Guido Tack (Ed.)

LNCS 15763

Integration of Constraint Programming, Artificial Intelligence, and Operations Research

22nd International Conference, CPAIOR 2025
Melbourne, VIC, Australia, November 10–13, 2025
Proceedings, Part II

Springer

Lecture Notes in Computer Science **15763**

The series Lecture Notes in Computer Science (LNCS), including its subseries Lecture Notes in Artificial Intelligence (LNAI) and Lecture Notes in Bioinformatics (LNBI), has established itself as a medium for the publication of new developments in computer science and information technology research, teaching, and education.

LNCS enjoys close cooperation with the computer science R & D community, the series counts many renowned academics among its volume editors and paper authors, and collaborates with prestigious societies. Its mission is to serve this international community by providing an invaluable service, mainly focused on the publication of conference and workshop proceedings and postproceedings. LNCS commenced publication in 1973.

Guido Tack
Editor

Integration of Constraint Programming, Artificial Intelligence, and Operations Research

22nd International Conference, CPAIOR 2025
Melbourne, VIC, Australia, November 10–13, 2025
Proceedings, Part II

Springer

Editor
Guido Tack
Monash University
Clayton, VIC, Australia

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-031-95975-2 ISBN 978-3-031-95976-9 (eBook)
https://doi.org/10.1007/978-3-031-95976-9

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

This book contains the proceedings of the 22nd International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR 2025). The conference was held in Melbourne, Australia, November 10–13, 2025. The conference organizers acknowledge that the conference was held on the unceded traditional lands of the Wurundjeri people of the Kulin Nation. We pay our respects to their Elders past, present and emerging. The conference was co-located with the 35th International Conference on Automated Planning and Scheduling (ICAPS) and the 22nd International Conference on Principles of Knowledge Representation and Reasoning (KR).

The conference received a total of 80 submissions of original unpublished research papers. After an initial screening, 68 papers were sent out to the Program Committee for single-blind peer review, with each paper receiving at least three reviews. The reviewing phase was followed by an author response period. The Program Committee then discussed each paper and made a recommendation for acceptance or rejection. At the end of this process, 32 papers were accepted for presentation at the conference and publication in these proceedings. In addition, a number of extended abstracts were accepted for presentation at the conference, but not included in these proceedings. The conference featured a masterclass and several joint invited talks that covered topics of interest at the intersection of Constraint Programming, Artificial Intelligence, Operations Research, Planning and Scheduling, and Knowledge Representation. It was preceded by a Summer School, which took place in San Remo, Victoria, Australia, November 3–7, 2025.

Of the accepted papers, the paper "Multitask Representation Learning for Mixed Integer Linear Programming" by Junyang Cai, Taoan Huang and Bistra Dilkina was selected for the Best Paper Award. The paper "A Dynamic Programming Approach for the Job Sequencing and Tool Switching Problem" by Emma Legrand, Vianney Coppé, Daniele Catanzaro and Pierre Schaus was selected for the Best Student Paper Award. The awards were selected by a sub-committee of seven members of the Program Committee.

I would like to thank our publicity chair, Jip Dekker, the sponsorship chair, Charles Gretton, and the masterclass chair, Buser Say, as well as the entire local organizing team for their contributions to making this conference a success. The conference would not have been possible to organise without the help of the local chairs of ICAPS, Nir Lipovetzky and Sebastian Sardina, whose hard work enabled us to run the two conferences as a single, joint event. Finally, I would like to thank our sponsors for their generous financial support.

May 2025 Guido Tack

Organization

Program and Conference Chair

Guido Tack	Monash University, Australia

Publicity Chair

Jip Dekker	Monash University, Australia

Masterclass Chair

Buser Say	Monash University, Australia

Sponsorship Chair

Charles Gretton	Australian National University, Australia

Program Committee

Chris Beck	University of Toronto, Canada
Nicolas Beldiceanu	IMT Atlantique, LS2N, France
Armin Biere	Universität Freiburg, Germany
Merve Bodur	University of Edinburgh, UK
Mats Carlsson	RISE Research Institutes of Sweden, Sweden
Margarida Carvalho	Université de Montréal, Canada
Margarita P. Castro	Pontificia Universidad Catolica de Chile, Chile
Andre Augusto Cire	University of Toronto, Canada
Simon de Givry	INRAE, France
Sophie Demassey	Mines Paris - PSL, France
Emir Demirović	Delft University of Technology, The Netherlands
María Andreína Francisco Rodríguez	Uppsala University, Sweden
Ambros Gleixner	HTW Berlin, Germany
Emmanuel Hebrard	LAAS-CNRS, France

John Hooker	Carnegie Mellon University, USA
Matti Järvisalo	University of Helsinki, Finland
Serdar Kadioglu	Brown University, USA
George Katsirelos	MIA Paris, INRAE, France
Zeynep Kiziltan	University of Bologna, Italy
Arnaud Lallouet	Huawei Technologies, France
Thi Thai Le	Zuse Institute Berlin, Germany
Pierre Le Bodic	Monash University, Australia
Jimmy H.M. Lee	The Chinese University of Hong Kong, China
Michele Lombardi	University of Bologna, Italy
Ciaran McCreesh	University of Glasgow, UK
Laurent Michel	University of Connecticut, USA
Nysret Musliu	Technische Universität Wien, Austria
Margaux Nattaf	Institut National Polytechnique de Grenoble, France
Barry O'Sullivan	University College Cork, Ireland
Sophie N. Parragh	Johannes Kepler University Linz, Austria
Justin Pearson	Uppsala University, Sweden
Laurent Perron	Google France, France
Gilles Pesant	École Polytechnique de Montréal, Canada
Claude-Guy Quimper	Université Laval, Canada
Jean-Charles Regin	Université Côte d'Azur, France
Louis-Martin Rousseau	École Polytechnique de Montréal, Canada
Domenico Salvagnin	University of Padua, Italy
Pierre Schaus	UCLouvain, Belgium
Thiago Serra	University of Iowa, USA
Paul Shaw	IBM, France
Mohamed Siala	INSA Toulouse & LAAS-CNRS, France
Helmut Simonis	University College Cork, Ireland
Christine Solnon	CITI, INSA Lyon/Inria, France
Kostas Stergiou	University of Western Macedonia, Greece
Peter J. Stuckey	Monash University, Australia
Michael Trick	Carnegie Mellon University, USA
Willem-Jan van Hoeve	Carnegie Mellon University, USA
Hélène Verhaeghe	UCLouvain, Belgium
Petr Vilím	OptalCP, Czech Republic
Mark Wallace	Monash University, Australia
Roland Yap	National University of Singapore, Singapore
Neil Yorke-Smith	Delft University of Technology, The Netherlands

