for modeling the objective function and quantifying uncertainty. Usually, Bayesian optimization models use only function values and not gradient information. An approach that also integrated gradient information into Gaussian processes in the context of Bayesian optimization is described in [17]. However, their algorithm is rather involved, and no code is provided. All these works did not consider line search as an application for Bayesian optimization. Here, we present a line search approach that uses Bayesian optimization and that is easy to implement.

3 Line Search Method

Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be a continuously differentiable function and bounded from below. Suppose we want to solve the unconstrained optimization problem

$$\min f(x) \tag{1}$$

Iterative methods usually proceed as follows. Given a starting point $x_0 \in \mathbb{R}^n$, in each iteration the current iterate x_k is updated by

$$x_{k+1} = x_k + \alpha_k p_k,$$

where α_k is the step length and p_k is a descent direction, i.e., a direction in which the function f decreases. Mathematically, it can expressed as $p_k^\top \nabla f(x_k) < 0$. An (approximate) line search method tries to find a good estimate of the step length parameter α_k such that sufficient progress is made in the current iteration. In our approach, we use the strong Wolfe conditions to ensure sufficient progress, i.e., the final α_k needs to satisfy

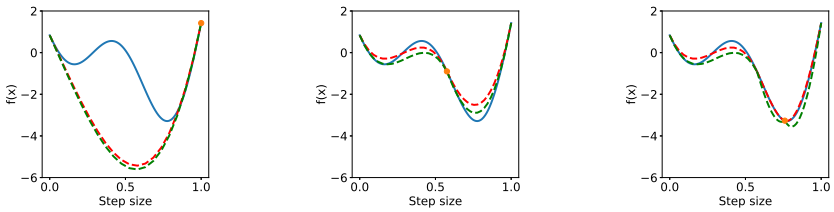
$$\begin{aligned} f(x_k + \alpha_k p_k) &\leq f(x_k) + c_1 \alpha_k \nabla f(x_k)^\top p_k \\ |\nabla f(x_k + \alpha_k p_k)^\top p_k| &\leq c_2 |\nabla f(x_k)^\top p_k|, \end{aligned}$$

where $0 < c_1 < c_2 < 1$ are two positive constants. The first inequality ensures a sufficient decrease in function value and the second ensures a point that is close to a stationary point along the search ray. Such a step length can be found by the following optimization problem

$$\min_{\alpha_k \geq 0} f(x_k + \alpha_k p_k), \tag{2}$$

i.e., the global minimizer along the search direction satisfies the strong Wolfe conditions. Usually, we do not know the structure of function f but only have

access to an oracle that computes function value and gradients for a given query point. Hence, finding an approximate solution to Problem (2) can be done using Bayesian optimization. For this, we create a model of the one-dimensional function along the search ray as follows: We approximate the true function by a cubic spline interpolation and additionally add an uncertainty estimate that depends on the distance to the closest query point and follows a Gaussian distribution. See Fig. 1 for an illustration.

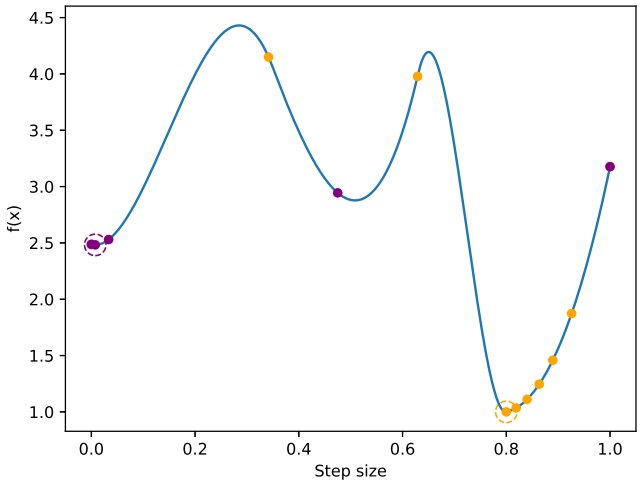


(a) First iteration. (b) Second iteration. (c) Third iteration.

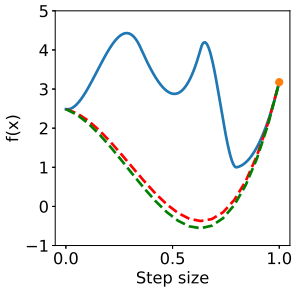
Fig. 1: The figures show three iterations of our proposed line search method. The blue line is the original function, the dashed red line is the cubic interpolation, and the dashed green line is the uncertainty. The x-axis shows the step length, while the y-axis shows the function value.

Suppose that in the current iteration we have already queried the function for a number of query points. There always exists exactly one solution for a cubic polynomial interpolating between two consecutive query points such that the polynomial matches function values and directional gradients in the given points. The uncertainty estimate for a given point follows a Gaussian distribution that is scaled with the function value of the corresponding cubic polynomial. In each iteration, the minimum of this Bayesian model, i.e., the spline interpolation minus the uncertainty estimate is computed and the result is the next query point. This process is repeated until a step length is found that satisfies the strong Wolfe conditions. In the practical implementation, an upper bound on the number of query points is fixed to account for numerical instabilities and round-off errors. This is common practice in line search methods.

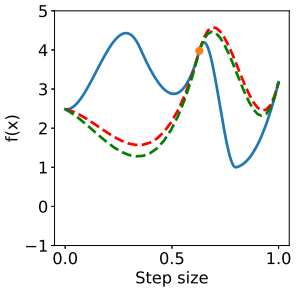
Let us compare standard line searches with our new line search method. Standard line searches typically concentrate on a specific interval containing a point that meets the Strong Wolfe conditions, effectively narrowing their focus. This approach, however, only uses the information, i.e., function value and gradient, available from the two query points that constitute the upper and lower bounds of the interval. All the information from the other query points is discarded. Consequently, there is a risk of missing a more optimal step length. This limitation is evident when examining Fig. 2. Here, our method is shown to identify a step length that yields a significantly better function value compared to the Moré-Thuente line search.



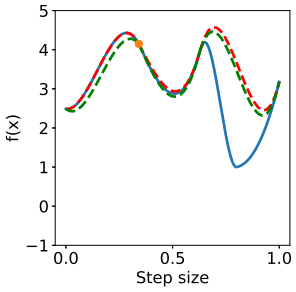
(a) Example of better step length.



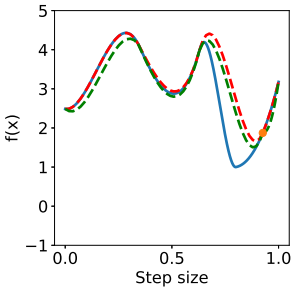
(b) First iteration.



(c) Second iteration.



(d) Third iteration.



(e) Fourth iteration.

Fig. 2: The figure compares our line search with the Moré-Thuente line search. In Subfigure 2a, orange points indicate our method's query steps, while purple points indicate points tried by the Moré-Thuente line search. Dashed circles indicate the final step size of the respective method.It can be seen that our method obtains a step length with a considerably better function value. Subfigures 2b- 2e depict four iterations of our method. The blue line is the original function, the dashed red line is the cubic interpolation, and the dashed green line is the spline plus uncertainty.

4 Experiments

To assess the effectiveness of our proposed line search, we integrated it into the GENO solver [7,8], which originally employs a quasi-Newton method equipped with the Moré-Thuente line search [10]. We also compare against other state-of-the-art methods, including L-BFGS-B, which also utilizes the Moré-Thuente line search, and Ipopt, which employs a filter line search [14,15]. We ran all solvers on the unconstrained problems from the CUTEst test set [5], a widely recognized benchmark featuring large-scale and numerically challenging problems. We focus primarily on large-scale problems common in machine learning applications, where computing the Hessian is impractical or infeasible. Consequently, all solvers could access function values and gradients but not Hessians.

We consider two criteria to evaluate convergence: the relative error in function value and the relative gradient norm. Convergence in function value is defined as the relative error to the optimum:

$$\frac{f_{\text{solver}} - f_{\text{opt}}}{1 + |f_{\text{opt}}|} < 10^{-4}. \tag{3}$$

Convergence in gradient is defined as the infinity norm of the gradient relative to the function value:

$$\frac{\|g\|_\infty}{1 + |f_{\text{solver}}|} < 10^{-6}. \tag{4}$$

We declare convergence if either of these criteria is satisfied.

The experiment was conducted on 285 out of the 289 unconstrained CUTEst problems. Four problems were discarded because their optimum was unbounded. Each method was executed on each problem until convergence, with a cutoff time of 10 minutes. Convergence was determined based on the relative error in function value or the infinity norm of the gradient relative to the function value.

The results are summarized in Table 1, which indicates the number of problems each method solved. As can be seen in Table 1, our proposed line search outperforms other state-of-the-art line search methods. Specifically, GENO combined with our new proposed method can solve 12 problems more than L-BFGS-B with the Moré-Thuente line search, the best competing approach.

Since L-BFGS-B outperformed Ipopt and GENO Moré-Thuente, we conducted a detailed comparison primarily with L-BFGS-B. Out of the 230 problems on which both GENO and L-BFGS-B converged, GENO needed more function evaluations on 84 and fewer on 146 of these problems. In order to compare the number of function evaluations for one problem needed by the different solvers, we use the following formula:

$$\frac{n_{\text{L-BFGS-B}} - n_{\text{GENO}}}{n_{\text{GENO}}} \tag{5}$$

where n denotes the number of function evaluations performed by each solver until one of the convergence criteria was met. The distribution of relative iteration counts between GENO and L-BFGS-B is depicted in Figure 3. This figure

Table 1: The table shows the number of problems each method solved. Column *function conv.* shows how many problems each of the methods solved by the function value criterion (see Equation 3), while column *gradient conv.* shows how many problems each solver solved by the gradient criterion (Equation 4). Column *convergence* shows the number of problems where either of the two equations are satisfied.

Solver	function conv.	gradient conv.	convergence
GENO this paper	247	208	257
GENO Moré-Thuente	227	175	242
Ipopt	191	185	226
L-BFGS-B	231	211	245

shows that while our proposed line search, on average, uses the same number of function evaluations, it solves more problems to optimality than competing state-of-the-art approaches.

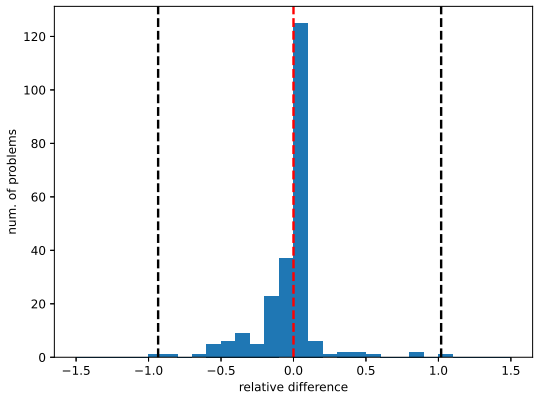


Fig. 3: The histogram of the relative running times of L-BFGS-B and GENO. Black lines represent minimum and maximum, while the red line represents the median. The x-axis represents the relative difference of the two methods computed as in Eqaution 5.

5 Conclusion

We presented a new line search approach. By integrating Bayesian optimization into the line search process, we have developed a method that efficiently utilizes all available data - including function values and gradients - and effectively

explores the search space to identify promising regions. This approach departs from traditional line search methods, which often overlook valuable information by focusing solely on refining step length intervals. Our method's efficacy is demonstrated through rigorous testing on the CUTEst test set, outperforming existing state-of-the-art methods regarding solution quality. Our approach's simplicity and ease of implementation allow for a seamless integration into existing optimization frameworks.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Armijo, L.: Minimization of functions having Lipschitz continuous first partial derivatives. Pacific Journal of Mathematics **16**(1), 1 – 3 (1966)
2. Barzilai, J., Borwein, J.M.: Two-Point Step Size Gradient Methods. IMA Journal of Numerical Analysis **8**(1), 141–148 (01 1988)
3. Giesen, J., Jaggi, M., Laue, S.: Regularization paths with guarantees for convex semidefinite optimization. In: International Conference on Artificial Intelligence and Statistics (AISTATS) (2012)
4. Giesen, J., Laue, S.: Combining ADMM and the augmented lagrangian method for efficiently handling many constraints. In: International Joint Conference on Artificial Intelligence (IJCAI) (2019)
5. Gould, N.I.M., Orban, D., Toint, P.L.: Cutest: a constrained and unconstrained testing environment with safe threads for mathematical optimization. Computational Optimization and Applications **60**(3), 545–557 (Apr 2015)
6. Hager, W.W., Zhang, H.: A new conjugate gradient method with guaranteed descent and an efficient line search. SIAM Journal on Optimization **16**(1), 170–192 (2005)
7. Laue, S., Mitterreiter, M., Giesen, J.: Geno – generic optimization for classical machine learning. In: Advances in Neural Information Processing Systems (NeurIPS). vol. 32 (2019)
8. Laue, S., Blacher, M., Giesen, J.: Optimization for classical machine learning problems on the GPU. In: Conference on Artificial Intelligence (AAAI) (2022)
9. Mockus, J.: On bayes methods for seeking an extremum. Avtomatika i Vychislitelnaja Technika **3**, 53–62 (1972)
10. Moré, J.J., Thuente, D.J.: Line search algorithms with guaranteed sufficient decrease. ACM Trans. Math. Softw. **20**(3), 286–307 (1994)
11. Rasmussen, C.E., Williams, C.K.I.: Gaussian processes for machine learning. MIT Press (2006)
12. Sra, S., Nowozin, S., Wright, S.J.: Optimization for Machine Learning. MIT Press (2012)
13. Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S.J., Brett, M., Wilson, J., Millman, K.J., Mayorov, N., Nelson, A.R.J., Jones, E., Kern, R., Larson, E., Carey, C.J., Polat, İ., Feng, Y., Moore, E.W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E.A., Harris, C.R., Archibald, A.M., Ribeiro, A.H., Pedregosa, F., van Mulbregt, P., SciPy 1.0 Contributors: SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods **17**, 261–272 (2020)

14. Wächter, A., Biegler, L.T.: Line search filter methods for nonlinear programming: Motivation and global convergence. SIAM Journal on Optimization **16**(1), 1–31 (2005)
15. Wächter, A., Biegler, L.T.: On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Mathematical Programming **106**(1), 25–57 (Mar 2006)
16. Wolfe, P.: Convergence conditions for ascent methods. SIAM Review **11**(2), 226–235 (1969)
17. Wu, J., Poloczek, M., Wilson, A.G., Frazier, P.: Bayesian optimization with gradients. In: Advances in Neural Information Processing Systems (NeurIPS) (2017)
18. Zhu, C., Byrd, R.H., Lu, P., Nocedal, J.: Algorithm 778: L-BFGS-B: fortran subroutines for large-scale bound-constrained optimization. ACM Trans. Math. Softw. **23**(4), 550–560 (1997)

Parallelizing High Dimensional Surrogate-Based Discrete Multi-Objective Optimization with Constraints

Rommel G. Regis[0000−0002−3468−5146]

Saint Joseph's University, Philadelphia PA 19131, USA
rregis@sju.edu

Abstract. The surrogate-based POSEIDON algorithm for high dimensional discrete multi-objective black-box optimization with constraints has shown good performance on the Mazda benchmark car structure design problem that has 222 discrete variables and 54 black-box inequality constraints. This paper explores the possibility of improving the wall clock time of POSEIDON to find an approximate Pareto front through parallel processing. The algorithm is modified to allow the selection of multiple points for simultaneous objective and constraint function evaluations in parallel. Within a given parallel iteration, the selection of points for function evaluations is done sequentially, using not only the information from previously evaluated sample points and also previously chosen points within the current iteration. Numerical experiments on the Mazda benchmark demonstrate that a parallel implementation of POSEIDON using up to 8 processors provides considerable improvement over the serial version on the set of nondominated points obtained as measured by the hypervolume metric based on a given reference point at various computational budgets. However, the improvements do not seem to scale very well as the number of processors increase on the Mazda benchmark. Nevertheless the parallel algorithm considered represents some progress that may be used as a stepping stone for designing a more scalable parallel algorithm for constrained discrete multi-objective black-box optimization.

Keywords: Constrained multiobjective optimization · ordinal discrete variables · high-dimensional optimization · surrogates · parallel implementation.

1 Introduction and Motivation

Many papers have been written on surrogate-based and Bayesian approaches for computationally expensive multiobjective optimization, including methods that handle constraints. Parallel approaches have also been employed to reduce the wall clock time of surrogate-based and Bayesian multiobjective optimization methods on engineering optimization problems. However, much of the work on surrogate-based multiobjective optimization do not handle discrete variables. Moreover, the proposed surrogate-based multiobjective optimization methods

have mostly been tested on relatively low dimensional problems (i.e., 30 decision variables or less). Part of the issue is that the surrogate used in the popular Bayesian approach typically requires a considerable amount of time to fit on high dimensional problems involving hundreds of decision variables.

The POSEIDON algorithm (Pareto Optimization using Surrogates for Expensive Inequality-constrained Discrete Ordinal and Nonlinear problems) [6] was proposed as a surrogate-based approach to handle high-dimensional expensive constrained multi-objective optimization involving hundreds of ordinal discrete variables. Here, ordinal means that the values of the discrete variable can be ordered. POSEIDON using Radial Basis Function (RBF) models has been successfully applied and has outperformed non-surrogate alternative approaches on the Mazda car structure design problem that has 222 ordinal discrete variables and 54 black-box inequality constraints, which is among the largest benchmark problems proposed in black-box optimization. The purpose of this article is to explore a parallel implementation of POSEIDON that can take advantage of multiple cores and reduce the wall clock time needed to find an approximate Pareto front for large-scale constrained discrete multi-objective optimization problems such as the Mazda benchmark.

Specifically, this paper explores a parallel approach for POSEIDON to solve the following constrained multi-objective optimization problem with ordinal discrete variables:

$$\begin{aligned} & \min_{x \in \mathbb{R}^d} \ F(x) = (f_1(x), \ldots, f_k(x)) \\ \text{s.t.} \quad & \\ & G(x) = (g_1(x), \ldots, g_m(x)) \leq 0 \\ & x^{(i)} \in D_i \subset \mathbb{R} \ i = 1, \ldots, d \end{aligned}$$

where $x^{(i)}$ is the ith coordinate of $x \in \mathbb{R}^d$ and D_i is an ordered finite set of possible settings of $x^{(i)}$.

2 The POSEIDON Algorithm for Constrained Discrete Multi-Objective Optimization

The POSEIDON algorithm [6] was designed for constrained discrete multiobjective optimization and it can handle problems even when a feasible solution is not available at the start via a two-phase approach. The first phase uses surrogates of the constraints to identify a feasible point while the second phase finds a set of nondominated objective vectors that approximate the Pareto front or that at least improves on the objective vector of the feasible point found.

In this paper, we consider the simpler scenario where a feasible point is available at the beginning, which is not uncommon in engineering applications where a feasible initial design is available and one seeks alternative solutions that improve on some or all of the objectives. That is, we focus on designing a parallel approach for Phase II of POSEIDON. Future work will incorporate a parallel approach for Phase I, which is focused on finding a feasible solution to the problem.

When a feasible initial point is available, POSEIDON begins by augmenting it with a set of points generated uniformly at random from the discrete search space. Alternatively, one can use a space-filling design that is tailored to the discrete search space. Then, among the feasible points, the initial set of nondominated points is identified. Each iteration of POSEIDON fits or updates multiple surrogates, one for each black-box objective function and one for each black-box constraint. In the case of the Mazda benchmark, this means maintaining a total of 56 surrogate models at any given time. Next, one of the current nondominated points in the decision space is chosen as a center point and many trial points are generated in some neighborhood of this nondominated center point. Each trial point is obtained by modifying a fraction of the coordinates of the chosen nondominated point. Three main strategies have been used to select a center point. The random (RND) strategy randomly selects one of the current nondominated points. The objective space distance (OSD) strategy chooses the nondominated point that is the most isolated in the objective space. The extreme objective vector (EOV) strategy randomly selects one of the nondominated points that minimizes one of the objective functions among all current nondominated points.

Next, among the trial points in each iteration, POSEIDON uses the surrogates for the constraints to determine those that are predicted to be feasible or those that are predicted to have the smallest number of constraint violations. Then, among these eligible trial points, POSEIDON uses the surrogates for the objectives to determine which ones are predicted to be nondominated by the current nondominated points and by the other eligible trial points in the current iteration. The next point for function evaluation is then selected from the eligible trial points that are predicted to be nondominated by weighing the scaled values of two criteria. One criterion is the MDOS, which is the distance between the predicted objective vector of the trial point and the current set of nondominated objective vectors in the objective space. The other criterion is the MDDS, which is the distance between the trial point and the current set of nondominated points in the decision space. In both cases, the distance of a point to a set of points is defined as the smallest of the distances between that point and the points in the given set. After the objective and constraint functions are evaluated at the chosen promising sample point, the current set of nondominated points is updated and POSEIDON iterates until the computational budget is reached.

Each trial point is obtained by modifying some of the coordinates of the chosen nondominated center point. Each coordinate of the nondominated center point is modified with a certain probability p_{pert}. This modification consists of changing the current coordinate by either increasing or decreasing its value by a few discrete steps according to the restrictions set by the neighborhood depth parameter $depth_{\text{nbhd}}$. This parameter is the fraction of the number of settings that the variable is allowed to increase or decrease. The values of the two control parameters p_{pert} and $depth_{\text{nbhd}}$ cycle through the iterations where some settings correspond to a more global search while others correspond to a more local search. One of the original implementations of POSEIDON employed

a local variant where the control parameters (p_{pert}, $depth_{\text{nbhd}}$) took on the values $\langle(0.5, 30\%), (0.1, 10\%), (0.05, 10\%), (0.01, 10\%)\rangle$ repeatedly in a cyclic manner.

3 A Parallel Approach for POSEIDON

The parallelization of POSEIDON consists of modifying the algorithm to generate multiple points to be evaluated simultaneously in a given iteration. This is achieved by choosing multiple nondominated center points where several groups of trial points will be generated in their respective neighborhoods. Moreover, various sizes of neighborhoods are used by considering different values of the control parameters p_{pert} and $depth_{\text{nbhd}}$ in a given parallel iteration. More precisely, the exact sequence of values for p_{pert} and $depth_{\text{nbhd}}$ as specified by the given cycle in the original version are used in the parallel implementation except that multiple sets of values (as many as there are processors or nodes) are considered in any parallel iteration.

Moreover, within each parallel iteration, the selection of points for function evaluation is done sequentially. The selection of a such a point uses some information not just from previously evaluated points but also from points selected earlier within the iteration before any function evaluations will take place in parallel. In particular, the MDOS criterion is modified so that it also incorporates the distance between the predicted objective vector of a trial point with the predicted objective vectors of the previously selected points within the current parallel iteration. For example, if POSEIDON is running with 4 processors, when choosing the third point for function evaluation among the trial points generated from the third nondominated center point, the MDOS criterion also incorporates the distance between the predicted objective vector of each trial point with the predicted objective vectors of the first and second points chosen earlier within the given parallel iteration. Similarly, the MDDS criterion is modified so that it also incorporates the distance between a trial point and the previously selected points (in the decision space) within the current parallel iteration.

4 Numerical Results on the Mazda Benchmark

A parallel implementation of the local variant of POSEIDON using radial basis function (RBF) surrogates is implemented in Matlab and tested on the Mazda 3-car structure design problem [1, 3–5] that has 222 ordinal discrete variables representing the thicknesses of structural parts, 54 black-box inequality constraints that includes collision safety requirements, and 2 objective functions: f_1 is the total weight of three types of Mazda cars, and f_2 is the number of common gauge parts. The possible values for each decision variable range from 4 to 18 different settings resulting in a discrete search space of size 4.4427×10^{198}, which is unimaginable even by astronomical standards. The particular RBF model used for the objectives and constraints is the same as in the original POSEIDON algorithm, which is a cubic RBF augmented by a linear polynomial. There is a way to implement the algorithm in such a way that the RBF models for the

two objective functions and 54 constraint functions can be updated very quickly. This involves storing distances between all previous sample points and using an efficient solver for the linear system needed to fit the model. This is one advantage of RBF methods over the more popular Bayesian optimization approaches where the surrogate models can take a considerable amount of time to fit on much smaller dimensional problems.

The performance of Parallel POSEIDON is evaluated by calculating the hypervolume corresponding to the set of nondominated points obtained by the algorithm with respect to the reference point provided in [4]. Figure 1 (left) shows the hypervolumes for the original (Serial) POSEIDON and Parallel POSEIDON using 2, 4 and 8 processors at various parallel iterations, which roughly correspond to various computational budgets. For truly computationally expensive problems, the overhead computing times are small relative to the total time spent on function evaluations, and so, the wall clock time of an algorithm at a given instant is roughly proportional to the number of parallel iterations so far (e.g., see [7]). Moreover, Figure 1 (right) shows the nondominated objective vectors obtained by the serial and parallel POSEIDON after 800 parallel iterations. From the figures, it can be seen that the parallel approach does significantly improve the performance of POSEIDON in the sense that better hypervolumes are obtained at fixed computational budgets (parallel iterations) when using 2 or 4 processors. However, the improvement obtained from 2 to 4 processors is much less than the improvement from 1 (serial) to 2 processors, and the result for 8 processors appears to stall. This indicates that the algorithm does not seem to scale well with more processors. However, since the results presented are the result of only one run, there is a possibility that the result for 8 processors was a fluke and that multiple trials might show that the average case performance is not so bad. Unfortunately, one trial run of these algorithms on the Mazda benchmark takes an enormous amount of time, so further investigations are needed involving more computing resources. In any case, the parallel version of POSEIDON still provides some improvement that can be used as a stepping stone for developing a more scalable parallel implementation.

References

1. Fukumoto, H., Oyama, A.: Benchmarking multiobjective evolutionary algorithms and constraint handling techniques on a real-world car structure design optimization benchmark problem. In: GECCO '18: Proceedings of the Genetic and Evolutionary Computation Conference Companion, pp. 177—178 (2018)
2. Haftka, R.T., Villanueva, D., Chaudhuri, A.: Parallel surrogate-assisted global optimization with expensive functions - a survey. Struct Multidisc Optim 54: 3–13 (2016)
3. Kohira, T., Kemmotsu, H., Oyama, A., Tatsukawa, T.: Proposal of benchmark problem based on real-world car structure design optimization. In: GECCO '18: Proceedings of the Genetic and Evolutionary Computation Conference Companion, pp. 183–184 (2018)
4. Mazda Benchmark Problem, https://ladse.eng.isas.jaxa.jp/benchmark/index.html. Last accessed 30 April 2020

5. Oyama, A., Kohira, T., Kemmotsu, H., Tatsukawa, T., Watanabe, T.: Simultaneous structure design optimization of multiple car models using the K computer. In: 2017 IEEE Symposium Series on Computational Intelligence (SSCI), Honolulu, HI, pp. 1–4 (2017)
6. Regis, R. G.: A two-phase surrogate approach for high-dimensional constrained discrete multi-objective optimization. GECCO '21: Proceedings of the Genetic and Evolutionary Computation Conference Companion (2021)
7. Regis, R. G. and Shoemaker, C. A.: Parallel Stochastic Global Optimization Using Radial Basis Functions. INFORMS Journal on Computing 21(3): 411–426 (2009)

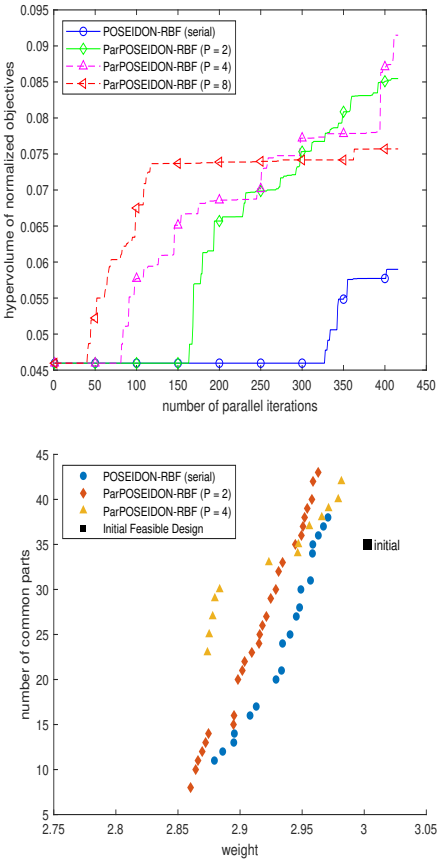


Fig. 1. Left: Hypervolumes corresponding to the nondominated points found by the Serial and Parallel POSEIDON algorithms at different computational budgets (parallel iterations). Right: Nondominated objective vectors found by the Serial and Parallel POSEIDON algorithms after 800 parallel iterations.

Synergies of Deep and Classical Exploratory Landscape Features for Automated Algorithm Selection

Moritz Seiler[1], Urban Škvorc[1], Carola Doerr[2], and Heike Trautmann[1,3]

[1] Machine Learning and Optimisation, University of Paderborn, Germany
{moritz.seiler,urban.skvorc,heike.trautmann}@uni-paderborn.de
[2] Sorbonne Université, CNRS, LIP6, Paris, France
carola.doerr@lip6.fr
[3] DMB Group, University of Twente, Netherlands

Abstract. Per-instance automated algorithm selection (AAS) aims at leveraging the complementarity of optimization algorithms with respect to different problem types. State-of-the-art AAS methods for numerical black-box optimization rely on supervised learning techniques that are supported by exploratory landscape analysis (ELA) feature sets. Recent works question the generalization ability of popular AAS approaches, which motivated the design of alternative feature sets.
In this work, we take a closer look at the recently proposed set of Deep ELA features and investigate the ways in which Deep ELA complements the classical ELA feature sets. To this end, we first study the correlation between the two feature collections, both through pairwise classification and through regression models. The complementarity observed in these analyses is confirmed by an AAS study, where models combining deep and classical features outperform those that are restricted to selecting from only of the two collections.

Keywords: Black-box Optimization · Exploratory Landscape Analysis · Automated Algorithm Selection · Deep Learning · Self-Supervised Learning · Feature Selection

1 Introduction

Automated Algorithm Selection (AAS) [24], aimed at automatically selecting the best performing algorithm for a given problem, as well as the closely related task of *Automated Algorithm Configuration* (AAC), which similarly aims at selecting the best performing configuration of a single algorithm, have long been of interest to the optimization community [2, 13]. This interest stems from the fact that both AAS and AAC can drastically reduce the runtime needed to solve a given optimization problem and simultaneously substantially improve optimization performance. As the algorithm performance is closely tied to the landscape of the optimization problem being solved, the task of selecting the best-performing algorithm or configuration requires some low-level information

about the optimization landscape which are usually described as numerical values, referred to as *(instance-)features*.

In continuous single-objective black-box optimization, AAS and AAC have recently seen a growth in popularity, fueled in large parts by the availability of dedicated feature sets such as *Exploratory Landscape Analysis* (ELA) [15], which allow for the transformation of problem samples into landscape features without requiring prior domain-expert knowledge of the problems. ELA features can be conveniently calculated using programming libraries such as *flacco* [13] and *pflacco* [22]. This has led to a large amount of research in this area [5, 10, 11, 12, 16, 19, 20, 27]. In these works, the authors utilize landscape features and algorithm selection or configuration to demonstrate an overall reduced runtime in comparison to relying solely on the *Single-Best Solver* (SBS). In other words, the algorithm selection model is capable of selecting specific algorithms for each problem in a problem suite to improve performance compared to only using a single overall best-performing algorithm.

Despite the benefits provided by ELA, the features it produces are not without flaws. Commonly criticized drawbacks include (1) a large correlation between features, (2) concerns over the expressiveness and robustness of the features [23], (3) additional computational costs to compute the features, and (4) a lack of generalizability of the features to problems sets outside of the one that they were developed for [14, 17, 30].

As a result, there is currently a large research interest in developing alternative approaches for automatically representing a problem landscape. For example, the authors of [3, 25, 28] proposed learned features for continuous optimization problems. Rather than relying on experts to design feature sets manually, those authors used different training tasks to automatically extract instance features. While van Stein et al. [28] used a simple multi-layered autoencoder architecture including an unsupervised training task, Cenikj et al. [3] used a deep transformer architecture [29] trained on the *Black-Box Optimization Benchmarking* suite [BBOB, 9] by predicting the function identifier. Seiler et al. [25] also utilized a transformer architecture but relied on a self-supervised learning strategy that does not require any labeling.

However, when developing such learned features, it is important to ensure that these features are not simply novel, but that they also complement each other, as well as the original ELA features. Without complementarity, such newly developed features would only restate the knowledge that is already contained in the traditional ELA features, without necessarily improving the overall understanding of a problem's landscape. Nevertheless, we must also ensure that these newly developed features do not just contain novel information, but also still capture existing knowledge. This ensures that the newly developed features can be used on their own, and applied to domains for which no current feature sets exist.

Another benefit of performing complementarity analysis is that, particularly in the case of approaches based on deep learning, interpreting their results can be difficult. Comparing these newly developed features to ELA features, which are

designed to correspond to a set of well-understood high-level problem properties, such as modality, global-to-local optima contrast, or separability, can help us better understand the behavior of the newly developed features.

Our contribution: In this paper, we will focus on the recently proposed *Deep ELA* feature set proposed in [25]. Deep ELA was shown to combine the benefits of both traditional ELA features (their ease of computation and lack of overfitting) with those of deep-learning-based feature-free approaches (in particular, their invariance to common problem transformations and their low correlation). Deep ELA has already shown promising results, drawing level with or even outperforming both traditional ELA and feature-free approaches in a single-objective AAS study [25]. However, the complementarity of the newly developed Deep ELA features and the original ELA features (which we will refer to as classical ELA features in the rest of the paper) is yet to be examined. Using correlation and regression analyses, we first observe that there is some overlap in the information captured by the classical ELA features and the Deep ELA ones. However, we also show that the sets complement each other, in that some features of the ELA set are difficult to predict by the Deep ones. We further verify this complementarity with an AAS study, which shows that combining the two approaches improves the performance over the stand-alone feature sets.

Paper organization: This paper is structured as follows. In general, we aim to analyze the complementarity and supplementarity of these two approaches, classical ELA and Deep ELA, in multiple ways. First, we provide the background necessary for the understanding of this paper, including an overview of classical ELA and Deep ELA (see Chapter 2). In Chapter 3, we compare the pairwise correlation of the Deep and classical ELA features. In Chapter 4 we train a *Support Vector Machine* (SVM) model to predict the classical ELA features using the Deep ELA features. Finally, we perform a study on AAS and compare the performance of models trained using the classical and Deep ELA features, as well as a model that uses the combination of both the classical and the Deep ELA features (see Chapter 5). If the classical and Deep ELA features are complementary, we would expect that the performance of the model increases when both types of features are used. On the other hand, we expect that Deep ELA features provide comparable performance if they mostly substitute classical ELA features. Last, we discuss in Chapter 6 the obtained results as well as their implications, summarize our results, and present avenues for future work.

2 Background and Methodology

To perform AAS using machine-learning models, we require some way of describing the landscape of an optimization problem that can be used by the model. One of the most common ways to do so in state-of-the-art research is the extraction of *Exploratory Landscape Analysis* (ELA) features. This approach, introduced by Mersmann et al. [15], allows for the automatic extraction of low-level ELA (instance-) features that correspond to specific high-level problem properties, such as the problem's modality or the ruggedness of its landscape. Further, the

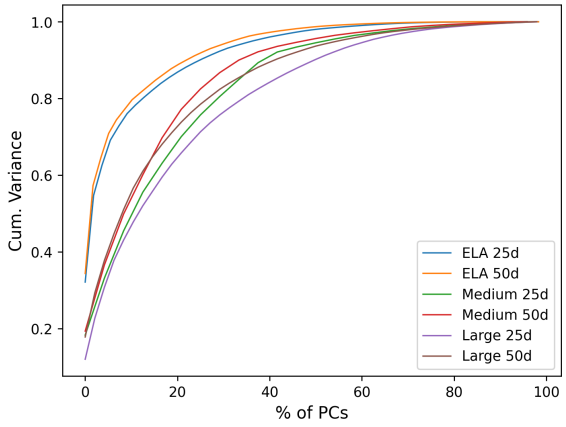


Fig. 1. Explainable Variance of classical (labeled as ELA **25d**/**50d**) and Deep ELA features (labeled as Medium/Large **25d**/**50d**). **25d** and **50d** correspond to the sample size while medium/large describes the model size of the Deep ELA models. Deep ELA features contain less redundant information as their principal components explain less of the overall variance.

ELA features can be calculated purely from a small amount of problem samples. Usually, a sample size of $50d$ or for improved robustness $250d$ are commonly chosen sample sizes that scale linearly with the dimensionality d of the decision space.

In the following, we will focus on a subset of these features, omitting categories that require additional samples (sampled iteratively) or the transformation from set-based into graph-based representation. These approaches differ substantially from Deep ELA which requires only a single set of samples and, hence, inhibit a fair comparison. In addition, the features that rely on graph-based representation are computationally expensive when applied to problems with a larger dimensionality. The used feature categories are:

Meta-Model features, which fit either a linear or a quadratic model to the provided problem samples, and use the metrics of this mode (such as the model's R^2 score) for the features.

PCA features, which compute the Principal Components of the sample, and use metrics such as the number of components required to explain 90% of the variance of the original samples as the features.

Nearest Better Clustering features, which examine the samples based on the distance between their nearest neighbors versus their better neighbors, where a better neighbor is a neighbor with better fitness.

Dispersion features, which compare the distribution of a subset of samples with the best fitness values against the distribution of the entire sample set.

Information Content features, based on a method for approximating the ruggedness of the problem landscape.

y-Distribution features, which represent descriptive statistics of the distribution of the fitness values, such as their skewness or kurtosis.

Linear model features, which divide the sample space into cells, and fit a linear model on each of the cells separately, then compare the various statistics of these models.

Fitness Distance Correlation features, which calculate the distances between samples in objective and in decision space, and compare various statistics between the two.

Specifically, in this paper, we consider the normalized variants of these features by `min-max` normalizing the objective space prior to the feature computation as proposed by [21].

However, as already noted in the introduction (see Chapter 1), ELA landscape features are not without flaws, and recent research has proposed several alternative methods. Hence, we will focus on an approach named Deep ELA [25], which proposed four pre-trained deep-learning models that were trained on 200 million randomly generated single- and multi-objective continuous optimization problems. Two of these models accept up to six input dimensions (medium models) while the other two accept up to twelve dimensions (large models). In addition, the authors experimented with input sizes of $25d$ and $50d$ for each medium and large architecture. The pre-trained models are transformer-based [29] and were trained using a self-supervised training task, closely following the idea proposed by [4].