Additional Reviewers

Valentin Antuori
Christian Artigues
Andrea Borghesi
Jip J. Dekker
Julien Ferry
Ernest Foussard
Maria Garcia de la Banda
Mohammed Ghannam
Luca Giuliani
Arthur Gontier
Richard Hua
Christoph Jabs
Edward Lam
Matthew J. McIlree
Xiao Peng
Angelo Quarta
Buser Say
Gaetano Signorelli
Mate Soos

Contents – Part II

Contents – Part I

Combining Constraint Programming and Metaheuristics for Aircraft Maintenance Routing with a Distribution Objective

Ida Gjergji[1(✉)], Lucas Kletzander[2], Hendrik Bierlee[3,4,5], Nysret Musliu[2], and Peter J. Stuckey[3,4]

[1] DBAI, TU Wien, Vienna, Austria
ida.gjergji@tuwien.ac.at
[2] Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling, DBAI, TU Wien, Vienna, Austria
{lucas.kletzander,nysret.musliu}@tuwien.ac.at
[3] Monash University, Melbourne, Australia
{hendrik.bierlee,peter.stuckey}@monash.edu
[4] OPTIMA ARC Industry Training and Transformation Centre, Melbourne, Australia
[5] KU Leuven, Leuven, Belgium
henk.bierlee@kuleuven.be

Abstract. In this paper we focus on a challenging version of the aircraft maintenance routing problem (AMRP) with a maintenance distribution objective (AMRP-D). For the AMRP-D, the flight legs with predefined start and end times are assigned to aircraft. In addition to the assigned flight legs, each aircraft has to satisfy certain regulations regarding the maintenance services that are mandatory in the scheduling period, while the maintenance should also be distributed evenly. We propose a two stage approach, where first we use a decomposition method that is solved using constraint programming, to cover the flight legs. To optimize the distribution objective, we propose two metaheuristic techniques based on Large Neighborhood Search (LNS) and Simulated Annealing (SA). The LNS method consists of different destroy operators and as repairer we use a constraint programming (CP) solver. The SA approach includes a novel neighborhood to deal with the distribution objective. Our experimental results show that the decomposition method is able to solve more instances than the exact approach, while SA provides better quality solutions for the optimization stage compared to LNS.

Keywords: Constraint programming · Scheduling · Metaheuristics

1 Introduction

Airlines operate in a challenging environment including strong regulations, variable demand, high competition, and high operational cost. Several optimization

G. Tack (Ed.): CPAIOR 2025, LNCS 15763, pp. 1–17, 2025.
https://doi.org/10.1007/978-3-031-95976-9_1

problems arise in this context, and high quality solutions are very important for successful operation [3]. Due to the overall complexity, several different problems are identified and usually solved separately. Typically these are the flight scheduling problem (FSP, generating a timetable for flights), the fleet assignment problem (FAP, assigning aircraft types to flights), the aircraft maintenance routing problem (AMRP, assigning flights and maintenance to individual aircraft), and finally the crew scheduling problem (CSP, assigning various types of crew). All these sub-problems are known to be NP-hard [16], and even small improvements in solutions can transfer to significant savings for airlines [15].

We focus on the AMRP, where flight legs and the type of aircraft to fly these are already fixed, but the specific aircraft still need to be assigned. Since aircraft need a range of different maintenance operations that depend on the sequence of flights, passed time and flight hours, and the current location, the assignment of flight legs, and maintenance operations is typically combined. We refer to reviews [15,16] for an overview of work in this area. Problem formulations date back many years [4]. Solution methods include heuristics [6], network-based MIP formulations [3,7], and branch-and-price techniques [13]. Heuristic approaches often outperform exact approaches on instances of realistic scale [1,2].

Capacity constraints were rarely considered, or only with upper limits [3]. Eltoukhy et al. [3] highlight in their overview regarding future research directions, that the AMRP rarely includes more considerations on how the maintenance workers are utilized. However, this can have very negative effects on the workforce, leading to both very high peak workloads, creating stress and fatigue, or requiring high overall manpower, while at other times excess capacity leads to increased cost. This paper investigates the extended problem that we recently introduced [9], with an additional objective to optimize the distribution of maintenance work, called the AMRP-D (AMRP with maintenance distribution). The additional objective greatly helps to provide a well distributed workload for the maintenance department, however, it also makes it even more challenging to provide good solutions for the problem, which motivates this research. To the best of our knowledge, no solution methods have been proposed in the literature yet for this extended problem. The main contributions of this paper are:

- A solver-independent model with different objective functions to optimize the maintenance distribution, including a novel over-approximation model.
- A decomposition approach based on the CP model to obtain feasible solutions for instances of realistic size.
- A Large Neighborhood Search that includes various novel destroy operators and a CP solver as a repair operator.
- A Simulated Annealing approach composed of three neighborhoods, including a novel neighborhood heuristic for maintenance distribution.
- Critical evaluation and comparison of the proposed methods on the diverse set of instances.

The experimental evaluation shows that while the CP based approach is crucial for obtaining feasible solutions, combining it with LNS and SA allows for significant improvements in results for most instances regarding the distribution.

2 Problem Description

The Aircraft Maintenance Routing Problem (AMRP) is specified by a set of flight legs, regulations for maintenance, and a set of aircraft including their recent flight and maintenance history. All times are given as integers.

2.1 Flight Legs

A set of n flight legs $T = \{t_1, \ldots, t_n\}$ is given, each leg t_i is associated with a start time s_i, an end time e_i, and a flight time f_i (with $f_i < e_i - s_i$, as the flight leg contains both the flight time and the preparation/turnaround times). As such, flight legs are not allowed to overlap for the same aircraft, but there is no minimum distance between legs on the same aircraft. The first m legs in T denote the last leg of each of the m aircraft from the previous scheduling period which are fixed to the particular aircraft. Note that in general, each flight leg is also associated with a start location and an end location, defining the routing aspect of the problem. However, in this paper we deal with a version using a single home base where outbound and following inbound flights are fused to one flight leg (with no maintenance at the destination). Therefore, all locations are equal and omitted.