Both ELA features and feature-free approaches have shown very promising results when applied to AAS and AAC. Kerschke et al. [11] provide an overview of AAS in the field of optimization, and note the success of using ELA to address the algorithm selection problem. In the field of single-objective continuous optimization, AAS has primarily focused on the 24 BBOB problems [9] of the COCO [8] benchmarking platform, a set of popular and well-known problems for benchmarking continuous optimization algorithms. Kerschke and Trautmann have shown that it is possible to reduce the expected running time on the set of 24 problems by half compared to a single best solver [12]. Prager et al. achieved only slightly worse results using a feature-free approach [20]. Finally, Deep ELA [25] consistently outperformed the feature-free approach described in [20], and in some cases outperformed the ELA-based approach described in [12]. Further, Deep ELA can be applied to AAS of multi-objective optimization problems (see Seiler et al. [25]) out of the box and have shown promise in combinatorial optimization [1]. Yet, we limit the scope of this paper to single-objective optimization problems.

3 Correlation Between Deep and Classical ELA

We begin our analysis with the comparison of the correlation between the Deep and classical ELA. In all that follows, we label the Deep ELA features as either *'Medium* $25d$*'*, *'Medium* $50d$*'*, *'Large* $25d$*'* or *'Large* $50d$*'*, depending on the size of

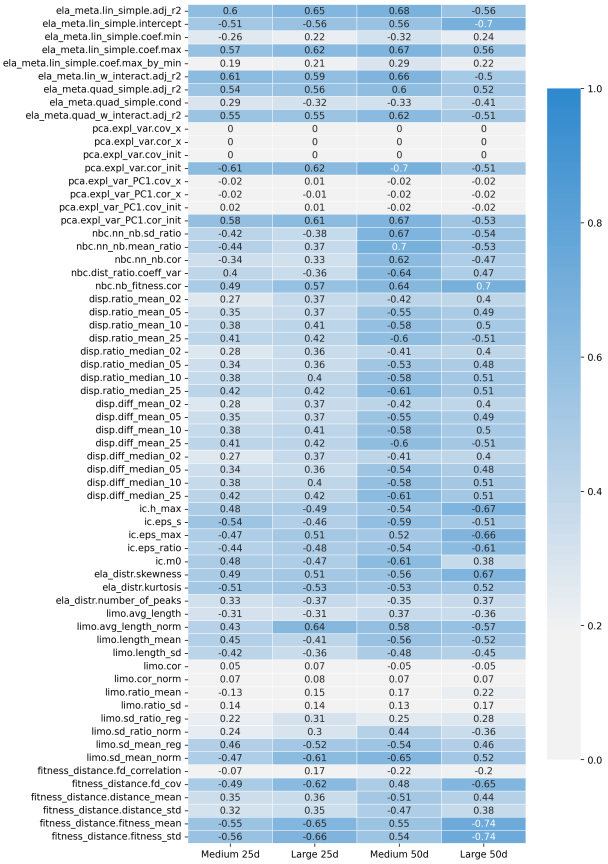
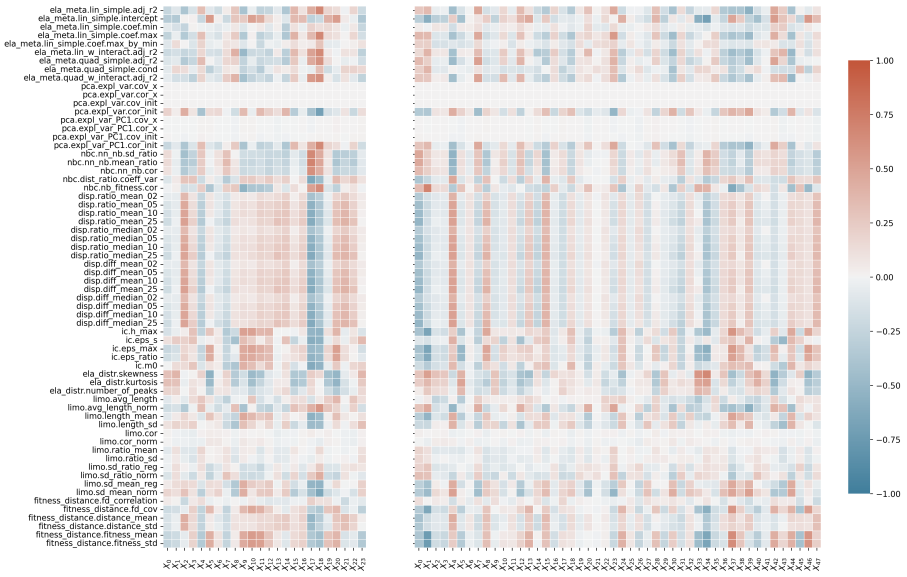


Fig. 2. Strongest correlation between the classical ELA features (rows) and the features of the Deep ELA models, for each of the four Deep ELA models (columns). The color scheme is based on absolute values; the darker the color, the stronger the (positive or negative) correlation.

the Deep learning model and the number of evaluated solutions per function that were used to train it, to which we refer to as *'sample size'* in the following. In this and all of the following experiments, both the classical and Deep ELA features are calculated on the 24 noiseless single objective BBOB problems, specifically on the first 20 instances of each function in dimensions 2, 3, 5 for the medium-sized model and, in addition, for dimension 10 for the large models. Following the recommendations made in [19, 20, 26], we base our feature computations on $50d$ and, in addition, also $25d$ samples, taken from the search space by *Latin Hypercube Sampling* (LHS). The Deep ELA features were computed by following closely the procedure described in [25].

Figure 1 shows the cumulative variance of the features calculated using the two approaches after performing *Principal Component Analysis* (PCA). We can



(a) Medium 50d (b) Large 50d

Fig. 3. Correlation heatmap between classical and Deep ELA features. In (a) the correlation between the Medium $50d$ model and the classical ELA features while in (b) the Large $50d$ model is depicted.

see that there is a distinct difference in the explained variance between the Deep and classical ELA. For the classical features, the first few components alone already explain about 60% of the variance, while the Deep features require a larger amount of components to reach this level. This indicates that the Deep ELA features contain less redundant information as PCA is efficient in reducing redundancy as well as noise. Hence, a low number of PCs that explain a large proportion of the original variance indicates that the original data contains many redundant and noisy information.

Further, Figure 3 shows the pairwise Spearman correlation between the classical and Deep ELA features using a sample size of $50d$ and the medium and large ELA models. We omit the comparison for the models with a sampling size of $25d$ as this produced similar results. Further, we can see that most classical ELA features have a weak to medium correlation to Deep ELA, which indicates that Deep ELA is somewhat representative of the characteristics of classical ELA. An exception to these are the `pca` and to an extent the `limo` feature groups, which are only weakly correlated with the Deep ELA features.

Finally, despite the fact that Deep ELA features are not comparable between different models as the training routine does not enforce any specific feature to be learned, we still see similar patterns emerge in both of the models. Further, as it is evident in Figure 2, the maximal correlation values between any Deep ELA feature and every classical ELA feature have similar absolute values across the

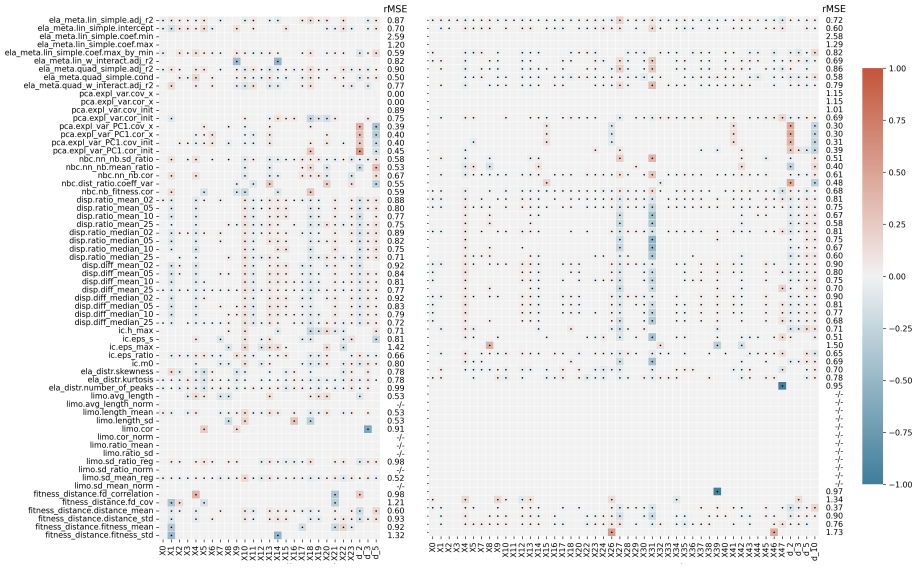
different models. This is particularly interesting as (again) the training routine of the Deep ELA models does not enforce any specific and pre-defined feature to be learned. Instead, it lets the models learn any meaningful representation given an optimization problem as input. Yet, we can see in Figure 3 and Figure 2, that all four models learn similar features as these features have a similar correlation to the expert-designed classical ELA features. Yet, pairwise correlations are only a weak method to verify that certain feature sets contain similar information.

4 Classical ELA Feature Prediction

In this chapter, we conduct an additional experiment to determine whether the information contained in the entire set of Deep ELA features matches up with the information provided by the classical ELA features. To achieve this, the Deep ELA features are used to train a linear SVM model that predicts the classical ELA features. Further, we use *Recursive Feature Elimination* [RFE, 7] to select the most relevant Deep ELA features for predicting the classical ones, which allows us to see which Deep ELA features are most closely related to the classical features. RFE removes the least relevant feature based on the SVM's coefficients, iteratively.

Figure 4 shows the result of the feature prediction experiment. Here, the labels on the side of the figure show the *standardized root Mean Square Error* (rMSE) between the predicted and the target feature. Individual cells show us which Deep ELA features were considered important by the model to predict each classical ELA feature, after performing recursive feature elimination according to the coefficients of the trained SVM model. One can see that most of the features can be well predicted using an SVM model, which reinforces the findings from the correlation analysis, i.e. that the Deep ELA features mostly capture the information provided by the classical ELA features. In addition, one can also see that the `pca` feature category is predicted well, despite the low correlation with individual Deep ELA features, which shows the importance of this type of analysis, as it is important to consider the interactions between individual ELA features.

Predicting the feature `ela_meta.lin_simple.coef.max` achieves somewhat worse performance than the rest of the features in both models. This indicates that, despite most classical ELA features can be well substituted, Deep ELA features still do not capture all classical ELA features. On the other hand, we can also identify Deep ELA features that are somewhat irrelevant for predicting classical ELA features. An example is the X_{11} feature of the Large $50d$ model depicted in Figure 4 (b). This deep feature may be unrelated to classical ELA features and may capture certain information that is not captured by classical ELA features at all. Again, we find similar rMSE values between the Medium $50d$ and Large $50d$ models, which strengthens our finding that the Deep ELA models learn similar representations although their training routine does not enforce learning to encode anything specific structures or information of a given optimization problem.



(a) Medium $50d$ (b) Large $50d$

Fig. 4. Feature importance of the Deep ELA features used to predict classical ELA ones using a Linear SVM in combination with recursive feature elimination. Selected Deep ELA features are indicated by a black dot, with the color of its surrounding box indicating the feature's importance. In some cases, ELA features contained missing values (which is within the definition of classical ELA features, e.g. `limo` features require a certain number of candidate solutions which is not always given in our setup) and, hence, no prediction was performed.

5 Automated Algorithm Selection Study

Finally, we examine the performance of the classical and Deep ELA features for the task of *Automated Algorithm Selection* (AAS). These experiments are performed using two different machine learning techniques to train the algorithm selectors: *k Nearest Neighbors* (kNN) and *Random Forests* (RF), using the same methodology as described in [12, 20, 25]. In addition, we applied *Sequential Forward Feature Selection* (SFFS) in combination with 5-fold cross-validation. The process of feature selection is in the outer loop. Therefore, the selected features are the same across the five folds and the five trained models. For both learners, kNN and RF, and for each of the two sampling sizes ($25d$ and $50d$) three different models are trained: (1) using classical ELA features (ELA), (2) using Deep ELA features (Deep), and (3) using a combination of both classical and Deep features (Comb.). The last model starts with the best set of Deep ELA features and sequentially adds classical ELA features. Other than that, we used the same setup as described in Kerschke et al. [12, 20, 25].

Table 1 lists classical ELA features that are often selected by either the classical ELA or the combined selectors. To be more precise, it lists all ELA

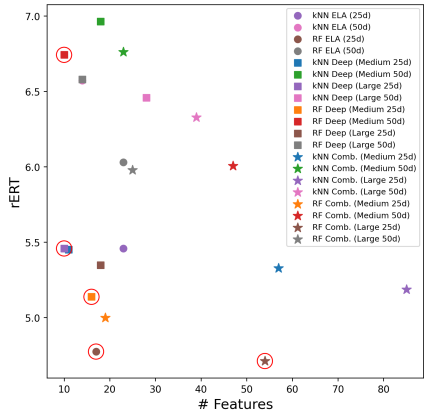
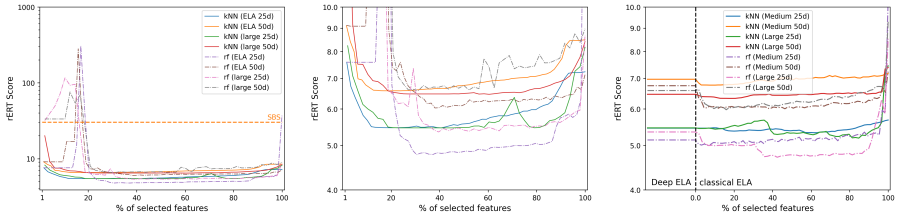


Fig. 5. Visualization of all trained AAS models based on the achieved performance (rERT) and the total number of features used. Non-dominated selectors are highlighted by a red circle.

features that were at least two times more often selected by one over the other. On the other hand, the dimensionality features as well as the PCA-based features are significantly less required by the combined selectors. This indicates again that the Deep ELA features represent similar meanings. On the other hand, the dispersion features are more often utilized by the combined selectors, noticeably. This indicates that the Deep ELA features do not substitute this information well.

Subfigures (a) and (b) in Figure 6 show how the performance of the models changes with the number of features selected. One can see that in the case of using just the classical or the Deep ELA features individually, only $20\% - 40\%$ of all available features produces close to optimal results. We also see that the kNN models work well even with a very small amount of features, while the random forests require a certain amount of features to reach performance better than the SBS. Subfigure (c) visualizes the improvements of a selector that utilizes only Deep ELA by the gradual addition of classical ELA features. We see that, for most models, an addition of only a few classical features results in a noticeable performance improvement, with the best-performing model achieving a further increase once around 30% of the features have been added. For most models, using all of the available landscape features produces a sharp decrease in performance. Figure 6 (c) also depicts a clear separation between $25d$ and $50d$ sample sizes, indicating that smaller sample sizes outperform larger ones. Selecting a sample size is always a trade-off between the accuracy of the computed features and additional costs. Larger sample sizes provide more (redundant) information which leads to higher stability of the compute features but also increases costs as more candidate solutions must be evaluated. In our case, we found that smaller costs ($25d$) outperform the higher feature quality ($50d$), indicating that ELA features (both Deep and classical) are even stable enough for small sample sizes.



(a) Large Models vs class. ELA

(b) Large Models vs class. ELA (Zoomed)

(c) Comb. of Deep and class. ELA

Fig. 6. Figures (a) and (b) show how the AAS model's rERT score changes with the number of features used, with Figure (b) showing only the rERT range of [4.0, 10.0] to allow for a more detailed examination once the rERT score starts to converge. Figure (c) depicts the impact of adding classical ELA features to the best set of Deep ELA features. Overall models, the best performance is achieved by a combined RF model 18 Deep ELA and 36 classical ELA features., for a total of 54 features.

Finally, Table 2 shows the full results of the algorithm selection study after the feature selection has been performed. Only a single model is trained for each column, with the rows representing the single model's performance separately by dimension and BBOB feature group. The selectors' performance is assessed by using the *relative Expected Running Time* (rERT), which measures the average time needed to reach a target precision (the found optimal solution is within a $\varepsilon = 0.01$ distance to the true global optima), normalized by the best-achieved average across the entire table. The rERT values also include the feature-costs which are in this case the number of sampled and evaluated candidate solutions.

One can see that all machine learning models mostly outperform the SBS baseline, with some exceptions in dimensions and feature groups where the SBS performance is close to the VBS. The final row presents the overall performance of each algorithm, with all algorithms that are stochastically tied to the best selector marked with a star using *Robust Ranking* with an $\alpha = 0.05$ [6]. Examples are (1) RF-based selector trained on classical ELA features and a sample size of $25d$ as well as (2) all models (kNN and RF as well as their Deep and Combined variants) trained on the Medium $25d$ features. The stochastic approach does not produce a clear winner in terms of the methodology used. However, in terms of pure performance, we can see that in all cases, the combined model using both classical and Deep ELA features outperforms the other two models. This indicates that neither Deep ELA nor classical ELA are complete substitutes for one another. In fact, both feature types encode some information that the other does not.

Overall, all types of models performed impressively compared to the SBS, outperforming it by a large margin both overall and in the majority of feature categories and dimensionalities. This matches prior work which has shown that ELA can be effectively used for automated algorithm selection on the BBOB problems, and also shows that Deep ELA achieves results competitive with the current state-of-the-art on this problem.

Table 1. List of the classical ELA features that are selected frequently by the classical ELA-based algorithm selectors and rarely by the Deep ELA-based selectors and vv.

Feature	ELA	Comb.	Feature	ELA	Comb.
d_2	7	3	ic.eps_max	6	4
d_3	6	2	ic.eps_ratio	3	5
disp.diff_median_02	0	2	ic.eps_s	7	4
disp.diff_median_05	1	3	ic.h_max	4	2
disp.diff_median_10	1	3	limo.avg_length	3	1
disp.diff_median_25	4	2	limo.length_mean	3	1
disp.ratio_mean_25	5	3	nbc.dist_ratio.coeff_var	1	3
ela_distr.number_of_peaks	4	2	pca.expl_var.cor_init	4	1
ela_distr.skewness	7	5	pca.expl_var.cor_x	6	3
ela_meta.lin_simple.coef.max	0	2	pca.expl_var.cov_init	6	3
fitness_distance.fd_correlation	2	0	pca.expl_var.cov_x	6	4
fitness_distance.fitness_std	4	6	pca.expl_var_PC1.cov_init	4	2

Figure 5 plots all of the models based on their performance versus the number of features used. We can see a set of non-dominated models consisting of five AAS models: *'RF Deep (Medium* 50*d)'*, *'kNN Deep (Large* 25*d)'*, *'RF Deep (Medium* 25*d)'*, *'RF (ELA* 25*d)'*, and *'RF Comb. (Large* 25*d)'*. From this, we see that the deep features provide us both with a model that achieves the absolute best performance, as well as the one with the fewest features. However, the *'RF (ELA* 25*d)'* model still achieves performance close to the best algorithm with only a small decrease in performance. We find those two models the most interesting: *'kNN Deep (Large* 25*d)'* and *'RF Comb. (Large* 25*d)'*. For the first one, the performance is still exceptionally good but only requires a minimal amount of features. On the other hand, *'RF Comb. (Large* 25*d)'* provides the best performance but requires multiple times more features — especially a mixture of classical and Deep ELA features. Nonetheless, a sample size of 25d and the large Deep ELA model seem to be (generally speaking) superior and will be our main focus for future work.

6 Discussion & Conclusion

In this paper, we have analyzed the complementarity between classical and Deep ELA features as well as the substitutability of classical ELA by Deep ELA features. We additionally analyzed their performance for the task of automated algorithm selection, both when used individually and when the two are combined and used in a single model. Our experiments have shown that there is some amount of complementarity between Deep and classical ELA. While there is a certain degree of overlap, we have shown that the Deep ELA features provide additional information not captured by the classical ELA features and vice versa. This can be seen both in the complementarity analysis, as well as in the consistent improvement in algorithm selection performance when using a combined model of both classical and Deep ELA features. In general, all of the examined models achieved impressive algorithm selection results, consistent with prior work which shows great promise in algorithm selection on the BBOB problem set. Of the examined models, the combined model using classical ELA and the Large Deep ELA model trained on a sample size of 25d achieved the absolute best performance, this performance was not significantly better than some of its competitors. Especially, all models trained on the Medium 25d Deep

Table 2. Results of the AAS study. The numbers of selected features are in brackets; * indicates a result that is stochastically tied to the best selector with $\alpha = 0.05$ using the robust ranking technique as proposed by [6]. The first column (D) indicates the dimensionality of the problem while the second column (FGrp) indicates the function groups as defined in BBOB.

			Classical ELA				Large ($25d$)				Large ($50d$)				Medium ($25d$)				Medium ($50d$)			
D	FGrp	SBS (HCMA)	(23) kNN ($25d$)	(14) kNN ($50d$)	(17) RF ($25d$)	(23) RF ($50d$)	(10) kNN Deep	(85) kNN Comb.	(18) RF Deep	(54) RF Comb.	(28) kNN Deep	(39) kNN Comb.	(14) RF Deep	(25) RF Comb.	(11) kNN Deep	(57) kNN Comb.	(16) RF Deep	(19) RF Comb.	(18) kNN Deep	(23) kNN Comb.	(10) RF Deep	(47) RF Comb.
2	1	**3.71**	9.12	12.37	9.61	14.00	9.03	8.81	9.35	8.49	14.98	14.41	14.59	12.71	10.35	8.41	7.93	8.31	16.77	14.44	14.35	11.92
	2	5.80	3.23	3.61	3.26	3.52	3.26	**2.63**	2.78	2.65	3.54	3.54	4.42	3.53	3.01	3.11	2.65	3.45	3.51	3.53	4.68	3.73
	3	6.29	3.07	4.38	3.08	4.12	4.51	3.32	3.06	**2.73**	4.26	4.28	4.68	4.14	3.87	3.25	3.85	3.23	4.62	4.85	4.52	4.34
	4	25.34	6.74	6.85	6.63	6.81	7.31	7.04	7.11	5.72	4.36	4.48	5.04	6.21	4.76	5.99	5.92	5.78	4.00	4.47	5.35	**3.83**
	5	44.95	4.91	4.82	3.20	2.86	4.28	3.99	4.15	3.46	4.04	2.82	4.65	3.21	4.22	4.07	3.72	2.69	3.69	3.23	6.28	**2.54**
	all	17.69	5.50	6.52	5.23	6.38	5.78	5.26	5.40	**4.69**	6.35	6.00	6.77	6.06	5.33	5.05	4.90	4.74	6.64	6.21	7.13	5.33
3	1	356.10	10.98	15.53	**10.16**	15.07	10.79	11.43	11.11	10.36	15.96	15.75	16.08	14.68	10.88	10.75	10.97	11.09	16.00	16.04	15.77	15.40
	2	4.46	2.65	3.34	**2.49**	3.31	2.61	2.79	2.52	2.55	3.45	3.44	3.46	3.18	2.59	2.62	2.83	2.60	3.36	3.31	3.74	3.41
	3	4.98	2.55	4.05	2.59	3.72	3.88	2.72	2.56	**2.43**	3.75	3.76	3.85	3.80	2.88	2.60	3.17	2.92	3.72	3.63	3.85	3.88
	4	**2.63**	6.58	6.71	5.43	4.98	5.66	5.99	6.31	6.32	5.72	5.44	5.46	5.24	5.93	6.02	4.02	3.69	5.95	5.89	4.23	4.29
	5	66.81	2.56	2.71	2.09	2.02	2.53	2.64	2.33	2.39	1.92	2.16	4.86	**1.69**	2.77	2.62	2.12	2.03	2.72	2.79	2.94	2.86
	all	90.43	5.16	6.60	4.64	5.92	5.20	5.21	5.07	4.90	6.27	6.22	6.88	5.82	5.11	5.02	4.70	**4.54**	6.47	6.46	6.20	6.07
5	1	**11.99**	17.59	22.95	16.79	22.01	19.06	17.42	17.52	16.72	23.19	23.25	23.50	21.97	17.52	17.40	17.81	17.27	24.40	23.43	25.57	23.04
	2	3.90	3.07	3.30	2.85	4.19	3.65	3.54	3.61	4.11	2.58	2.77	3.28	3.36	2.49	2.49	**2.46**	2.47	3.71	3.70	3.38	2.98
	3	4.21	3.72	4.91	**3.17**	4.80	4.80	3.52	4.85	3.48	5.06	4.87	5.56	4.95	3.70	3.52	3.67	3.64	4.47	4.47	3.91	4.40
	4	4.29	4.86	4.15	4.00	3.86	4.28	4.17	4.09	3.95	3.95	3.89	4.50	3.44	2.96	3.78	3.70	3.09	4.00	3.89	2.88	**2.61**
	5	7.67	6.08	2.11	1.60	2.33	1.71	2.39	2.61	1.38	1.17	**1.16**	1.48	1.75	2.18	1.71	1.45	1.90	1.47	1.79	1.25	2.10
	all	6.52	7.23	7.66	**5.80**	7.57	6.83	6.32	6.66	6.00	7.38	7.37	7.84	7.25	5.91	5.92	5.96	5.81	7.77	7.61	7.56	7.19
10	1	**2.74**	9.54	16.34	8.63	15.42	9.54	9.60	9.50	8.60	16.76	16.58	16.20	15.41	-/-	-/-	-/-	-/-	-/-	-/-	-/-	-/-
	2	**2.16**	2.43	2.71	2.40	2.70	2.43	2.42	2.42	2.41	2.69	2.69	2.63	2.61	-/-	-/-	-/-	-/-	-/-	-/-	-/-	-/-
	3	**2.76**	3.62	4.20	2.79	3.72	3.62	3.60	3.59	2.83	4.88	4.56	4.06	3.67	-/-	-/-	-/-	-/-	-/-	-/-	-/-	-/-
	4	2.02	2.05	1.95	1.86	1.93	2.05	2.04	2.06	**1.85**	2.05	2.06	2.08	2.01	-/-	-/-	-/-	-/-	-/-	-/-	-/-	-/-
	5	23.64	1.74	1.78	1.68	1.76	2.16	1.77	4.76	**1.40**	2.15	2.07	2.49	1.50	-/-	-/-	-/-	-/-	-/-	-/-	-/-	-/-
	all	6.85	3.94	5.51	3.52	5.20	4.02	3.95	4.55	**3.46**	5.83	5.71	5.61	5.14	-/-	-/-	-/-	-/-	-/-	-/-	-/-	-/-
all	1	93.63	11.81	16.80	11.30	16.62	12.11	11.82	11.87	**11.04**	17.72	17.49	17.59	16.20	12.92	12.19	12.24	12.22	19.06	17.97	18.56	16.79
	2	4.08	2.84	3.24	2.75	3.43	2.99	2.84	2.83	2.93	3.06	3.11	3.45	3.17	2.70	2.74	**2.65**	2.84	3.53	3.51	3.93	3.37
	3	4.56	3.24	4.38	2.91	4.09	4.20	3.29	3.51	**2.87**	4.49	4.37	4.54	4.14	3.49	3.12	3.57	3.26	4.27	4.32	4.09	4.20
	4	8.57	5.06	4.91	4.48	4.40	4.83	4.81	4.89	4.46	4.02	3.97	4.27	4.22	4.55	5.27	4.55	4.19	4.65	4.75	4.15	**3.57**
	5	35.77	3.82	2.86	2.14	2.24	2.67	2.70	3.46	2.16	2.32	2.05	3.37	**2.04**	3.06	2.80	2.43	2.22	2.63	2.60	3.49	2.50
	all	30.37	5.46	6.57	4.80*	6.27	5.46	5.19	5.42	**4.76***	6.46	6.33	6.78	6.07	5.45*	5.33*	5.19*	5.03*	6.96	6.76	6.97	6.20

ELA models are stochastically tied to the Large $25d$ model, indicating superior performance of the $25d$ models versus the $50d$ ones.

The models trained using only the Deep ELA features also performed well, with some achieving performance that is stochastically tied to the best-performing model, and with one particular model achieving such performance using only 11 features total. The fact that the Deep ELA features were able to match the performance of the classical features makes them an important candidate for research into domains that cannot be tackled yet by classical ELA, such as multi-objective optimization, opening doors to a large amount of potential future research.

Another area that still needs to be assessed is the generalizability of Deep ELA to problems outside of the BBOB benchmark suite, where classical ELA features have been shown to struggle. In addition, Prager et al. [20] have demonstrated the successful application of cost-sensitive learning. In future work, we will investigate this approach more closely — especially in combination with Deep ELA features.

Finally, it will also be important to assess the complementarity of Deep ELA with other recently developed approaches for describing problem landscapes, such as TransOPT [3] and Topological Landscape Analysis [18]. However, the methods presented in this paper should be easily applicable to such a comparison, and we plan to conduct such an analysis in the future.

References

[1] Alissa, M., Sim, K., Hart, E.: Automated Algorithm Selection: from Feature-Based to Feature-Free Approaches. Journal of Heuristics **29**(1), 1–38 (2023)

[2] Belkhir, N., Dréo, J., Savéant, P., Schoenauer, M.: Per Instance Algorithm Configuration of CMA-ES with limited Budget. In: Proceedings of the 19th Annual Conference on Genetic and Evolutionary Computation (GECCO). pp. 681–688. ACM (2017)

[3] Cenikj, G., Petelin, G., Eftimov, T.: TransOpt: Transformer-based Representation Learning for Optimization Problem Classification (2023)

[4] Chen, X., Xie, S., He, K.: An Empirical Study of Training Self-Supervised Vision Transformers (2021)

[5] Derbel, B., Liefooghe, A., Vérel, S., Aguirre, H.E., Tanaka, K.: New Features for Continuous Exploratory Landscape Analysis based on the SOO Tree. In: Friedrich, T., Doerr, C., Arnold, D.V. (eds.) Proceedings of the 15th ACM/SIGEVO Conference on Foundations of Genetic Algorithms, FOGA. pp. 72–86. ACM (2019)

[6] Fawcett, C., Vallati, M., Hoos, H.H., Gerevini, A.E.: Competitions in AI – Robustly Ranking Solvers Using Statistical Resampling (Aug 2023)

[7] Guyon, I., Weston, J., Barnhill, S., Vapnik, V.: Gene Selection for Cancer Classification using Support Vector Machines. Machine learning **46**, 389–422 (2002)

[8] Hansen, N., Brockhoff, D., Mersmann, O., Tusar, T., Tusar, D., ElHara, O.A., Sampaio, P.R., Atamna, A., Varelas, K., Batu, U., Nguyen, D.M., Matzner, F., Auger, A.: COmparing Continuous Optimizers: numbbo/COCO on Github (Mar 2019). `https://doi.org/10.5281/zenodo.2594848`, `https://doi.org/10.5281/zenodo.2594848`

[9] Hansen, N., Finck, S., Ros, R., Auger, A.: Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions. Research Report RR-6829, INRIA (2009), `https://hal.inria.fr/inria-00362633`

[10] Jankovic, A., Popovski, G., Eftimov, T., Doerr, C.: The Impact of Hyper-Parameter Tuning for Landscape-Aware Performance Regression and Algorithm Selection. In: Proceedings of the 23rd Annual Conference on Genetic and Evolutionary Computation (GECCO). pp. 687–696 (2021)

[11] Kerschke, P., Hoos, H.H., Neumann, F., Trautmann, H.: Automated Algorithm Selection: Survey and Perspectives. ECJ **27**, 3 – 45 (2019)

[12] Kerschke, P., Trautmann, H.: Automated Algorithm Selection on Continuous Black-Box Problems by Combining Exploratory Landscape Analysis and Machine Learning. Evolutionary Computation **27**(1), 99–127 (2019), pMID: 30365386

[13] Kerschke, P., Trautmann, H.: Comprehensive Feature-based Landscape Analysis of Continuous and Constrained Optimization Problems using the R-package flacco. Applications in Statistical Computing: From Music Data Analysis to Industrial Quality Improvement pp. 93–123 (2019)

[14] Kostovska, A., Jankovic, A., Vermetten, D., de Nobel, J., Wang, H., Eftimov, T., Doerr, C.: Per-run Algorithm Selection with Warm-starting using Trajectory-based Features. In: Parallel Problem Solving from Nature – PPSN XVII: 17th International Conference, PPSN, 2022. pp. 46–60. Springer (2022)

[15] Mersmann, O., Bischl, B., Trautmann, H., Preuss, M., Weihs, C., Rudolph, G.: Exploratory Landscape Analysis. In: Proceedings of the 13th annual conference on Genetic and evolutionary computation (GECCO). pp. 829–836 (2011)

[16] Muñoz, M.A., Smith-Miles, K.: Generating New Space-Filling Test Instances for Continuous Black-Box Optimization. Evolutionary Computation (ECJ) **28**(3), 379–404 (2020)

[17] Petelin, G., Cenikj, G.: How Far Out of Distribution Can We Go With ELA Features and Still Be Able to Rank Algorithms? In: 2023 IEEE Symposium Series on Computational Intelligence (SSCI). pp. 341–346 (2023)
[18] Petelin, G., Cenikj, G., Eftimov, T.: TLA: Topological Landscape Analysis for Single-Objective Continuous Optimization Problem Instances. In: 2022 IEEE Symposium Series on Computational Intelligence (SSCI). pp. 1698–1705 (2022)
[19] Prager, R.P., Seiler, M.V., Trautmann, H., Kerschke, P.: Towards Feature-Free Automated Algorithm Selection for Single-Objective Continuous Black-Box Optimization. In: 2021 IEEE Symposium Series on Computational Intelligence (SSCI). IEEE (2021)
[20] Prager, R.P., Seiler, M.V., Trautmann, H., Kerschke, P.: Automated Algorithm Selection in Single-Objective Continuous Optimization: A Comparative Study of Deep Learning and Landscape Analysis Methods. In: Parallel Problem Solving from Nature–PPSN XVII: 17th International Conference, PPSN 2022. Springer (2022)
[21] Prager, R.P., Trautmann, H.: Nullifying the Inherent Bias of Non-invariant Exploratory Landscape Analysis Features. In: International Conference on the Applications of Evolutionary Computation (Part of EvoStar). pp. 411–425. Springer (2023)
[22] Prager, R.P., Trautmann, H.: Pflacco: Feature-Based Landscape Analysis of Continuous and Constrained Optimization Problems in Python. Evolutionary Computation pp. 1–25 (07 2023)
[23] Renau, Q., Dréo, J., Doerr, C., Doerr, B.: Expressiveness and Robustness of Landscape Features. In: Proceedings of the 21st Annual Conference on Genetic and Evolutionary Computation (GECCO) Companion. pp. 2048 – 2051. ACM (2019)
[24] Rice, J.R.: The Algorithm Selection Problem. Advances in Computers **15** (1976)
[25] Seiler, M.V., Kerschke, P., Trautmann, H.: Deep-ELA: Deep Exploratory Landscape Analysis with Self-Supervised Pretrained Transformers for Single- and Multi-Objective Continuous Optimization Problems *(First revision submitted to ECJ)* (2024)
[26] Seiler, M.V., Prager, R.P., Kerschke, P., Trautmann, H.: A Collection of Deep Learning-based Feature-Free Approaches for Characterizing Single-Objective Continuous Fitness Landscapes. In: Proceedings of the 24th Annual Conference on Genetic and Evolutionary Computation (GECCO) (2022)
[27] Seiler, M.V., Rook, J., Heins, J., Preuß, O.L., Bossek, J., Trautmann, H.: Using Reinforcement Learning for Per-Instance Algorithm Configuration on the TSP. In: 2023 IEEE Symposium Series on Computational Intelligence (SSCI). pp. 361–368. IEEE (2023)
[28] van Stein, B., Long, F.X., Frenzel, M., Krause, P., Gitterle, M., Bäck, T.: DoE2Vec: Deep-learning Based Features for Exploratory Landscape Analysis. In: Silva, S., Paquete, L. (eds.) Proceedings of the 25th Annual Conference on Genetic and Evolutionary Computation (GECCO) Companion. pp. 515–518. ACM (2023)
[29] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is All You Need. Advances in neural information processing systems **30** (2017)
[30] Vermetten, D., Ye, F., Bäck, T., Doerr, C.: MA-BBOB: A Problem Generator for Black-Box Optimization Using Affine Combinations and Shifts. CoRR **abs/2312.11083** (2023)

Sustainable Development Index - using MILP to assign relative weight to different UNSDG parameters

Keyaan Shah[1][0000-1111-2222-3333]

[1] Dhirubhai Ambani International School, Mumbai 400098, India;
1keyaanshah@gmail.com

lncs@springer.com

Abstract. This report presents a novel approach to quantifying and comparing national efforts towards the United Nations Sustainable Development Goals (UNSDGs) by developing a Sustainable Development Index (SDI) using Mixed-Integer Linear Programming (MILP). The methodology uniquely integrates MILP to optimize the weighting of various sustainability indicators, offering a robust and mathematical foundation for the SDI. This approach not only captures the discrete and linear aspects of sustainability indicators but also aligns the resulting index with existing measures such as the Human Development Index (HDI) and Environmental Performance Index (EPI). Through meticulous data collection, normalization, and MILP-based analysis, the report examines 17 unique indicators, each representing a specific UNSDG. The study employs advanced optimization tools, including Gurobi Optimizer and Julia programming language, to compute the SDI, ensuring an objective and reliable comparison of countries' sustainability performances. The report reveals insightful findings, with the SDI offering a clear and quantifiable measure of progress towards the UNSDGs. The outcome of this research has significant implications for policymakers, researchers, and the public, providing a nuanced understanding of national sustainability efforts. It underscores the utility of MILP in addressing complex global challenges and sets a precedent for future research in sustainable development assessment.

Keywords: Sustainable Development Index (SDI), Mixed-Integer Linear Programming (MILP), Mathematical optimization in sustainability

1 Introduction

Sustainable development, first clearly defined in the 1987 Brundtland Report by the United Nations Brundtland Commission, has been at the forefront of global agendas for over three decades (United Nations, n.d.). This foundational document underscored the urgent need for strategies that harmonize development

with environmental sustainability. Since then, the concept has expanded to include three interconnected dimensions—environmental, social, and economic sustainability—which collectively address the multifaceted challenges of our global community. The adoption of the United Nations Sustainable Development Goals (UNSDGs) by all Member States in 2015 further solidified the world's commitment to this agenda. These goals offer a comprehensive blueprint for peace and prosperity for people and the planet, now and into the future.

This report introduces a novel quantitative framework—the Sustainable Development Index (SDI)—using Mixed-Integer Linear Programming (MILP) to analyze and compare national efforts toward achieving the UNSDGs. MILP is an optimization framework that is crucial for assigning appropriate weights to diverse sustainability indicators, thus forming a robust mathematical base for the SDI. This method ensures that both the discrete decisions and the continuous inputs are optimally integrated, reflecting the multi-faceted nature of sustainability metrics. The SDI is meticulously developed through a methodology that integrates MILP to optimize the weighting of various sustainability indicators, thereby providing a robust and mathematically sound foundation for national sustainability assessments. This integration is crucial as it encapsulates the linear and non-linear aspects of sustainability measures and aligns them with established indices like the Human Development Index (HDI) and the Environmental Performance Index (EPI).

The organization of this paper is as follows: After this introduction, the background section expands on the evolution of the sustainability concept and the development of the UNSDGs. It also details the role of MILP in optimization challenges, which is critical for the theoretical underpinning of the SDI. The subsequent sections methodically discuss the MILP-based formulation of the SDI, the data collection and normalization processes employed, and the index calculation. The results section then presents the findings from the application of this methodology, offering insights into national performances and disparities in sustainability efforts.

The main contribution of this paper lies in its innovative approach to quantifying sustainability through advanced mathematical programming. Unlike typical assessments that may rely on subjective or unevenly weighted indicators, the SDI provides a scalable, objective measure of sustainability that can be tailored for comparative analysis across different national and regional contexts. This paper does not only contribute to academic discourse but also serves as a practical tool for policymakers and researchers who aim to evaluate and enhance sustainability initiatives effectively.

2 Background and Literature Review

The literature on the use of MILP in sustainable development highlights its effectiveness in optimizing complex, multi-dimensional problems [1]. Studies

have shown MILP's capabilities in applications ranging from optimizing energy systems and resource management to evaluating economic development and environmental performance [2]. By integrating these methodologies with the UNSDG framework, this paper advances the field of sustainable development assessment, proposing a rigorous and scalable approach to measuring and comparing national sustainability efforts [3].

A review of related literature demonstrates a growing recognition of the need for robust tools to assess sustainability. The SDI developed in this report is informed by previous works that have utilized similar optimization techniques to address global challenges. For example, the HDI, introduced by the United Nations Development Programme in 1990, and the EPI, developed by Yale University, have both provided foundational models for integrating various indicators into a cohesive assessment tool [4,5]. This report builds upon these methodologies, using MILP to refine and weight sustainability indicators comprehensively, thereby enhancing the ability to conduct detailed analyses of how countries are performing against the UNSDGs.

Thus, the integration of MILP into the development of the SDI represents a significant advancement in the quantitative assessment of sustainability. By leveraging this advanced mathematical tool, the SDI not only provides a reliable and objective measure of national sustainability efforts but also sets a precedent for future research and policy-making in sustainable development.

3 MILP & Sustainable Development

To effectively assess and track countries' sustainability efforts against the United Nations Sustainable Development Goals (UNSDGs), it's essential to create a Sustainable Development Index (SDI) using a rigorous MILP model. MILP's integration captures both discrete and continuous elements of sustainability metrics, optimizing for goal alignment and exploring solutions to derive optimal index values for each country. This method lays a solid mathematical groundwork for the SDI, ensuring objective and reliable assessments of national sustainability achievements. This SDI should be formulated using a combination of the parameters and indicators associated with the UNSDGs. By integrating these indicators using MILP methods, the index can objectively assess each country's performance. The SDI development process incorporates key aspects of MILP such as:

i. ***Indicator Selection:*** Choosing relevant and measurable indicators from the global indicator framework that reflect the progress toward each SDG.

ii. ***Data Collection and Normalization:*** Gathering and normalizing data for each indicator to ensure comparability across different countries and over time.

iii. ***Weighting of Indicators using MILP:*** Assigning appropriate weights

to indicators based on their relative importance to sustainable development. Utilizing MILP for computing weights for different indicators.

iv. ***Index Calculation:*** Computing SDI based on weights and indicator values

The SDI computed by following this approach can serve as a valuable tool for policymakers, researchers, and the public, offering a clear and quantifiable measure of sustainability performance and progress towards the UNSDGs.

4 Methodology

The SDI, calculated using MILP, leverages data from global databases relevant to the SDGs. To streamline computational demands while ensuring a broad analysis, one indicator was selected from the available 231 for each SDG, resulting in 17 distinct indicators evaluated worldwide. The HDI and EPI also played roles in determining the weights for these indicators in the SDI computation. The HDI, created by the UN Development Programme in 1990, assesses development with a focus on life expectancy, education, and income rather than merely economic growth [3]. Conversely, the EPI, developed by Yale University, quantifies the environmental performance of national policies, focusing on human health and ecosystem protection, divided into two main categories with various indicators [4].

For the SDI calculation using MILP, the latest 2021 data was primarily used, though the most recent data available before 2021 was used for countries lacking 2021 data. Indicator values were normalized between 0 and 1 to standardize data for weight determination and SDI calculation. Microsoft Excel facilitated data preparation, while Gurobi Optimization solved the optimization problem using the Julia programming language. Additionally, HiGHS was employed to confirm Gurobi's results for consistency. Gurobi Optimization is renowned for its advanced linear programming solvers, capable of efficiently and robustly tackling complex optimization challenges. Julia, known for its speed and user-friendliness, is a high-level, high-performance programming language ideal for technical computing, including mathematics, statistics, and data analysis tasks [5].

The MILP model was formulated with an objective function to minimize the difference between the SDI and a combined measure of HDI and EPI. Constraints were imposed to ensure the sum of weights equals one and to bound individual weights within specific ranges to avoid extreme values. The model also incorporated constraints to ensure that each indicator had a minimal threshold value to maintain balance. Multiple MILP runs were conducted to refine the index. Each run adjusted constraints and weights to achieve an optimal balance. The process included:

- Run 1: Initial MILP with basic constraints, resulting in weights that

highlighted disparities among indicators.

- Run 2: Excluded duplicate indicators and refined constraints based on initial findings.
- Run 3: Introduced positive and negative constraints to reflect the intended impact of each indicator on sustainability.
- Run 4: Added minimal threshold values for all indicators to ensure no indicator was disregarded.
- Run 5: Further refined to remove redundant constraints, optimizing alignment with HDI and EPI.
- Run 6: Final run with constraints limiting extreme values to ensure a balanced and equitable weighting of indicators

5 Computational Analysis

5.1 Indicator Selection and Data Collection

The data for this report was primarily sourced from the Our World in Data website, a reputable and comprehensive platform providing global data and insights aimed at tackling major challenges such as poverty, disease, hunger, and climate change. It includes a dedicated section for the Sustainable Development Goals (SDGs), with current data on 232 SDG indicators, and collaborates with a global network of scholars to offer well-researched and accessible information [6]. Table 1 shows the indicators considered as representative of the respective SDGs for the purpose of this report.

Table 1. List of UNSDGs

UNSDG	Indicator		UNSDG	Indicator
i. No Poverty	*Poverty headcount ratio*		x. Reduced Inequality	*Gini coefficient*
ii. Zero Hunger	*Prevalence of undernourishment*		xi. Sustainable Cities and Communities	*Access to Clean Water & Sanitation*
iii. Good Health and Well-being	*Mortality rate, under-5*		xii. Responsible Consumption / Production	*Domestic material consumption per capita*
iv. Quality Education	*Literacy rates*		xiii. Climate Action	*Annual CO2 emissions per capita*
v. Gender Equality	*Gender Inequality Index*		xiv. Life Below Water	*Marine Protected Area*

vi. Clean Water and Sanitation	*Access to Clean Water Access to Sanitation*		xv. Life on Land	*Forest Cover Ratio*
vii. Affordable and Clean Energy	*Share of renewable energy in electricity consumption*		xvi. Peace, Justice, and Strong Institutions	*Corruption Perception Index*
viii. Decent Work and Economic Growth	*Gross Domestic Product (GDP) per capita*		xvii. Partnerships for the Goal	*% of government expenditure on education*
ix. Industry, Innovation, and Infrastructure	*Patent applications per million people*			

In addition to the above, other sources of information were used as follows:

- Population of all countries from the World Bank database [7]
- EPI from Yale University's website [4]

5.2 Data Normalization

The dataset considered for this report has been filtered by making adjustments for materiality from the global database. In order to conduct meaningful analysis with reasonable data availability and impact on a larger segment of the world population, countries with a population of 2.5 million and higher have been considered for this report. In addition, only for which countries HDI and EPI are published have been considered. Accordingly, 133 out of 191 countries covering 97.8% of world population have been considered.

Table 2. Dataset considered for creating the network

	Number of countries	Population	GNI (2021 / USD bn)
Aggregate	191	7,872 mn	83,966
Excluded	58	173 mn	2,500
Included	133	7,699 mn	81,466
Percentage included	70%	98%	97%

For undertaking the MILP, one of the key steps involves the normalization of data by assigning values between 0 and 1. This is achieved using one of several methods:

- Min-Max Normalization: This method rescales each indicator based on relative position between the minimum and maximum values of the indicator.
- Z-Score Normalization: This involves standardizing the data points based on their mean and standard deviation, although it doesn't strictly rescale to 0-1.
- Proportional Scaling: This method involves dividing each value by the sum of all values in its dataset.

Equation used for computations for this report for Min-Max Normalization:

$$Normalized\ Value\ x\ of\ indicator\ y = \frac{(Value(x) - Min(y))}{(Max(y) - Min(y))} \quad (1)$$

5.3 Application of MILP

In creating the MILP problem, the primary goal is to develop SDI that evaluates and compares countries based on their performance towards the UNSDGs. The MILP model incorporates constraints to ensure that the solution is both feasible and reflective of real-world conditions. These constraints include the normalization bounds of the indicators, limitations on the sum of the weights to prevent skewness, and the integrality constraints for decision variables. The objective is to find an optimal set of weights for various SDI indicators such that the resulting index aligns closely with HDI and EPI.

MILP for assigning weights to compute the SDI – Run 1: MILP to calculate Weights for all Indicators such that the SDI has maximum alignment with EPI + HDI subject to the following constraints: i. Sum of all weights is 1; ii. All individual weights are range bound between +1 and -1

Objective: Calculate values for W_n such that

$$Minimize \sum_{i=0}^{133} \left(SDI_i - (HDI_i + EPI_i)\right)^2, \text{ where } SDI_i = \sum_{n=1}^{19} (W_n \times I_n), \text{ where } \textit{(2)}$$

W_n=Weight for Indicator;I_n=Value for respective Development Indicator

Subject to the following constraints:

$$i. \sum_{n=1}^{19} W_n = 1;$$

$$ii. W_n \leq 1$$

$$iii. W_n \geq -1$$

Summarizing outcome of the MILP solution on Gurobi Optimizer in Table 3.

Table 3. Outcome Summary - Run 1

Dimension: 1 row x 19 columns x 19 non-zeros Quadratic objective terms: 190	• Objective Values[1]: **2.4985** • Objective Ratio[2]: 1.97%

Weights for each indicator based on this optimization run are shown in Table 4.

Table 4. Weights for all indicators as per Table 2– Run 1

SDG / Indicator	**Weight**	**SDG / Indicator**	**Weight**
1/ Poverty headcount ratio	-0.147	10/ Gini coefficient	0.121
2/ Undernourishment	0.056	*11a/ Access to Clean Water*	-0.020
3/ Mortality rate, under-5	0.109	*11b/ Access to Sanitation*	-0.050
4/ Literacy rates	0.787	12/ Domestic material consumed	-0.170
5/ Gender Inequality Index	-0.398	13/ Annual emissions per capita	-0.074
6a/ Access to Clean Water	0.121	14/ EPI -Marine Protected Area	0.227
6b/ Access to Sanitation	-0.020	15/ Forest Cover Ratio	0.052
7/ Share of renewable energy	-0.008	16/ Corruption Perception Index	0.610
8/ GDP per capita	0.587	17/ Govt expenditure - education	-0.070
9/ Patent applications	-0.710		

Indicators 6a and 6b are common with 11a and 11b, and by including them twice, the weights could be skewed. In order, the address this, the model optimization was ran again removing the duplicate entries, and its outcome is summarized in Table 5

Table 5. Outcome Summary- Run 2

Dimension: 1 row x 17 columns x 17 non-zeros Quadratic objective terms: 153	• Objective Values: **2.4985** • Objective Ratio: 1.97%

[1]Objective Value = Sum of least squares as per the objective function
[2]Objective Ratio = Objective value / Sum all EPI+HDI

Table 6. Weights for all indicators by excluding duplicate indicators - Run 2

SDG / Indicator	Weight	SDG / Indicator	Weight
1/ Poverty headcount ratio	-0.147	9/ Patent applications	-0.710
2/ Undernourishment	0.056	10/ Gini coefficient	0.121
3/ Mortality rate, under-5	0.109	12/ Domestic material consumed	-0.170
4/ Literacy rates	0.787	13/ Annual emissions per capita	-0.074
5/ Gender Inequality Index	-0.398	14/ EPI -Marine Protected Area	0.227
6a/ Access to Clean Water	0.121	15/ Forest Cover Ratio	0.052
6b/ Access to Sanitation	-0.020	16/ Corruption Perception Index	0.610
7/ Share of renewable energy	-0.008	17/ Govt expenditure - education	-0.070
8/ GDP per capita	0.587		

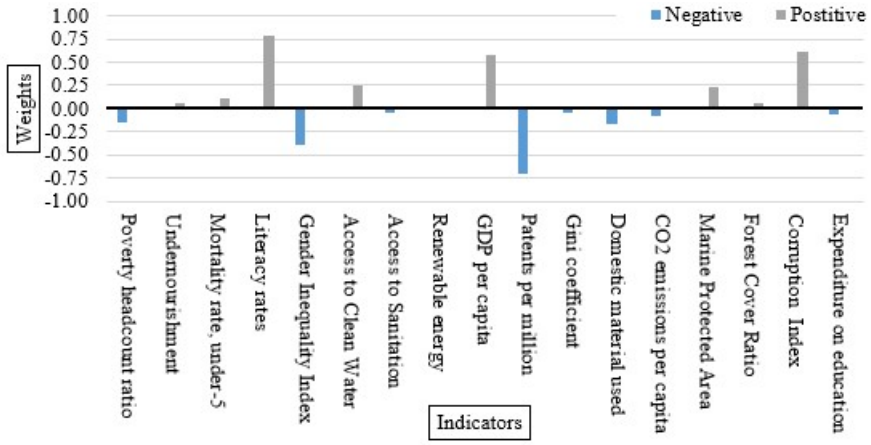


Fig. 1. Weights for indicators considered as per the MILP Run 2

Table 6 and Figure 1 illustrate that the MILP model produces both positive and negative weights based on the defined constraints, affecting the SDI values accordingly. However, this approach might not accurately reflect the intended impact of certain indicators on sustainability. For example, Patent Applications and Access to Sanitation should positively influence the SDI, whereas Undernourishment and Infant Mortality Rate should detract from it. This discrepancy suggests a need to refine the MILP model by introducing additional constraints tailored to the specific nature of each indicator. Table 7 categorizes each of the indicators based on its positive or negative contribution to sustainable development.

Table 7. Categorization of indicators as sustainability positive or negative

SDG / Indicator	**Weight**	**SDG / Indicator**	**Weight**
1/ Poverty headcount ratio	Negative	9/ Patent applications	Positive
2/ Undernourishment	Negative	10/ Gini coefficient	Negative
3/ Mortality rate, under-5	Negative	12/ Domestic material consumed	Positive
4/ Literacy rates	Positive	13/ Annual emissions per capita	Negative
5/ Gender Inequality Index	Negative	14/ EPI -Marine Protected Area	Positive
6a/ Access to Clean Water	Positive	15/ Forest Cover Ratio	Positive
6b/ Access to Sanitation	Positive	16/ Corruption Perception Index	Negative
7/ Share of renewable energy	Positive	17/ Govt expenditure - education	Positive
8/ GDP per capita	Positive		

The MILP problem restated to include constraints based on the above categorization of indicators is presented hereunder.

Restated MILP for assigning weights with positive/negative constraints – Run 3: MILP to calculate Weights for all Indicators such that the SDI has maximum alignment with EPI + HDI subject to the following constraints: **i.** Sum of all weights is 1; **ii.** Individual weights are range bound; **iii.** Negative and Positive values based on the Indicator

Objective: Calculate values for W_n such that

$Minimize \sum_{i=0}^{133} \left(SDI_i - (HDI_i + EPI_i)\right)^2, where\ SDI_i = \sum_{n=1}^{17} \left(W_n \times I_n\right), where$ *(3)*

W_n = Weight for Indicator n; I_n = Value for respective Development Indicator

Subject to the following constraints:

$$i. \sum_{n=1}^{17} W_n = 1;$$

$$ii.\ \text{If } l_n = Positive, 0 \leq W_n \leq 1$$

$$iii.\ \text{If } l_n = Negative, 0 \geq W_n \geq 1$$

The MILP generates revised weights are shown in Table 9 and Figure 2.

Table 8. Outcome Summary- Run 3

Objective Values: 4.44513	Objective Ratio: 3.50%

Table 9. Weights for all indicators with positive-negative constraints - Run 3

SDG / Indicator	Weight	SDG / Indicator	Weight
1/ Poverty headcount ratio	0.0000	9/ Patent applications	-0.710
2/ Undernourishment	-0.1379	10/ Gini coefficient	0.121
3/ Mortality rate, under-5	0.0000	12/ Domestic material consumed	-0.170
4/ Literacy rates	0.9656	13/ Annual emissions per capita	-0.074
5/ Gender Inequality Index	-0.5210	14/ EPI -Marine Protected Area	0.227
6a/ Access to Clean Water	0.3165	15/ Forest Cover Ratio	0.052
6b/ Access to Sanitation	0.0000	16/ Corruption Perception Index	0.610
7/ Renewable energy	0.0000	17/ Govt expenditure - education	-0.070
8/ GDP per capita	0.5451		

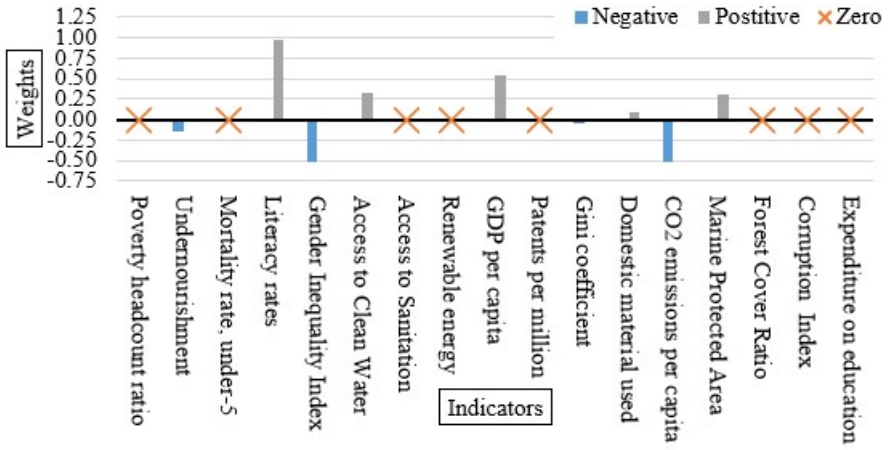


Fig. 2. Weights for Indicators with positive-negative constraints - Run 3

This MILP run has 8 out of 17 indicators with 0 value. The model is thus disregarding several variables. To have a balanced view, the model should be constrained to ensure that each indicator has a certain minimal value. Accordingly, the MILP model further revised to include constraints based minimal value for all indicators is as follows.

Restated MILP for assigning weights with threshold value for all Indicators – Run 4: MILP to calculate Weights such that SDI has alignment with EPI + HDI subject to the following: **i.** Sum of all weights is 1; **ii**. All weights range bound; **iii**. Negative and Positive values based on Indicator; and **iv.** Minimal threshold values for all Indicators.

Objective: Calculate values for W_n such that

$Minimize \sum_{i=0}^{133} \left(SDI_i - (HDI_i + EPI_i)\right)^2, where\ SDI_i = \sum_{n=1}^{17} \left(W_n \times I_n\right), where$ *(4)*

W_n = Weight for Indicator n; I_n = Value for respective Development Indicator

Subject to the following constraints:

$$i. \sum_{n=1}^{17} W_n = 1;$$

$$ii.\ \text{If } l_n > 0, \text{ then } 0.05 \leq W_n \leq 1$$

$$iii.\ \text{If } l_n < 0, \text{ then } -0.05 \geq W_n \geq -1$$

The MILP generated revised weights are shown in Table 11 and Figure 3.

Table 10. Outcome Summary- Run 4

Objective Values: 5.0393	Objective Ratio: 3.97%

Table 11. Weights for all indicators with threshold value constraints - Run 4

SDG / Indicator	**Weight**	**SDG / Indicator**	**Weight**
1/ Poverty headcount ratio	-0.0500	9/ Patent applications	0.0500
2/ Undernourishment	-0.1404	10/ Gini coefficient	-0.0737
3/ Mortality rate, under-5	-0.0500	12/ Domestic material consumed	0.0890
4/ Literacy rates	0.9388	13/ Annual emissions per capita	-0.4773
5/ Gender Inequality Index	-0.4764	14/ EPI -Marine Protected Area	0.3104
6a/ Access to Clean Water	0.2717	15/ Forest Cover Ratio	0.0500
6b/ Access to Sanitation	0.0500	16/ Corruption Perception Index	-0.0500
7/ Renewable energy	0.0500	17/ Govt expenditure - education	0.0500

8/ GDP per capita	0.4580		

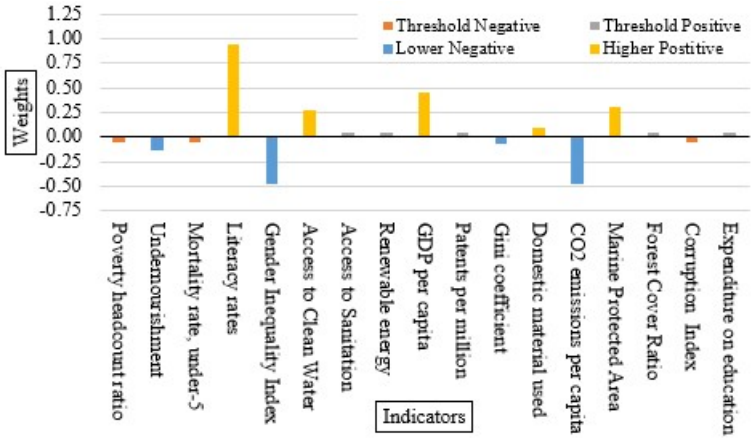


Figure 3. Weights for Indicators with threshold value constraints - Run 4

As observed in Table 10, the Objective Value and Objective Value Ratio have nearly doubled as compared to Run 2, making the outcome suboptimal. This is because multiple constraints have been added to the function. The MILP can be further modified to exclude constraints which may be redundant as shown hereunder.

Restated MILP for assigning weights after removing redundant constraints– Run 5: MILP to calculate Weights for all Indicators such that the SDI has maximum alignment with EPI + HDI subject to the following constraints: **i.** Negative and Positive values based on category of the Indicator; **ii.** Minimal threshold values for all Indicators

Objective: Calculate values for W_n such that

$Minimize \sum_{i=0}^{133} \left(SDI_i - (HDI_i + EPI_i)\right)^2, where\ SDI_i = \sum_{n=1}^{17} \left(W_n \times I_n\right), where$ *(5)*

W_n = Weight for Indicator n; I_n = Value for respective Development Indicator

Subject to the following constraints:

$$i.\ \text{If } l_n > 0, \text{ then } 0.05 \leq W_n$$

$$ii.\ \text{If } l_n < 0, \text{ then } -0.05 \geq W_n$$

The MILP generated revised weights are shown in Table 13 and Figure 4.

Table 12. Outcome Summary- Run 5

Objective Values: **2.92**	Objective Ratio: 2.31%

Table 13. Weights for all indicators after removing redundant constraints - Run 5

SDG / Indicator	Weight	SDG / Indicator	Weight
1/ Poverty headcount ratio	-0.0500	9/ Patent applications	0.0500
2/ Undernourishment	-0.0500	10/ Gini coefficient	-0.0500
3/ Mortality rate, under-5	-0.0500	12/ Domestic material consumed	0.0500
4/ Literacy rates	0.6744	13/ Annual emissions per capita	-0.2352
5/ Gender Inequality Index	-0.2084	14/ EPI -Marine Protected Area	0.2705
6a/ Access to Clean Water	0.2724	15/ Forest Cover Ratio	0.0600
6b/ Access to Sanitation	0.0500	16/ Corruption Perception Index	-0.0500
7/ Renewable energy	0.0500	17/ Govt expenditure - education	0.0500
8/ GDP per capita	2.0261		

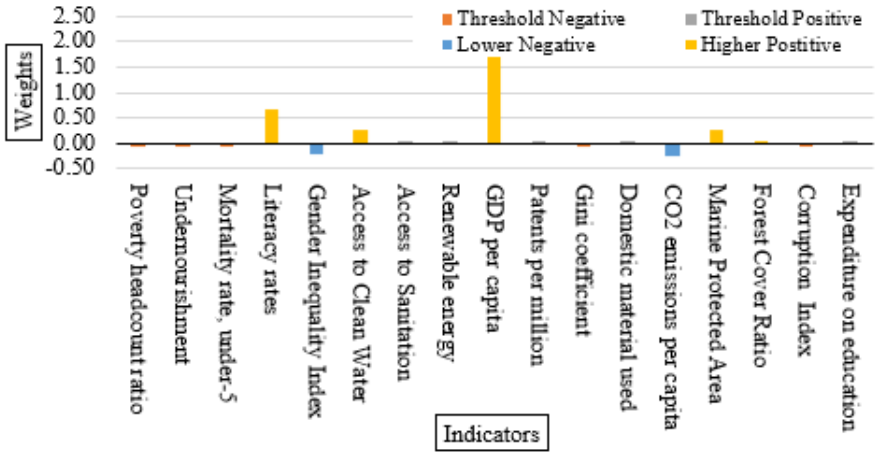


Figure 4. Weights for Indicators after removing redundant constraints - Run 5

The outcome of this MILP run 5 considers all the parameters almost equitably, with extra emphasis on a few indicators on either side. For example, GDP per capita and Literacy rates are distinctly much higher than the other positive

indicators. On the other side, Gender Inequality and CO_2 emission are the most negative indicators. One thing that comes out starkly is the significant disparity of weights of indicators, especially on the positive side.

Table 14. Range of weights computed in Run 5

	Positive Indicators	**Negative Indicators**
	Lowest / Average / Highest +ve Weight	Lowest / Average / Highest -ve Weight
Calculated Values	0.0500 / 0.3553 / 2.0261	-0.0500 / -0.0991 / -0.2352
Ratio Highest/Lowest	40.5x	4.7x
Ratio: Highest/Average	5.7x	2.4x

Such extreme values give extra emphasis on few indicators at the cost of ignoring the others. Hence, it would be pertinent to optimize the differences of weights across indicators. The MILP can be further modified to include constraints which restrict extreme values as shown below.

Restated MILP for assigning weights to include constraints for restricting extremities – Run 6: MILP to calculate Weights for all Indicators such that the SDI has maximum alignment with EPI + HDI subject to the following constraints: i. Negative and Positive values based on category of the Indicator; ii. Minimal threshold values for all Indicators; iii. Maximum value for all Indicators 10x the minimum threshold value

Objective: Calculate values for W_n such that

$Minimize \sum_{i=0}^{133} \left(SDI_i - (HDI_i + EPI_i)\right)^2, where\ SDI_i = \sum_{n=1}^{17} \left(W_n \times I_n\right), where$ *(6)*

W_n = Weight for Indicator n; I_n = Value for respective Development Indicator

Subject to the following constraints:

$$i.\ \text{If } l_n > 0, \text{ then } 0.05 \leq W_n \leq 0.50$$

$$ii.\ \text{If } l_n < 0, \text{ then } -0.05 \geq W_n \geq -0.50$$

The MILP generated revised weights after including the constraints as above, are shown in Table 16 and Figure 5.

Table 15. Outcome Summary- Run 6

Objective Values: **4.2577**	Objective Ratio: 3.36%

Table 16. Weights for indicators after adding constraints to limit extremities-Run 6

SDG / Indicator	Weight	SDG / Indicator	Weight
1/ Poverty headcount ratio	-0.0500	9/ Patent applications	0.4162
2/ Undernourishment	-0.0500	10/ Gini coefficient	-0.0500
3/ Mortality rate, under-5	-0.0500	12/ Domestic material consumed	0.2143
4/ Literacy rates	0.5000	13/ Annual emissions per capita	-0.0500
5/ Gender Inequality Index	-0.3404	14/ EPI -Marine Protected Area	0.3441
6a/ Access to Clean Water	0.4738	15/ Forest Cover Ratio	0.0917
6b/ Access to Sanitation	0.0500	16/ Corruption Perception Index	-0.0500
7/ Renewable energy	0.1119	17/ Govt expenditure - education	0.0500
8/ GDP per capita	0.5000		

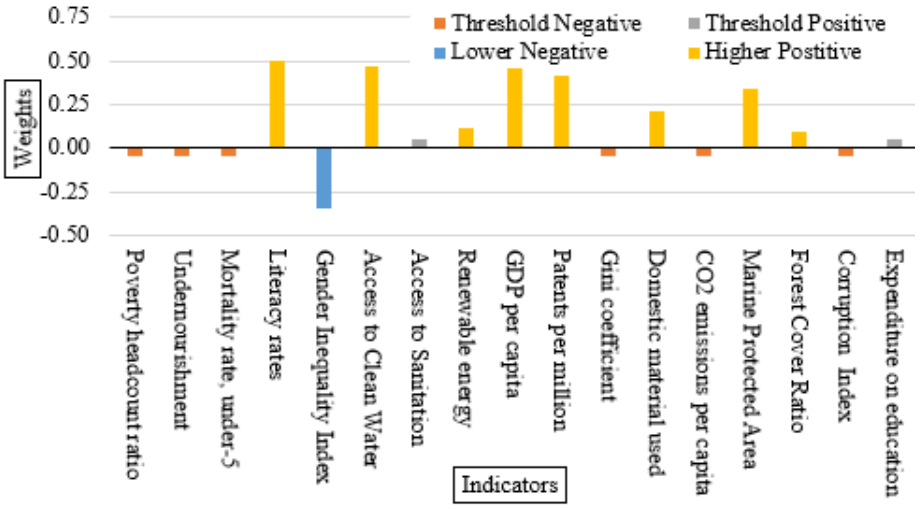


Figure 5: Weights for Indicators after adding constraints to limit extremities-Run 6

Summarizing outcomes of Runs 1 to 6 in Table 17 for a comparative assessment of each and select the one with optimum values for weights of indicators.

Table 17. Summary of MILP optimization runs

Run#	Objec -tive Terms	Object- ive Value	Objec- tive Ratio	Average Modulus Value	Std. Devn. Modulus Values	Positive Indicators		Negative Indicators	
						Highest / Lowest	Highest/ Average	Highest / Lowest	Highest/ Average
Run 1	190	2.4985	1.97%	0.23	0.25	15.2x	2.65x	86.7x	4.2x
Run 2	153	2.4985	1.97%	0.26	0.26	15.2x	2.36x	86.7x	3.8x
Run 3	153	4.4451	3.50%	0.20	0.28	10.1x	4.3x	11.1x	3.0x
Run 4	153	5.0393	3.97%	0.21	0.25	18.8x	4.1x	9.6x	2.5x
Run 5	153	2.9296	2.31%	0.25	0.49	40.5x	5.7x	4.7x	2.4x
Run 6	153	4.2577	3.36%	0.18	0.20	10.0x	1.8x	6.8x	3.7x

From Table 17, Run 6 comes across as one that gives an optimal solution. While the Objective Value is moderately high at 4.25, the value is not very significant since the objective ratio is reasonably low at 3.36%. All indicators have at the very least, a threshold value and extreme values have been restricted. Further, this run considers all variables equitably with average modulus value of 0.18 and standard deviation of 0.20.

5.4 Creating the SDI

The SDI for all countries based on the weights listed in Table 16 is presented in below in Table 18.

Table 18. SDI based on the Weights from MILP Run 6

Place	**SDI | EPI+HDI**
Finland	**1.7** | 1.94
Japan	1.7 | 1.58
Sweden	1.69 | 1.89
Denmark	1.66 | 1.98
Norway	1.66 | 1.68
Australia	1.63 | 1.68
New Zealand	1.62 | 1.6
U.S.A.	1.61 | 1.47
Canada	1.59 | 1.48
Germany	1.58 | 1.7
Netherlands	1.56 | 1.7
Chile	1.55 | 1.28
Belgium	1.54 | 1.62
Lithuania	1.52 | 1.47
France	1.5 | 1.64
U.K.	1.49 | 1.94
Spain	1.48 | 1.54
Poland	1.46 | 1.39

Place	SDI | EPI+HDI
Switzerland	1.45 | 1.8
U.A.E.	1.44 | 1.48
Austria	1.41 | 1.73
Croatia	1.4 | 1.52
Italy	1.38 | 1.54
Brazil	1.37 | 1.05
Romania	1.36 | 1.38
Ireland	1.34 | 1.62
Singapore	1.32 | 1.5
Mexico	1.29 | 1.09
Portugal	1.28 | 1.36
Panama	1.28 | 1.26
Ecuador	1.26 | 1.08
Colombia	1.26 | 1.03
Qatar	1.24 | 1.05
Czechia	1.24 | 1.57
Greece	1.23 | 1.5
Ukraine	1.23 | 1.19
Bulgaria	1.22 | 1.27
Slovakia	1.22 | 1.5
Belarus	1.2 | 1.23
Dominica	1.2 | 1.05
Kazakhstan	1.19 | 1.11
Hungary	1.18 | 1.41
Bosnia	1.18 | 1.03
Costa Rica	1.18 | 1.2
Israel	1.18 | 1.42
Serbia	1.17 | 1.14
Armenia	1.16 | 1.14
Turkiye	1.14 | 0.91
Paraguay	1.14 | 0.94
Georgia	1.13 | 1.06
Albania	1.13 | 1.19
Argentina	1.13 | 1.17
China	1.13 | 0.82
Malaysia	1.12 | 0.99
South Africa	1.11 | 0.87
Thailand	1.1 | 1.04
Turkmenistan	1.1 | 0.92
Uruguay	1.09 | 1.04
Mongolia	1.07 | 0.79
Kyrgyzstan	1.07 | 0.81
Russian Fed.	1.06 | 1.07
Saudi Arabia	1.05 | 1.17

Place	SDI | EPI+HDI
Azerbaijan	1.05 | 0.95
Uzbekistan	1.05 | 0.91
Moldova	1.05 | 1.06
Kuwait	1 | 1.17
Honduras	1 | 0.7
Bolivia	0.99 | 0.88
Tajikistan	0.98 | 0.82
Vietnam	0.98 | 0.56
Cuba	0.97 | 1.14
Jordan	0.95 | 0.99
Peru	0.95 | 1
Venezuela	0.93 | 0.99
Indonesia	0.92 | 0.71
El Salvador	0.91 | 0.87
Lao PDR	0.88 | 0.58
Philippines	0.88 | 0.71
Tunisia	0.87 | 0.96
Oman	0.86 | 0.94
Sri Lanka	0.84 | 0.95
Botswana	0.84 | 1.12
Nepal	0.84 | 0.53
Jamaica	0.83 | 1.01
Iraq	0.82 | 0.66
Lebanon	0.81 | 0.77
Namibia	0.8 | 0.93
Iran	0.8 | 0.93
Egypt	0.77 | 0.87
Guatemala	0.76 | 0.56
Nicaragua	0.73 | 0.8
Myanmar	0.72 | 0.34
Algeria	0.69 | 0.8
Morocco	0.66 | 0.67
Bangladesh	0.65 | 0.54
Sudan	0.64 | 0.35
Cambodia	0.63 | 0.54
Zambia	0.62 | 0.63
India	0.61 | 0.42
Congo	0.6 | 0.67
Ghana	0.59 | 0.57
Zimbabwe	0.57 | 0.81
Cameroon	0.56 | 0.51
Burundi	0.54 | 0.25
Malawi	0.53 | 0.58
Senegal	0.52 | 0.46

Place	SDI | EPI+HDI
Pakistan	0.5 | 0.36
Rwanda	0.5 | 0.48
Angola	0.48 | 0.53
Gambia	0.47 | 0.48
Togo	0.45 | 0.51
Uganda	0.45 | 0.52
Kenya	0.44 | 0.52
Nigeria	0.43 | 0.41
DR of Congo	0.41 | 0.45
Tanzania	0.41 | 0.53
Mali	0.4 | 0.22
Afghanistan	0.38 | 0.57
Ethiopia	0.33 | 0.4
Mozambique	0.32 | 0.31
Liberia	0.31 | 0.25
Guinea	0.3 | 0.34
Mauritania	0.29 | 0.44
Sierra Leone	0.27 | 0.38
Benin	0.26 | 0.41
Cote d'Ivoire	0.25 | 0.51
Haiti	0.24 | 0.37
Madagascar	0.19 | 0.34
P. N. Guinea	0.13 | 0.39
Burkina Faso	0.1 | 0.38
Chad	0.06 | 0.16
C African Rep	0.03 | 0.46
Niger	-0.03 | 0.33

Table 19. Countries with Top 10 & Bottom 10 SDI

Top 10	SDI	SDI Rank	EPI+ HDI	EPI+HDI Rank	Bottom 10	SDI	SDI Rank	EPI+ HDI	EPI+HDI Rank
Finland	1.70	1	1.94	3	Niger	-0.03	133	0.33	128
Japan	1.70	2	1.58	15	C African Rep	0.03	132	0.46	112
Sweden	1.69	3	1.89	4	Chad	0.06	131	0.16	133
Denmark	1.66	4	1.98	1	Burkina Faso	0.10	130	0.38	121
Norway	1.66	5	1.68	9	PN Guinea	0.13	129	0.39	119
Australia	1.63	6	1.68	10	Madagascar	0.19	128	0.34	126

Top 10	SDI	SDI Rank	EPI+HDI	EPI+HDI Rank	Bottom 10	SDI	SDI Rank	EPI+HDI	EPI+HDI Rank
New Zealand	1.62	7	1.60	14	Haiti	0.24	127	0.37	122
United States	1.61	8	1.47	26	Cote d'Ivoire	0.25	126	0.51	108
Canada	1.59	9	1.48	23	Benin	0.26	125	0.41	116
Germany	1.58	10	1.70	8	Sierra Leone	0.27	124	0.38	120

The above data shows several top ranked countries as per SDI were lower on EPI+HDI ranking, and conversely some of the countries in the bottom 10 ranked marginally better on EPI+HDI ranking. These positions seem to have changed because the SDI considers additional indicators listed below in Table 20 which are not included either in EPI or HDI which could account for the swing in the relative ranking of countries.