2.2 Maintenance

Aircraft require different types of maintenance. Each maintenance type takes a different amount of time to complete. A maintenance on an aircraft cannot overlap with its other maintenance or flight legs. Some maintenance compete for a single hangar. Furthermore, some types are due periodically (regardless of flight time), while others are due based on the cumulative flight time. The following types of maintenance are used in this paper:

- Regular: An aircraft can fly for at most $regl = 47$ h after the end of the previous regular maintenance, then it needs regular maintenance taking $regd = 2.5$ h before taking off again.
- Weekly: An aircraft can fly for at most $weekl = 156$ h (6.5 days) after the end of the previous weekly maintenance, then it needs weekly maintenance taking $weekd = 7$ h before taking off again. Weekly maintenance includes regular maintenance.
- Major: There are four different types of major maintenance. Each of them is independent from the others. Each follows the same rule regarding time: After at most $majl = 950$ h of cumulative flight time since the last maintenance of the same type, the aircraft needs major maintenance taking $majd = 14$ h before taking off again. Each type of major maintenance includes regular and weekly maintenance. There are further differences regarding subtypes:
  - `MH1` and `MH2`: These require a single hangar, meaning that only one aircraft can perform any of these two types of maintenance at once.
  - `MR1` and `MR2`: These two types do not require the hangar.

2.3 Aircraft

A set of m aircraft $A = \{a_1, \ldots, a_m\}$ is given, each aircraft a_j is preassigned to the history flight leg t_j, and is associated with the time since the start of the planning horizon that the last regular maintenance ended r_j, the time since the start of the planning horizon that the last weekly maintenance ended w_j, and the cumulative flight time since each of the major maintenance types (including the last assigned leg) $majp_{kj}$ with $k \in \{1, 2, 3, 4\}$, where 1 and 2 correspond to `MH1` and `MH2`, while 3 and 4 correspond to `MR1` and `MR2`.

2.4 Solution

The legs for $j \in \{1, \ldots, m\}$ are fixed to the corresponding aircraft. A feasible solution assigns all remaining flight legs to aircraft, and the required types of maintenance to specific aircraft and time intervals, such that:

- No overlapping flight legs are assigned to any aircraft.
- No maintenance intervals are violated.
- At most one aircraft is assigned `MH1` or `MH2` at any point in time.

Traditionally, the AMRP features objectives like minimizing the total maintenance duration, or minimizing the earliness of each maintenance compared to its latest possible assignment. In contrast, the version with distribution objective (AMRP-D) includes a focus on even distribution of maintenance. This requirement stems from a real-world application, where the issue of uneven maintenance distribution leads to difficult assignments of maintenance personnel: Peaks need to be covered, requiring a large workforce at the same time, while in times with few maintenance tasks costly overstaffing is caused. Further, there might also be other resources like required machinery that get scarce during peaks.

For the optimization goal of AMRP-D, the total number of aircraft in any type of maintenance m_s is calculated for each minute s in the planning period, and $\sum_s m_s^2$ is minimized. This optimizes the distribution of maintenance tasks (since peaks are penalized more), and also the total amount of maintenance (leading to maximization of the available intervals between maintenance tasks).

Major maintenance of each type, however, typically occurs at most once per scheduling period (a scheduling period is up to a month, while the maximum flight time is well over a month of non-stop flying). Therefore, we limit each type of major maintenance to at most once per aircraft and type. In addition, it should be as late as possible within the allowed window. The earliness in minutes is added to the objective.

3 Constraint Model

The standalone model for the AMRP-D uses the following variables:

- T': the set of flight legs excluding the previously assigned legs $T' = T \backslash \{t_j \mid j \in A\}$.

$$as_{jj} = s_j, \quad j \in A \tag{1}$$

$$\texttt{alternative}(s_i, e_i - s_i, \overline{as}_i, [e_i - s_i \mid j \in A]), \quad i \in T' \tag{2}$$

$$\texttt{disj}([majs_{kj} \mid k \in 1..2, j \in A], [majd \mid k \in 1..2, j \in A]) \tag{3}$$

$$majp_{kj} + \sum_{i \in T',\ as_{ij} \neq \circ,\ as_{ij} < majs_{kj}} f_i \leq majl, \quad k \in 1..4, j \in A \tag{4}$$

$$\texttt{strictly\_increasing}([ms_{lj} \mid l \in 1..mm]), \quad j \in A \tag{5}$$

$$ms_{lj} = \circ \rightarrow ms_{l+1,j} = \circ, \quad j \in A, 1 \leq l < mm \tag{6}$$

$$ms_{1j} = w_j - weekd \wedge mt_{1j} = 5, \quad j \in A \tag{7}$$

$$ms_{2j} = r_j - regd \wedge mt_{2j} = 6, \quad j \in A, r_j \neq w_j \tag{8}$$

$$\texttt{element}(mt_{lj}, [majd, majd, majd, majd, weekd, regd], md_{lj}), \quad l \in 1..mm, j \in A \tag{9}$$

$$\texttt{disj}(\overline{as}_j +\!\!+ \overline{ms}_j, [e_i - s_i | i \in T] +\!\!+ \overline{md}_j), \quad j \in A \tag{10}$$

$$as_{ij} \neq \circ \rightarrow (\exists_{l \in 1..mm} ms_{lj} \neq \circ \wedge ms_{lj} < as_{ij} \wedge ms_{lj} + md_{lj} + regl \geq e_i), \quad i \in T', j \in A \tag{11}$$

$$as_{ij} \neq \circ \rightarrow (\exists_{l \in 1..mm} ms_{lj} \neq \circ \wedge mt_{lj} \neq 6 \wedge ms_{lj} < as_{ij} \wedge ms_{lj} + md_{lj} + weekl \geq e_i), \quad i \in T', j \in A \tag{12}$$

$$majs_{kj} \neq \circ \rightarrow \exists_{l \in 1..mm} ms_{lj} = majs_{kj} \wedge mt_{lj} = k, \quad k \in 1..4, j \in A \tag{13}$$

$$mt_{lj} \in 1..4 \rightarrow majs_{(mt_{lj})j} = ms_{lj}, \quad l \in 1..mm, j \in A \tag{14}$$

$$m_s = \sum_{l \in 1..mm, j \in A} (ms_{lj} \neq \circ \wedge ms_{lj} \leq s \wedge ms_{lj} + md_{lj} > s), \quad s \in 0..\max_{i \in T}(e_i) \tag{15}$$

Fig. 1. Model of AMRP-D. `disj` shorthand for `disjunctive` for layout reasons.