Table 20. Indicators not included in EPI or HDI, but included for SDI

SDG / Indicator	**Weight**	**SDG / Indicator**	**Weight**
1 /Poverty ratio	-0.05	10 / Gini coefficient	-0.05
2 / Undernourishment	-0.05	12 / Domestic material consumption	0.21
5 / Gender Inequality Index	-0.34	16 / Corruption Perception Index	-0.05
9 / Patent applications	0.42	17 / Expenditure on education	0.05

5.5 Results

The MILP methodology significantly advanced the development of the SDI, offering deep insights into countries' progress toward the UNSDGs. By applying distinct weights to 17 key indicators, this approach effectively quantified national achievements, emphasizing the critical role of each indicator in the sustainability framework. This process resulted in a ranking system that highlighted sustainability leaders and those countries needing to direct more attention to specific SDGs. Moreover, the SDI facilitated comparisons with other indices like the HDI and EPI, uncovering unique insights and differences that emphasize the SDI's all-encompassing approach to sustainable development. A sensitivity analysis pinpointed indicators with substantial effects on the overall SDI score, providing policymakers with actionable information to enhance performance in areas lagging behind. Altogether, MILP-based SDI emerges as a nuanced and quantifiable tool for assessing sustainable development.

6 Conclusions & Future Works

The SDI's development through MILP marks a significant step in accurately assessing global progress toward the UNSDGs, leveraging MILP to refine and weight sustainability indicators comprehensively. This process analyzed 17 key indicators, each linked to a specific UNSDG, to highlight their roles and global sustainability impact. The SDI offers a nuanced view of global sustainability efforts, identifying leaders in certain SDGs and areas needing more focus. The MILP's meticulous approach ensures the SDI's reliability and relevance, providing a foundation for future evaluations and serving as a crucial tool for policymakers and researchers. This facilitates targeted sustainable development strategies and further studies. The research promotes a data-driven approach to enhance global sustainability efforts, making the MILP-based SDI a pivotal contribution to sustainable development goals.

Future research will focus on enhancing SDI with real-time, specific data, and dynamic modeling, involving advanced techniques like machine learning. Engaging with stakeholders for model validation and conducting longitudinal studies will track the SDI's long-term impact, providing a comprehensive view of sustainable development progress and areas for improvement. This effort positions the MILP application in creating the SDI as a significant advancement towards sustainable development, offering a detailed, adaptable framework for navigating evolving sustainability goals and data complexities, guiding future actions and research in this field.

Study Limitations: SDI's accuracy is contingent on the quality of data. In some cases, data might be outdated, incomplete, or inconsistent across different countries, potentially affecting the robustness of the index. Further, the process of assigning weights to various indicators, despite being systematic, might not fully reflect the complex interdependencies and varied impacts of different sustainability aspects. The SDI may not adequately account for regional and cultural differences in sustainability practices and priorities. This could lead to oversimplification. The long-term validity of the SDI is yet to be tested.

References

1. Gurobi Optimization. "Mixed-Integer Programming", www.gurobi.com/resources/mixed-integer-programming-mip-a-primer-on-the-basics/. Accessed 16 Nov. 2023.
2. MathWorks. "Mixed-Integer Linear Programming Algorithms" Www.mathworks.com, www.mathworks.com/help/optim/ug/mixed-integer-linear-programming-algorithms.html. Accessed 13 Nov. 2023.
3. Lashmar, Hayley. "The Human Development Index – a Better Indicator for Success?" SDG Action, United Nations Association – UK, 19 Mar.

2018, sdg-action.org/the-human-development-index-a-better-indicator-for-success/. Accessed 30 Nov. 2023.

4. Wolf, M. J., et al. "About the EPI | Environmental Performance Index." Epi.yale.edu, Yale Center for Environmental Law & Policy, New Haven, CT, 2022, epi.yale.edu/about-epi. Accessed 12 Nov. 2023.
5. Julia Programming Language. "Julia Documentation · the Julia Language." The Julia Project, Dec. 2023, docs.julialang.org/en/v1/. Accessed 20 Dec. 2023.
6. Roser, Max. "Measuring Progress towards the Sustainable Development Goals." Our World in Data, 18 July 2023, ourworldindata.org/sdgs. Accessed 1 Nov. 2023.]
7. The World Bank. "Population, Total." Worldbank.org, World Bank Group, 2022, data.worldbank.org/indicator/SP.POP.TOTL. Accessed 16 Jan. 2023.

An Evaluation of Domain-agnostic Representations to Enable Multi-task Learning in Combinatorial Optimisation

Christopher Stone[2][0000−0002−9512−9987], Quentin Renau[1][0000−0002−2487−981X], Ian Miguel[2][0000−0002−6930−2686], and Emma Hart[1][000−0002−5405−4413]

[1] Edinburgh Napier University, Edinburgh, Scotland, UK
{q.renau,e.hart}@napier.ac.uk
[2] University of St Andrews, Scotland, UK {cls29,ijm}@st-andrews.ac.uk

Abstract. We address the question of multi-task algorithm selection in combinatorial optimisation domains. This is motivated by a desire to simplify the algorithm-selection pipeline by developing a more general classifier that does not require specialised information per domain, and the potential for transfer learning. A minimum requirement to achieve this is to find a common representation for describing instances from multiple domains. We assess the strengths and weaknesses of three candidate representations (text, images and graphs) which can all be used to describe three different application domains. Two setups are considered: single-task selection where one classifier is trained per domain, each using the *same* representation, and multi-task selection where a single classifier is trained with data from all three domains to output the best solver per instance. We find that the domain-agnostic representations perform comparably with domain-specific feature-based classifiers with the benefit of providing a generic representation that does not require feature identification or computation, and could be extended to additional domains in future.

Keywords: Algorithm Selection, Multi-Task Learning, Combinatorial Optimisation.

1 Introduction

It is well known that for most combinatorial optimisation problems for which there are multiple solvers available, no single algorithm will perform best across all instances. Any portfolio of applicable solvers will exhibit *performance complementarity*, thus giving rise to the per-instance algorithm-selection problem. Although this was first articulated by Rice in 1976 [22], research in the field has accelerated over recent years, driven by advances in machine-learning, with the state-of-the-art summarised in a recent survey paper [12]. From the latter, it is clear that the most typical approach is to train one model per domain on a set of representative instances to predict the best solver from a portfolio. However, many practical domains falling under the umbrella of combinatorial

optimisation share characteristics, e.g. routing, scheduling and assignment are essentially ordering problems. This raises the possibility that knowledge encapsulated in different (but related) domains could be exploited to improve selection in per-instance algorithm selection. Multi-task learning (MTL) is of course well studied in the field of machine learning, specifically for classification (see [28]) but to the best of our knowledge has not been addressed in the context of algorithm selection. It has several potential benefits: it enables a system to handle multiple existing tasks with minimum software engineering effort and facilitates the possibility of being able to handle unanticipated tasks (i.e. new domains) that appear downstream (assuming they can be manipulated into the chosen representation).

A minimal requirement for multi-tasking is a unified (i.e. domain-agnostic) representation [9]. We therefore address the design of this component of an MTL system: how to find a representation that is common to multiple domains, so that in future, existing algorithmic 'machinery' can be reused without further engineering by domain experts. A unified representation for all domains has two benefits. First, it avoids having to define domain-specific features for every new domain in order to train classifiers. Second, with respect to a classifier, a single fixed-architecture neural network can be trained that accepts instances of different length and from multiple domains. Hence, there is no need to re-engineer the solving pipeline if new domains appear: instances from the new domain need only to be translated into the representation used. We consider three candidate domain-agnostic representations — text, images and graphs — each of which can be used to directly describe instances from multiple domains to train an algorithm selector. Three domains are used as a test-bed: TSP, knapsack, and graph colouring. We evaluate the strengths and weaknesses of the candidate representations in two pipelines. The first has two stages: given a single representation r a classifier is trained to perform *domain recognition* on the presented instance (i.e. output the domain). The instance is then passed to one of three specialised classifiers that outputs the best solver: crucially, each domain-specific classifier uses the *same input representation* r. This is denoted *single-task* selection. Second, we compare these results to a more streamlined pipeline in which a single *multi-task* classifier takes as input an instance from any of three domains (all represented using the same representation r) and outputs the best solver for that instance.

Results demonstrate that it is possible to design a single-task and multi-task system to handle a multi-domain problem using a single representation. We show that in multi-task settings, a trained algorithm selector never selects an algorithm from the incorrect domain for two of the three representations considered — even though domain labels are not provided. Furthermore, we show that in the case of image-based representation the drop in performance is below 1% when going from the single-task to multi-task settings. We therefore demonstrate the potential for cross-domain learning in minimising engineering effort, and propose that this lays a strong foundation for future development of a system that also supports more complex scenarios where new tasks appear sequentially.

2 Related Work

The vast majority of previous work in per-instance algorithm-selection relies on extracting domain-specific features from an instance before training a selector to map features to a solver [13] — a process which requires significant domain knowledge and can also be computationally costly [24]. To circumvent this issue, a recent line of work has begun to emerge that implements algorithm selection using *feature-free* methods, i.e. directly passing a description of an instance to a classifier [2,24,30,20,3]. Many of these approaches have been inspired by the success of deep learning models, particularly in the areas of computer vision and natural language processing [14]. For example, Alissa *et al* [2] use a *text-based* representation in conjunction with an Long-Short-Term-Memory (LSTM) network in bin-packing, showing it can outperform feature-based approaches. Feature-free approaches using *image-based input* tend to dominate the literature, exploiting deep convolutional neural networks as selectors. This is particularly prevalent in the TSP domain [18,24,30] due to the natural placement of coordinates on a 2-d map. Various approaches to image transformation are explored in order to improve classification accuracy, with results indicating the potential of the feature-free direction using images in algorithm selection tasks [24,30]. In [16], textual descriptions of SAT instances are converted to a grey-scale images and used to train an algorithm-selector. Outside of algorithm selection *Graph-Based neural networks (GNN)* have been shown to have general applicability [26] in classification tasks, with applications ranging from social nets, to computer vision to text classification. For example, Richter *et. al.* use a graph representation of programs when applying algorithm selection in the domain of software-validation [23].

Most algorithm selection is performed on a per-instance basis [12], where the instances belong to a single domain. In the field of machine-learning, Multi-Task Learning (MTL) focuses on learning multiple tasks simultaneously [28], usually by combining or blending features into a single input. Much of the work in this area focuses on leveraging knowledge from similar domains to improve performance. However, a more compelling need derives from the necessity to deal with additional, unanticipated tasks following training. For example, in the context of multi-task learning, [29] designed a multi-task learning framework to effectively learn robust feature representations by jointly optimising coarse-grained, fine-grained and ultra-fine-grained image classification tasks, reporting that the multi-task classifier outperformed the state-of-the-art.

This paper attempts to advance the field of multi-task algorithm selection by evaluating the strengths and weaknesses of three domain-agnostic representations: specifically, we compare images, text and graphs as candidate representations, drawing on previous literature in feature-free algorithm selection.

3 Representations

Recall that the goal of this study is to assess the strengths and weaknesses of three candidate representations in being able to represent instances from multiple

domains to conduct algorithm selection. The candidates are chosen (text, images, graphs) based on a review of the related literature. These are described below.

3.1 Text

The simplest of the three representations chosen is that no transformation of raw data is required: each instance (irrespective of the domain) is described in a comma-separated variable file and is therefore simply a sequence of characters. For TSP instances, one pair of coordinates is listed per row. For knapsack, a single value that specifies the size of the container is followed by rows containing pairs of profit/weights. For graph colouring, instances are described by edge lists in the form of pairs of values. None of the files contain meta-information that could potentially provide information to the classifier about the domain.

Note that for each of the domains, the information describing the instances (e.g. city coordinates) can be listed in the file in any order without altering the instance itself. However, the sequencing of characters might well have an influence on the behaviour of a text-based classifier depending on the type of machine-learning model used. For simplicity, we only consider a single description of each instance which has no explicit ordering of data imposed on it, and leave it to further work to understand whether training a classifier with versions of the same file containing different orderings is beneficial.

The limits of this representation depend on the number of tokens or characters accepted by the machine learning model. If one were to continue increasing the size of the instances beyond a certain point some special input concatenation method must be implemented in order to process larger instances.

3.2 Images

A transformation is required to turn the natural text-based description of an instance into an image. For TSP, the coordinates of each city are plotted as points. For knapsack, each item was plotted as a point, using the profit and weight of the item as its coordinates. The size of the knapsack was normalised to fall in the range of the items profit and weight values and plotted as an individual point. For graph colouring, the graph of each instance was transformed to an adjacency matrix of 0 and 1 values for adjacent/non-adjacent vertices.

All the images in all domains were transformed to greyscale with a resolution equal to 8 dpi (for graph colouring, the adjacency matrix is first transformed by converting 0 and 1 values to 0 and 255). The size of the images for all domains was set to 32×32, after a brief preliminary experimentation with both smaller and larger sizes. Examples of images produced from each domain are shown in Figure 1.

As discussed above with respect to the text representation, multiple images can be created for a single instance by rotating/flipping the original image. Again, for simplicity we restrict our transformation to creating a single image per instance from the textual description.

The input size of this representation is fixed regardless of the size of the instances, this means that arbitrary size instances can be processed by the machine learning model. However, beyond a certain point the lossy nature of the transformation would blur almost all information requiring higher resolution images and neural networks.

(a) TSP (b) GC (c) KP

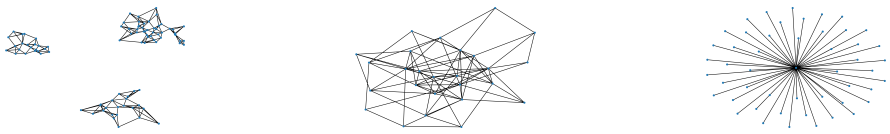
Fig. 1. Grey scale images of a selected instance from each domain. The TSP image uses the locations of the cities. The graph colouring image is a conversion of the corresponding adjacency matrix of the instance; a point indicates the presence of a connection. The knapsack image uses weight and profit values.

3.3 Graphs

All three domains can also be transformed into a graph representation with some manipulation.

TSP: A graph is constructed with cities as vertices. The coordinates of each city are assigned to each respective vertex feature vector. The TSP is naturally a complete graph with edge lengths equal to the Euclidean distance between two nodes. For simplicity instead of using all the edges we connect only the edges of 5 nearest neighbours for each vertex, as per previous literature [24], significantly decreasing the overall size of the graph. **Knapsack**: Each object is represented as a vertex, with a feature vector comprising the normalised profit and weight of the object. The knapsack is also represented as a vertex. Its associated feature vector contains the capacity plus an extra 0 used as padding to keep the dimensionality of knapsack and objects equivalent. An edge is added between each object and the knapsack. **Graph Colouring**: this is straightforward as the domain is naturally a graph. Each vertex is assigned its degree as its feature vector. Edges are added as specified by edge list in the problem description. An extra 0 is added to the vertices feature vector as padding in the experiments that consider all the domains at once to keep the dimensionality equal among all domains. Examples of graphs produced from each domain are shown in Figure 2. This representation is lossless therefore beyond a certain instance size the memory required to process the input will become too large to be processed.

In order to scale beyond that point some graph simplification or segmentation would need to be implemented.



(a) TSP (b) GC (c) KP

Fig. 2. Graphs generated from a selected instance from each domain. For TSP the plot retains the coordinates of the points. Otherwise a force based layout is used to plot the position of the vertices and highlight structures.

3.4 Feature-based representations

In order to provide a baseline to compare the use of domain-agnostic representations to domain-specific representations, we extract a set of common features for each domain and use these to train a random forest classifier as a baseline. Recall however the disadvantages of defining problem domain specific features described earlier in Section 2.

For TSP, we use the 64 features defined in the TSPMETA R package,[3] which describe the geometrical features of a TSP instance. For the knapsack, we used the feature extraction code supplied by MATILDA[4] which is implemented in Matlab and extracts 44 features (see [25]). For graph colouring, we also used the feature set (3 features) proposed by MATILDA, implementing them using the Python Networkx package [8] for networks and graph processing. The reader is referred to these publications for a detailed description of each feature.

4 Datasets

We reiterate that the goal is to evaluate the potential of each representation chosen to represent multiple domains, rather than develop state-of-the-art algorithm selectors or solvers. We opt to work with *two* solvers for each domain that return high-quality solutions quickly, and use balanced instance data-sets in which 50% of instances are best solved by each solver, in order to craft a discriminating instance set [1,7] to simplify training. A large set of instances is required per domain to train networks: we deemed it important for instances to vary in size to ensure generalisation and real-world applicability. While for each

[3] https://github.on.publication

[4] `https://matilda.unimelb.edu.au/matilda/`

domain there are many benchmark instances suites available both for TSP and GC, they do not meet all of the above criteria (e.g. TSP benchmarks often use state-of-the-art stochastic solvers with long running times [24], contain instances of the same size [24] or are too few in number and too large in size[21]). Hence, we use a mix of instance-generation and collation as described below per domain. All instances are provided in the accompanying github repository.[5]

4.1 Instance Generation

Graph Colouring - we create instances of graphs with n vertices ($10 \leq n \leq 200$). For each value of n 1K graphs are created, each with a uniform random number of edges between n and $(n^2)/2$. Extreme values that could produce empty or complete graphs are avoided as they would both be trivial to solve. Disjoint vertices and self-loops are removed if present.

TSP - to generate instances with varied structure we use the following method: c clusters are identified, where c is a random number in the range (1,10), and the centre of each is drawn from a uniform random distribution. Next, a distribution is randomly chosen to generate cities per cluster. Four distributions are considered: a 2D Gaussian distribution, a 2d grid, a streak of points in a line and a spiral. The first three distributions are taken from the work in [4] on evolving TSP instances. We add a spiral distribution as they are common in the ML literature and difficult to recognise using naive linear classifiers. The coordinates of n cities are generated, by drawing n/c samples from the chosen distribution centered on each cluster. 50K instances were created and solved.

Knapsack - for this domain, a dataset that met the criteria outlined above was available from the literature. Instances were collated from two sources: (1) the Pisinger repository[6], which includes varied instances in size and profit/weight coefficients [19]. 8,000 instances from the large, small coefficient, and hard instances ranging in size between 20-5,000 items were selected. (2) 4,000 instances (with fixed size 1,000 items) downloaded from MATILDA: see [25] for a detailed description of these instances.

4.2 Solvers

Solver selection is guided by the criteria in Section 4: to use solvers that produce high-quality results quickly, that provide differential performance in different regions of the instance space and have complementary performances with respect to criteria relevant to the real world (e.g. speed or quality). All solvers selected are deterministic and do not require extensive computation: thousands of instances from each domain could be solved in less than two hours. For the range of **knapsack** sizes in the instance set, several solvers are available that solve to optimality in milli-second timescales. We select the ExpKnap (EXP)

[5] GitHub supplied on acceptance

[6] available from: http://hjemmesider.diku.dk/ pisinger/codes.html

algorithm (based on branch and bound) and the Combo (COM) method (a dynamic programming approach) [17]. These have also been used in the MATILDA methodology for instance-space generation and algorithm selection of the knapsack domain [25]. For **TSP**, two constructive heuristics are chosen for their ability to quickly generate solutions. The Greedy heuristic (GRD) [11] is a well-known constructive heuristic algorithm for solving the TSP, in which the tour is constructed by adding the node that provides the minimal tour length and provides very fast solutions. The Christofides algorithm (CHR) [6] is an approximation algorithm based on minimum spanning trees. While slower than Greedy, it provides improved solutions.

As the graph-colouring instances are relatively small we also solve via constructive heuristics. The Largest First heuristic (LF) [15] is a well-known heuristic that assigns the colours by constructing each colour class one at a time. DSATUR(DST) [5] is a greedy algorithm based on the saturation degree of the vertices.

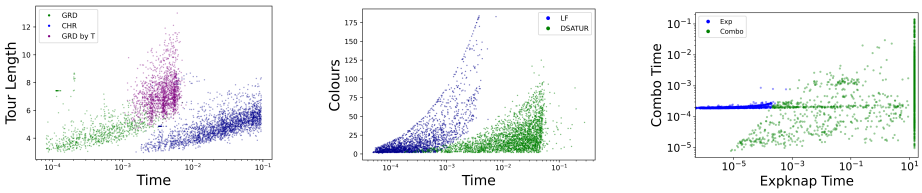
4.3 Instance Labelling

Instances were labelled using a method that accounted for both solving time and objective value (as both are relevant criteria from a practical perspective) as follows.

- Knapsack: All instances are solved to optimality. Those that required more than 15 seconds to solve (a criteria set by MATILDA) were excluded. The winner is then determined by solving time only.
- TSP: a cut-off time for solving was set, equal to the average of the recorded times to solve all instances using the slower heuristic method Christofides (0.095 seconds). If only one algorithm solves the instance within this time it is labelled as the winner; otherwise, the winner is the algorithm that provides the shortest tour length.
- Graph colouring: the algorithm that uses the fewest colours wins, with time used to break ties.

The instances generated by the methods described in Section 4.1 were solved by both solvers in each domain. 2,500 instances of each label were then randomly selected for each domain, resulting in a balanced set of 5,000 instances per domain.

Figures in 3 illustrate the performance of the solvers per domain on the 5,000 instances selected for each of them. For TSP/GC, this is shown on two plots from the perspective of solving time and objective value, to illustrate the diverse behaviour of each heuristic and how they occupy different regions of the performance space. In the case of TSP instances we highlight in purple the instances that are won by the Greedy heuristic due to the timeout of the CHR heuristic as opposed to the length of the tour. In the case of KP instances we plot the time of each heuristic against each other as this is the only criteria used to rank them.



(a) TSP (b) GC (c) KP

Fig. 3. Each dot is an instance coloured and positioned based on the winning solver's performance. *Left*: TSP instances. Purple dots win by timeout of the other solver. *Centre*: GC instances. *Right*: KP instances.

5 Experimental Method

Figure 4 summarises the approaches used to investigate the relative performance of each representation in single-task and multi-task classification. For experiments using image and text representations as input, classifiers were developed using AutoKeras [10], an AutoML Python framework based on Keras. The framework provides a set of well-established architectures and parameters from the literature for both image and text recognition tasks, saving considerable effort in selecting architectures and parameter tuning. The specific architectures used are available via GitHub previously mentioned. For the graph representation, Graph Neural Network (GNN) classifiers were developed using StellarGraph[7]. The GNN uses a well-established Deep Graph Convolutional Neural Network taken from [27], we modify their parameters only in the dense layer, which is reduced to 64 units from 128 in order to reduce the memory footprint. All neural networks are trained for 100 epochs. Finally, to conduct baseline experiments using extracted features for comparison with the domain-agnostic representations, we use a Random Forest (RF) classifier, implemented using scikit-learn with default settings which creates an ensemble of 100 estimators. Throughout, classifiers were trained using cross-validation with 10 folds. Results are reported using the average accuracy and F1-scores[8] of the 10-folds.

6 Results

Experiments were conducted to: (1) compare the classification accuracy/F1-score values obtained by training a classifier per *representation* to output the *domain* associated with an instance (i.e. *task-recognition*); (2) compare the classification accuracy/F1-score obtained by training one classifier per domain (each with the *same* representation) to select the best solver for the instance (*Single task classification*); (3) determine whether a single classifier trained once with instances

[7] `https://github.com/stellargraph/stellargraph`

[8] combines the precision and recall of a classifier into a single metric by taking their harmonic mean

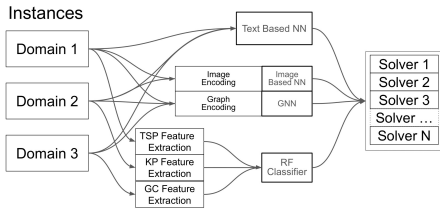


Fig. 4. Data pipelines: for single-task experiments, only one domain and its two corresponding solvers are wired at a time.

Table 1. Domain recognition results averaged over 10 folds. Standard deviation in parenthesis.

Representation	Accuracy	F1-score
Text	92.8(0.9)%	0.93 (0.01)
Images	100.0(0.00)%	1.00 (0.00)
Graph	100.0(0.00)%	1.00 (0.00)

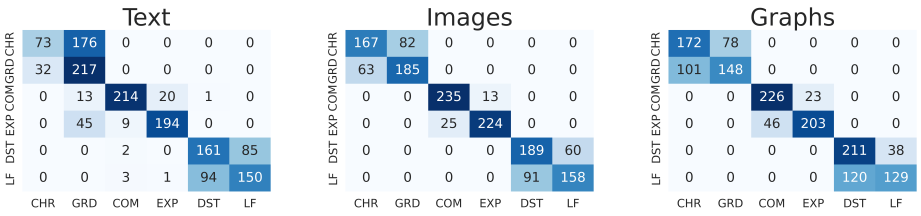
from *all three domains* with a specific representation can select the best solver for a given instance (*multi-task classifier*). Experiments (2) and (3) were repeated for each of the three chosen representations.

6.1 Task Recognition

To determine whether a classifier with a domain-agnostic input representation can identify the domain associated with an instance (task recognition), AutoKeras was used to train one classifier for each representation using 15K instances collated from the three domains (5K from each domain). A single integer representing the domain is used as output. The results (Table 1) show that the classifier identifies the correct domain with very high accuracy for all three representations. The graph and image representations result in 100% averaged accuracy over the 10 folds. There are obvious differences in structure/patterns between the three domains when represented as graphs (Fig. 2) which poses a trivial challenge for the classifier regarding domain prediction, in fact for all training rounds the accuracy reaches 100% by the 5th epoch. This is less obvious for image representations (Fig.1) where differences are more subtle and creating instances from two different domains that are indistinguishable after transformation is technically possible. These results provide confidence that given an instance from an unknown domain defined by a domain-agnostic representation, this step could be followed by single-task algorithm-selection to identify an appropriate solver. This is explored in the next section.

6.2 Single-Task Algorithm Selection

These experiments compare the performance of each *representation* when used as input to a domain-specific classifier to select the most appropriate solver for



(a) TSP (b) GC (c) KP

Fig. 5. Confusion Matrices in Multi-task settings. Rows represent the winning solver, columns represent the solver that was selected on average over 10 folds.

the instance. We measure the accuracy and the F1 score of each classifier. Results are reported in Table 2, which shows the average accuracy and F1-scores of the 10 folds for the single-task classifier per representation per domain. Ranking each representation based on F1-scoring, we observe that:

- **TSP**: *Image* > *Graph* > *Text*
- **Knapsack**: *Text* > *Image* > *Graph*
- **Graph Colouring**: *Graph* > *Image* > *Text*
- **Average Across Domains**: *Image* > *Graph* > *Text*

The relative strengths and weaknesses of each representation appear strongly domain-related, with the image-based representation doing best on average. We highlight that the largest difference in performance across representations is recorded in the TSP domain with values going from 0.60(0.02) for Text to 0.72(0.02) for images. While in the KP domain the difference is smallest, going from 0.89(0.01) for Graphs to 0.93(0.01) for Text. Table 2 also shows the results of learning a feature-based classifier in each domain for the single-task problem. Recall that useful features are defined using considerable domain-knowledge/expertise and have been the subject of much academic literature in the past. However, all four classifiers perform comparably on the Knapsack problem. This is particularly surprising for the textual representation, considering that *no* pre-processing or domain knowledge is provided. This demonstrates that for some domains and specific instance sets features can be learned directly by the classifier with good performance regardless of the representation used. In the case of the graph colouring domain both graphs and images perform on a par with the feature-based method, providing further evidence of the viability of using a domain-agnostic representation. The feature-representation clearly surpasses all three domain-agnostic representations for TSP: this is perhaps the domain where there has been most effort in collating useful features (64 features are used from the R tsplib package) and therefore the result is not surprising.

Table 2. Mean accuracy and macro F1-score over 10 folds. Performance that are above or on par to features are highlighted in bold.

Representation	Domain	Accuracy %	St.d.	F1-Score	St.d
Text	TSP	60.38	2.47	0.60	0.02
Text	KP	92.66	1.04	**0.93**	0.01
Text	GC	64.08	2.56	0.64	0.03
Text	Avg-Single	72.3	2.11	0.72	0.02
Text	Multi-Task	67.49	1.10	0.67	0.01
Images	TSP	71.80	1.75	0.72	0.02
Images	KP	92.18	0.93	0.92	0.01
Images	GC	**69.96**	2.21	**0.70**	0.02
Images	Avg-Single	**77.98**	1.63	**0.78**	0.02
Images	Multi-Task	**77.40**	1.15	**0.77**	0.01
Graphs	TSP	65.92	2.72	0.66	0.03
Graphs	KP	89.04	1.34	0.89	0.01
Graphs	GC	**71.22**	3.66	**0.71**	0.04
Graphs	Avg-Single	75.39	2.57	0.75	0.02
Graphs	Multi-Task	72.81	1.22	0.72	0.01
Features	TSP	99.38	0.36	0.99	0.03
Features	KP	93.46	0.89	0.91	0.01
Features	GC	69.90	2.16	0.70	0.02
Features	Avg-Single	87.58	1.14	0.87	0.02

6.3 Multi-Task Algorithm Selection

Finally, we investigate multi-task algorithm selection, i.e. training a single classifier simultaneously on all instances from multiple domains (where all instances are defined using the same representation), with the goal of selecting the best solver for each instance. 15k instances (5k from each domain) labelled with a single integer output representing the solver were used as training data.

Average accuracy/F1 calculated over the 10 test-fold is also shown in Table 2. Comparing the three representations directly, we observe the relative ranking remains unvaried with respect to the single task performance where on average *Images* > *Graph* > *Text*. It should be noted that all three representations produce an accuracy which is lower than the corresponding average over three domains for each representation obtained in the single-task experiments. However, we highlight that in the case of the image-based learner the drop in performance, measured by the F1-score, only goes from 0.78(0.02) to 0.77(0.01) due to a decrease in accuracy in the CHR class alone as shown in Table 3. The largest decrease in performance is observed in the text-based representation which goes from 0.72(0.01) to 0.67(0.01), while graphs sit in the middle going from 0.75(0.02) to 0.72(0.01). It can be observed from the same table how TSP is the domain that causes the most decrease in performance across all representations when going from single-task to multi-task setting.

Figure 5 shows confusion matrices for each representation in the multi-task setting, indicating how instances were misclassified. Rows represent the expected

Table 3. Solver wise F1-scores for single task and multi task models, averaged over 10 folds. T: Text, I:Images, G:Graphs. S:Single-Task, M:Multi-task.

Dom.	Solver	Features	T-S	T-M	I-S	I-M	G-S	G-M
TSP	CHR	0.99(0.01)	0.59(0.04)	0.41(0.04)	0.72(0.03)	0.69(0.03)	0.65(0.03)	0.65(0.05)
TSP	GRD	0.99(0.01)	0.62(0.02)	0.62(0.02)	0.72(0.02)	0.72(0.03)	0.67(0.04)	0.62(0.04)
KP	CMB	0.93(0.01)	0.92(0.01)	0.9(0.01)	0.92(0.01)	0.92(0.01)	0.9(0.01)	0.87(0.01)
KP	EXP	0.89(0.01)	0.93(0.01)	0.83(0.02)	0.92(0.01)	0.92(0.01)	0.88(0.02)	0.85(0.02)
GC	DST	0.70(0.01)	0.63(0.03)	0.63(0.03)	0.71(0.03)	0.71(0.02)	0.73(0.04)	0.73(0.02)
GC	LF	0.69(0.01)	0.65(0.03)	0.62(0.03)	0.68(0.02)	0.68(0.01)	0.69(0.05)	0.62(0.04)

classification and columns the average predicted classification over 10 folds. Remarkably both image and graph-based representations never select a solver from the wrong domain when misclassifying an instance. Differently from the task-recognition experiments, here no information was provided to the network regarding the specific domains, hence the grouping emerges spontaneously.

7 Conclusion

The goal of this paper was to investigate the strengths and weaknesses of three domain-agnostic representations (text, images, and graphs) for describing instances from multiple domains. As set out in the introductory section, a common representation is necessary to facilitate the introduction of new tasks to a multi-task selector in a manner that does not require significant engineering effort to develop a new classifier architecture specific to a domain, or the domain expertise necessary to extract features. A common representation can also facilitate cross-domain learning and makes steps towards a future vision of developing a general solver that can solve instances from a range of domains.

Results are presented in tables 2 and 3. While there is no overall 'winner', the *text*-based representation performs surprisingly well in the single-task KP domain and the *image*-based representation is on par with the feature-based methods in two out of three domains (KP, GC) in both single and multi-task settings. Although the multi-task classification task tends to provide lower accuracy/F1 than the related single-task experiments, in the *image*-based representation the drop in performance is almost negligible with the variance of the two results overlapping. Investigating how many instances were misclassified revealed that both *image*-based and *graph*-based representations made no mistakes in any of the test sets.

These findings highlight that there is still considerable research to be done to build optimisation systems that can effectively handle multiple tasks without having to significantly re-engineer systems if new tasks arrive — an inevitability in any real-world setting. This suggests that further work is required to both find appropriate domain-agnostic representations and also to develop new architectures specific to the multi-task setting (which may differ substantially from state-of-the-art approaches in domain-specific single-task setting).

The work presented only begins to scratch the surface with respect to creating a solver capable of handling multiple tasks, including those unanticipated during design. Specifically, we intend to conduct additional tests to understand how the system scales with the number of tasks it has to handle, as well as considering more complex scenarios where new tasks arrive sequentially, requiring the system to adapt to the new task. Finally, understanding the limitations of such an approach is also important — for example, by developing metrics that measure similarity between domains in order to quantify the amount of task 'diversity' a system might handle.

Acknowledgements This work is supported by UK Engineering and Physical Sciences Research Council grant nos. EP/V026534/1 and EP/V027182/1.

References

1. Akgün, Ö., Dang, N., Miguel, I., Salamon, A.Z., Spracklen, P., Stone, C.: Discriminating instance generation from abstract specifications: A case study with cp and mip. In: Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 17th International Conference, CPAIOR 2020. pp. 41–51 (2020)
2. Alissa, M., Sim, K., Hart, E.: Algorithm selection using deep learning without feature extraction. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 198–206 (2019)
3. Alissa, M., Sim, K., Hart, E.: Automated algorithm selection: from feature-based to feature-free approaches. Journal of Heuristics pp. 1–38 (2023)
4. Bossek, J., Kerschke, P., Neumann, A., Wagner, M., Neumann, F., Trautmann, H.: Evolving diverse tsp instances by means of novel and creative mutation operators. In: Proceedings of the 15th ACM/SIGEVO Conference on Foundations of Genetic Algorithms. pp. 58–71 (2019)
5. Brélaz, D.: New methods to color the vertices of a graph. Communications of the ACM **22**(4), 251–256 (1979)
6. Christofides, N.: Worst-case analysis of a new heuristic for the travelling salesman problem. Tech. rep., Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group (1976)
7. Dang, N., Akgün, Ö., Espasa, J., Miguel, I., Nightingale, P.: A framework for generating informative benchmark instances. In: 28th International Conference on Principles and Practice of Constraint Programming, CP. LIPIcs, vol. 235, pp. 18:1–18:18 (2022)
8. Hagberg, A., Swart, P., S Chult, D.: Exploring network structure, dynamics, and function using networkx. Tech. rep., Los Alamos National Lab.(LANL), Los Alamos, NM (United States) (2008)
9. Hart, E., Miguel, I., Stone, C., Renau, Q.: Towards optimisers that 'keep learning'. In: Proceedings of the Companion Conference on Genetic and Evolutionary Computation. pp. 1636–1638 (2023)
10. Jin, H., Song, Q., Hu, X.: Auto-keras: An efficient neural architecture search system. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp. 1946–1956. ACM (2019)
11. Johnson, D.S., McGeoch, L.A.: The traveling salesman problem: A case study in local optimization. Local search in combinatorial optimization **1**(1), 215–310 (1997)

12. Kerschke, P., Hoos, H.H., Neumann, F., Trautmann, H.: Automated algorithm selection: Survey and perspectives. Evolutionary computation **27**(1), 3–45 (2019)
13. Kotthoff, L.: Algorithm selection for combinatorial search problems: A survey. In: Data mining and constraint programming, pp. 149–190. Springer (2016)
14. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. nature **521**(7553), 436–444 (2015)
15. Leighton, F.T.: A graph coloring algorithm for large scheduling problems. Journal of research of the national bureau of standards **84**(6), 489 (1979)
16. Loreggia, A., Malitsky, Y., Samulowitz, H., Saraswat, V.: Deep learning for algorithm portfolios. In: Proceedings of the aaai conference on artificial intelligence. vol. 30 (2016)
17. Martello, S., Pisinger, D., Toth, P.: Dynamic programming and strong bounds for the 0-1 knapsack problem. Management science **45**(3), 414–424 (1999)
18. Miki, S., Ebara, H.: Solving traveling salesman problem with image-based classification. In: 2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI). pp. 1118–1123. IEEE (2019)
19. Pisinger, D.: Where are the hard knapsack problems? Computers & Operations Research **32**(9), 2271–2284 (2005)
20. Prager, R.P., Seiler, M.V., Trautmann, H., Kerschke, P.: Towards feature-free automated algorithm selection for single-objective continuous black-box optimization. In: 2021 IEEE Symposium Series on Computational Intelligence (SSCI). pp. 1–8. IEEE (2021)
21. Reinelt, G.: Tsplib—a traveling salesman problem library. ORSA journal on computing **3**(4), 376–384 (1991)
22. Rice, J.R.: The algorithm selection problem. In: Advances in computers, vol. 15, pp. 65–118. Elsevier (1976)
23. Richter, C., Hüllermeier, E., Jakobs, M.C., Wehrheim, H.: Algorithm selection for software validation based on graph kernels. Automated Software Engineering **27**(1), 153–186 (2020)
24. Seiler, M., Pohl, J., Bossek, J., Kerschke, P., Trautmann, H.: Deep learning as a competitive feature-free approach for automated algorithm selection on the traveling salesperson problem. In: International Conference on Parallel Problem Solving from Nature. pp. 48–64. Springer (2020)
25. Smith-Miles, K., Christiansen, J., Muñoz, M.A.: Revisiting where are the hard knapsack problems? via instance space analysis. Computers & Operations Research **128**, 105184 (2021)
26. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Philip, S.Y.: A comprehensive survey on graph neural networks. IEEE transactions on neural networks and learning systems **32**(1), 4–24 (2020)
27. Zhang, M., Cui, Z., Neumann, M., Chen, Y.: An end-to-end deep learning architecture for graph classification. In: Proceedings of the AAAI conference on artificial intelligence. vol. 32 (2018)
28. Zhang, Y., Yang, Q.: A survey on multi-task learning. IEEE Transactions on Knowledge and Data Engineering (2021)
29. Zhao, J., Peng, Y., He, X.: Attribute hierarchy based multi-task learning for fine-grained image classification. Neurocomputing **395**, 150–159 (2020)
30. Zhao, K., Liu, S., Yu, J.X., Rong, Y.: Towards feature-free tsp solver selection: A deep learning approach. In: 2021 International Joint Conference on Neural Networks (IJCNN). pp. 1–8. IEEE (2021)

Heuristic algorithms for the planar intermodal p-hub location: a possibilistic clustering approach*

Mario J. Basallo-Triana[1], Carlos J. Vidal-Holguín[2], Juan J. Bravo-Bastidas[2], and Yesid F. Basallo-Triana[3]

[1] School of Industrial Engineering, Universidad del Valle sede Buga, Carrera 13 No. 5–21, Guadalajara de Buga, Colombia. ario8906@gmail.com.
[2] School of Industrial Engineering, Universidad del Valle, Calle 13 No 100–00, Santiago de Cali, Colombia carlos.vidal@correounivalle.edu.co.
[3] Engineering Department, Universidad Autónoma de Occidente, Calle 25 No 115–85, Santiago de Cali, Colombia yesid.basallo@hotmail.com.