- as_{ij}: (optional) start time if flight leg i is assigned to aircraft j.
- $majs_{kj}$: (optional) start time for major maintenance type k for aircraft j.
- mm: The maximum number of maintenance tasks for each aircraft in schedule. Setting this value too high has negative effect on the performance as maintenance calculation is very challenging. Setting it too low leads to restrictions in the search space, cutting off optimal or even all feasible solutions. Based on the most frequent type of maintenance (regular), we use Eq. (16) for the definition, where $start_h$ is the earliest timeslot referenced in the history, and the addition of 5 was determined empirically based on a large set of possible sub-problems.

$$mm = \frac{\max_i(e_i) - start_h}{regl} + 5 \tag{16}$$

- ms_{lj}: (optional) start time for l^{th} maintenance task for aircraft j.
- md_{lj}: the duration of the l^{th} maintenance task for aircraft j.
- mt_{lj}: type of l^{th} maintenance on aircraft j in {1..6} where {1..4} are major, 5 is weekly, 6 is regular.

We use notation $s = \circ$ for an optional start time variable s to mean it is *absent*, that is does not occur. Constraints involving absent variables are considered to always hold.

The mathematical model is shown in Fig. 1. Equation (1) ensures that aircraft j is assigned their last assigned flight leg starting at the correct time. We use $\overline{as}_i$ to represent the i^{th} row of matrix as, similarly $\overline{as}_j$ represents the j^{th} column, and similarly for other matrices. Equation (2) enforces that each leg is assigned to some aircraft, the `alternative` constraint enforces that exactly one of the optional start times for that leg equals the given start time. Equation (3) enforces that only one major maintenance task requiring the hangar is executing at any time; the optional task `disjunctive` enforces this, ignoring absent tasks. Equation (4) enforces that the total flight time of legs performed by aircraft j before a major maintenance of type k fits within the limit $majl$; note that if $majs_{kj} = \circ$ we assume the inequality $as_{ij} < majs_{kj}$ always holds. Equation (5) enforces that the start times for the list of maintenance tasks for each aircraft are strictly increasing; note absent tasks always satisfy this constraint. Equation (6) ensures that absent maintenance tasks are at the end of the list. Equation (7) makes the first maintenance task for aircraft j a weekly maintenance task ending at time w_j; note that this happens before all tasks to be scheduled. Similarly if $r_j \neq w_j$, Equation (8) makes the second maintenance task for aircraft j a regular maintenance task ending at time r_j. Equation (9) enforces that the duration of the l-th maintenance task for aircraft j matches the type of maintenance; the $\texttt{element}(i, a, x)$ constraint enforces that $x = a[i]$. Equation (10) enforces that there is no overlap in the (optional) flight and maintenance tasks for each aircraft j; here $+\!\!+$ represents sequence concatenation. Equation (11) enforces that there is a maintenance task before each leg i flown by aircraft j so that the end of the flight leg is within the regular maintenance limit. Similarly Equation (12) enforces that there is a (non-regular) maintenance task before each leg i flown by aircraft j so that the end is within the weekly limit. Equation (13) enforces that for each major maintenance task of type k that is actually scheduled for aircraft j, that it occurs in the list of maintenance tasks with the same start time. Equation (14) enforces that if a major maintenance type occurs in the list of maintenance tasks, then that major maintenance is scheduled.

3.1 Optimization Objectives

The objective function is to minimize the sum of squares of the number of simultaneous maintenance tasks occurring at each minute s, m_s. The calculation of m_s is given by Eq. (15), leaving the basic objective as:

$$\text{minimize} \sum_{s \in 0..\max_i(e_i)} m_s^2 \tag{17}$$

There is a secondary requirement, to make the major maintenance happen as late as possible which is achieved by an additional objective term:

$$\sum_{k \in 1..4, j \in A} majl - \left(majp_{kj} + \sum_{i \in T', as_{ij} \neq \circ, as_{ij} < majs_{kj}} f_i \right) \tag{18}$$

Note that the expression calculates the remaining flight time allowed for each aircraft and major maintenance type (the same term is used in Eq. (4) and the model defines the expression only once).

3.2 Modeling the Objective Function

Using a sum of squares over all timeslots in the planning horizon is not a feasible approach in CP, as the number of timeslots is too high (flights are in granularity of 5 min, the horizon is up to a month), and the squaring is not efficient for most CP solvers either. Equation (18) is easy to handle and is included in all options described here.

$$\text{minimize} \sum_{l \in 1..mm, j \in A, ms_{lj} \neq \circ} md_{lj} \tag{19}$$

Equation (19) shows a simple objective to minimize the total duration of the maintenance tasks. This is easy to model, however, it does not directly help with the distribution objective. It can be combined with a `cumulative` constraint to minimize the maximum number of concurrent maintenance tasks as well. This approach has been conducted in our preliminary work [9], however, the distribution objective was only poorly optimized with this version.

A more efficient way to capture the true objective function is to introduce an additional array $maint$ of length $2 \cdot mm \cdot m + 1$ which contains the sorted array of all maintenance start and end times of all aircraft (plus a dummy at the end). This can be achieved with the `sort` constraint, but is more efficient in a custom mapping that is skipped here for brevity.

$$act_s = \sum_{l \in 1..mm,\ j \in A} ms_{lj} \neq \circ \ \wedge \ ms_{lj} \leq maint_s \wedge \ ms_{lj} + md_{lj} > maint_s \tag{20}$$

$$\text{minimize} \sum_{s \in 1..2 \cdot mm \cdot m} act_s \cdot act_s \cdot (maint_{s+1} - maint_s) \tag{21}$$

Equation (20) defines the number of active maintenance tasks at point $maint_s$ in time (the sum counts the number of times the expression evaluates to true), while Eq. (21) only sums at all points where a maintenance starts and ends, instead of all timeslots.

However, Eq. (21) is still costly to evaluate, therefore, we propose a new overestimation of the objective that is much less computationally involved, but optimizes in a similar way to the exact objective.

$$appr_{lj} = \sum_{x \in 1..mm,\ y \in A} ms_{lj} \neq \circ \ \wedge \ \text{overlap}(ms_{lj}, md_{lj}, ms_{xy}, md_{xy}) \tag{22}$$

$$\text{minimize} \sum_{l \in 1..mm, j \in A} appr_{lj} \cdot md_{lj} \tag{23}$$

Equation (22) approximates the number of active maintenance during maintenance lj by all maintenance (including itself) that overlap in any way with this one. In reality, they will not overlap for the whole duration, which leads to an over-approximation. In Eq. (23) the approximate number is assumed for the whole duration. For each maintenance, each overlapping one contributes to the overall sum, therefore the square is not needed in the sum itself.

4 Obtaining an Initial Feasible Solution

An initial solution is required for local search procedures. One of the challenges of this problem is the attainment of a *feasible* initial solution. As AMRP is a *hard to satisfy* problem, getting an initial feasible solution with a greedy heuristic is often not possible due to its complexity. Therefore, in the first part of this work, we present a decomposition approach with backtracking to provide initial feasible solutions. The goal is to solve smaller (and therefore faster) subproblems when possible, while iteratively increasing the size when needed.

The planning horizon is divided into D slices. To allocate the legs to slices, we first identify the leg with the earliest start time and then assign this leg and all overlapping unassigned legs to a slice. Another strategy we explore is the use of 2 legs with the earliest start time to define the legs per slice. The total number of slices D varies from instance to instance. Each leg can belong to only one slice. Then, in an iteration we consider the legs of slice $d \in \{1, 2, \ldots, D\}$ and all the aircraft, sequentially moving through the slices in the planning horizon. At slice d we consider the legs of that slice whilst the legs from the slice 1 to slice $d-1$ are fixed to their previously assigned aircraft. Maintenance tasks from previous slices are not fixed, they are re-scheduled at each slice. If a feasible solution at slice d is found, we proceed to the subsequent slice, otherwise we *backtrack*. With this method, the aircraft are always in a feasible status, however it can happen that legs can not be assigned to the aircraft. One reason for this is that in between flights the aircraft need to be maintained (regular, weekly or major maintenance), so a rearrangement of the legs is needed before moving on to the next slice. If backtracking occurs at slice d, then the flights belonging to slice *d-1* need to be reassigned. More slices are iteratively unfixed until a feasible assignment of legs and maintenance tasks for all aircraft can be scheduled according to the regulations up to the current slice. For both approaches, the model is solved to find feasible solution (no objective function). After an initial solution has been reached, the procedure optimizes the maintenance for each aircraft individually using Eq. (23) (as the initial feasible solution can contain redundant maintenance in the absence of an objective function).

5 Large Neighborhood Search for AMRP-D

Large Neighborhood Search (LNS) is a local search algorithm proposed by Shaw [14]. LNS starts with an initial solution that commonly is reached with a greedy heuristic. With the principle of *destroy* and *repair*, LNS selects a portion of the solution using the destroy operator/s and removes fully or partially the assignments in this portion. The other part of the solution remains fixed. Then, using repair operator/s the selected portion of the solution is restored. Multiple destroy and repair operators can be included in the LNS. If the new solution improves the current one, the solution is updated, otherwise the next iteration of the algorithm starts from the old solution. This procedure is carried out repeatedly until the stopping criterion is met.

5.1 Destroy Operators

By using the destroy operators, a part of the solution at any iteration is selected. While the rest of the solution remains intact, this part of the solution is destroyed which means that those existing assignments are removed. The design of the destroy operators is crucial for achieving good results with LNS. Typically, the destroyers should explore the structural properties of the problem.

After a careful analysis of the AMRP-D, we propose five different destroy operators, which are utilized in the LNS based on their respective weight using the *roulette wheel approach* [5]. Each destroy operator is used only on some consecutive days of the scheduling period, determined by a parameter d, in order to maintain a reasonable runtime per iteration. The other assigned flight legs and maintenance tasks for the aircraft present in the sub-problem are considered as *history* (or future) assignment. The parameter k is used to identify up to k aircraft of interest based on the respective operator. For each selected aircraft, all legs in the selected time window are part of the sub-problem. Considering a *destroy size* of k aircraft and a duration of d days (illustrated in Table 1), the following operators are used:

- `Non-overlapping Legs (NLegs)`: For a randomly selected flight leg t, identify $k-1$ flight legs that do not overlap with t which are closest to the start/end time of t, each aircraft covering any of these legs is part of the sub-problem. The period included in the sub-problem starts at earliest start time of these legs.
- `Overlapping Legs (OLegs)`: For a randomly selected flight leg t, select $k-1$ overlapping flight legs, and add the aircraft these are assigned to. The period included in the sub-problem starts at the earliest start time of these legs.
- `Peak Aircraft (PAir)`: Peak is defined to be the time in the planning period when the highest number of maintenance tasks occurs simultaneously and peak aircraft are the aircraft being maintained in the peak time. Then in the sub-problem an aircraft in the peak time and $k-1$ aircraft not in the peak time are selected. The start time for the sub-problem begins one day earlier than the peak time.

- `Size Aircraft (SAir)`: Select the aircraft with the maximum number of flight legs and the minimum number of flight legs assigned. The start time is randomly selected within the scheduling period.
- `Random Aircraft (RAir)`: For a randomly selected aircraft, select one of its maintenance start times (either regular or weekly maintenance). For the picked maintenance task, select $k-1$ flight legs that overlap with it, add the initial aircraft and those having one of the selected legs in their assignment. The start time of this operator is the minimum start time among these legs and the maintenance.

5.2 Repair Operator

In the repair phase, the focus is on the sub-problem which is formed based on the destroy operators. Some additional data regarding the aircraft not present in the subproblem is needed. Specifically, the start time and duration of the maintenance tasks of other aircraft is passed each iteration to the CP model in order to use the *overestimation* the objective function based on Eq. (23). Using the exact objective function is not time efficient even for the sub-problem. Further, the next fixed leg, and the latest possible times for maintenance are specified for each aircraft similar to the history assignments.

5.3 Parameters

The parameters used in the LNS are described in Table 1. In the first column, the destroy operators are listed. The number of aircraft and the number of days per operator, given in column two and three, are determined based on preliminary experiments to keep the runtime per iteration moderate while maximizing the effect of the operator. Probabilities p_1, p_2, p_3, p_4 represent the selection probabilities for the destroy operators `NLegs`, `OLegs`, `PAir`, and `SAir` respectively. Then, the `RAir` destroy operator is used with $p_5 = 1 - (p_1 + p_2 + p_3 + p_4)$. The selection probabilities are determined with the automatic parameter configuration tool `irace` [10]. For this setup, we use 17 instances with a budget of 10000 experiments, where each experiment has a timeout of 600 s. The domains given to the selection probabilities are given in the fourth column of Table 1 while in the fifth column we display the chosen values from `irace`. Lastly, the repair time given to the CP solver is set with manual trials to $t_r = 200$ s.