Abstract. The design of intermodal hub networks is of paramount importance in logistic operations involving multiple transportation modes like trains and trucks. In this work, we consider two non-linear optimization models for the intermodal p-hub network design problem assuming that hubs can be located anywhere in a two-dimensional space. The first formulation focuses on minimizing congestion in transport activities, while the second formulation aims to maximize hub usage using the concept of entropy. To solve the problem, we propose two heuristic algorithms which are generalizations of well-known possibilistic clustering algorithms. These heuristics are relatively easy to implement and provide a practical way to efficiently solve problems.

Keywords: Possibilistic clustering · Intermodal hub network design · Planar hub location · Lagrangian relaxation.

1 Introduction

Intermodal hub network design problems refer to the location of hub facilities and the allocation of demand or spoke, nodes to hubs to optimize some objective function that depends on the location of hubs and the routing of flows [4]. Hubs serve as consolidation centers with an inter-hub transport cost lower than the hub-and-spoke transport cost due to economies of scale. Hub location problems are complicated NP-hard optimization problems and there is a high interest in the research community in finding efficient solution algorithms for solving them. Surveys in hub location modeling include [2], [1], [3], and [7].

* This work was supported by Fondo de Ciencia Tecnología e Innovación of Sistema General de Regalías (FCTeI-SGR) of Colombia and Ministerio de Ciencia, Tecnología e Innovación (MINCIENCIAS) of Colombia (project code: BB1.112.957.930-1), and the Universidad del Valle, Cali, Colombia.

Compared with the discrete hub location research, the literature on planar hub location is very scarce. One of the first heuristic algorithms for solving the planar hub location was proposed by [14], who studied the location of two interacting hubs with single allocations. [13] considered a hard clustering approach for solving the planar hub location with single allocations. More recently, [9] proposed a probabilistic clustering approach for solving the planar hub location with direct transportation and multiple allocations. Previous authors consider an Euclidean distance metric to obtain the total distance in the objective function.

[6] developed a genetic algorithm for solving the single allocation planar hub location. [8] proposed an iterative Weszfield-type algorithm along with a particle swarm optimization algorithm for solving the planar hub-location routing problem with single allocations. For their part, [5] studied the theoretical properties and solution algorithms of several hub network structures.

In this work, we propose heuristic algorithms for solving the planar multiple allocation p-hub location problem with direct transportation decisions. We follow a possibilistic clustering approach. To the best of our knowledge, there are no studies considering possibilistic clustering approaches for hub location. We also consider two variants of the problem including congestion and entropy aspects. The proposed algorithms can be understood as generalizations of existing possibilistic clustering algorithms.

Section 2 proposes a heuristic algorithm for solving the hub location problem under congestion. Section 3 develops a heuristic algorithm considering an entropy-maximizing approach. Section 4 provides numerical experiments for evaluating the effectiveness and efficiency of the heuristics. Finally, Section 5 concludes the paper.

2 A congestion approach

We consider the planar p-hub location problem in which we have n nodes or points in a plane and we want to determine the coordinates for the optimal location of p hubs. The coordinates of the hubs do not need to coincide with a given subset of nodes and might be located anywhere in the plane. Each pair of nodes has an origin-destination (OD) demand $W_{ij} \geq 0$ associated with it. The demand between nodes i and j might be satisfied using truck-only transport or using the intermodal transportation option by performing transshipment operations at hubs. The advantage of using the intermodal transport option is the reduced transportation costs in the inter-hub transportation part, which is achieved due to the consolidation at hubs and the effect of the economies of scale at transport activities.

The planar hub location problem consists of locating p hubs in a continuous space, and determining direct and intermodal transport flows to minimize transportation costs, which are proportional to the transport distance and the amount of flow. In this paper, we consider the squared Euclidean distance as distance metric. Then, the distance between nodes i and j is:

$$d_{ij}^2 = \|\mathbf{s}_i - \mathbf{s}_j\|_2^2, \tag{1}$$

where $\mathbf{s}_i$ is the vector containing the coordinates of node i. We also refer to this quantity as the direct or truck-only transport distance, which can be used to satisfy the transport demand between nodes i and j. Alternatively, the demand between nodes i and j can be satisfied using the intermodal (train-truck) transport option through hubs l and m, which are located at points $\mathbf{v}_l$ and $\mathbf{v}_m$, respectively, the total squared Euclidean transport distance D^2_{ijlm} in this case is:

$$D^2_{ijlm}(\mathbf{v}_l, \mathbf{v}_m) = \|\mathbf{s}_i - \mathbf{v}_l\|_2^2 + \alpha\|\mathbf{v}_l - \mathbf{v}_m\|_2^2 + \|\mathbf{s}_j - \mathbf{v}_m\|_2^2, \tag{2}$$

where the first term denotes the collection distance from node i to hub l, which is referred to as collection distance and it is performed by trucks. The second term is the inter-hub distance between hubs l and m, which is affected by a discount factor α to account for the economies of scale in inter-hub train transportation. The last term is the final stage of the transport activities from hub m to destination node j, which is also referred to as distribution distance, which is again performed by trucks.

One aspect of interest in hub location theory is the mitigation of the impact of congestion, which increases transportation costs. There are many alternatives in the literature for the modeling of congestion (see [2]). In this work, we consider an alternative approach, which is based on the power law congestion function [10].

In our formulation, congestion is accounted for in each source of flow in the network, and then individual contributions of congestion are aggregated. In this sense, congestion costs are measured as a power function of the individual flows, with exponent $q > 1$. This differs from the traditional power law congestion function, which quantifies congestion costs according to the total flow instead (see [10]).

Let $x_{ijlm} \in [0, 1]$ and $y_{ij} \in [0, 1]$ be decision variables that represent the fraction of flow transported between nodes i and j using the intermodal (train-truck) and truck only transport option, respectively. For their part, $\mathbf{v}_l, l = 1, \ldots, p$, are 2 by 1 decision vectors used for selecting the coordinates (location) of hubs. The *congestion minimizing hub location problem* (CHL) is defined as follows as follows:

$$(CHL):$$

$$\min_{\mathbf{x},\mathbf{y},\mathbf{v}} \quad \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{l=1}^{p}\sum_{m=1}^{p}(W_{ij}x_{ijlm})^q D^2_{ijlm}(\mathbf{v}_l, \mathbf{v}_m)\sum_{i=1}^{n}\sum_{j=1}^{n}(W_{ij}y_{ij})^q d^2_{ij}, \tag{3}$$

$$\text{s.t.} \quad y_{ij} + \sum_{l=1}^{p}\sum_{m=1}^{p} x_{ijlm} = 1, \qquad \forall i, j = 1, \ldots, n, \tag{4}$$

$$x_{ijlm}, y_{ij} \geq 0, \quad \forall i, j = 1, \ldots n; l, m = 1 \ldots p. \tag{5}$$

Objective Function (3) measures the total cost for the direct and intermodal transport alternatives measuring the impact of congestion using a power law congestion function for each transport path. Constraints (4) guarantee that the

demand is satisfied exactly by the intermodal, direct transport, or by a combination of these alternatives. Constraints (5) establish the domain of the decision variables. It is not necessary to explicitly state that $x_{ijlm} \leq 1$ and $y_{ij} \leq 1$ given that those variables are non-negative implying that previous constraints are implicitly imposed by Constraints (4).

We consider a Lagrangian relaxation approach considering Constraints (4) as complicating constraints. The Lagrangian function of the Optimization Problem (3)–(5) is:

$$L = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{l=1}^{p}\sum_{m=1}^{p}(W_{ij}x_{ijlm})^q D_{ijlm}^2(\mathbf{v}_l,\mathbf{v}_m)\sum_{i=1}^{n}\sum_{j=1}^{n}(W_{ij}y_{ij})^q d_{ij}^2 - \sum_{i=1}^{n}\sum_{j=1}^{n}\lambda_{ij}\left(y_{ij}+\sum_{l=1}^{p}\sum_{m=1}^{p}x_{ijlm}-1\right). \tag{6}$$

Necessary conditions for x_{ijlm} and y_{ij} to be a minimum of the Problem (3)–(5) are:

$$\frac{\partial L}{\partial x_{ijlm}} = qW_{ij}^q D_{ijlm}^2 x_{ijlm}^{q-1} - \lambda_{ij} = 0, \tag{7}$$

$$\frac{\partial L}{\partial y_{ij}} = qW_{ij}^q d_{ij}^2 y_{ij}^{q-1} - \lambda_{ij} = 0. \tag{8}$$

Solving for x_{ijlm} and y_{ij} in Equations (7) and (8), we obtain:

$$x_{ijlm} = \left(\frac{\lambda_{ij}}{qW_{ij}^q D_{ijlm}^2}\right)^{\frac{1}{q-1}}, \tag{9}$$

$$y_{ij} = \left(\frac{\lambda_{ij}}{qW_{ij}^q d_{ij}^2}\right)^{\frac{1}{q-1}}. \tag{10}$$

The substitution of the expressions for x_{ijlm} and y_{ij} from Equations (9) and (10) into Constraint (4) produces:

$$\left(\frac{\lambda_{ij}}{qW_{ij}^q d_{ij}^2}\right)^{\frac{1}{q-1}} + \sum_{r=1}^{p}\sum_{s=1}^{p}\left(\frac{\lambda_{ij}}{qW_{ij}^q D_{ijrs}^2}\right)^{\frac{1}{q-1}} = 1 \tag{11}$$

Solving for λ_{ij} in Equation (11), we obtain:

$$\lambda_{ij} = \left(\left(qW_{ij}^q d_{ij}^2\right)^{\frac{1}{1-q}} + \sum_{r=1}^{p}\sum_{s=1}^{p}\left(qW_{ij}^q D_{ijrs}^2\right)^{\frac{1}{1-q}}\right)^{1-q}. \tag{12}$$

Note that we can substitute the value of λ_{ij} from Equation (12) into Equations (9) and (10) to compute x_{ijlm} and y_{ij}. Clearly, the lagrange multiplier λ_{ij} is

non-negaitive, and given that λ_{ij} satisfy (12), this guarantee that $x_{ijlm} \in [0,1]$ and $y_{ij} \in [0,1]$.

On the other hand, a necessary condition of $\mathbf{v}_l$ to be a minimum is:

$$\frac{\partial L}{\partial \mathbf{v}_l} = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{m=1}^{p} W_{ij}^q x_{ijlm}^q \left((1+\alpha)\mathbf{v}_l - (\mathbf{s}_i + \alpha \mathbf{v}_m)\right) = 0. \tag{13}$$

Solving for $\mathbf{v}_l$ in the above equation we obtain:

$$\mathbf{v}_l = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{m=1}^{p} W_{ij}^q x_{ijlm}^q (\mathbf{s}_i + \alpha \mathbf{v}_m)}{(1+\alpha)\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{m=1}^{p} W_{ij}^q x_{ijlm}^q}. \tag{14}$$

Equations (9), (10) and (14) are coupled and it is difficult to obtain a solution for the resulting system. Alternatively, it is possible to obtain an approximate solution of the system by an iterative procedure for computing x_{ijlm}, y_{ij} and $\mathbf{v}_l$, as it is described as follows:

Heuristic algorithm for solving CHL

1. Set $t = 0$ and choose the initial estimates of the location of hubs $\mathbf{v}_l(t), l = 1, \ldots, p$.
2. Compute the transportation fractions x_{ijlm} as follows (Equation (9)):

$$x_{ijlm}(t) = \left(\frac{\lambda_{ij}}{qW_{ij}^q D_{ijlm}^2(t)}\right)^{\frac{1}{q-1}},$$

   where,

$$\lambda_{ij}(t) = \left(\left(qW_{ij}^q d_{ij}^2\right)^{\frac{1}{1-q}} + \sum_{r=1}^{p}\sum_{s=1}^{p}\left(qW_{ij}^q D_{ijrs}^2(t)\right)^{\frac{1}{1-q}}\right)^{1-q}.$$

3. Update the location of hubs (Equation (14)):

$$\mathbf{v}_l(t+1) = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{m=1}^{p} W_{ij}^q x_{ijlm}^q(t)(\mathbf{s}_i + \alpha \mathbf{v}_m(t))}{(1+\alpha)\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{m=1}^{p} W_{ij}^q x_{ijlm}^q(t)}.$$

4. set $t = t + 1$ and repeat Steps 2 and 3 until the termination criteria is satisfied.

This iterative algorithm is a generalization of the well-known possibilistic clustering algorithm [11]. Although the algorithm does not consider directly the computation of y_{ij}, this value can be easily obtained from Constraint (4).

3 An entropy maximizing approach

[15] suggested the use of the entropy maximizing principle for selecting the location of hubs in discrete models. The idea is to use the available information regarding transport flows to describe the probability distribution of hub usage and then select the p hub locations with the smallest bias. This is done by maximizing a so-called entropy function. According to [15], the natural logarithm of the entropy E is defined as follows:

$$\ln E = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{l=1}^{p}\sum_{m=1}^{p} x_{ijlm}(1 - \ln x_{ijlm}) + \sum_{i=1}^{n}\sum_{j=1}^{n} y_{ij}(1 - \ln y_{ij}). \tag{15}$$

It is noted that maximizing the natural logarithm of the entropy is equivalent to maximizing the entropy.

Given that the location of hubs should be affected by the transportation cost, in this paper we also account for the impact of transport cost in the objective function. Hence, we aim to provide hub network designs with as low a transportation cost as possible and as high entropy as possible. For this formulation, we change the definition of flow variables. In this case, x_{ijlm} and y_{ij} should be understood as the total flow transported between nodes i and j using the intermodal and truck-only transport options, respectively. The *entropy maximizing hub location problem* (EHL) is defined as follows as follows:

$$(EHL):$$

$$\min \quad \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{l=1}^{p}\sum_{m=1}^{p} x_{ijlm} D^2_{ijlm}(\mathbf{v}_l, \mathbf{v}_m) + \sum_{i=1}^{n}\sum_{j=1}^{n} y_{ij} d^2_{ij} + \eta\left(\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{l=1}^{p}\sum_{m=1}^{p} x_{ijlm}(\ln x_{ijlm} - 1) + \sum_{i=1}^{n}\sum_{j=1}^{n} y_{ij}(\ln y_{ij} - 1)\right), \tag{16}$$

$$\text{s.t.} \quad (5),$$

$$y_{ij} + \sum_{l=1}^{p}\sum_{m=1}^{p} x_{ijlm} = W_{ij}, \quad \forall i, j = 1, \ldots, n. \tag{17}$$

The first term in Objective Function (16) refers to the total transportation cost. The second term is the negative of the natural logarithm of the entropy where the negative value is taken given that the objective function is of minimization and we intend to maximize entropy. Minimizing the negative entropy is equivalent to maximizing the entropy. The entropy term is affected by a factor η outside the parenthesis as a weighting factor for the transport and entropy terms. For their part, Constraints (17) guarantee that the total transport flow between nodes i and j satisfies the corresponding demand.

To solve the problem (16)–(17), we follow a similar procedure to Section 2 and propose the following iterative algorithmic scheme:

Heuristic solution for EHL

1. Set $t = 0$ and choose the initial estimates of the location of hubs $\mathbf{v}_l(t), l = 1, \ldots, p$.
2. Compute the transportation flows x_{ijlm} as follows:

$$x_{ijlm}(t) = e^{\frac{\lambda_{ij} - D^2_{ijlm}}{\eta}},$$

where,

$$\lambda_{ij}(t) = \ln\left(\frac{W_{ij}}{e^{-d^2_{ij}/\eta} + \sum_{r=1}^{p}\sum_{s=1}^{p} e^{-D^2_{ijrs}/\eta}}\right).$$

3. Update the location of hubs:

$$\mathbf{v}_l(t+1) = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{m=1}^{p} x_{ijlm}(t)(\mathbf{s}_i + \alpha\mathbf{v}_m(t))}{(1+\alpha)\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{m=1}^{p} x_{ijlm}(t)}.$$

4. set $t = t+1$ and repeat Steps 2 and 3 until termination criteria are satisfied.

This iterative algorithm, is a generalization of the possibilistic clustering algorithm suggested by [12].

The termination criterion of the algorithms was considered according to the value of flow variables x_{ijlm} at each iteration. The algorithm terminates when $\|X(t) - X(t-1)\|_2^2 < \epsilon$, where $X(t)$ represents the multidimensional array for the flow decision variables x_{ijlm} at iteration t, and ϵ is some threshold; we consider $\epsilon = 10^{-2}$. The algorithms terminate whenever the ϵ threshold or the maximum number of iterations criteria is reached.

4 Numerical experiments

We generate instances by randomly generating a set of demand nodes in the plane. The coordinates of demand nodes are generated to follow p spherical-like clusters, where p is the number of required hubs to open. To this end, we generate random data following bivariate normal distributions with mean $\boldsymbol{\mu}_l$ and covariance matrix $\Sigma_l = \sigma^2 I$, for $l = 1, \ldots p$. The cluster centroids $\boldsymbol{\mu}_l$ were uniformly generated in the box $[0, 100] \times [0, 100]$. The size of the clusters was adjusted by varying the dispersion parameter σ^2. Each cluster contains an evenly distributed number of demand nodes. For their part, the transport demand between each node was generated randomly with the uniform distribution $U \sim [0, 1\ 000]$.

An implementation of the algorithms was conducted using MATLAB R2020b. We used a computer with CPU AMD Ryzen 5 at 2.10 GHz, 8.00 GB RAM, Windows 10 Pro, and 64 bits.

To analyze the efficiency of the proposed algorithms, the number of demand nodes n was evaluated at 11 different values ranging from 10 to 1 000. The number of hubs to locate is evaluated from 3 to 6. Additionally, 30 repetitions were conducted. Each repetition is a completely new randomly generated problem. In

this sense, a total of 1 320 problem instances were evaluated for each algorithm. We use $\sigma^2 = 500$, generating high overlap between clusters. For the congestion-minimizing model (CHL), we consider an exponent $q = 1.1$. For their part, for the entropy maximizing model (EHL), we consider $\eta = 200$. In both cases, the discount factor was set to $\alpha = 0.75$.

Our experience by testing the algorithms indicates that the congestion minimizing hub location (CHL) algorithm takes on average more iterations than the entropy maximizing hub location (EHL) algorithm. For this reason, we set the maximum number of iterations for the CHL algorithm to 200, while for the EHL the maximum number of iterations is set to 400. Table 1 shows the results for the average number of iterations and the average processing time for CHL and EHL. Increasing the number of hubs to locate notoriously increases the processing time of the algorithms, and the iteration count (see [8]). On average, the EHL algorithm takes 2.75 times more iterations than the CHL algorithm. Despite this, the average processing time of the EHL algorithm is 1.93 times faster. One aspect that diminishes the efficiency of the CHL algorithm is the computation of the power function for the demand matrix W and the matrix of decision variables X in steps 2 and 3 of the algorithm.

Table 1: Computational results.

		CHL†		EHL‡	
n	p	**Iterations***	**Time (s)**	**Iterations***	**Time (s)**
100	3	39.7	1.82	86.1	0.71
	4	49.7	4.44	106.0	2.78
	5	60.9(1)	7.88	143.1	5.85
	6	75.6	15.29	166.8(1)	9.76
200	3	45.6	9.96	86.8	5.30
	4	63.9(2)	24.74	147.9(1)	15.58
	5	71.8	42.05	207.5(2)	33.73
	6	80.4(1)	67.27	204.4(1)	47.51
300	3	41.6	20.79	130.5(2)	17.64
	4	70.2(1)	63.60	148.8(2)	34.82
	5	78.7	103.92	197.5(1)	71.50
	6	99.1(2)	182.76	215.6(3)	111.14
400	3	56.1(1)	47.83	108.2	25.85
	4	79.3(1)	117.58	149.1	61.78
	5	86.3(1)	197.88	183.8(1)	117.61
	6	110.9(6)	357.34	218.4(5)	200.42
500	3	51.6	68.58	117.6(1)	43.74
	4	107.9(4)	245.25	181.8(2)	117.70
	5	111.7	419.89	214.8(3)	215.72
	6	123.1(5)	623.79	232.0(5)	334.20
600	3	56.8(1)	108.26	107.1	57.74
	4	95.5(4)	321.41	180.0(2)	168.5
	5	98.4(3)	516.17	196.8(1)	285.37

Table 1 continued from previous page

		CHL†		EHL‡	
n	p	**Iterations***	**Time (s)**	**Iterations***	**Time (s)**
700	6	110.9(6)	357.34	252.5(6)	518.71
	3	59.7(1)	152.50	127.8(1)	93.03
	4	95.3	422.50	190.7(1)	245.08
	5	119.0(5)	798.98	245.0(4)	483.80
	6	129.1(5)	1,215.65	244.8(1)	684.75
800	3	69.4(1)	229.94	114.0(1)	108.33
	4	109.9(4)	630.08	184.7(1)	303.35
	5	111.7(2)	995.96	219.3(2)	558.5
	6	127.8(6)	1,589.18	263.2(6)	940.47
900	3	64.2	306.38	144.6	172.40
	4	84.6(2)	638.65	211.0(4)	436.17
	5	117.4(6)	1,294.75	217.5(1)	690.34
	6	130.6(7)	2,134.40	250.0(4)	1,132.48
1,000	3	63.8(1)	327.74	145.0(1)	220.02
	4	100.7(1)	886.56	211.9(3)	552.51
	5	115.2(5)	1,540.10	225.2(2)	876.52
	6	137.7(5)	2,603.52	288.2(10)	1,598.10

* Number in parentheses indicate the number of times the algorithm reaches the maximum number of iterations.

Because of the heuristic nature of the algorithms, there is some variation in the results when the algorithm initializes at random, as was the case for the results in Table 1. In this sense, the solution of the algorithm corresponds to a local optima.

According to Figure 1, the magnitude of the discount factor has a clear impact on the network structure. When the discount factor is small, the hubs tend to be located far from each other and in the center of dense regions in terms of the proximity and number of demand nodes. In this case, the inter-hub traffic is high. On the other hand, when the discount factor is high, hubs tend to be closer to each other and the inter-hub traffic is considerably reduced. However, contrary to [9], when α approaches 1, hub locations do not necessarily collapse in a single point (see also [8]). This might be due to the different nature of the objective functions between this paper and the one of [9].

5 Conclusion

We studied the intermodal p-hub network design problem considering congestion-minimizing and entropy-maximizing approaches. The purpose of the first approach is to avoid the negative impact of congestion on transport activities. The latter approach seeks to maximize the hub usage in the network. We propose two non-linear formulations considering that hubs can be located anywhere in a two-dimensional space. We proposed heuristic algorithms based on Lagrangian

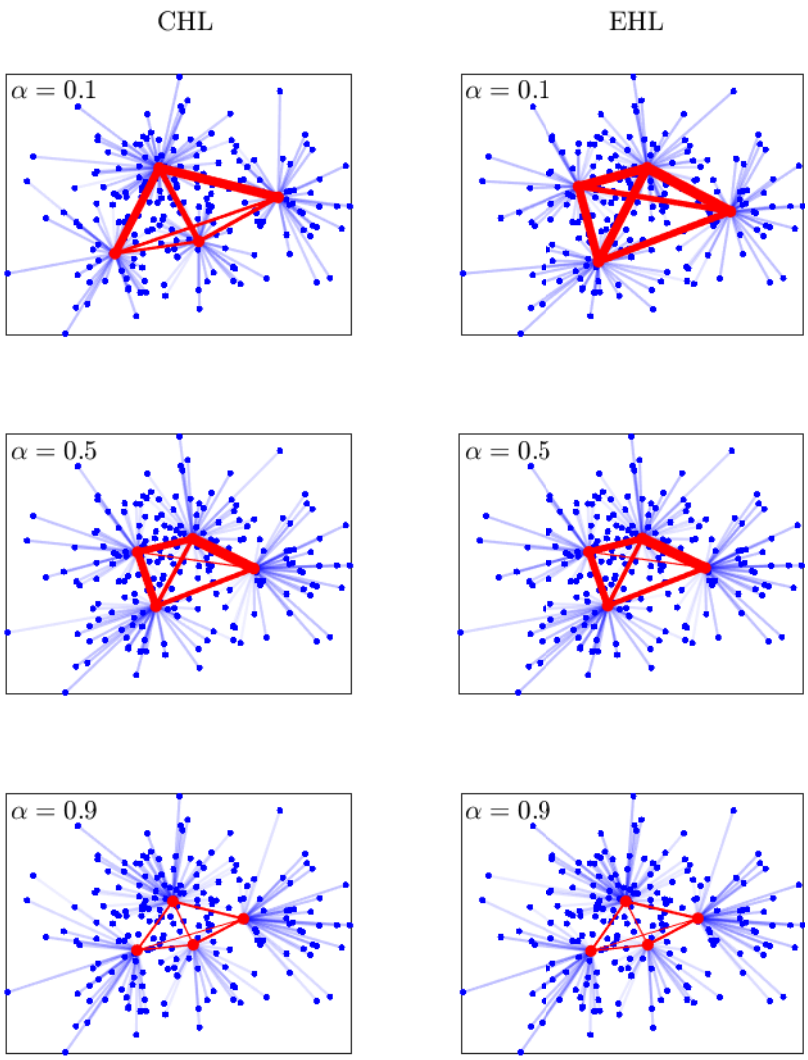


Fig. 1. Solutions for different values of the discount factor α. We show only the allocation (blue link) of a non-hub node to its most representative hub. The brightness of the links in blue is proportional to the total flow in the link. The thickness of inter-hub links in red is proportional to the total flow passing through the arc. Parameters: $n = 200, p = 4, q = 1.1, \eta = 200$.

relaxation to solve each problem. The heuristics are generalizations of the well-known possibilistic and possibilistic c-means clustering algorithms widely used in the literature in the pattern recognition field. The algorithms provide local

optimal solutions to hub network design problems of up to 1 000 nodes and 6 hubs in a few minutes of computational time.

References

1. Alumur, S.A., Campbell, J.F., Contreras, I., Kara, B.Y., Marianov, V., O'Kelly, M.E.: Perspectives on modeling hub location problems. European Journal of Operational Research **291**(1), 1 – 17 (2021). https://doi.org/10.1016/j.ejor.2020.09.039
2. Basallo-Triana, M., Vidal-Holguín, C., Bravo-Bastidas, J.: Planning and design of intermodal hub networks: A literature review. Computers and Operations Research **136** (2021). https://doi.org/10.1016/j.cor.2021.105469
3. Campbell, J.: Modeling economies of scale in transportation hub networks. Proceedings of the Annual Hawai International Conference on System Sciences pp. 1154–1163 (2013). https://doi.org/10.1109/HICSS.2013.411
4. Campbell, J., O'Kelly, M.: Twenty-five years of hub location research. Transportation Science **46**(2), 153–169 (2012). https://doi.org/10.1287/trsc.1120.0410
5. Carlsson, J., Jia, F.: Euclidean hub-and-spoke networks. Operations Research **61**(6), 1360–1382 (2013). https://doi.org/10.1287/opre.2013.1219
6. Damgacioglu, H., Dinler, D., Evin Ozdemirel, N., Iyigun, C.: A genetic algorithm for the uncapacitated single allocation planar hub location problem. Computers and Operations Research **62**, 224–236 (2015). https://doi.org/10.1016/j.cor.2014.09.003
7. Farahani, R., Hekmatfar, M., Arabani, A., Nikbakhsh, E.: Hub location problems: A review of models, classification, solution techniques, and applications. Computers and Industrial Engineering **64**(4), 1096–1109 (2013). https://doi.org/10.1016/j.cie.2013.01.012
8. Ghaffarinasab, N., Van Woensel, T., Minner, S.: A continuous approximation approach to the planar hub location-routing problem: Modeling and solution algorithms. Computers and Operations Research **100**, 140–154 (2018). https://doi.org/10.1016/j.cor.2018.07.022
9. Iyigun, C.: The planar hub location problem: A probabilistic clustering approach. Annals of Operations Research **211**(1), 193–207 (2013). https://doi.org/10.1007/s10479-013-1394-4
10. Kian, R., Kargar, K.: Comparison of the formulations for a hub-and-spoke network design problem under congestion. Computers and Industrial Engineering **101**, 504–512 (2016). https://doi.org/10.1016/j.cie.2016.09.019
11. Krishnapuram, R., Keller, J.: A possibilistic approach to clustering. IEEE Transactions on Fuzzy Systems **1**(2), 98–110 (1993). https://doi.org/10.1109/91.227387
12. Krishnapuram, R., Keller, J.: The possibilistic c-means algorithm: Insights and recommendations. IEEE Transactions on Fuzzy Systems **4**(3), 385–393 (1996). https://doi.org/10.1109/91.531779
13. O'Kelly, M.: A clustering approach to the planar hub location problem. Annals of Operations Research **40**(1), 339–353 (1992). https://doi.org/10.1007/BF02060486
14. O'Kelly, M.E.: Location of interacting hub facilities. Transportation Science **20**(2), 92–106 (1986). https://doi.org/10.1287/trsc.20.2.92
15. Teye, C., Bell, M., Bliemer, M.: Locating urban and regional container terminals in a competitive environment: An entropy maximising approach. Transportation Research Part B: Methodological **117**, 971–985 (2018). https://doi.org/10.1016/j.trb.2017.08.017

Machine Learning Optimized Orthogonal Basis Piecewise Polynomial Approximation

Hannes Waclawek[1,2] and Stefan Huber[1]

[1] Josef Ressel Centre for Intelligent and Secure Industrial Automation, Salzburg University of Applied Sciences, Austria
[2] Paris Lodron University Salzburg, Austria
{hannes.waclawek, stefan.huber}@fh-salzburg.ac.at

Abstract. Piecewise Polynomials (PPs) are utilized in several engineering disciplines, like trajectory planning, to approximate position profiles given in the form of a set of points. While the approximation target along with domain-specific requirements, like $\mathcal{C}^k$-continuity, can be formulated as a system of equations and a result can be computed directly, such closed-form solutions posses limited flexibility with respect to polynomial degrees, polynomial bases or adding further domain-specific requirements. Sufficiently complex optimization goals soon call for the use of numerical methods, like gradient descent. Since gradient descent lies at the heart of training Artificial Neural Networks (ANNs), modern Machine Learning (ML) frameworks like TensorFlow come with a set of gradient-based optimizers potentially suitable for a wide range of optimization problems beyond the training task for ANNs. Our approach is to utilize the versatility of PP models and combine it with the potential of modern ML optimizers for the use in function approximation in 1D trajectory planning in the context of electronic cam design. We utilize available optimizers of the ML framework TensorFlow directly, outside of the scope of ANNs, to optimize model parameters of our PP model. In this paper, we show how an orthogonal polynomial basis contributes to improving approximation and continuity optimization performance. Utilizing Chebyshev polynomials of the first kind, we develop a novel regularization approach enabling clearly improved convergence behavior. We show that, using this regularization approach, Chebyshev basis performs better than power basis for all relevant optimizers in the combined approximation and continuity optimization setting and demonstrate usability of the presented approach within the electronic cam domain.

Keywords: Piecewise Polynomials · Gradient Descent · Chebyshev Polynomials · Approximation · TensorFlow · Electronic Cams

1 Introduction

1.1 Motivation

PPs are of special interest in several science and engineering disciplines, where the latter are particularly interesting, as they come with additional physical constraints. Path and trajectory planning for machines in the field of mechatronics

are just two examples. Path planning is the task of finding possible waypoints of a robot or automated machine to move through its environment without collision. Trajectory planning computes time-dependent positional, velocity or acceleration profiles that hold setpoints for controllers of robots or automated machines to move joints from one waypoint to the other. Electronic cams are a subfield of the latter, describing repetitive motion executed by servo drives within industrial machines. In this context, positional, velocity or acceleration profiles are defined as input point clouds and approximated by PPs, which are then processed by industrial servo drives, like B&R Industrial Automation's ACOPOS series. Conventionally, the approximation target along with domain-specific requirements such as continuity, cyclicity or periodicity are formulated as a system of equations and a closed-form solution is computed. While computational effort may be low for computing such closed-form solutions, they posses limited flexibility with respect to polynomial degrees, polynomial bases or adding further domain-specific requirements. Sufficiently complex optimization goals soon call for the use of numerical methods, like gradient descent.

Training an ANN is the iterative process of adapting weights of node connections in order to fit given training data. With the advent of ML frameworks, this iterative training task is commonly carried out by calculating gradients using automatic differentiation and updating weights according to the gradient descent algorithm. Since gradient descent lies at the heart of this process, modern ML frameworks like TensorFlow or PyTorch therefore come with a set of gradient-based optimizers potentially suitable for a wide range of optimization problems beyond the training task for ANNs. Since advances in deep learning rely on these optimizers, state-of-the art insights within the field are continuously implemented, holding the potential of better performance than classical gradient descent optimizers.

Our approach therefore is to utilize the versatility of PP models and combine it with the potential of modern ML optimizers for the use in function approximation in 1D trajectory planning in the context of electronic cam design. We utilize available optimizers of the ML framework TensorFlow directly, outside of the scope of ANNs, to optimize model parameters of our PP model. This allows for an optimization of trajectories with respect to dynamic criteria using state of the art methods, while at the same time working with an explainable model, allowing to directly derive physical properties like velocity, acceleration or jerk. Although our work documented in [6] shows that our approach is basically feasible, experiments show that clean convergence especially regarding continuity optimization requires further practical considerations and regularization techniques. In an attempt to overcome these shortcomings, in this paper, we show how an orthogonal polynomial basis contributes to significantly improving approximation and continuity optimization performance.

1.2 An Orthogonal Basis

Orthogonal polynomial sets are beneficial for determining the best polynomial approximation to an arbitrary function f in the least squares sense, resulting

from the fact that every polynomial in such an orthogonal system possesses a minimal L_2 property with respect to the system's weight function, see [8] for details. Among different sets of orthogonal polynomials, Chebyshev polynomials of the first kind are of special interest. Besides favorable numerical properties, like the absolute values such polynomials evaluate to staying within the unit interval, their roots are well known as nodes (Chebyshev nodes) in polynomial interpolation to minimize the problem of Runge's phenomenon. This effect is interesting for us, as we are anticipating reduced oscillating behavior beneficial for our electronic cam approximation setting. Additionally and contrary to Chebyshev polynomials of the second kind, numerous efficient algorithms exist for the conversion between orthogonal basis and power basis representations [2]. (We need this in the end, since our solution targets servo drives like B&R Industrial Automation's ACOPOS series processing cam profiles as a power basis PP function.) With respect to potential future work, the relationship of Chebyshev series to Fourier cosine series has the potential of allowing a more straightforward formal approach to spectral analysis and optimization of generated cam profiles (e.g. to reduce resonance vibrations), since it inherits theorems and properties of Fourier series, allowing for a Chebyshev transform analogue to Fourier transform [4]. These arguments make Chebyshev polynomials of the first kind an attractive subject of investigation for this paper.

1.3 Related work

There is a lot of published work on approximation using PP in the cam approximation domain that rely on a closed-form expression in order to non-iteratively calculate a solution, like [9]. Although computational effort is reduced using such approaches, as mentioned earlier, they posses limited flexibility with respect to polynomial degrees, bases and complexity of loss functions. There also is work on PP approximation using "classical" gradient based approaches. In [3], Voronoi tessellation is used for partitioning the input space before approximating given 2D surfaces using a gradient-based approach with multivariate PPs. The authors state that an orthogonal basis is beneficial for this task, however, develop their approach using power basis. Contrary to our work, the loss function only comprises the approximation error, which implies that continuity is not achieved by their approach. Furthermore, the authors state that magnitudes of derivatives of the error function may vary greatly, thus making classical gradient descent approaches unusable. We show in this paper, that regularization may mitigate this effect, at least for continuity optimization in the 1D setting.

Other work utilizes parametric functions for approximation of given input data, like B-Spline or NURBS curves, as in [10]. This has some benefits over non-parametric PPs: For one, $\mathcal{C}^k$-continuity is given simply by the choice of order of the curve. For another, fewer parameters (usually control points) need to be optimized. However, PP models are utilized in several engineering disciplines and conversion from parametric curves is not generally possible. As an example, servo drives, like B&R Industrial Automation's ACOPOS series, process electronic cams in a PP format.

ANNs are investigated for the use in function approximation. In [1], the authors provide an overview of the state-of-the art with respect to aspects like numerical stability or accuracy and develop a computational framework for examining the performance of ANNs in this respect. Although results look promising and other recent works, like [11], already provide algorithmic bounds for the ANN approximation setting, one of the main drivers for the use of a PP model outside of the context of neural networks for our work is explainability. When utilizing Artificial Intelligence (AI) methods in the domain of electronic cams, there is a strong necessity for explainability, mainly rooted in the fact that movement of motors and connected kinematic chains like robotic arms affects its physical environment with the potential of causing harm, therefore calling for predictable movement. This, in turn, calls for a "predictable model", that allows to reliably derive physical properties and generate statements about expected behavior. Another recent development tackling these challenges are Physics-informed Neural Networks (PINNs), where resulting ANNs can be described using partial differential equations. For the domain of 1D electronic cam approximation covered in this work, however, using PP models is simpler and more intuitive considering that servo drives directly utilize PPs.

There are some preliminary non-scientific texts (e.g. personal blog articles) on gradient descent optimization for polynomial regression utilizing ML optimizers for a single polynomial segment. However, to the best of our knowledge, there is no published work on utilizing gradient descent optimizers of modern ML frameworks for $\mathcal{C}^k$-continuous PP approximation other than our previous research published in [6].

1.4 Contributions

Although Chebyshev basis holds the potential of improved approximation optimization performance, our experiments show that clean convergence, especially regarding continuity optimization, requires further practical considerations and regularization techniques. In this work, we therefore

1. adapt our base approach introduced in [6] to utilize an orthogonal basis using Chebyshev polynomials of the first kind,
2. develop a novel regularization approach enabling clearly improved convergence behavior,
3. show that, using this regularization approach, Chebyshev basis performs better than power basis for all relevant optimizers in the combined approximation and continuity optimization setting and
4. demonstrate usability of the presented approach within the electronic cam domain by comparing remaining approximation and continuity losses to least squares optima and strictly establishing continuity via an algorithm with impact only on local polynomial segments.