6 Simulated Annealing for AMRP-D

As a second improvement method we propose to use Simulated Annealing [8]. It starts from a given initial solution, with an initial temperature $t = T_0$, and randomly selects moves with given probabilities. The new solution is accepted if it is better than before, or with probability $e^{-\Delta/t}$, where Δ is the change in solution value, and t is the current temperature. A number of inner iterations

Table 1. Parameter values used for the LNS

Operator	Nr aircraft	Nr days	Domain	Tuned
`NLegs`	3	5	$p_1 \in [0, 0.5]$	0.50
`OLegs`	5	3	$p_2 \in [0, 0.5]$	0.22
`PAir`	3	5	$p_3 \in [0, 0.5]$	0.06
`SAir`	2	6	$p_4 \in [0, 0.5]$	0.18
`RAir`	3	5	$p_5 \in [0, 0.5]$	0.04

inner is performed on the same temperature, then the temperature is reduced by $t' = c \cdot t$, where c is the cooling rate. Hard constraint violations can occur and are treated with high penalties (missing maintenance or overlap of flight legs).

Three different types of moves are used in our Simulated Annealing. The first two deal with the assignment of flight legs and either reassign one flight leg (`move_leg`), or a sequence of consecutive legs (`move_legs`, the length of the sequence is randomly chosen), from one aircraft a_1 to another a_2. All legs overlapping with the moved legs in a_2 are moved back to a_1. The third move `recalc_maintenance`, which is a novel addition for the AMRP-D, recalculates all maintenance for a randomly selected aircraft with the procedure introduced in the following, while the assignment of legs is fixed. This allows us to reoptimize the maintenance distribution.

6.1 Maintenance Optimization

Maintenance optimization for aircraft a is performed in a sequential pass in the order of time. The last possible maintenance location (denoted by the gap between legs t_x and t_y consecutively assigned to aircraft a) for each type of maintenance is kept in memory. For each flight leg t_j, first for each type of maintenance the last possible location is updated to (t_i, t_j) (where t_i is the predecessor of t_j on a) if the location is suitable. If adding t_j would trigger a maintenance violation, the last possible location is chosen for the maintenance. The assignment might be infeasible if no possible location was found before t_j. This part of the assignment procedure ensures that maintenance is not done excessively early, but just when needed.

The second part of the assignment procedure picks the best spot between the consecutive legs t_i and t_j where the maintenance is assigned. In general, the interval $[e_i, s_j]$ is available for the assignment, but the beginning might be restricted further to $e_i + x$ if necessary for the feasible assignment of the currently evaluated leg. Within this interval, the maintenance is added at the latest possible spot where the maintenance distribution objective is minimal. For efficiency, the evaluation is limited to all points in time in the interval where the overall maintenance assignment of the problem changes.

This concept was also used in a greedy heuristic, which sorts the flight legs by start time, and then assigns each leg to the aircraft with the least increase in

Table 2. Parameter values used for the SA

Parameter	Values	Default	Tuned
T_0	$\{1, 10, 100, 1000, 10000\}$	100	100
c	$\{0.9, 0.95, 0.99, 0.995, 0.999, 0.9995, 0.9999\}$	0.999	0.9999
p_1	$(0; 1)$	0.05	0.22
p_2	$(0; 1)$	0.475	0.52

the objective, using the above procedure for maintenance assignment. However, this approach was not successful in avoiding conflicts.

6.2 Parameters

The parameters used for SA are described in Table 2. The two parameters T_0 and c for the cooling scheme, as well as the probabilities p_1 for `move_leg` and p_2 for `move_legs` are tuned. `recalc_maintenance` is chosen with $p_3 = 1 - (p_1 + p_2)$. The number of inner iterations $inner = |T|$ is fixed (with a lower bound of 100, and an upper bound of 1000) to allow scaling for the instance size, but using more inner iterations with faster cooling would be similar to less inner iterations with slower cooling, therefore, only one of the parameters is tuned. Simulated Annealing is implemented with efficient delta evaluation, only reevaluating parts of aircraft schedules when needed. Additionally, there is an early stopping criterion: if there is no improvement in 100 temperature steps, the algorithm stops, since further improvements are very unlikely. The parameters are again tuned with `irace` [10] on the same 17 instances as for LNS, but with a budget of 1000 experiments with the full timeout of 1 h (the cooling scheme needs the full runtime to be properly tuned).

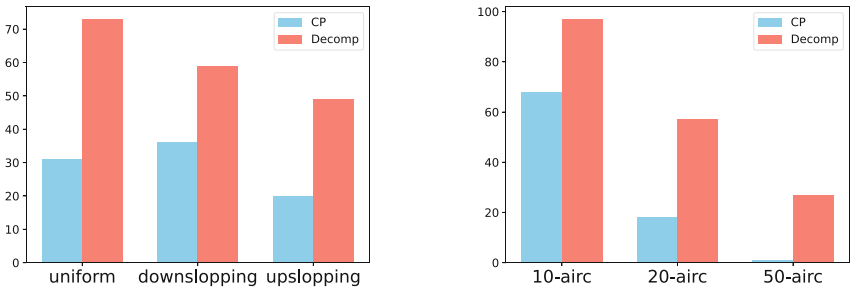
7 Experimental Evaluation

The experiments were done on a computing cluster, equipped with two Intel Xeon E5-2650v4 @ 2.20 CPUs with 12 cores. CP-SAT 9.10 [12] is used for solving all versions of the CP model described in Sect. 3 which was modeled in MiniZinc [11] and PyPy 7.3.11 for running the other code written in Python. The runs of the decomposition method were executed in 8 threads using a time budget of 12000 s. These runs are deterministic and performed only once. The runs of LNS and of SA were executed in a single thread using a time budget of 3600 s. As the LNS and the SA are not deterministic, we perform 10 runs per instance for each method. The seeds are stored for reproducibility.