We do so by studying convergence behavior and properties of generated curves in the context of controlled experiments utilizing the popular ML framework TensorFlow. All experimental results discussed in this work are available at [12].

2 Piecewise Polynomial Model

Let $d \in \mathbb{N}_0$. The Chebyshev polynomials of the first kind are defined via the recurrence relation $2x\, T_{d-1}(x) - T_{d-2}(x)$, with $T_d(x) = 1$ if $d = 0$ and $T_d(x) = x$ for $d = 1$. The Chebyshev polynomials form an orthogonal set of functions on the interval $[-1, 1]$ with respect to the weighting function $w(x) = \frac{1}{\sqrt{1-x^2}}$. This means that two Chebyshev polynomials $T_c(x)$ and $T_d(x)$ are orthogonal with respect to the inner product $T_d(x)\rangle = \int_{-1}^{1} T_c(x) T_d(x)\, \frac{dx}{\sqrt{1-x^2}}$.

Considering n samples at $x_1 \leq \cdots \leq x_n \in \mathbb{R}$ with respective values $y_i \in \mathbb{R}$, we ask for a PP $f\colon I \to \mathbb{R}$ on the interval $I = [x_1, x_n]$ approximating the input samples well and fulfilling additional domain-specific properties, like $\mathcal{C}^k$-continuity. The polynomial boundaries of f are denoted by $\xi_0 \leq \cdots \leq \xi_m$, where $\xi_0 = x_1$ and $\xi_m = x_n$. With $I_i = [\xi_{i-1}, \xi_i]$, the PP f is modeled by m polynomials $p_i\colon I \to \mathbb{R}$ that agree with f on I_i for $1 \leq i \leq m$. The Chebyshev polynomials up to d form a basis of the vector space of real-valued polynomials up to degree d in the interval $[-1, 1]$. This means that an arbitrary real-valued polynomial $p(x)$ up to degree d defined on the interval $[-1, 1]$ can be expressed as a linear combination of Chebyshev polynomials of the first kind. We therefore can adapt our PP model introduced in [6] and construct each polynomial segment p_i via Chebyshev polynomials of the first kind as

$$p_i = \sum_{j=0}^{d} c_{i,j} T_j(x - \mu_i). \tag{1}$$

We rescale input data and shift polynomials to the mean of the respective segment by $\mu_i = \frac{\xi_{i-1}+\xi_i}{2}$, as outlined in Sect. 3. The $c_{i,j}$ are the to be trained model parameters of the PP ML model. We investigate the convergence of these model parameters $c_{i,j}$ with respect to the loss function defined below.

2.1 Loss function

We utilize the loss function

$$\ell = \alpha \ell_{\mathrm{CK}} + (1 - \alpha)\ell_2 \tag{2}$$

with $0 \leq \alpha \leq 1$ in order to establish $\mathcal{C}^k$-continuity and allow for curve fitting via least squares approximation, where the approximation error ℓ_2 is defined as

$$\ell_2 = \frac{1}{n} \sum_i |f(x_i) - y_i|^2. \tag{3}$$

Note that, as outlined in Sect. 3.3, optimization results with $\alpha = 0$ are of less practical relevance, because remaining approximation errors are significantly higher after strictly establishing continuity utilizing the algorithm introduced in Sect. 2.3. Summing up discontinuities at all ξ_i across relevant derivatives as

$$\ell_{\mathrm{CK}} = \frac{1}{m-1} \sum_{i=1}^{m-1} \sum_{j=0}^{k} \left(\frac{\Delta_{i,j}}{r_k} \right)^2 \quad \text{with} \quad \Delta_{i,j} = p_{i+1}^{(j)}(\xi_i) - p_i^{(j)}(\xi_i) \tag{4}$$

quantifies the amount of discontinuity throughout the overall PP function. This loss definition allows for a straightforward extension towards $\mathcal{C}^k$-cyclicity or periodicity commonly required for electronic cam profiles: We can achieve periodicity in (4) by adapting $m-1$ to m and generalizing $\delta_{i,j} = p^{(j)}_{1+(i \bmod m)}(\xi_{i \bmod m}) - p^{(j)}_i(\xi_i)$. For cyclicity, we ignore the case $j = 0$ when $i = m$. (While periodicity requires matching the values of all derivatives at the endpoints of a motion profile, with cyclicity, we can have a positional offset.) For the sake of generality, we define $\ell_{\mathrm{CK}} = 0$ for $m = 1$.

Experiments documented in Sect. 3 show that the derivative-specific regularization factor r_k is required in order to reduce oscillating behavior and improve convergence behavior. In the following, we develop a formula describing this derivative-specific regularization factor.

2.2 Regularization of Loss

Magnitudes of discontinuities at boundary points may vary significantly between derivatives, with higher derivatives potentially having higher magnitudes, thus impairing convergence. With polynomials, the potentially highest contributing factor is the one of the highest order term. We can express a derivative k of a polynomial of degree d in power basis form using Horner's method as $p_i = \sum_{i=k}^{d} c_i x^{i-k} \cdot R_k$, where $R_k = (i \cdot (i-1) \cdot (i-2) \cdot \ldots \cdot (i-k-1))$. For the highest order term, this leaves us with

$$r_k = \frac{d!}{(d-k)!}. \tag{5}$$

Our approach is to regularize each derivative by r_k defined in eq. (5) as outlined in eq. (4). We could apply a similar approach to Chebyshev basis. Since, for calculating ℓ_{CK}, we are only interested in functional values at the boundaries of individual polynomial segments, due to symmetry properties of Chebyshev polynomials, we receive the same absolute values at both ends of the interval $[-1, 1]$. We therefore don't need a closed formula also for Chebyshev basis, but could simply evaluate $\|T_d(1)\|$ in order to retrieve the regularization factor for derivative d. However, experiments indicate that r_k defined in eq. (5) performs better also for Chebyshev basis. We therefore utilize this factor for regularization of continuity loss ℓ_{CK} with both polynomial bases.

2.3 Enforcing Continuity

We can eliminate possible remaining discontinuities, i.e., $\ell_{\mathrm{CK}} > 0$ in the sense of eq. (4), by applying corrective polynomials that strictly enforce $\Delta(\xi_i) = 0$ at all ξ_i. Iterating through polynomial segments from left to right, we construct a corrective polynomial q_i of degree $2k+1$ by formulating a system of equations taking the values of $k+1$ derivatives per endpoint, respectively, as shown in Algorithm 1 in Appendix A. Note that the corrective polynomials themselves are of degree at most d.

A favorable property of Algorithm 1 is that it only has local effects: only derivatives relevant for $\mathcal{C}^k$-continuity are modified, while other derivatives remain unchanged. This allows us to apply the corrections at each ξ_i independently as they have only local impact. This is a nice property in contrast to interpolation methods like natural cubic splines or parametric curves lacking the local control property, like Bézier curves. For an extension towards $\mathcal{C}^k$-cyclicity or periodicity commonly required for electronic cam profiles, we can naturally modify the first cases of b^L and b^R of Algorithm 1 as outlined in Sect. 2.1, respectively.

2.4 Discussion of Continuity Loss Characteristics

Let us denote by θ the list of model parameters $c_{1,0}, c_{1,1}, \ldots, c_{m,d}$ and let us denote by f_θ the resulting PP. Using the mean-square-error loss function outlined in eq. (3), a natural approach would be to solve the optimization problem

$$\theta^* = \arg\min_\theta \ell_2, \tag{6}$$

to obtain the optimal model parameters θ^*. Since this problem is convex, there is a unique solution. However, f_{θ^*} is in general not $\mathcal{C}^k$-continuous, or even $\mathcal{C}^0$-continuous, as the individual p_i are optimized individually.

We can capture the amount by which $\mathcal{C}^k$-continuity is violated by the ℓ_{CK} loss defined in eq. (4). Note that $\Delta_{i,j}$ captures the discontinuity of the j-th derivative at ξ_i. Also note that $\ell_{\mathrm{CK}} = 0$ iff $\mathcal{C}^k$-continuity holds.

Then we ask for the ℓ_2-minimizing $\mathcal{C}^k$-continuous PP, which is the solution of the constrained optimization problem

$$\begin{aligned} \min_\theta \quad & \ell_2 \\ \text{s.t.} \quad & \ell_{\mathrm{CK}} = 0. \end{aligned} \tag{7}$$

We turn this into an unconstrained optimization problem by adding ℓ_{CK} to the loss function in eq. (2) and obtain

$$\theta^*_\alpha = \arg\min_\theta \ell \tag{8}$$

with $0 \leq \alpha \leq 1$. For $\alpha = 0$, we again have the problem in eq. (6). If $\alpha = 1$ then any $\mathcal{C}^k$-continuous PP is an optimal solution, such as the zero function, but also the solution of eq. (7). Any $\alpha > 0$ leaves us, in general, with a non-convex optimization problem prone of getting stuck in local optima. Let us denote by f^*_α a solution of eq. (8) with a fixed α. In this sense, we are interested in a f^*_α approximating the input point set well after running the algorithm introduced in Sect. 2.3, since the result is guaranteed to be numerically continuous. In Sect. 3 we therefore perform experiments analyzing suitable initialization methods and α values for different input point sets and preconditions, with a subsequent strict establishment of continuity via Algorithm 1.

3 Experimental Results

The Chebyshev polynomials up to d form a basis of the vector space of real-valued polynomials up to degree d in the interval $[-1, 1]$. We therefore rescale the x-axis of input data so that every segment is of span 2. Considering this, for a PP consisting of 4 polynomial segments, as an example, we scale the input data such that $I = [0, 8]$. Splitting the interval equally into 4 segments we receive a width of 2 for each individual segment, and, by μ_i in eq. (1), we shift polynomials to the mean of the respective segment, leading to $I_i = [-1, 1]$ for each respective PP segment. We do the same for power basis and skip back-transformation as we would do in production code. (In our experiments, convergence is not significantly impaired for interval widths of 2 ± 0.5 for Chebyshev basis with tested data sets.)

Considering our focus on the electronic cam domain, a position profile with high acceleration values leads to the kinematic system being exposed to high forces. A position profile with high jerk values, on the other hand, will make the system prone to vibration and excessive wear [7]. In order to address the latter, continuos jerk curves are highly beneficial, if not a prerequisite. We therefore look closer at $\mathcal{C}^3$-continuity in all conducted experiments with $\alpha > 0$. Looking at Algorithm 1, this implies a PP of degree 7. We utilize the Tensorflow *GradientTape* environment for the creation of custom training loops utilizing available optimizers directly, as outlined in [6]. All experimental results discussed in this section where created with TensorFlow version 2.13.0, Python 3.10 and are available at [12].

3.1 Input Data and Optimization Goal

Computing eq. (6) for each polynomial segment, we denote by

$$\ell_2^* = \min_{\theta} \quad \ell_2 \tag{9}$$

the segment-wise approximation loss optimum. The resulting PP is not C^k-continuous, since each polynomial segment is fitted individually. Utilizing method CKMIN described in Algorithm 1, we enforce continuity for this result and denote by $\widetilde{\ell_2^*} = \ell_2(\text{ckmin}(\arg\min_\theta \ell_2))$ its approximation loss. It is easy to see that $\ell_2^* \leq \widetilde{\ell_2^*}$. Considering this, our optimization goal is for results to lie in the margin between ℓ_2^* and $\widetilde{\ell_2^*}$. The closer a result is to ℓ_2^*, the better. The value of ℓ_2^* depends on the amount of variance / noise in the input data along with the choice of number of polynomial segments. In the following, we therefore refer to the value of ℓ_2^* as *variance*, i.e., input data with a higher value of ℓ_2^* is referred to as input data with higher variance. Table 1 gives an overview of input data used for our experiments along with the respective baseline values of ℓ_2^* and $\widetilde{\ell_2^*}$ when enforcing $\mathcal{C}^3$-continuity. In addition to the input data described in Table 1, we utilize two different noise levels for each dataset, generated by adding random samples from a normal Gaussian distribution using the package `numpy.normal` with scales 0.1 and 0.5, respectively, and a fixed seed of 0 for reproducibility.

Dataset	I	n	m	ℓ_2^*	$\widetilde{\ell_2^*}$
A: $\sin(x)$	$[0, \frac{\pi}{2}]$	50	2	$4.110 \cdot 10^{-21}$	$8.450 \cdot 10^{-17}$
B: $\sin(x)$	$[0, 2\pi]$	100	2	$2.240 \cdot 10^{-11}$	$3.630 \cdot 10^{-7}$
C: $\sin(x^2 4\pi)$	$[0, 1]$	100	3	$3.540 \cdot 10^{-6}$	$4.230 \cdot 10^{-2}$

Table 1: Different input data sets used for performing experiments.

3.2 Optimizing ℓ_2 only

We first compare performance of Chebyshev basis and power basis in the single approximation target scenario, i.e., one polynomial segment and $\alpha = 0$. With rising polynomial degree, the remaining ℓ_2 optimum approximation error gets lower. This is in accordance with Chebyshev basis performance observed in our experiments, where results manage to converge to a lower loss with rising degree. This is not the case for power basis. While performance with degree 5 is competitive, higher degrees perform clearly worse than the Chebyshev basis counterparts. As expected, higher polynomial degrees generally require more epochs to converge for both bases. For Chebyshev basis, however, results generally converge to lower remaining losses, and, for all observed degrees ($[3, 9]$), within 1000 epochs. Experiments also show that a learning rate of 1.0 is a reasonable choice for all observed degrees and both observed polynomial bases in the sole approximation optimization target scenario. Raising the number of input points has no effect to the highest learning rate we can achieve. Of course, the number of input points has to be sufficiently high for a polynomial degree to enable a well-conditioned fit in the first place. Looking at the impact of the amount of variance in the input data, generally speaking, it takes longer to converge to the optimal approximation result for both observed polynomial bases with less noise in the input data. However, while optimization with Chebyshev basis reaches the least squares optimum for all observed noise levels, power basis is only competitive with noise added to the input curve. The experimental test run documenting this behavior for dataset A is depicted in Fig. 1.

Looking at the performance of all optimizers available with TensorFlow version 2.13.0 shows that Chebyshev basis clearly outperforms power basis with all observed datasets and optimizers converging to the optimum within 5000 epochs. While none of the optimizers manage to reach the ℓ_2 optimum with power basis in the given 5000 epochs, there are several optimizers achieving this with Chebyshev basis. Interestingly, with Chebyshev basis, Vanilla SGD is outperforming more "elaborate" adaptive optimizers, like AMSGrad and other Adam-based optimizers. Sorted by quickest convergence, candidate optimizers are Nadam, Adagrad, FTRL, Vanilla SGD, SGD with Nesterov momentum, Adam, Adamax, AMSGrad SGD with momentum and Adadelta. As our main goal is performing combined approximation and continuity optimization, we do not look into optimizing hyperparameters of individual optimizers in the sole approximation optimization target scenario.

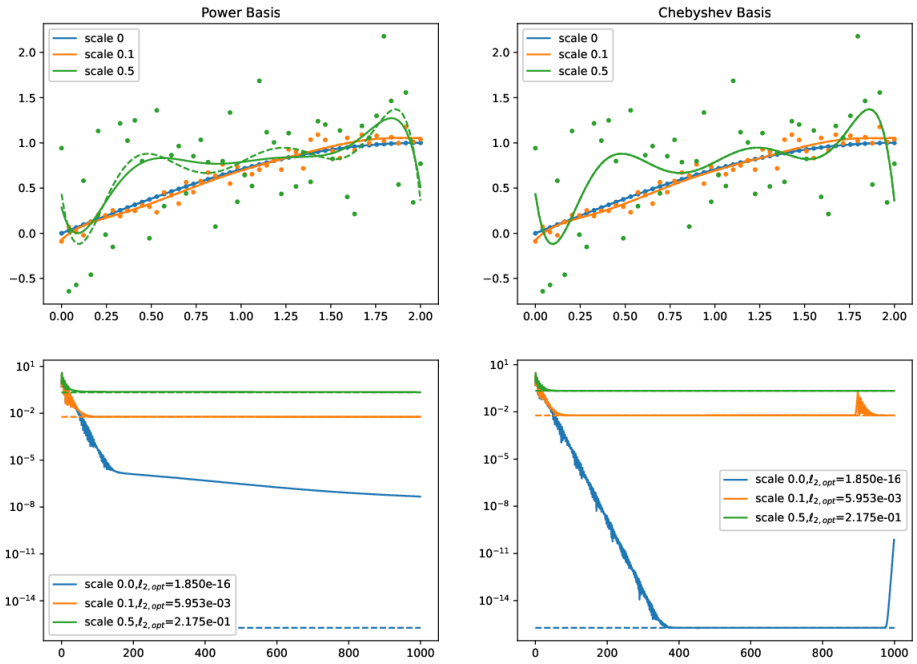


Fig. 1: Results for different noise levels, AMSGrad optimizer, learning rate 1.0, dataset A, 1 segment, 1000 epochs, $\alpha = 0$. Top: Derivative 0 curve shapes. Bottom: ℓ_2 losses. Optima at dashed lines.

3.3 Optimizing ℓ

Contrary to the single approximation target discussed in the previous section, we see that a lower learning rate of 0.1 is beneficial. Chebyshev basis results again converge to lower losses, however, now 2000 epochs are required. Increasing the number of segments does not affect the highest possible learning rates.

Looking at the performance of all optimizers available with TensorFlow version 2.13.0, Chebyshev basis is again clearly outperforming power basis in regard to all relevant optimizers and input point sets. Fig. 3 in Appendix B shows this in more detail for dataset A: While none of the optimizers manage to surpass ℓ_2^* with power basis in the given 5000 epochs, there are several optimizers achieving this with Chebyshev basis. Interestingly, however, with Chebyshev basis, SGD does not converge in this combined approximation and continuity optimization target any longer. Results with regularization of $\mathcal{C}^k$-loss, as introduced in Sect. 2.2, clearly outperform results without $\mathcal{C}^k$-loss regularization for candidate optimizers.

Running experiments with all datasets and noise levels defined in Table 1, we see the trend that more optimizers (also with power basis) manage to surpass the ℓ_2^* baseline for input data with higher ℓ_2^* values. Experiments also clearly

show that results with regularization of $\mathcal{C}^k$-loss outperform results without $\mathcal{C}^k$-loss regularization for candidate optimizers also for datasets other than dataset A. The higher ℓ_2^* of the input data, the closer is the gap between regularized and non-regularized losses, as well as power basis and Chebyshev basis results. Considering all observed input data, AMSGrad with default parameters is the best candidate for both polynomial bases in this combined approximation and continuity optimization target. Since spikes in its total loss curve frequently occur after the optimizer has reached lowest remaining losses, however, early stopping with reverting to the best PP coefficients achieved for the training run is strongly recommended. Investigating different methods of initializing polynomial coefficients (ℓ_2 optimum, zero, random), ℓ_2 optimum initialization turns out to be the best choice.

A simple approach for $\mathcal{C}^k$-continuous solutions would be to apply Algorithm 1 to the segment-wise ℓ_2 optimum. However, looking at ℓ_2 versus ℓ_{CK} error for different datasets, as depicted in Fig. 2 for dataset A, we find that strictly establishing continuity using Algorithm 1 for a result with $\alpha = 0$ increases the remaining approximation error significantly. Strictly establishing continuity for an optimization result with $\alpha > 0$, on the other hand, affects prior approximation errors only mildly. This is an important finding, as it not only substantiates the need for ℓ_{CK} optimization, but also tells us that if we strictly establish continuity after optimization, it is beneficial to choose an α value that is greater but close to 0.

Looking at the experimental setup of Fig. 2, the margin between remaining approximation error ℓ_2^* of a conventional segment-wise least squares fit and approximation error $\widetilde{\ell}_2^*$ of this fit after strictly establishing continuity via algorithm is higher for input data with higher variance. Taking this margin as a baseline, higher variance input data leaves us with with a wider band of where our optimization leads to better results than performing a conventional segment-wise least squares fit with strict continuity establishment. In case of a wider band, we can also have a wider range of α values that make sense. Leaving out some scenarios where local optima of higher α values have a lower approximation error than lower ones, we generally see a tendency of rising approximation errors with rising α values. Considering the experimental setup of Fig. 2, the two bottom plots show results for input dataset A with a noise scale of 0.5. Here, we observe results that lie within the aforementioned margin throughout the complete α range $0 \leq \alpha \leq 1$. Without noise, as shown in the two top plots, the ℓ_2^* value of the input data is lower, and we see α values of $>\approx 0.3$ tendentially having ever higher approximation errors, thus leaving the margin, which leaves us with a smaller range of possible values of α that make sense.

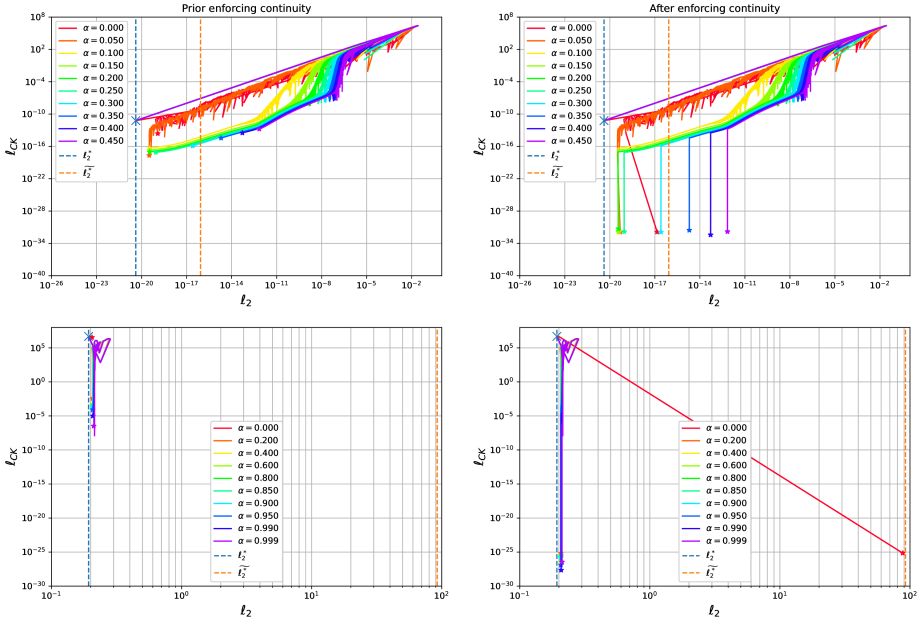


Fig. 2: Continuity over approximation error for ℓ_2 initialization with learning rate = 0.1, α = 0.1, dataset A, 2 segments, 2000 epochs and early stopping enabled with a patience of 500. Top: Noise scale = 0. Bottom: Noise scale = 0.5.

4 Conclusion and Outlook

Our results show that optimization of PPs utlizing Chebyshev basis clearly outperforms power basis with tested data sets and various noise levels. Increasing the number of polynomial segments can further boost this advantage, as Chebyshev basis benefits from input data with lower variance. Considering its better performance with degree 7 polynomials, this also makes Chebyshev basis the favorable candidate for the generation of $\mathcal{C}^3$-continuos PP position profiles for the use in electronic cam approximation, as a polynomial degree of 7 is required to strictly establish $\mathcal{C}^3$-continuity after optimization using Algorithm 1. Looking at Chebyshev basis convergence results for different data sets, we see that the regularization method introduced in Sect. 2.2 is required to reduce oscillating behavior and boost optimizer performance.

As outlined in section Sect. 2.4, setting $\alpha > 0$ leaves us with a non-convex optimization problem prone of getting stuck in unfavorable local optima. Experimental results documented in Sect. 3 show, however, that we can effectively guard against this by (i) pre-initializing polynomial coefficients with the segment-wise ℓ_2 optimum, (ii) applying early stopping and reverting to best PP coefficients achieved during the training run and (iii) strictly establishing continuity after optimization with $\alpha > 0$ running the algorithm introduced in Sect. 2.3.

Our results show that strictly establishing continuity for a result with $\alpha = 0$ increases the remaining approximation error significantly, while a result with $\alpha > 0$ is only affected mildly. This substantiates the need for ℓ_{CK} optimization also when strictly establishing continuity after optimization.

Looking at possible future work, additional terms in ℓ can accommodate for further domain-specific goals. For instance, we can reduce oscillations in f by penalizing the strain energy $\ell_{\text{strain}} = \int_I f''(x)^2 \, \mathrm{d}x$, which, in the context of electronic cams, can reduce induced forces and energy consumption. While our approach benefits from optimizers provided with modern ML frameworks, it also has the potential of contributing back to the field of ANNs. In their preprint [5], Daws and Webster initialize Neural Networks via polynomial approximation using Legendre basis and show that subsequent training results benefit from such an initialization. Our approach could be used for such an initialization. Also, our approach is compatible to other iterative ML methods. In this way, we can utilize it for Reinforcement Learning (RL) methods based on policy gradients, like Trust Region Policy Optimization or Proximal Policy Optimization, and model the policy landscape using multivariate polynomials in an effort of improving sample efficiency of modern RL algorithms.

Appendix A Strictly Establishing Continuity

Algorithm 1 CKMIN: Strictly Establish $\mathcal{C}^k$-Continuity

Input: PP, k ▷ Piecewise polynomial, continuity class

1: **procedure** CKMIN()
2: $(p_1, \ldots, p_m) \leftarrow PP$ ▷ m polynomial pieces
3: **for** $i \leftarrow 1$ **to** m **do** ▷ Correct all p_i over $[\xi_{i-1}, \xi_i]$
4: $A^L \leftarrow \left(\frac{j!}{(l-j)!} \xi_{i-1}^{l-j} \right)_{j,l=0,j}^{k,2k+1}$ ▷ Left conditions at ξ_{i-1}
5: $A^R \leftarrow \left(\frac{j!}{(l-j)!} \xi_{i}^{l-j} \right)_{j,l=0,j}^{k,2k+1}$ ▷ Right conditions at ξ_i
6: $b^L \leftarrow \left(\begin{cases} 0 & \text{if } i = 0 \\ p_{i-1}^{(j)}(\xi_{i-1}) - p_i^{(j)}(\xi_{i-1}) & \text{otherwise} \end{cases} \right)_{j=0}^{k}$ ▷ Match p_{i-1}
7: $b^R \leftarrow \left(\begin{cases} 0 & \text{if } i = m \\ \frac{p_{i+1}^{(j)}(\xi_i) - p_i^{(j)}(\xi_i)}{2} & \text{otherwise} \end{cases} \right)_{j=0}^{k}$ ▷ Match mean of p_i, p_{i+1}
8: $A, b \leftarrow \begin{pmatrix} A^L \\ A^R \end{pmatrix}, \begin{pmatrix} b^L \\ b^R \end{pmatrix}$ ▷ Stacking equation systems
9: $c \leftarrow A^{-1} \cdot b$ ▷ Solve $A \cdot c = b$ for c
10: $q_i \leftarrow \sum_{j=0}^{2k+1} c_j x^j$ ▷ Corrective polynomial
11: $p_i \leftarrow p_i + q_i$ ▷ Apply correction
12: **end for**
13: **return** $(p_1, \ldots, p_m)$ ▷ $\mathcal{C}^k$-continuous piecewise polynomial
14: **end procedure**

Appendix B Optimizer Performance

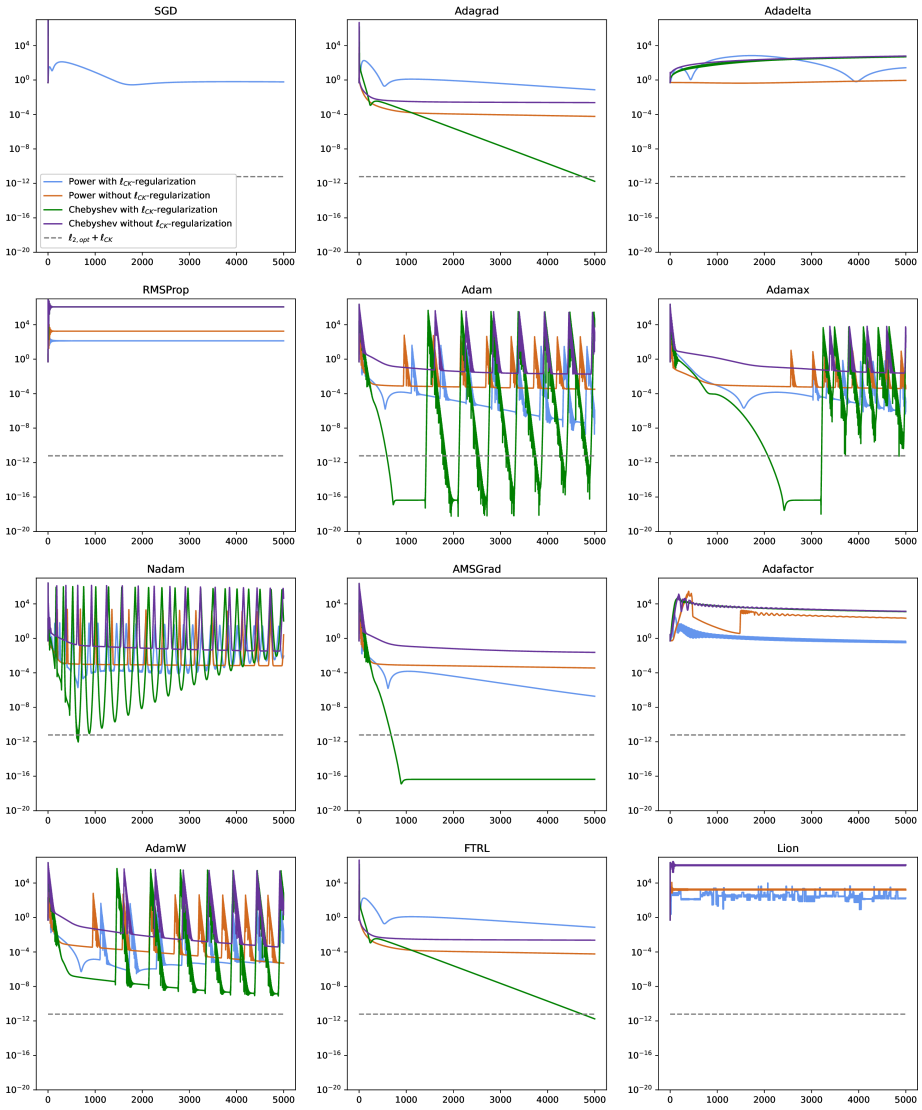


Fig. 3: Losses over epochs with different optimizers with learning rate = 0.1, $\alpha = 0.1$, dataset A, 2 segments. Dashed lines denote $\ell_2 + \ell_{CK}$ for the respective segment-wise least squares optimum.

Acknowledgements. This work was supported by the Christian Doppler Research Association (JRC ISIA), the federal state of Salzburg WISS-FH project IAI and the European Interreg Österreich-Bayern project BA0100172 AI4GREEN.

References

1. Adcock, B., Dexter, N.: The Gap between Theory and Practice in Function Approximation with Deep Neural Networks. SIAM Journal on Mathematics of Data Science **3**(2), 624–655 (2021). https://doi.org/10.1137/20M131309X
2. Bostan, A., Salvy, B., Schost, E.: Fast conversion algorithms for orthogonal polynomials. Linear Algebra and its Applications **432**(1), 249–258 (2010). https://doi.org/10.1016/j.laa.2009.08.002
3. Chen, Z., Xiao, Y., Cao, J.: Approximation by piecewise polynomials on Voronoi tessellation. Graphical Models **76**(5), 522–531 (2014). https://doi.org/10.1016/j.gmod.2014.04.006, Geometric Modeling and Processing 2014
4. Datta, K., Mohan, B.: Orthogonal Functions In Systems And Control. Advanced Series In Electrical And Computer Engineering, World Scientific Publishing Company (1995)
5. Daws, J., Webster, C.G.: A Polynomial-Based Approach for Architectural Design and Learning with Deep Neural Networks (2019). https://doi.org/10.48550/arXiv.1905.10457
6. Huber, S., Waclawek, H.: $\mathcal{C}^k$-continuous Spline Approximtion with TensorFlow Gradient Descent Optimizers. In: EUROCAST 2022. pp. 577–584. Springer Nature Switzerland, Cham (2023)
7. Josephs, H., Huston, R.: Dynamics of Mechanical Systems. CRC Press, USA, 1 edn. (2002)
8. Mason, J., Handscomb, D.: Chebyshev Polynomials. CRC Press (2002)
9. Mermelstein, S., Acar, M.: Optimising cam motion using piecewise polynomials. Engineering With Computers **19**, 241–254 (02 2004). https://doi.org/10.1007/s00366-003-0264-0
10. Nguyen, T., Kurtenbach, S., Hüsing, M., Corves, B.: A general framework for motion design of the follower in cam mechanisms by using non-uniform rational B-spline. Mechanism and Machine Theory **137**, 374–385 (2019). https://doi.org/10.1016/j.mechmachtheory.2019.03.029
11. Shen, Z., Yang, H., Zhang, S.: Deep Network Approximation Characterized by Number of Neurons. Communications in Computational Physics **28**(5), 1768–1811 (2020). https://doi.org/10.4208/cicp.OA-2020-0149
12. Waclawek, H., Huber, S.: Experimental results for publication "Machine Learning Optimized Orthogonal Basis Piecewise Polynomial Approximation". `https://github.com/hawaclawek/ml-optimized-orthogonal-basis-pp` (2024), accessed: 2024-02-15

An Approximate-and-Optimize Method for Security-Constrained AC Optimal Power Flow

Jinxin Xiong[1,2], Shunbo Lei[1,3], Akang Wang[1,2[0000−0002−3325−8441]⋆], and Xiaodong Luo[1,2]

[1] Shenzhen Research Institute of Big Data, Guangdong, China
wangakang@sribd.cn
[2] School of Data Science, The Chinese University of Hong Kong, Shenzhen, Guangdong, China
[3] School of Science and Engineering, The Chinese University of Hong Kong, Shenzhen, Guangdong, China

Abstract. The security-constrained alternating current optimal power flow problem (SC-ACOPF) involves determining the minimum-cost operating conditions for power systems while ensuring system security. This problem is challenging due to its formulation as a large-scale, non-linear, and non-convex model, presenting difficulties for traditional optimization approaches. In this study, we introduce an approximate-and-optimize method to tackle the SC-ACOPF. We decompose the SC-ACOPF model into a two-stage problem formulation, where the first stage addresses the base case, and the second stage focuses on post-contingency decisions. To handle the complexity, we employ a neural network to approximate the objective cost associated with all contingencies. Subsequently, we optimize the first-stage problem after incorporating the neural network surrogate function. Computational experiments conducted on a 500-bus system showcase the effectiveness of our proposed framework. The results demonstrate its superior performance compared to baseline algorithms, achieving a remarkable 60% reduction in computational time and a 50% decrease in objective costs.

Keywords: Security-constrained alternating current optimal power flow · Approximate-and-optimize · Neural networks · Surrogate function.

1 Introduction

The *Alternating Current Optimal Power Flow* (ACOPF) is dedicated to determining an economically optimal operating configuration for power system controllers while adhering to essential physical constraints. In order to mitigate the risk of system failure, practitioners commonly adopt a pre-contingency strategy by incorporating *security constraints*. The widely utilized $N-1$ security criterion ensures that the power system remains operational even in the event of the failure of a single generator or branch. This gives rise to the *Security-Constrained*

⋆ Corresponding author: wangakang@sribd.cn

ACOPF (SC-ACOPF) problem, which poses a formidable challenge due to its formulation as a large-scale, non-linear, non-convex, and potentially non-smooth model. The complexity of the model escalates with the number of contingencies, making the SC-ACOPF problem particularly demanding to address. However, operational requirements dictate frequent solutions to SC-ACOPF problems to maintain stable and economically efficient power system operation. This necessity propels our efforts to develop an efficient and scalable algorithm capable of delivering high-quality feasible solutions to the SC-ACOPF.

In this study, our attention is directed towards the SC-ACOPF problem as addressed in the Grid Optimization (GO) competition [6]. The targeted SC-ACOPF problem is conceptualized as a two-stage stochastic optimization problem. In the first stage, the objective is to determine cost-effective bus voltage magnitudes, bus voltage angles, active and reactive power from generators, and controllable shunt susceptances under normal operating conditions. The second stage comes into play during contingencies, where the aforementioned settings are adjusted, and violations of second-stage constraints are penalized in the objective function. Addressing a multitude of contingencies necessitates the adoption of filtering methods, as seen in prior works such as [11, 8, 3]. However, existing filtering strategies either involve iterative procedures or can only handle a limited number of contingencies. An alternative approach proposed by the authors of [19] involves the elimination of second-stage variables and constraints. They establish a mapping between first-stage variables and second-stage costs using a quartic approximation function, resulting in a considerably reduced formulation that requires solving. Despite its merits, this approach has two notable drawbacks:

i. the necessity for updating approximation functions in each iteration leads to solving numerous small yet nonlinear optimization problems;
ii. a quartic approximation might not accurately capture the intricate nature of the second-stage cost function.

Recently, *machine learning* techniques have emerged as powerful tools for tackling ACOPF problems, both through end-to-end prediction and by enhancing the efficiency of existing optimization algorithms [26]. Similar advancements have extended to the domain of SC-ACOPF problem-solving. To address a large number of contingency scenarios, the work of [7] proposed data-driven methods for fast contingency selection. However, this approach might produce SC-ACOPF solutions that violate security constraints. To mitigate this issue, the work of [22] suggested a two-step process: first predicting network voltage magnitudes and angles of buses using a multi-input multi-output random forest model and then calculating the remaining variables by solving physics-based power flow equations. The authors of [22] demonstrated the effectiveness of their approach on a 500-bus system. Despite its success, it is essential to note that the random forest model might predict voltages and angles in a way that the resulting system of power flow equations has no solution, potentially leaving practitioners without complete SC-ACOPF solutions.

Since SC-ACOPF can be viewed as a two-stage stochastic program, we review how machine learning techniques enhance the solving of stochastic programs. In the work of [10], a learning-based approach is proposed to address two-stage stochastic Mixed-Integer Linear Programs (MILPs). The authors leverage a neural network with Rectified Linear Unit (ReLU) neurons to approximate the expected cost function. This results in a model that can be easily translated into an MILP, facilitating optimization through commercial MILP solvers. However, it is important to note that if the first-stage problem involves non-linear and non-convex constraints, the resulting model would be converted into a mixed-integer non-linear program, posing challenges in terms of computational handling.