The available benchmark instances are created to resemble structures of real-life instances from a practical use case (single hub, mid- to long-distance flights, with seasonal fluctuations and daily departure peaks). The instance set combines four different features as shown in Table 3, leading to 324 instances. More dense instances include more flight legs for the same number of aircraft, making it

Table 3. Instance characteristics

Number of aircraft	10, 20, 50
Number of days	7, 14, 28
Daily departure peaks	0 (no) or 1 (yes)
Density	5–10, d4–d9, u4–u9



(a) Instance structure (b) Instance size

Fig. 2. Number of solved instances based on instance structure and size

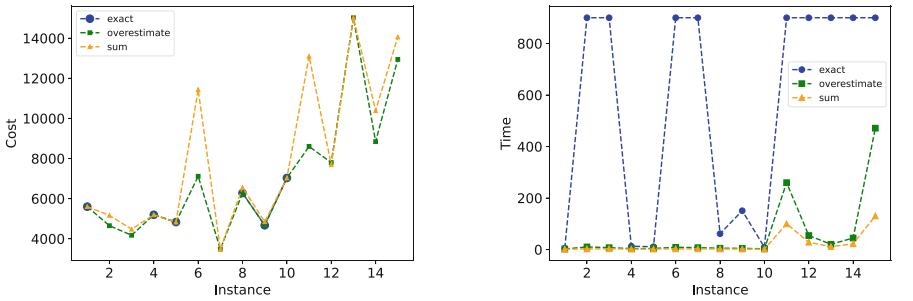
harder to even find feasible solutions. d4–d9 represents demand that reduces during the scheduling period (downsloping), while u4–u9 represents upsloping demand. These combinations result in 70 to 1695 flight legs per instance.

7.1 Feasibility Results

Figure 2 shows a comparison for the number of feasible solutions, solving only for satisfaction, using the model from Sect. 3 (CP) and the decomposition approach from Sect. 4 (Decomp) that uses the same model at each slice d. Note that the greedy approach mentioned in Sect. 6 could only provide 7 feasible solutions for instances of very low density and is not shown here. CP mostly covers small sized instances with a total of 87 feasible instances, while the decomposition approach with two strategies provides 181 feasible instances. There is only one instance solved only by CP and not the decomposition method. CP struggles with instances that have upsloping demand (Fig. 2a) and with the medium to large sized instances (Fig. 2b). The decomposition technique handles instances of diverse structures well, but as the size of the instances increases, fewer feasible solutions are obtained.

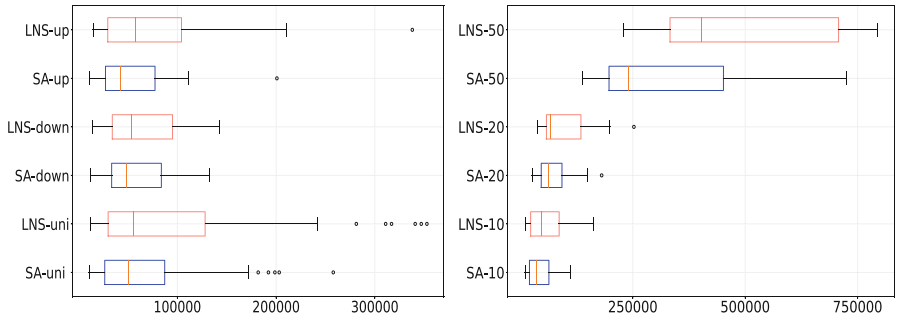
7.2 Reformulating the Objective in MiniZinc

In order to evaluate the proposed objective function formulations we perform experiments using a timeout of 900 s on LNS sub-problems. In Fig. 3a the exact cost for 15 different sub-problems obtained from the exact Eq. (21), overestimate



(a) Cost from different formulations (b) Time from different formulations

Fig. 3. Comparing different formulations of objective function



(a) Average values boxplot (structure) (b) Average values boxplot (size)

Fig. 4. Boxplots for the average values obtained from SA and LNS

Eq. (23) and sum approaches Eq. (19) is shown whilst in Fig. 3b the respective time of each approach for the same sub-problems is displayed. Note that in Fig. 3a the missing values for the exact approach indicate that no solution was returned in this timeout, which happens for 9 out of 15 sub-problems. This indicates that the exact approach might not be adequate for the iterative procedure of LNS. For the sub-problems where the exact method returns a solution, the overestimate approach and the sum approach match their cost. Nevertheless, their runtime is lower than the one of the exact method. Comparing the overestimate and the sum approach, the sum approach runs faster, but it provides lower quality solutions. Therefore, we include the overestimate approach in the MiniZinc model used for the decomposition approach to remove redundant maintenance tasks, and for the LNS procedure.

7.3 Optimization Results

In this part we compare the results from SA and LNS, when both algorithms use the initial solutions obtained from the decomposition approach.

In Fig. 4, we present the boxplots for the average values from SA and LNS based on the instance structure (Fig. 4a) and based on the instance size (Fig. 4b). Some outliers are outside the visualized range, almost all for LNS. For the minimum values achieved in 10 runs, LNS provides better results than SA for 14 instances, provides equal results with SA for 6 instances and SA offers the best minimum values for the remaining 161 instances. Regarding the average values over 10 runs, LNS provides better results than SA for 14 instances, provides same results with SA for 3 instances and for the other 164 instances SA delivers better average results. Most of the instances for which LNS provides better minimum or average values are instances with 10 aircraft and uniform density. As shown in Fig. 4a, SA and LNS perform similarly for instances with downsloping density, while for instances with upsloping or uniform density their performance differences are more noticeable. In Fig. 4b, the largest performance difference is for instances with 50 aircraft. The average improvement of SA and LNS compared to the objective returned by the decomposition approach for instances with 10 aircraft is -23.94% for SA and -13.89% for LNS; for instances with 20 aircraft -33.46% for SA and -14.64% for LNS; for instances with 50 aircraft -36.66% for SA and -7.20% for LNS. Moreover, LNS provides feasible results for all instances over all runs unlike SA that at times return infeasible solutions. This is due to the fact that SA explores also infeasible regions and might get stuck in such regions. When this happens the cost returned by SA is the cost of the initial solution. For LNS, subproblems of larger size might be needed for better results, however, these quickly become very time-consuming.

Additionally, we perform the Wilcoxon signed-rank test with the Python module `scipy`. The obtained p-value is 5.08×10^{-25}. With a significance value of $\alpha = 0.05$, these results show that SA and LNS perform statistically different.

For some smaller instances, we could find the optimum for the total maintenance duration (Eq. (19)), and verified that optimizing the distribution does not increase this metric too much (average within 2.3%), while the distribution is well optimized in our new results.