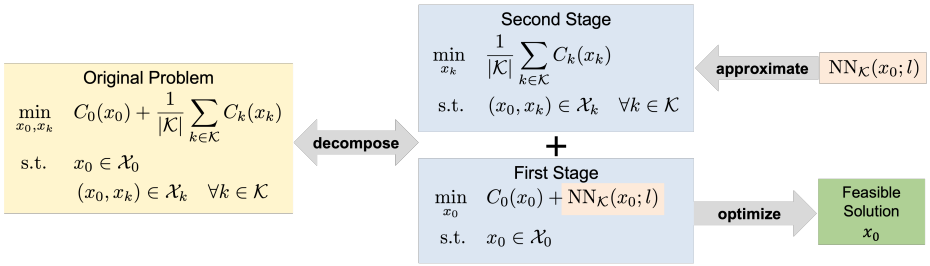


Fig. 1. A schematic of the approximate-and-optimize framework. The expected cost function is approximated by a neural network $\mathrm{NN}_{\mathcal{K}}(x_0; l)$, and is directly embedded into the first-stage model.

In this study, we introduce an innovative machine learning-based approximate-and-optimize method to efficiently derive high-quality, feasible solutions for the SC-ACOPF. The entire process is illustrated in Fig. 1. Following a similar strategy to [19], we decompose the SC-ACOPF model into a first-stage problem representing the base case and a second-stage problem associated with contingencies. Here, x_0 ($\mathcal{X}_0$) and x_k ($\mathcal{X}_k$) denote decision variables (feasible regions), respectively. In contrast to direct prediction methods, we employ regularized neural networks featuring *smooth activation functions* to approximate the second-stage cost. The well-trained neural network function, denoted as $\mathrm{NN}_{\mathcal{K}}(x_0; l)$, acts as a surrogate and seamlessly integrates into the first-stage problem, yielding a non-linear, smooth, and non-convex model of significantly reduced size. Subsequently, the resulting problem, incorporating an explicit neural network function in the objective, is solved using off-the-shelf non-linear solvers, such as Ipopt [28]. The proposed method aligns with the category of integrating data-driven surrogate models into mathematical optimization [17]. It is noteworthy that our approximate-and-optimize method exhibits versatility, making it applicable to various problems formulated as continuous two-stage stochastic programs [16].

We summarize the distinct contributions of our work as follows.

– We propose a learning based approximate-and-optimize framework to efficiently identify high-quality solutions to SC-ACOPF problems.

- We propose to utilize smooth and regularized neural networks as approximation functions to alleviate the computational burden and boost performances.
- We conduct computational experiments on a 500-bus network from the GO Competition and demonstrate that our proposed method can produce high-quality base case solutions faster than baseline methods.

The remainder of the paper is organized as follows. Section 2 introduces an SC-ACOPF formulation of our interest. Section 3 presents the approximate-and-optimize framework and the neural network models. Section 4 reports our computational results and associated datasets. We conclude in Section 5 with some final remarks.

2 Problem Formulation

In this work, we adopt the SC-ACOPF formulation proposed in the GO Competition. For a detailed formulation, readers are referred to [6]. Let $\mathcal{K}$ denote a set of contingencies, and we use subscripts 0 and $k \in \mathcal{K}$ to represent the base case and a contingency case, respectively. The decision variables x encompass bus voltage magnitudes and angles, active and reactive power from generators, and controllable shunt susceptances. The SC-ACOPF model of interest is presented in (1):

$$\min_{x_0, s_0^g, s_0^h x_k, s_k^g, s_k^h} \quad C(x_0) + C_0\left(s_0^h, s_0^g\right) + \frac{1}{|\mathcal{K}|}\sum_{k\in\mathcal{K}} C_k\left(s_k^h, s_k^g\right) \tag{1a}$$

$$\text{s.t.} \quad h_0(x_0) = s_0^h, \quad g_0(x_0) \le s_0^g, \quad s_0^g \ge 0 \tag{1b}$$

$$h_k(x_k) = s_k^h, \quad g_k(x_k) \le s_k^g, \quad s_k^g \ge 0 \qquad \forall k \in \mathcal{K} \tag{1c}$$

$$\phi_k(x_0, x_k) = 0 \qquad \forall k \in \mathcal{K} \tag{1d}$$

The equality constraints $h(x) = 0$ define power flow into lines and transformers, power flow balance at generators, while the inequalities $g(x) \le 0$ enforce line current ratings and transformer power ratings constraints. Slack variables s^h and s^g are introduced to quantify the violation of constraints $h(x) = 0$ and $g(x) \le 0$, respectively, as shown in (1b) and (1c). These slack variables are then penalized in the objective (1a).

Constraints (1d) establish relationships between the decision variables x_0 in the base case and their counterparts x_k during contingency k. The real and reactive power of each generator in a post-contingency state are subject to constraints imposed by two controllers: a droop control system and a voltage-control system [2]. In particular, the adjustment of real power in the post-contingency state follows a predefined droop slope while being confined within its operational bounds. Mathematically, this adjustment is captured by *complementarity*

constraints as depicted in (2).

$$\begin{aligned} &\rho^{+}_{g,k} - \rho^{-}_{g,k} = p_{g,k} - (p_{g,0} + A_g \delta_k), \\ &0 \le \rho^{-}_{g,k} \perp \overline{P}_g - p_{g,k} \ge 0, \\ &0 \le \rho^{+}_{g,k} \perp p_{g,k} - \underline{P}_g \ge 0, \end{aligned} \tag{2}$$

where $p_{g,0}$ and $p_{g,k}$ denote the active power for generator g in the base case and contingency k, respectively. The variable $p_{g,k}$ is bounded by $\underline{P}_g$ and $\overline{P}_g$. A_g is the predefined droop slope for generator g, and δ_k is the frequency deviation in contingency k.

Furthermore, if a generator is active in a contingency, it will attempt to maintain its bus voltage as in the base case by adjusting its reactive power, unless it reaches the operational bound. The resulting constraints are formulated in (3):

$$\begin{aligned} &\nu^{+}_{n,k} - \nu^{-}_{n,k} = v_{n,k} - v_{n,0}, \\ &0 \le \nu^{-}_{n,k} \perp \overline{Q}_g - q_{g,k} \ge 0, \\ &0 \le \nu^{+}_{n,k} \perp q_{g,k} - \underline{Q}_g \ge 0, \end{aligned} \tag{3}$$

where $v_{n,k}$ and $v_{n,0}$ are the voltage for bus n in contingency k and the base case, respectively. The variable $q_{g,k}$ represents the reactive power for generator g in contingency k, with lower bound $\underline{Q}_g$ and upper bound $\overline{Q}_g$. $\nu^{+}_{n,k}$ and $\nu^{-}_{n,k}$ are the voltage increase and decrease from the base case for bus n in contingency k.

Let piece-wise linear functions $C(x), C_0(s^h_0, s^g_0)$ and $C_k(s^h_k, s^g_k)$ denote the operating cost for generators, penalty cost for the base and contingency cases, respectively. The SC-ACOPF problem aims to find operating conditions for the base case and every contingency case, such that relevant power system constraints are satisfied and the total cost is minimized.

We are faced with three main challenges when solving model (1).

i. **Large size**. The number of variables and constraints grows proportionally with the size of the contingency set $\mathcal{K}$, which is of the same order of magnitude as the number of buses and branches in a power network. Hence, (1) usually becomes a very large-scale optimization model even for a small-sized power system. For example, in the GO Competition, networks with thousands of generators, buses, and branches are considered, and the corresponding SC-ACOPF problems can have millions of variables and constraints.
ii. **Non-linearity and non-convexity**. Functions $h(x)$ and $g(x)$ rely on products of variables and trigonometric functions to describe power flow. As a result, the feasible region is highly non-linear and non-convex.
iii. **Complementarity**. A large number of complementarity constraints $\phi_k(x_0, x_k) = 0$ exist, but mathematical programs with complementarity constraints are notoriously difficult to solve due to the non-satisfaction of constraint qualification [21], such as the Mangasarian–Fromovitz constraint qualification.

Therefore, solving such a large-scale, non-linear and non-convex program (1) is extremely demanding.

3 Solution Approach

3.1 An Approximate-and-Optimize Framework

We first cast model (1) as a two-stage optimization problem as follows:

$$\begin{aligned} \min_{x_0, s_0^h, s_0^g} \quad & C(x_0) + C_0\left(s_0^h, s_0^g\right) + r_{\mathcal{K}}(x_0) \\ \text{s.t.} \quad & \text{constraints (1b)}, \end{aligned} \tag{4}$$

where $r_{\mathcal{K}}(x_0)$ is defined as

$$\begin{aligned} r_{\mathcal{K}}(x_0) \coloneqq \min_{x_k, s_k^h, s_k^g} \quad & \frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} C_k\left(s_k^h, s_k^g\right) \\ \text{s.t.} \quad & \text{constraints (1c) - (1d)}. \end{aligned} \tag{5}$$

Specifically, the constraints and objective penalties associated with contingencies are collectively replaced by a single function $r_{\mathcal{K}}(x_0)$ in (4) and (5). This function represents the optimal value for the second-stage optimization problem and is dependent solely on the base case solution x_0. Consequently, the model size of (4) is significantly smaller than that of (1). Unfortunately, obtaining a closed form for $r_{\mathcal{K}}(x_0)$ is not feasible. Given a fixed x_0, evaluating $r_{\mathcal{K}}(x_0)$, as shown in model (5), involves solving $|\mathcal{K}|$ independent ACOPF-like optimization problems in parallel. Using this, one can collect samples $\{x_0^1, x_0^2, ..., x_0^m\}$ from the x_0 space and obtain $\{r(x_0^1), r(x_0^2), ..., r(x_0^m)\}$. Motivated by the well-known *universal approximation theorem* [14], we choose to approximate the function $r_{\mathcal{K}}(x_0)$ using neural networks.

For a given set of contingencies $\mathcal{K}$, the system load vector l (including active and reactive power of load buses), and a base case solution x_0, the function $r_{\mathcal{K}}(x_0)$ is approximated by a well-trained neural network function $\mathrm{NN}_{\mathcal{K}}(x_0, l)$. This neural network is explicitly embedded into model (4) as a surrogate function for $r_{\mathcal{K}}(x_0)$. Subsequently, off-the-shelf nonlinear optimization solvers can be employed to address the resulting problem (6) and obtain a high-quality solution x_0. This methodology is referred to as an "approximate-and-optimize" method.

$$\begin{aligned} \min_{x_0, s_0^h, s_0^g} \quad & C(x_0) + C_0\left(s_0^h, s_0^g\right) + \mathrm{NN}_{\mathcal{K}}(x_0; l) \\ \text{s.t.} \quad & \text{constraints (1b)}. \end{aligned} \tag{6}$$

The proposed approximate-and-optimize framework offers several advantages when tackling SC-ACOPF problems:

- Only variables and constraints associated with the base case are involved in model (6), and all information related to each contingency is incorporated into a single approximation term. This reduction in problem size significantly facilitates solution efforts.
- The use of an approximation term eliminates complementarity constraints (1d), which are known to pose considerable challenges for state-of-the-art nonlinear optimization solvers.

– The resulting model (6) takes a form similar to classic ACOPF models, allowing the utilization of well-developed techniques, such as interior point methods, commonly employed for ACOPF problems [23].

3.2 Neural Networks

In the approximate-and-optimize framework for solving SC-ACOPF problems, we leverage fully connected *Neural Networks (NNs)* to approximate $r_{\mathcal{K}}(x_0)$. An NN consists of multiple layers of inter-connected neurons. Data is fed into the input layer in an NN and passes through several parameterized hidden layers, ultimately returning an output. Activation functions introduce non-linearity and often non-convexity, enabling neural networks to approximate complex functions with the desired accuracy. The design of a properly engineered loss function is crucial, as it is minimized to identify well-parameterized NNs that effectively capture the relationship between inputs and outputs. It has been established that, under very mild assumptions, NN models exhibit the universal approximation property [24, 25].

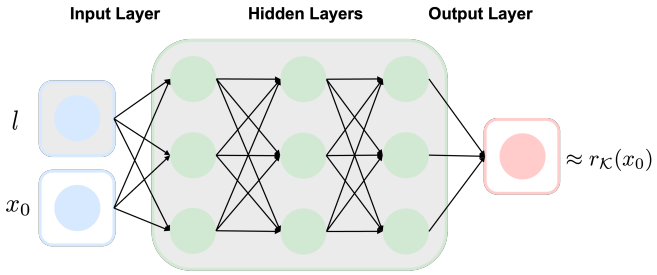


Fig. 2. An NN designed for approximating the contingency penalty in SC-ACOPF.

Using historical data x_0, we first compute $r_{\mathcal{K}}(x_0)$ and then train a neural network to serve as an approximation function for $r_{\mathcal{K}}(x_0)$. In this study, neural networks with two hidden layers are employed to approximate $r_{\mathcal{K}}(x_0)$, represented as in (7). Specifically, as illustrated in Fig. 2, the power system load vector l is concatenated with the base case variables x_0 and fed into an NN as the input. Let W_h and b_h denote weights and bias for the h-th layer, and σ represent the activation function.

$$\mathrm{NN}_{\mathcal{K}}(x_0; l) := \sigma\left(\sigma\left([x_0, l]W_1 + b_1\right)W_2 + b_2\right)W_3 + b_3 \tag{7}$$

Once a well-trained neural network $\mathrm{NN}_{\mathcal{K}}(x_0; l)$ is obtained, it can replace $r_{\mathcal{K}}(x_0)$. Notably, in the SC-ACOPF problem of interest, the power system load vector l is given, making $\mathrm{NN}_{\mathcal{K}}(x_0; l)$ solely dependent on x_0 and easily incorporated into model (6). Regularized neural networks with smooth activation functions are adopted, resulting in a model (6) suitable for interior point methods.

Smooth Activation Function Since model (6) will be solved using an interior point method, the function $\mathrm{NN}_{\mathcal{K}}(x_0; l)$ needs to be smooth, preferably twice differentiable. To achieve this, we opt for the *Gaussian Error Linear Unit* (GELU) [13], a smoother variant of rectified linear units [18], as the activation function. The GELU formula is defined as (8).

$$\mathrm{GELU}(y) \coloneqq y\Phi(y), \tag{8}$$

Here, $\Phi(y) \coloneqq \mathbb{P}(Y \leq y)$ represents the cumulative function with Y following a standard normal distribution. The GELU activation function is smooth, twice differentiable, non-convex, and exhibits curvature at all points [15]. With these desirable properties, GELU activation functions have demonstrated their effectiveness in various state-of-the-art neural network architectures, such as GPT [4, 20].

Regularization To mitigate numerical challenges when solving model (6), we opt to incorporate regularization into NN training. *Gradient regularization* is a widely used method for preventing overfitting and enhancing smoothness by penalizing substantial changes in the output of a neural network [9, 12]. This regularization technique has demonstrated its efficacy in promoting stable training processes and improving classification accuracy in computer vision tasks, particularly when dealing with small training datasets [27]. We propose to regularize the input gradient, $\nabla_{x_0} \mathrm{NN}_{\mathcal{K}}(x_0; l)$. The loss function during training is then defined as follows:

$$\mathrm{Loss}\left(x_0, \{W_h\}_{h=1}^{H}\right) \coloneqq \left(\mathrm{NN}_{\mathcal{K}}(x_0; l) - r_{\mathcal{K}}(x_0)\right)^2 + \lambda \left\|\nabla_{x_0} \mathrm{NN}_{\mathcal{K}}(x_0; l)\right\|_2^2.$$

4 Computational Experiments

4.1 Test Set-up

All experiments in this section were conducted on a server equipped with a 12th Gen Intel(R) Core(TM) i9-12900K and NVIDIA GeForce RTX 3090 GPU. We utilized Ipopt version 3.14.13 [28], operating with the linear solver MUMPS 5.6.0, as the nonlinear solver for all experiments. The training of neural networks was performed using PyTorch v2.0.1, and Scikit-learn 1.2.2 was employed for various tasks related to machine learning. We make our code publicly available at `https://github.com/NetSysOpt/Approximate-and-Optimize-for-SCOPF`.

4.2 Baseline

The PowerModelsSecurityConstrained.jl v0.10.0 [5] serves as the baseline algorithm in the GO competition, and we adopt it as our benchmark algorithm, denoted by `Benchmark`. This algorithm commences by solving a *Direct Current Optimal Power Flow* (DCOPF) problem, and the solution is then utilized to formulate an SC-DCOPF problem. Solving this SC-DCOPF problem provides an initial solution for the SC-ACOPF. Subsequently, an ACOPF problem is solved as a post-processing step to obtain the final solution for the base case.

4.3 Dataset Generation

We have chosen the Network_01R-10 instance from the GO competition datasets [1] to create a benchmark dataset. This particular instance features 500 buses, 599 branches, 90 generators, 51 generator contingencies, and 326 branch contingencies. In practical scenarios, the set of contingencies is typically predefined by experts and often includes the failure of individual vulnerable generators and branches. Therefore, for our dataset, the contingency set remains the same for every instance. We introduce perturbations to the load demand of each bus by a factor uniformly distributed over $[0.9, 1.1]$, resulting in a total of 50 instances.

To generate the dataset for training the surrogate neural network for the contingency penalty, we need to collect feasible base case solutions, which will serve as part of the input to the neural network. In practice, feasible base case solutions can be obtained from historical data for a specific power network under various load profiles. In this work, we follow the procedure outlined below. First, we execute the `Benchmark` algorithm to obtain a feasible base case solution $\hat{x}_0$. Next, we randomly sample a subset of bus nodes and generators, and denote the indices of these variables as I. We then perturb the voltage, active power, and reactive power of the sampled buses and generators by factors uniformly sampled from $[0.9, 1.1]$, denoted as $\hat{x}_{0,I}$. Subsequently, a feasible base case solution is obtained by solving model (9).

$$\begin{aligned} \min_{x_0, s_0^h, s_0^g} \quad & \|x_{0,I} - \hat{x}_{0,I}\|_2^2 + \|s_0^h\|_2^2 + \|s_0^g\|_2^2 \\ \text{s.t.} \quad & \text{constraints (1b)} \end{aligned} \tag{9}$$

For each collected x_0, we solve the second-stage problem (5) using the `Benchmark` algorithm and obtain $r_{\mathcal{K}}(x_0)$. The histogram of $\log_{10}(r_{\mathcal{K}}(x_0))$ is presented in Fig. 3. Notably, the values of $r_{\mathcal{K}}(x_0)$ concentrate around 1.2×10^6.

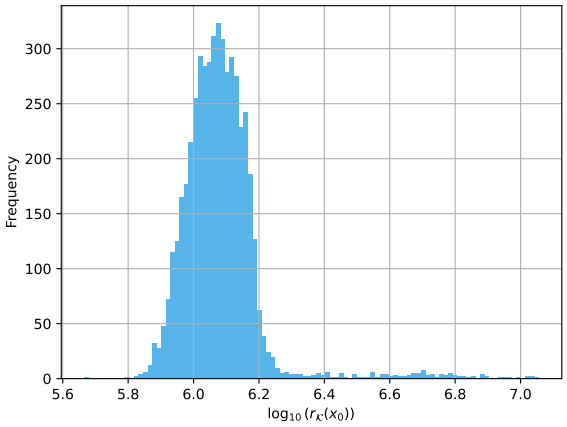


Fig. 3. Histogram of $\log_{10}(r_{\mathcal{K}}(x_0))$ in the training data.

4.4 Configuration

We trained our neural networks using the Adam optimizer with a learning rate of 3×10^{-4} and a batch size of 64 for 1000 epochs. The parameter λ was chosen from the set $\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$. For the neural network architecture, we considered a simple two-layer fully connected neural network with N hidden neurons, where $N \in \{16, 32, 64, 128\}$. We generated 100 scenarios for training, 20 scenarios for validation, and 50 scenarios for testing. In each training scenario, 50 feasible base case solutions were collected. After training and validation, the neural network with 16 hidden neurons in the first layer, 64 hidden neurons in the second layer, and $\lambda = 10^{-3}$ was chosen in the surrogate function $\mathrm{NN}_{\mathcal{K}}(x_0; l)$. The corresponding approximate-and-optimize algorithm is denoted by `A&O-NN`. To illustrate how utilizing neural networks with regularization as approximation functions affects algorithmic performance, we replaced them with the following two surrogate functions:

- `A&O-Lasso`: a lasso model;
- `A&O-NN(w/o reg.)`: a neural network that has the same architecture but is trained without regularization.

4.5 Results and Discussion

We present the computational results in Table 1. Columns "Gen. Cost", "Base Penalty", and "Contingency Penalty" represent geometric means of generation costs, constraint violation penalties for the base case and contingencies, respectively. The total objective cost (geometric mean) and its normalized value (with respect to `A&O-NN`) are reported in columns "Obj." and "Ratio", respectively. The

average total time (including reading data, loading and solving models, writing solutions, and solving sub-problems) across 50 testing instances is reported in column "Time (sec)", while its breakdown is displayed in Fig 4.

Table 1. Computational results for the 500-bus system

Method	Gen. Cost	Penalty		Obj.	Ratio	Time (sec)
		Base	Contingency			
`A&O-NN`	46,196	0.32	561,824	608,038	1.00	12.0
`Benchmark`	34,427	0.03	1,362,096	1,396,569	2.30	34.6
`A&O-Lasso`	45,719	0.45	1,425,178	1,471,041	2.42	12.0
`A&O-NN(w/o reg.)`	46,042	0.73	621,574	667,634	1.10	12.8

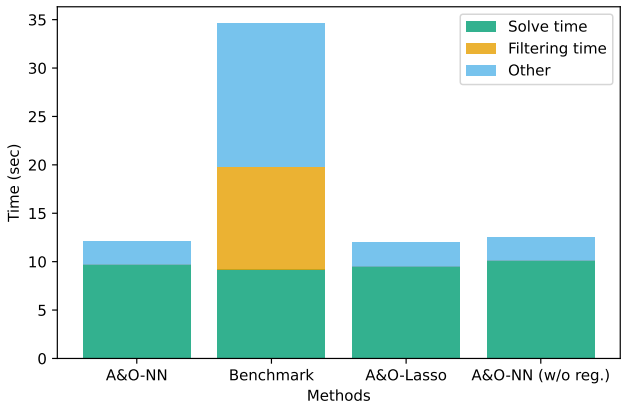


Fig. 4. The breakdown of solution times for each method. "Solve time" represents the cumulative time for constructing and solving models in each method, while "Filter time" represents the contingency-filtering time in `Benchmark`. "Other" denotes the total time for loading the data, writing solutions to output files, and it also includes solution-recovering time in `Benchmark`.

`A&O-NN` v.s. `Benchmark` Firstly, we compare the computational performances of our proposed method against that of `Benchmark`. With the use of `A&O-NN`, the average solution time is reduced from 34.6 seconds to 12.0 seconds. Moreover, `A&O-NN` yields solutions with objective values that are 50% smaller than those obtained with `Benchmark`. It is noteworthy that while the operational cost for generators in `Benchmark` is smaller than that in `A&O-NN`, the penalty for constraint violations in the second-stage problem is significantly larger (the same

order of magnitude as training samples, as indicated in Fig 3). This difference in objective costs is expected since `Benchmark` follows a greedy approach that primarily optimizes the base case problem, whereas our approximate-and-optimize framework `A&O-NN` seeks a balance between generation costs and constraint violation penalties. Additionally, since `A&O-NN` involves solving a small-sized nonlinear model, its computational time is 60% less. On the contrary, `Benchmark` employs a filtering procedure to introduce contingencies and needs to iteratively solve SC-DCOPF models as well as recover feasible solutions, resulting in a computationally expensive process, as indicated in Fig. 4.

`A&O-NN` **v.s.** `A&O-Lasso` To showcase the effectiveness of neural networks in approximating highly non-linear and non-convex functions, we compare the computational performance of `A&O-NN` with that of `A&O-Lasso`. While `A&O-Lasso` exhibits similar solution times, it incurs 1.42 times more objective costs. As anticipated, a simple linear surrogate function lacks the capacity to accurately approximate $r_{\mathcal{K}}(x_0)$. The substantial reduction in objective costs underscores the advantage of employing sophisticated neural networks as surrogate functions, enabling more informed decision-making compared to using basic linear approximators.

`A&O-NN` **v.s** `A&O-NN(w/o reg.)` We now assess the impact of regularizing neural networks by comparing `A&O-NN` against `A&O-NN (w/o reg.)`. The two methods exhibit similar monetary costs. However, `A&O-NN` demonstrates the ability to identify solutions with smaller base and contingency penalties slightly faster. This observation suggests that solvers, such as Ipopt, may derive benefits from the use of regularized neural networks when addressing embedded problems within the approximate-and-optimize framework.

Out of sample We evaluate our proposed algorithm on an out-of-sample dataset, comparing its performance with that of `Benchmark`, `A&O-Lasso`, and `A&O-NN (w/o reg.)`. In this new dataset, load profiles are obtained by multiplying the nominal load demand by a factor sampled from $[0.7, 1.3]$. The computational results, presented in Table 2, once again demonstrate that `A&O-NN` outperforms the other three methods in producing high-quality solutions and is computationally more efficient than `Benchmark`. The results affirm that `A&O-NN` exhibits superior generalization capabilities, efficiently addressing SC-ACOPF problems with previously unseen load demands.

5 Conclusions

In this study, we introduce a novel machine learning-based approximate-and-optimize framework for addressing SC-ACOPF problems. The SC-ACOPF model is systematically decomposed into a two-stage stochastic program, where the first stage corresponds to base case decisions, and the second stage addresses

Table 2. Out-of-sample results for the 500-bus system

Method	Gen. Cost	Penalty		Obj.	Ratio	Time (sec)
		Base	Contingency			
`A&O-NN`	46,095	0.32	605,414	651,648	1.00	12.1
`Benchmark`	34,509	0.03	1,421,137	1,455,849	2.23	32.1
`A&O-Lasso`	45,699	0.49	1,488,845	1,534,974	2.36	11.8
`A&O-NN(w/o reg.)`	45,628	0.8	672,342	720,324	1.11	12.8

post-contingency scenarios. To facilitate this, a neural network is trained using system loads and base case decisions as inputs, incorporating smooth activation functions and regularization techniques. The well-trained neural network is seamlessly integrated into the objective function of the first-stage problem, serving as a surrogate for contingency penalties. The resulting model is subsequently solved using established nonlinear solvers. Our proposed method is evaluated on a 500-bus power system from the GO competition. Computational results demonstrate the superiority of our approach, denoted as `A&O-NN`, outperforming the benchmark method in the GO Competition with a 50% reduction in the objective value and a 60% decrease in computational time. It is worth noting that, in this work, we assume a fixed contingency set for a power network across all scenarios. In future work, we aim to incorporate dynamic contingency information into the neural network training process. Additionally, when deploying our approximate-and-optimize framework for larger power networks, the challenge of a rapidly growing number of neurons in the input layers will be addressed, requiring exploration of alternative network architectures and training strategies for improved generalization capabilities.

Acknowledgment. This research is supported by the National Key R&D Program of China (Grant No. 2022YFA1003900), Shenzhen Science and Technology Program (Grant No. RCBS20221008093309021), and National Natural Science Foundation of China (Grant No. 12301416).

References

1. Datasets: Challenge 1 (2019), `https://gocompetition.energy.gov/challenges/22/datasets`
2. Aravena, I., Molzahn, D.K., Zhang, S., Petra, C.G., Curtis, F.E., Tu, S., Wächter, A., Wei, E., Wong, E., Gholami, A., et al.: Recent developments in security-constrained ac optimal power flow: Overview of challenge 1 in the arpa-e grid optimization competition. Operations Research (2023)
3. Bazrafshan, M., Baker, K., Mohammadi, J.: Computationally efficient solutions for large-scale security-constrained optimal power flow. arXiv preprint arXiv:2006.00585 (2020)

4. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners. Advances in neural information processing systems **33**, 1877–1901 (2020)
5. Coffrin, C.: Powermodelssecurityconstrained.jl (2020), `https://github.com/lanl-ansi/PowerModelsSecurityConstrained.jl`
6. Competition, G.O.: Scopf problem formulation: Challenge 1, `https://gocompetition.energy.gov/sites/default/files/SCOPF\_Problem\_Formulation\_\_Challenge\_1\_20190412.pdf`
7. Crozier, C., Baker, K., Du, Y., Mohammadi, J., Li, M.: Data-driven contingency selection for fast security constrained optimal power flow. In: 2022 17th International Conference on Probabilistic Methods Applied to Power Systems (PMAPS). pp. 1–6. IEEE (2022)
8. Curtis, F.E., Molzahn, D.K., Tu, S., Wächter, A., Wei, E., Wong, E.: A decomposition algorithm with fast identification of critical contingencies for large-scale security-constrained ac-opf. Operations Research (2023)
9. Drucker, H., Le Cun, Y.: Improving generalization performance using double backpropagation. IEEE transactions on neural networks **3**(6), 991–997 (1992)
10. Dumouchelle, J., Patel, R., Khalil, E.B., Bodur, M.: Neur2SP: Neural two-stage stochastic programming. Advances in Neural Information Processing Systems **35** (2022)
11. Gholami, A., Sun, K., Zhang, S., Sun, X.A.: An admm-based distributed optimization method for solving security-constrained alternating current optimal power flow. Operations Research **71**(6), 2045–2060 (2023)
12. Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., Courville, A.C.: Improved training of wasserstein gans. Advances in neural information processing systems **30** (2017)
13. Hendrycks, D., Gimpel, K.: Gaussian error linear units (gelus). arXiv preprint arXiv:1606.08415 (2016)
14. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural networks **2**(5), 359–366 (1989)
15. Lee, M.: Gelu activation function in deep learning: A comprehensive mathematical analysis and performance. arXiv preprint arXiv:2305.12073 (2023)
16. Li, H., Cui, Y.: A decomposition algorithm for two-stage stochastic programs with nonconvex recourse functions. SIAM Journal on Optimization **34**(1), 306–335 (2024)
17. Misener, R., Biegler, L.: Formulating data-driven surrogate models for process optimization. Computers & Chemical Engineering **179**, 108411 (2023)
18. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th international conference on machine learning (ICML-10). pp. 807–814 (2010)
19. Petra, C.G., Aravena, I.: A surrogate-based asynchronous decomposition technique for realistic security-constrained optimal power flow problems. Operations Research (2023)
20. Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al.: Improving language understanding by generative pre-training (2018)
21. Raghunathan, A.U., Biegler, L.T.: An interior point method for mathematical programs with complementarity constraints (mpccs). SIAM Journal on Optimization **15**(3), 720–750 (2005)
22. Rahman, J., Feng, C., Zhang, J.: Machine learning-aided security constrained optimal power flow. In: 2020 IEEE Power & Energy Society General Meeting (PESGM). pp. 1–5. IEEE (2020)

23. Sadat, S.A.: Evaluating the performance of various acopf formulations using nonlinear interior-point method. In: 2021 IEEE International Smart Cities Conference (ISC2). pp. 1–7 (2021). https://doi.org/10.1109/ISC253183.2021.9562928
24. Scarselli, F., Tsoi, A.C.: Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results. Neural networks **11**(1), 15–37 (1998)
25. Sonoda, S., Murata, N.: Neural network with unbounded activation functions is universal approximator. Applied and Computational Harmonic Analysis **43**(2), 233–268 (2017)
26. Van Hentenryck, P.: Machine learning for optimal power flows. Tutorials in Operations Research: Emerging Optimization Methods and Modeling Techniques with Applications pp. 62–82 (2021)
27. Varga, D., Csiszárik, A., Zombori, Z.: Gradient regularization improves accuracy of discriminative models. arXiv preprint arXiv:1712.09936 (2017)
28. Wächter, A., Biegler, L.T.: On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Mathematical programming **106**, 25–57 (2006)

Auto-sktime: Automated Time Series Forecasting

Marc-André Zöller[1,3][0000−0001−8705−9862], Marius Lindauer[2][0000−0002−9675−3175], and Marco F. Huber[3,4][0000−0002−8250−2092]

[1] USU GmbH, Rüppurrer Str. 1, Karlsruhe, Germany mazoeller@gmail.com
[2] Institute of Artificial Intelligence, Leibniz University Hannover, Welfengarten 1, Hannover, Germany m.lindauer@ai.uni-hannover.de
[3] Institute of Industrial Manufacturing and Management IFF, University of Stuttgart
[4] Center for Cyber Cognitive Intelligence CCI, Fraunhofer IPA, Nobelstr. 12, Stuttgart, Germany marco.huber@ieee.org

Abstract. In today's data-driven landscape, time series forecasting is pivotal in decision-making across various sectors. Yet, the proliferation of more diverse time series data, coupled with the expanding landscape of available forecasting methods, poses significant challenges for forecasters. To meet the growing demand for efficient forecasting, we introduce AUTO-SKTIME, a novel framework for automated time series forecasting. The proposed framework uses the power of automated machine learning (AutoML) techniques to automate the creation of the entire forecasting pipeline. The framework employs Bayesian optimization to automatically construct pipelines from statistical, machine learning (ML) and deep neural network (DNN) models. Furthermore, we propose three essential improvements to adapt AutoML to time series data. First, pipeline templates to account for the different supported forecasting models. Second, a novel warm-starting technique to start the optimization from prior optimization runs. Third, we adapt multi-fidelity optimizations to make them applicable to a search space containing statistical, ML and DNN models. Experimental results on 64 diverse real-world time series datasets demonstrate the effectiveness and efficiency of the framework, outperforming traditional methods while requiring minimal human involvement.

Keywords: Automated Machine Learning · Time Series · Forecasting.

1 Introduction

Time series forecasting is crucial for applications like economics, finance, and manufacturing. It involves analyzing historical data and using statistical, machine learning (ML) or deep neural network (DNN) models to predict future trends. With accurate forecasts, businesses can efficiently allocate resources, plan production schedules, and manage inventories, reducing waste and costs. Moreover, time series forecasting can help manufacturers detecting potential issues before they become problematic. For example, predictive maintenance models can analyze sensor data to anticipate when a machine will need maintenance.

Due to the tedious and error-prone process of creating well-performing models [3], organizations struggle to create forecasting models. This issue is further

amplified by a shortage of skilled data scientists [2]. This has led to an increasing demand for automation that can enable businesses to develop accurate forecasting models without requiring a high level of technical expertise. Automated machine learning (AutoML) is an emerging field aiming to automate the process of creating, evaluating, and deploying ML models. By leveraging heuristics and techniques from optimization theory, AutoML tools search for the best models and hyperparameters to create accurate forecasts with minimal human interaction. Optimally, the AutoML system provides a complete end-to-end automation covering steps like data cleaning, model selection, and hyperparameter tuning.

More formally, AutoML aims at generating a pipeline, described by its hyperparameters $\boldsymbol{\lambda}^\star$, minimizing

$$\boldsymbol{\lambda}^\star \in \underset{\boldsymbol{\lambda} \in \Lambda}{\arg\min}\ \mathcal{L}(\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{valid}}, \boldsymbol{\lambda}) \tag{1}$$

with Λ the configuration space containing all possible hyperparameter combinations, $\mathcal{L}$ a validation loss function, and $\mathcal{D}$ a dataset split into a train ($\mathcal{D}_{\text{train}}$) and validation part ($\mathcal{D}_{\text{valid}}$). In general, the features of Equation (1) are not known as they depend on $\mathcal{D}$, making the use of efficient solvers impossible.

While prior work on AutoML mainly focused on classification and regression, e.g., [15,17], some work has been done to apply AutoML techniques to time series forecasting, e.g., [14,40,42]. Yet, those methods mainly apply AutoML procedures developed for tabular data to time series data. Consequently, the potential of AutoML adapted to time series is currently not fully utilized. For example, multi-fidelity approximations, which are an integral part of AutoML [16], usually use either a random subset of the training data or a number of fitting iterations as budget. Yet, both are not easily applicable to time series: random subsets distort the temporal relation of the data, and many forecasting models do not support iterative fitting. Therefore, we propose AUTO-SKTIME, an AutoML framework for end-to-end time series forecasting. Our contributions can be summarized as follows:

1. We propose AUTO-SKTIME, a framework combining statistical, ML, and DNN state-of-the-art (SOTA) techniques for time series forecasting, making it applicable to a wide range of time series. We use a *templating* method to select appropriate pipelines given the input data. Each template encodes specific best practices to ensure an efficient yet flexible creation of pipelines.
2. We propose a novel method for warm-starting the AutoML optimization based on prior optimizations to increase the sampling efficiency of the optimization. While this method is specifically designed for time series, it can also be transferred to other AutoML problems like classification.
3. Current multi-fidelity optimization techniques are not applicable to common time series data. We propose a novel multi-fidelity budget enabling the benefits of multi-fidelity approximations for all kinds of time series data.

The rest of this work is structured as follows: Section 2 introduces related work in time series forecasting and AutoML. The proposed AUTO-SKTIME framework is described in Section 3 and validated in Section 4. Finally, the results are discussed in Section 5.

2 Background and Related Work

First, we provide an introduction to existing forecasting models and techniques for automating time series forecasting. Finally, we also provide a short introduction to Bayesian optimization.

2.1 Models for Time Series Forecasting

The first approaches for time series forecasting used statistical models like autoregressive integrated moving average (ARIMA), exponential smoothing (ES), and seasonal decomposition (STL). These models assume that future observations are related to past observations and can be predicted using the patterns observed in the data. The ARIMA model [5], for example, captures the autocorrelation in the data and uses it to predict future observations. ES [36] assigns different weights to past observations, with more recent observations being given more weight. STL models decompose the time series into seasonal, trend, and residual components [21]. While statistical models offer interpretability and insights into the underlying dynamics of the data, they often struggle to handle complex relationships between variables [21].

ML models have gained significant attention in time series forecasting due to their ability to capture complex patterns and making fewer assumptions about the data. By modeling time series forecasts as a regression problem, supervised learning methods can be applied. Time series forecasting has seen extensive exploration of various ML models, such as random forests [34] or gradient boosting trees [22]. ML models offer flexibility, scalability, and the potential to handle diverse time series data with varying characteristics [31]. However, achieving success with them often requires careful tuning of hyperparameters and access to large amounts of training data [22], which usually is not available if only a single time series is considered.

More recently, DNNs have emerged as powerful models for time series forecasting due to their ability to model temporal relations within the data. Specifically, recurrent neural networks (RNNs), like long-term short memory (LSTM) [18] and gated recurrent units (GRU) [12], have shown remarkable performance in capturing long-term dependencies in sequential data [48]. Alternatively, transformers have been applied more recently to time series forecasting [50]. Additionally, feedforward networks can forecast time series by transforming the sequential input into a fixed-size feature representation [4]. However, the successful application of neural networks for time series forecasting requires careful architecture design, appropriate activation functions, and regularization techniques. Ongoing research focuses on hybrid architectures, ensemble methods, and attention mechanisms to enhance the forecasting accuracy of DNN models, e.g., [50].

2.2 Automated Time Series Forecasting

Extensive studies have examined how to automate the creation and fine-tuning of statistical time series forecasting algorithms. For example, researchers have

employed the Box-Jenkins methodology [5] to find optimized ARIMA models while the state-space methodology [36] can be applied to ES models. Yet, this prior work does not consider automating the selection of the underlying statistical model. The demand for automation in forecasting increases due to the various processes generating time series data and the ever-increasing number of models for time series forecasting (ML and DNN models).

While researchers have proposed various AutoML approaches, the majority of these focus on standard learning tasks such as tabular or image classification, e.g., [15,17]. These approaches typically treat the generation of ML pipelines as a black-box optimization problem: Given a dataset $\mathcal{D}$ and loss function $\mathcal{L}$, AutoML searches within a search space Λ to find a pipeline $\boldsymbol{\lambda}^\star$ minimizing the validation loss. Researchers often address this optimization process using sample-efficient Bayesian optimization (BO) [6]. Yet, time series forecasting, especially when forecasting single univariate time series, often has too little training data for complex regression models.

Multiple dedicated AutoML frameworks for time series forecasting exist. TSPO [13] transforms the forecasting problem into a tabular regression problem compatible with standard AutoML. Auto-PyTorch [14] constructs ensembles of DNNs. While it contains many standard features from AutoML—like multi-fidelity approximations and ensemble learning—it confines itself to DNNs. [42] propose automatic forecasting of time series based on genetic optimization. By combining statistical and ML models to a single search space, forecasting of time series panel data is possible. Similarly, AutoTS [10] also employs genetic optimization combined with creating an ensemble of evaluated candidates. BOAT [24] uses BO to automatize the creation of statistical and ML models. The optimization is warm-started using the typical AutoML approach via meta-features of the given dataset. HyperTS [49] also uses genetic optimization of statistical and DNN models combined with a greedy ensemble construction for univariate and multivariate time series. AutoGluon-TS [41] combines DNNs, ML, and statistical models for forecasting. Finally, AutoAI-TS [40] offers an end-to-end automation for forecasting of univariate and multivariate time series. However, it uses a fixed ensemble of diverse models and does not optimize an internal objective function as it is characteristic for AutoML. Similarly, PyAF [8] builds best-practice pipelines by combining different preprocessing steps with ML regression models using their default hyperparameters.

While different approaches of AutoML for forecasting exist, no framework combines all SOTA performance improvements of AutoML frameworks—namely ensembling, multi-fidelity approximations, and meta-learning. Frameworks that implement some of these improvements usually do not adapt them to the time series data format. With auto-sktime, we aim to overcome these limitations.

2.3 Bayesian Optimization

Bayesian optimization (BO) [6] is often used to solve the AutoML optimization problem described in Equation (1). It aims to find the global minimum of an unknown function $f : \Lambda \to \mathbb{R}$ using an initial experiment design $S_0 = \{(\boldsymbol{\lambda}_i, l_i)\}_{i=1}^{M}$

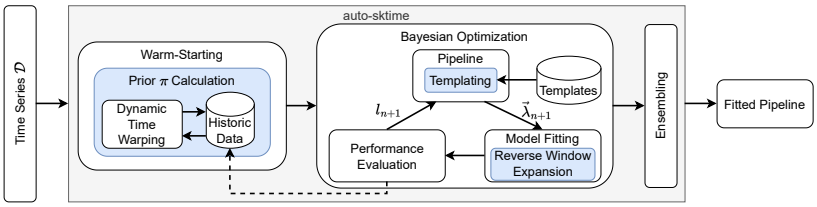


Fig. 1: General architecture of AUTO-SKTIME. Highlighted in blue are our three proposed improvements for automatic time series forecasting.

followed by a sequential selection of new candidates $\boldsymbol{\lambda}_{n+1}$ and a corresponding function evaluation $l_{n+1} = f(\boldsymbol{\lambda}_{n+1})$ to generate $S_{n+1} = S_n \cup \{(\boldsymbol{\lambda}_{n+1}, l_{n+1})\}$. After each new observation, a probabilistic surrogate model $p(f, S_{1:n})$ of f is constructed. Prominent examples of p are Gaussian processes [35], random forests [19], or Bayesian neural networks [26]. p is used in combination with an acquisition function $\alpha(\boldsymbol{\lambda}, S_{1:n})$ to select the most promising $\boldsymbol{\lambda}_{n+1}$ by trading off exploration and exploitation. By maximizing the acquisition function

$$\boldsymbol{\lambda}^{\star}_{n+1} \in \underset{\boldsymbol{\lambda} \in \Lambda}{\arg\max}\ \alpha\left(\boldsymbol{\lambda}; S_{1:n}, p\right) \tag{2}$$

in each iteration, a candidate $\boldsymbol{\lambda}^{\star}_{n+1}$ is selected for evaluation. Prominent examples of acquisition functions are knowledge gradients, entropy search, expected improvements, and upper confidence bound.

3 AUTO-SKTIME Methodology

An overview of the proposed system architecture for AUTO-SKTIME is displayed in Figure 1. This architecture is similar to existing SOTA AutoML frameworks [15,17]: Given the input data $\mathcal{D}$, meta-learning is used to warm-start the BO, more specifically SMAC [28]. The optimization loop itself generates pipeline candidates $\boldsymbol{\lambda}_{n+1}$ from the set of predefined templates with corresponding hyperparameters. Candidates are fitted using multi-fidelity approximations to calculate their performance l_{n+1}. This loop is repeated until a predefined budget is exhausted. Finally, an ensemble is created using ensemble selection [9]. To adapt this methodology to time series forecasting, we need to introduce three new ideas, highlighted in blue in Figure 1, in more detail. Yet, first, we present a formal problem formulation for time series forecasting.

3.1 Problem Formulation

Let a dataset $\mathcal{D} = \{D_i\}_{i=1}^{N}$ contain $N \in \mathbb{N}$ time series. Each time series D_i is defined by $D_i = \{\boldsymbol{y}_{i,1:T_i}, \boldsymbol{x}^{(p)}_{i,1:T_i}, \boldsymbol{x}^{(f)}_{i,T_i+1:T_i+H}\}$ with T_i being the number of observations in D_i and H the forecasting horizon. Let $\boldsymbol{y}_{i,1:T_i}$, with $\boldsymbol{y} \in \mathbb{R}^d$, denote the set of observed targets. In addition, let $\boldsymbol{x}^{(p)}_{i,1:T_i}$ and $\boldsymbol{x}^{(f)}_{i,T_i+1:T_i+H}$, with

$\boldsymbol{x} \in \mathcal{X}^e$, denote the set of observed features and forecasted features, respectively. A forecasting model $f_{\mathcal{D}}$ aims to predict future target values

$$\hat{\boldsymbol{y}}_{i,T_i+1:T_i+H} = f_{\mathcal{D}}\Big(\boldsymbol{y}_{i,1:T_i}, \boldsymbol{x}_{i,1:T_i+H}; \boldsymbol{\lambda}\Big) \tag{3}$$

with $\boldsymbol{\lambda}$ being the hyperparameters describing the model. This definition covers univariate and multi-multivariate time series with or without exogenous data and even panel data containing multiple time series depending on the selected dimensions. The quality of the forecasts $l \in \mathbb{R}$ is measured by the distance between the predicted and actual future targets according to a loss function $\mathcal{L}$.

3.2 Templates for Time Series

Historically, three general approaches for time series forecasting have been proposed: statistical, ML, and DNN models. In general, it is not possible to simply focus on just one of these approaches as no single best approach exists [30], but the best approach depends on the actual dataset. For example, both ML and DNN methods are not applicable for short univariate time series, while statistical models are not able to generalize over panel data. In addition, as we aim to provide end-to-end automation for time series forecasting, just selecting the forecaster itself is not sufficient. Instead, multiple preprocessing steps for data cleaning and feature engineering are necessary. Yet, the actual pipeline steps depend on the selected forecasting method as different preprocessing is necessary for the three approaches. AutoML tools often use a single fixed pipeline structure, i.e., [15,17], which is not able to capture the wide variety of potential input data, i.e., univariate/multivariate and endogenous/exogenous/panel data. Alternatively, AutoML tools build arbitrary pipelines with genetic programming, i.e., [38], which is not able to capture the implicit dependencies between the preprocessing steps and the used forecasting method.

We propose a templating approach to be used instead. Following the CASH notation introduced by Thornton et al. [44] for a single pipeline template i, the search space Λ_i aggregates a set of algorithms $\mathcal{A}_i = \{A_i^1, \ldots, A_i^n\}$ with according hyperparameters $\Lambda_i^1, \ldots, \Lambda_i^n$ into a single search space as $\Lambda_i = \Lambda_i^1 \cup \cdots \cup \Lambda_i^n \cup \{\lambda_{i,r}\}$, with $\lambda_{i,r}$ being a hyperparameter selecting the algorithms in $\mathcal{A}_i$. Instead of using a single pipeline template, we propose a joint search space $\Lambda = \Lambda_1 \cup \cdots \cup \Lambda_k \cup \{\lambda_r\}$, with λ_r selecting a pipeline, for k different templates.

Although our approach would allow for an arbitrary number of templates, we opted to implement three different pipeline templates, as there are three different classes of forecasting models with different requirements regarding data preprocessing. Each template is inspired by best-practice pipelines in the literature, e.g., [32]. An overview of the templates is depicted in Figure 2. While some steps are similar for all templates, e.g., encoding of categorical features, other steps only occur in a single template, like a reduction of time series data to tabular data for using ML forecasters. These basic pipelines can be further expanded with more specialized steps (e.g., [7]) in future work. No hard mapping of input data to a pipeline template exists. Meta-learning (see Section 3.4) can steer the optimization to a favorable combination of input data and template.

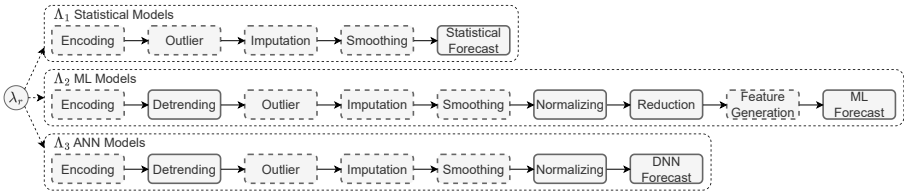


Fig. 2: Overview of the available templates. Steps marked by a solid border are mandatory, steps marked by a dotted border are optional. For each step, multiple algorithms with corresponding hyperparameters are available.

3.3 Multi-Fidelity Approximation for Time Series

Creating a forecast using Equation (3) can become expensive for sufficiently large input data $\mathcal{D}$ as training a model with the according hyperparameters is necessary. Especially if many forecasts have to be created, as is the case with BO, this procedure becomes quite expensive. Often, it is possible to define easier-to-evaluate proxies $\tilde{f}_{\mathcal{D}}(\cdot, b)$ of $f_{\mathcal{D}}(\cdot)$ that are parametrized by a budget $b \in [b_{\min}, b_{\max}]$ with $0 < b_{\min} < b_{\max} \leq 1$. These multi-fidelity approximations $\tilde{f}_{\mathcal{D}}$ are used in AutoML to speed up the evaluation of a single configuration $\boldsymbol{\lambda}$ by discarding unpromising models early, e.g., [25]. The lowest budget $b_{\min}$ is assigned to all initial models, and higher budgets are successively assigned to well-performing models until $b_{\max}$ is reached and $\tilde{f}(\cdot, b_{\max}) = f(\cdot)$.

To actually calculate $\tilde{f}_{\mathcal{D}}$, a mapping of b to a cheaper function evaluation has to be defined. For tasks on tabular data often the number of training samples, e.g., [23], or number of iterations (i.e., epochs for neural networks or number of trees in random forests) are used, e.g., [3]. For time series forecasting, these interpretations are not possible: many statistical models do not support iterative fitting. Selecting only some of the time series for training is only possible for panel data but not for univariate or multivariate forecasting. Similarly, reducing the length of a time series by selecting observations at random or selecting just every n-th sample is not always reasonable, as both methods could distort the seasonality of the data. Instead, we propose a reverse expanding window interpretation for multi-fidelity budgets.

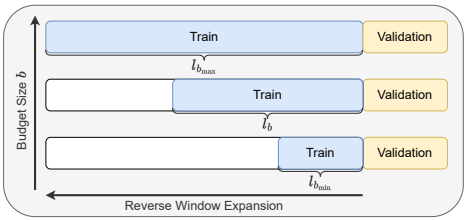


Fig. 3: Interpretation of multi-fidelity budgets as reverse expanding windows.

The complete procedure of reverse expanding windows is visualized in Figure 3. Given the current budget b, a desired window length $l_b = \min(b \cdot T_i, l_{\min})$ is calculated. While multi-fidelity approximations can speed up the evaluation of forecasting models on long time series, the reduced computation time for short time series is only marginal. Therefore, they are deactivated if the time series is shorter than a user-defined constant $l_{\min}$. Forecasts are created using only the latest l_b observations with $\tilde{f}_{\mathcal{D}}$ being defined as

$$\boldsymbol{y}_{i,T_i+1:T_i+H} = \tilde{f}_{\mathcal{D}}\Big(\boldsymbol{y}_{i,T_i-l_b1:T_i}, \boldsymbol{x}_{i,T_i-l_b:T_i+H}; \boldsymbol{\lambda}\Big) \ .$$

With increasing budgets, the window is expanding backward in time until, finally, the complete time series is included. The exact procedure to select budgets is handled using successive halving [25].

3.4 Warm-Starting the Optimization

In the context of AutoML, warm-starting usually refers to the practice of using meta-learning to initialize the optimization process with prior knowledge about well-performing regions of the search space. This may lead to an accelerated optimization convergence and potentially better-performing models. Meta-learning, in the context of BO, is commonly implemented by using specific configurations during initialization instead of a random initial design S_0. Often, those configurations are a portfolio of well-performing configurations on a variety of datasets or similar datasets identified via meta-learning [45].

Instead of using meta-learning only during the initialization and relying on the optimizer to derive well-performing regions during the optimization, Hvarfner et al. [20] proposed to include a user-defined prior π in the acquisition function, displayed in Equation (2), to make prior information available

$$\boldsymbol{\lambda}_n^\star \in \arg\max_{\boldsymbol{\lambda} \in \Lambda} \ \alpha\left(\boldsymbol{\lambda}; S_{1:n}, p\right) \pi\left(\boldsymbol{\lambda}\right)^{\beta/n} \tag{4}$$

with β being a decay factor to fade out the prior in longer optimizations. Hvarfner et al. [20] were able to prove that the introduction of $\pi(\boldsymbol{\lambda})^{\beta/n}$ does not negatively impact the worst-case convergence rate of BO while showing improved performance empirically.

Equation (4) relies on user-provided priors π for each hyperparameter, which contradicts the end-to-end automation we pursue. Furthermore, providing reasonable priors, for instance, for all 98 hyperparameters included in AUTO-SKTIME, is virtually impossible for users. We propose to automatically extract priors from historic experiments to achieve a fully automated warm-starting instead of manually crafting priors. The proposed approach is outlined in Algorithm 1.

For each element of a set of historical time series, let a set of the n_c best historic hyperparameters be given as $\mathcal{C}_{\text{meta}} = \left\{\left(\{\boldsymbol{\lambda}_j\}_{j=1}^{n_c}, \mathcal{D}_i\right)\right\}_{i=1}^{k}$. Furthermore, let a new dataset $\mathcal{D}_{\text{new}}$ be given that is going to be optimized. We aim to identify time series $\mathcal{D}_i \in \mathcal{C}_{meta}$ with a low distance $d : \mathcal{D} \times \mathcal{D} \to \mathbb{R}$ to $\mathcal{D}_{\text{new}}$(Line 1 and 2).

Algorithm 1 Automatic prior calculation for warm-starting the forecasting.

Require: Results of optimization runs on other datasets $\mathcal{C}_{\text{meta}}$, new time series $\mathcal{D}_{\text{new}}$
1: **for** $\mathcal{D}_i \in \mathcal{C}_{\text{meta}}$ **do**
2: Calculate Distance $d_i = d(\mathcal{D}_i, \mathcal{D}_{\text{new}})$; Eq. (5)
3: **end for**
4: $\mathcal{C}_{\text{sim}} \leftarrow$ Select n_d closest datasets
5: $\Lambda' \subset \Lambda \leftarrow$ Concatenate all samples $\{\boldsymbol{\lambda}\}$ from $\mathcal{C}_{\text{sim}}$
6: $(\Lambda', \boldsymbol{w}) \leftarrow$ Weight $\boldsymbol{\lambda} \in \Lambda'$ with d_i ; see Eq. (6)
7: $\pi \leftarrow$ Fit KDE using $\{(\Lambda', \boldsymbol{w})\}$

Instead of using a distance based on hard-to-select meta-features, as it is usually done for tabular data, we propose to use native distance metrics for time series. In the context of this work, a distance

$$d(\mathcal{E}, \mathcal{F}) = \frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} \text{DTW}(E_{i,T_i-h:T_i}, F_{j,T_i-h:T_i}) \tag{5}$$

based on dynamic time warping (DTW) [46] is used with M and N being the number of time series in $\mathcal{E}$ and $\mathcal{F}$, respectively, and h a user-provided maximum sequence length. DTW is used to compute the optimal alignment between two time series by finding the warping path that minimizes the cumulative distance between their data points. Similar distance measures would also be possible given that time series with different lengths are supported. To limit the introduced computational overhead for long time series, only the latest h observations of each time series are considered. Let $\mathcal{C}_{\text{sim}} = \{(\{\boldsymbol{\lambda}\}_i, \mathcal{D}_i)\}_{i=1}^{n_d} \subset \mathcal{C}_{\text{meta}}$, with n_d being a user-provided hyperparameter, denote the set of time series most similar to $\mathcal{D}_{\text{new}}$ (Line 4) and $\boldsymbol{d}_{n_d}$ the set of according distances. Next, all configurations selected in $\mathcal{C}_{\text{sim}}$ are combined $\Lambda' = \bigcup_{\{\boldsymbol{\lambda}_j\}, \cdot \in \mathcal{C}_{\text{sim}}} \{\boldsymbol{\lambda_j}\}$ (Line 5). Furthermore, the normalized distances

$$\boldsymbol{w} = 1 - \frac{\boldsymbol{d}_{n_d} - \min(\boldsymbol{d}_{n_d})}{\max(\boldsymbol{d}_{n_d}) - \min(\boldsymbol{d}_{n_d})} \tag{6}$$

are used to weight Λ' to give higher importance to configurations tested on data very similar to $\mathcal{D}_{\text{new}}$. Finally, based on Λ' and $\boldsymbol{w}$ we construct the prior π using kernel density estimation (KDE) [11] (Line 7), modeling promising areas of the configuration space. As [33] were able to show that there are only a few interactions between the different hyperparameters in $\boldsymbol{\Lambda}$, we also model their priors independently using univariate KDEs. The performance of configurations in Λ' is not further considered, but only the distribution of Λ' is relevant.

4 Experiments

To show the viability of AUTO-SKTIME, we perform a thorough evaluation on 64 real-world datasets. AUTO-SKTIME is compared with four SOTA forecasting frameworks, namely PMDARIMA [43], ETS [36], DEEPAR [39], and TFT

[27], as a baseline. Additionally, we also test the aforementioned frameworks already employing AutoML techniques which have published source code, namely, Auto-PyTorch (APT-TS), AutoTS, HyperTS, PyAF, and AutoGluon-TS. In contrast to benchmarks used in the related work, we consider a mix of univariate, multivariate, and panel data. Furthermore, we focus on rather short time series as they are more prevalent in real-world applications [47]. Each benchmark dataset comes with a predefined forecasting horizon that is used as a holdout test set. In most cases, this forecasting horizon corresponds to one natural period, e.g., 24 observations for hourly data. The performance of each framework is measured using mean absolute scaled error (MASE). All evaluations are repeated five times with different initial seeds to account for non-determinism. Each evaluation was limited to a computational budget of 5 minutes. Evaluations still running after an additional grace period of 60 seconds were pruned with the worst result produced by any of the competitors on the same dataset with an additional small failure penalty. While these optimization durations are rather short for typical AutoML applications, preliminary results showed fitting time series forecasting models on the rather short datasets selected for the benchmark often required only a few seconds, and the used DNN models converged in over 80% of all evaluations before hitting the time limit. The priors for warm-starting are generated using a leave-one-out procedure by only considering the remaining 63 datasets for each evaluated dataset to prevent leaking knowledge. All frameworks are used with their default parameters to emphasize the idea of end-to-end optimization. The source code, scripts for the experiments, and detailed results are available on Github to ensure the reproducibility of all presented results.[5]

4.1 Datasets

We conducted experiments using 64 real-world datasets sourced from different domains—including, for example, social media, finance, and commerce—encompassing both univariate, multivariate, and panel time series that are publicly available [1,29,37]. Due to the lack of common time series forecasting benchmark datasets, we reused datasets used in related work. None of the selected datasets were used during the development of auto-sktime. The number of data samples in the univariate time series ranges from 144 to 145 366, while the multivariate time series, with up to 111 dimensions, range from 48 to 7588 samples. Panel datasets contain 32 to 1 428 time series with 20 to 942 samples. The list of used datasets, with results, is available in the supplementary material.

4.2 Experiment Results

Table 1 shows the final test performance of all evaluations. For each framework, the mean MASE score, ranking, and fitting time in seconds with the according standard deviations are given. Bold face represents the best mean value.

[5] See https://github.com/Ennosigaeon/auto-sktime.

Table 1: Performance overview on all time series while enforcing timeouts.

Framework	MASE	Ranking	Time
APT-TS	3.12 ± 5.41	4.51 ± 2.49	242.2 ± 134.9
AUTO-SKTIME	$\mathbf{1.59 \pm 1.61}$	$\mathbf{3.22 \pm 1.94}$	326.1 ± 43.4
AUTOGLUON	6.00 ± 18.17	6.74 ± 2.34	144.9 ± 160.1
AUTOTS	$\underline{4.26 \pm 10.94}$	5.38 ± 2.35	321.1 ± 52.7
DEEPAR	11.54 ± 51.51	7.23 ± 1.99	61.3 ± 106.0
ETS	2.57 ± 2.78	5.82 ± 2.34	$\mathbf{2.0 \pm 7.6}$
HYPERTS	2.88 ± 5.03	4.57 ± 2.60	300.0 ± 89.4
PMDARIMA	2.63 ± 3.21	5.42 ± 2.62	47.1 ± 98.4
PYAF	3.23 ± 5.33	5.38 ± 2.56	22.6 ± 56.5
TFT	9.80 ± 50.62	6.73 ± 1.95	76.9 ± 118.3

Results not significantly worse, according to a t-test with $\alpha = .05$ and Bonferroni correction, are underlined. AUTO-SKTIME significantly outperforms all other frameworks both regarding to the absolute performance (MASE) and rank. Surprisingly, ETS and PMDARIMA reached a better average MASE than some of the AutoML tools. This can be explained as the AutoML tools produced bad results on a minor subset of the dataset while ETS and PMDARIMA significantly outperform them on very few data sets (see Table 1 in the online appendix). When considering the median performance, the AutoML tools outperform the baseline methods. This is also reflected in the average ranking in Table 1. We consider this a limitation of our small benchmark rather than a general issue of AutoML tools. TFT and DEEPAR perform badly due to failures on many time series. If a prediction was created, they often have a competitive performance.

A major issue for many AutoML tools is the missing support for end-to-end automation of time series forecasting. Multiple frameworks are missing methods for data cleaning, e.g., imputation of missing values. As 27 time series contained missing values, the average performance of those frameworks appears to be significantly worse. Table 2 contains the results of all tested frameworks on only time series not necessarily requiring preprocessing and data cleaning. Most methods achieve a better performance. As a consequence, AUTO-SKTIME does not significantly outperform many of the baseline methods anymore, even though it still has the best average performance. Interestingly, methods like DEEPAR and TFT even profited from failing runs, as the worst performance of the competitors, on average, is still better than their own predictions.

Another major issue was the imposed time limit of 300 seconds. Even though a grace period of additional 60 seconds was introduced, many tools did not adhere to the maximum budget with single optimization runs requiring over 5 000 seconds. In total, 10.36% of all evaluations violated the timeout with AUTOTS and AUTOGLUON exceeding it on nearly 33.3% of all runs worsening the reported results significantly. Table 2 summarizes the average performance of all tested frameworks when the timeout is not enforced. AUTOTS and AUTOGLUON gained a significant performance boost as they regularly exceeded the configured

Table 2: Performance overview for two different scenarios.

Framework	Without Missing Values			Without Enforcing Timeouts		
	MASE	Ranking	Time	MASE	Ranking	Time
APT-TS	2.4 ± 6.1	4.7 ± 2.8	276.5 ± 109.2	3.4 ± 6.3	5.1 ± 2.5	246.3 ± 141.6
AUTO-SKTIME	$\mathbf{1.0 \pm 0.9}$	$\mathbf{3.5 \pm 1.8}$	327.7 ± 12.4	$\mathbf{1.4 \pm 1.6}$	$\mathbf{2.9 \pm 1.8}$	334.6 ± 54.3
AUTOGLUON	6.9 ± 23.4	$\underline{6.7 \pm 3.0}$	224.9 ± 144.9	2.8 ± 4.7	5.2 ± 2.3	585.3 ± 1480.3
AUTOTS	$\underline{4.6 \pm 14.0}$	6.3 ± 2.0	318.7 ± 29.4	$\underline{3.3 \pm 10.5}$	4.8 ± 2.5	512.5 ± 421.4
DEEPAR	16.3 ± 66.7	7.5 ± 2.5	103.0 ± 121.5	11.9 ± 51.5	7.7 ± 1.8	61.3 ± 106.0
ETS	1.4 ± 1.1	5.3 ± 2.8	$\mathbf{0.3 \pm 0.7}$	2.7 ± 2.9	6.3 ± 2.2	$\mathbf{2.0 \pm 7.6}$
HYPERTS	2.4 ± 5.9	5.3 ± 2.5	311.0 ± 54.3	2.5 ± 5.8	4.0 ± 2.8	342.0 ± 155.5
PMDARIMA	$\underline{1.2 \pm 1.1}$	4.5 ± 3.1	51.3 ± 86.0	2.9 ± 4.6	5.8 ± 2.6	88.6 ± 291.3
PYAF	2.3 ± 5.9	$\underline{4.4 \pm 2.9}$	36.4 ± 70.1	3.6 ± 6.3	5.9 ± 2.4	23.4 ± 61.6
TFT	$\underline{13.3 \pm 65.7}$	6.7 ± 2.5	129.1 ± 130.1	10.2 ± 50.7	7.2 ± 1.8	76.9 ± 118.2

Table 3: Performance of AUTO-SKTIME variants.

Framework	MASE	Ranking	Time
APT-TS	2.67 ± 3.45	3.44 ± 1.61	$\mathbf{242.2 \pm 134.9}$
AUTO-SKTIME	$\mathbf{1.54 \pm 1.60}$	$\mathbf{2.23 \pm 1.20}$	326.1 ± 43.4
TEMPLATES	3.21 ± 9.02	3.41 ± 1.15	324.9 ± 43.4
TEMPLATES & MULTI-FIDELITY	$\underline{1.99 \pm 2.22}$	3.19 ± 1.26	326.7 ± 44.3
TEMPLATES & WARM STARTING	$\underline{2.16 \pm 3.12}$	2.73 ± 1.24	327.5 ± 44.0

timeout on larger time series. Similarly, HYPERTS's and AUTO-SKTIME's performance also slightly improved. The remaining frameworks basically never utilized the configured optimization budget fully and consequently also did not improve their performance. Even so ETS only achieves a mediocre ranking, it is still able to beat some of the contestants in a fraction of the computational budget, proving the value of considering statistical forecasting methods.

In both examined alternative scenarios, AUTO-SKTIME still obtains the best average performance even though differences are often not significant anymore. Yet, in all scenarios, AUTO-SKTIME still achieves the overall best ranking.

4.3 Ablation Study

In Section 3.2–3.4, we proposed three potential AutoML improvements for time series forecasting. To study their impact, we perform an ablation study by introducing three variants of AUTO-SKTIME: TEMPLATES using only the templating approach, TEMPLATES & MULTI-FIDELITY using templating and multi-fidelity approximations but not warm-starting, and TEMPLATES & WARM-STARTING using templating and warm-starting but not multi-fidelity. These three variants are evaluated on the same datasets as the other forecasting frameworks and compared with the original AUTO-SKTIME version, including all proposed improve-

ments. We also provide the performance of Auto-PyTorch, the second-best method, as a reference for assessing the impact of the different versions.

As shown in Table 3, the templating base version performs worse than Auto-PyTorch. Adding either multi-fidelity approximations or warm-starting improves the performance significantly, with both versions outperforming Auto-PyTorch. By combining all three improvements, the performance of auto-sktime is again slightly improved, making all three proposed improvements useful for time series forecasting.

5 Conclusion and Limitation

By adapting existing AutoML techniques to the domain of time series forecasting, instead of just applying techniques developed for tabular data, we were able to outperform existing AutoML frameworks for time series forecasting. auto-sktime proved to be viable for forecasting on a wide variety of time series data, i.e., univariate, multivariate, and panel data with or without exogenous data. The ablation study showed that a combination of at least two proposed adaptations of AutoML techniques for time series forecasting are necessary to outperform the current SOTA while using templating, multi-fidelity approximations and warm-starting together yields an even better performance.

During the experiments, we were able to show that a robust handling of common data errors is important for good performance. Yet, while we strove for end-to-end automation of forecasting, auto-sktime is still not applicable to time series with arbitrary data defects. For example, auto-sktime is currently not able to handle measurements collected with an irregular frequency. We plan to overcome this and similar common data defects in the near future.

Besides forecasting, time series are also relevant for other learning tasks like, for example, time series classification or regression. While the presented framework was already used successfully for time series regression in the context of remaining useful life predictions of mechanical systems [51], further work to validate the usefulness of auto-sktime on a variety of domains is necessary.

Acknowledgement Marc Zöller was funded by the Federal Ministry for Economic Affairs and Climate Action in the project AutoQML, Marius Lindauer by the Federal Ministry for Economic Affairs and Energy of Germany in the project CoyPu (Grant 01MK21007L), and Marco Huber by the Baden-Wuerttemberg Ministry for Economic Affairs, Labour and Tourism in the project KI-Fortschrittszentrum "Lernende Systeme und Kognitive Robotik" (Grant 036-140100).

References

1. Antoine Carme: Time Series Datasets (2023)
2. Bauer, M., van Dinther, C., Kiefer, D., van Dinther: Machine Learning in SME: An Empirical Study on Enablers and Success Factors Success Factors. In: Americas' Conference on Information Systems. pp. 1–10 (2020)

3. Bischl, B., Binder, M., Lang, M., Pielok, T., Richter, J., Coors, S., Thomas, J., Ullmann, T., Becker, M., Deng, D., Lindauer, M.: Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. WIRE **13**(2) (2023)
4. Bontempi, G., Ben Taieb, S., Le Borgne, Y.A.: Machine Learning Strategies for Time Series Forecasting. LNBIP **138**, 62–77 (2013)
5. Box, G.E., Jenkins, G.M., Reinsel, G.C., Ljung, G.M.: Time Series Analysis: Forecasting and Control. Wiley, 5 edn. (2015)
6. Brochu, E., Cora, V.M., de Freitas, N.: A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning. arXiv:1012.2599 pp. 1–49 (2010)
7. Candelieri, A., Archetti, F.: Global optimization in machine learning: the design of a predictive analytics application. Soft Computing **23**(9), 2969–2977 (2019)
8. Carme, A.: PyAF: Python Automatic Forecasting (2016)
9. Caruana, R., Niculescu-Mizil, A., Crew, G., Ksikes, A.: Ensemble Selection from Libraries of Models. International Conference on Machine Learning p. 18 (2004)
10. Catlin, C.: AutoTS (2020)
11. Chen, Y.C.: A Tutorial on Kernel Density Estimation and Recent Advances. Biostatistics & Epidemiology **1**(1), 161–187 (2017)
12. Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In: EMNLP. pp. 1724–1734 (2014)
13. Dahl, S.M.J.: TSPO: An AutoML Approach to Time Series Forecasting (2020)
14. Deng, D., Karl, F., Hutter, F., Bischl, B., Lindauer, M.: Efficient Automated Deep Learning for Time Series Forecasting. In: ECML (2022)
15. Erickson, N., Mueller, J., Shirkov, A., Zhang, H., Larroy, P., Li, M., Smola, A.: Robust and Accurate AutoML for Structured Data. In: ICML. pp. 1–28 (2020)
16. Feurer, M., Hutter, F.: Hyperparameter Optimization. In: Automatic Machine Learning: Methods, Systems, Challenges, pp. 3–38. Springer (2018)
17. Feurer, M., Klein, A., Eggensperger, K., Springenber, J.T., Blum, M., Hutter, F.: Efficient and Robust Automated Machine Learning. In: NeurIPS (2015)
18. Hochreiter, S., Schmidhuber, J.: Long Short-Term Memory. Neural Computation **9**(8), 1735–1780 (1997)
19. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential Model-Based Optimization for General Algorithm Configuration. In: LION. pp. 507–523 (2011)
20. Hvarfner, C., Stoll, D., Souza, A., Lindauer, M., Hutter, F., Nardi, L.: piBO: Augmenting Acquisition Functions with User Beliefs for BO. In: ICLR. pp. 1–15 (2022)
21. Hyndman, R.J., Athanalos, G.: Forecasting: Principles and Practice. OTexts (2021)
22. Januschowski, T., Wang, Y., Torkkola, K., Erkkilä, T., Hasson, H., Gasthaus, J.: Forecasting with trees. IJF **38**(4), 1473–1481 (2022)
23. Klein, A., Falkner, S., Bartels, S., Hennig, P., Hutter, F.: Fast BO of ML Hyperparameters on Large Datasets. In: AISTATS. pp. 528–536 (2016)
24. Kurian, J.J., Dix, M., Amihai, I., Ceusters, G., Prabhune, A.: BOAT: A Bayesian Optimization AutoML Time-series Framework for Industrial Applications. In: IEEE BigDataService. pp. 17–24 (2021)
25. Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A.: Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. JMLR **18**(1), 6765–6816 (2017)
26. Li, Y.L., Rudner, T.G.J., Wilson, A.G.: A Study of Bayesian Neural Network Surrogates for Bayesian Optimization. In: AABI. pp. 1–41 (2023)
27. Lim, B., Arık, S., Loeff, N., Pfister, T.: Temporal Fusion Transformers for interpretable multi-horizon time series forecasting. IJF **37**(4), 1748–1764 (2021)

28. Lindauer, M., Eggensperger, K., Feurer, M., Biedenkapp, A., Deng, D., Benjamins, C., Ruhkopf, T., Sass, R., Hutter, F.: SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization. JMLR **23**, 1–9 (2022)
29. Maciag, P.S., Kryszkiewicz, M., Bembenik, R., Lobo, J.L., Del Ser, J.: Unsupervised Anomaly Detection in Stream Data with Online Evolving Spiking Neural Networks. Neural Networks **139**, 118–139 (2021)
30. Makridakis, S., Spiliotis, E., Assimakopoulos, V.: The M4 Competition: 100,000 time series and 61 forecasting methods. IJF **36**(1), 54–74 (2020)
31. Masini, R.P., Medeiros, M.C., Mendes, E.F.: Machine learning advances for time series forecasting. Journal of Economic Surveys **37**(1), 76–111 (2023)
32. Meisenbacher, S., Turowski, M., Phipps, K., Rätz, M., Müller, D., Hagenmeyer, V., Mikut, R.: Review of automated time series forecasting pipelines. WIRE (2022)
33. Pushak, Y., Hoos, H.: AutoML Loss Landscapes. ACM Transactions on Evolutionary Learning and Optimization **2**(3), 1–30 (2022)
34. Rady, E.H.A., Fawzy, H., Fattah, A.M.A.: Time series forecasting using tree based methods. Journal of Statistics Applications and Probability **10**(1), 229–244 (2021)
35. Rasmussen, C., Williams, C.: Gaussian Processes for ML. MIT Press (2006)
36. Rob J. Hyndman, Anne B. Koehler, J. Keith Ord, Ralph D. Snyder: Forecasting with Exponential Smoothing: The State Space Approach. Springer (2008)
37. Rob Mulla: Time Series Datasets (2018)
38. de Sá, A., Pinto, W., Oliveira, L., Pappa, G.: RECIPE: A Grammar-Based Framework for Automatically Evolving Classification Pipelines. In: EuroGP (2017)
39. Salinas, D., Flunkert, V., Gasthaus, J., Januschowski, T.: DeepAR: Probabilistic forecasting with autoregressive recurrent networks. IJF **36**(3), 1181–1191 (2020)
40. Shah, S., Patel, D., Vu, L., Dang, X., Chen, B., Kirchner, P., Samwitz, H., Wood, D., Bramble, G., Gifford, W., Ganapavarapu, G., Vaculin, R., Zerfos, P.: AutoAI-TS: AutoAI for Time Series Forecasting. In: SIGMOD. pp. 2584–2596 (2021)
41. Shchur, O., Turkmen, C., Erickson, N., Shen, H., Shirkov, A., Hu, T., Wang, Y.: AutoML for Probabilistic Time Series Forecasting. In: AutoML Conf (2023)
42. da Silva, F., Vieira, A., Bernardino, H., Alencar, V., Pessamilio, L., Barbosa, H.: Automated Machine Learning for Time Series Prediction. In: IEEE CEC (2022)
43. Smith, M.J., Wedge, R., Veeramachaneni, K.: FeatureHub: Towards collaborative data science. In: IEEE DSAA. pp. 590–600 (2017)
44. Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Auto-WEKA: Combined Selection and HPO of Classification Algorithms. In: ACM KDD. pp. 847–855 (2013)
45. Vanschoren, J.: Meta-Learning. In: Automatic Machine Learning: Methods, Systems, Challenges, pp. 35–61. Springer (2019)
46. Vintsyuk, T.: Speech discrimination by dynamic programming. Cybernetics **4**(1), 52–57 (1968)
47. Vogelsgesang, A., Haubenschild, M., Finis, J., Kemper, A., Leis, V., Muehlbauer, T., Neumann, T., Then, M.: Get real: How benchmarks fail to represent the real world. In: Workshop on Testing Database Systems (2018)
48. Yamak, P.T., Yujian, L., Gadosey, P.K.: A comparison between ARIMA, LSTM, and GRU for time series forecasting. In: ACAI. pp. 49–55 (2020)
49. Zhang, X., Wu, H., Yang, J.: HyperTS: A Full-Pipeline Automated Time Series Analysis Toolkit (2022)
50. Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H., Zhang, W.: Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting. In: AAAI. pp. 11106–11115 (2021)
51. Zöller, M.A., Mauthe, F., Zeiler, P., Lindauer, M., Huber, M.F.: Automated Machine Learning for Remaining Useful Life Predictions. In: IEEE SMC (2023)