8 Conclusions

In this paper we investigated a specific version of the aircraft maintenance routing problem that focuses on the distribution of maintenance tasks. We provided a solver-independent model that can be used to solve small instances directly, as well as in our decomposition approach, which resulted in far more feasible solutions. We proposed an LNS method with new destroy operators and a CP solver in the repair stage. We analyzed different ways of formulating the objective function in the MiniZinc model to improve the efficiency of the repair stage, and showed that our over-approximation method can provide a large speed-up with only moderate deviations in the objective. Furthermore, we presented an SA method with three distinct neighborhoods. While both improvement methods can achieve large improvements in the distribution objective, Simulated Annealing provides the best results. Our methods contribute to a better integration of

aircraft routing and scheduling of maintenance work. For future work we would like to investigate the branch-and-price technique for the AMRP-D and investigate multi-hub scenarios, where maintenance can be done in multiple locations.

Acknowledgments. The financial support by the Austrian Federal Ministry of Labour and Economy, the National Foundation for Research, Technology and Development and the Christian Doppler Research Association is gratefully acknowledged. This research was also funded by the Austrian Science Fund (FWF), project I 5443-N, Grant-DOI 10.55776 and by the Doctoral Program Vienna Graduate School on Computational Optimization, Austrian Science Foundation (FWF), under grant No.: W1260-N35 (https://vgsco.univie.ac.at/). The work of P.J. Stuckey and H. Bierlee is partially supported by the Australian Research Council (OPTIMA ITTC IC200100009).

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

References

1. Başdere, M., Bilge, Ü.: Operational aircraft maintenance routing problem with remaining time consideration. Eur. J. Oper. Res. **235**(1), 315–328 (2014)
2. Cui, R., Dong, X., Lin, Y.: Models for aircraft maintenance routing problem with consideration of remaining time and robustness. Comput. Industr. Eng. **137**, 106045 (2019)
3. Eltoukhy, A.E., Chan, F.T., Chung, S.H.: Airline schedule planning: a review and future directions. Industr. Manage. Data Syst. **117**(6), 1201–1243 (2017)
4. Feo, T.A., Bard, J.F.: Flight scheduling and maintenance base planning. Manage. Sci. **35**(12), 1415–1432 (1989)
5. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning, 1st edn. Addison-Wesley Longman Publishing Co., Inc., USA (1989)
6. Gopalan, R., Talluri, K.T.: The aircraft maintenance routing problem. Oper. Res. **46**(2), 260–271 (1998)
7. Haouari, M., Shao, S., Sherali, H.D.: A lifted compact formulation for the daily aircraft maintenance routing problem. Transp. Sci. **47**(4), 508–525 (2013)
8. Kirkpatrick, S., Gelatt, C.D., Jr., Vecchi, M.P.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983)
9. Kletzander, L., Gjergji, I., Musliu, N.: Combining aircraft maintenance routing with a distribution objective. In: Proceedings of the 14th International Conference on the Practice and Theory of Automated Timetabling, PATAT 2024, pp. 325–328 (2024)
10. López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., Stützle, T.: The irace package: iterated racing for automatic algorithm configuration. Oper. Res. Perspect. **3**, 43–58 (2016)
11. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: MiniZinc: towards a standard CP modelling language. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 529–543. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74970-7_38
12. Perron, L., Didier, F., Gay, S.: The CP-SAT-LP solver (invited talk). In: Yap, R.H.C. (ed.) 29th International Conference on Principles and Practice of Constraint Programming, CP 2023, 27–31 August 2023, Toronto, Canada. LIPIcs, vol. 280, pp. 3:1–3:2. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2023)

13. Sarac, A., Batta, R., Rump, C.M.: A branch-and-price approach for operational aircraft maintenance routing. Eur. J. Oper. Res. **175**(3), 1850–1869 (2006)
14. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: International Conference on Principles and Practice of Constraint Programming (CP-98), pp. 417–431. Springer (1998)
15. Temucin, T., Tuzkaya, G., Vayvay, O.: Aircraft maintenance routing problem–a literature survey. Promet-Traffic Transp. **33**(4), 491–503 (2021)
16. Xu, Y., Wandelt, S., Sun, X.: Airline scheduling optimization: literature review and a discussion of modeling methodologies. Intell. Transp. Infrastruct. (2023)

Reducing Income Variability in Natural Resource Portfolios via Integer Programming

Laura Greenstreet[1(✉)], Qinru Shi[1], Marc Grimson[1], Franz W. Simon[2,3], Suresh A. Sethi[2,4], Carla P. Gomes[1], Andrea Lodi[5], and David B. Shmoys[1,6]

[1] Department of Computer Science, Cornell University, Ithaca, NY, USA
leg86@cornell.edu
[2] Department of Natural Resources and the Environment, Cornell University, Ithaca, NY, USA
[3] Department of Biology, University of Kentucky, Lexington, USA
[4] Aquatic Research and Environmental Assessment Center, Department of Earth and Environmental Sciences, Brooklyn College, Brooklyn, NY, USA
[5] Jacobs Technion-Cornell Institute, Cornell Tech, New York, NY, USA
[6] School of Operations Research and Info Engineering, Cornell University, Ithaca, NY, USA

Abstract. Alaskan fishing communities are heavily impacted by income variability, where studies suggest benefits to individuals and communities through diversification by participating in multiple fisheries. However, Alaska uses a limited entry permit system, allowing only a fixed number of individuals to participate in each fishery. This motivates a resource allocation problem to determine how to allocate fishing permits to minimize a global measure of income variability. Further, experts and policy makers are interested in what interventions are most effective for enabling diversification. In collaboration with Alaskan fisheries experts, we developed a quadratic constrained resource allocation problem to reduce community-level income variability and model financial and vocational training interventions. Using over 20 years of fisheries data, we demonstrate an integer programming approach can solve instances up to the state level, involving over 10,000 permits and 170 communities. The model shows the potential for a 30–75% reduction in community-level average fishery income variance and provides a flexible framework for resource managers to explore the impacts of financial and vocational training interventions to support natural resource portfolio adaptation.

Keywords: resource allocation · discrete optimization · quadratic integer programming · Alaskan fisheries · resource management

1 Introduction

Fisheries play important economic and social roles worldwide, including being responsible for more than 250 million primary livelihoods and being the main source of protein for over 20% of the global population [5,10,24]. In Alaska,

G. Tack (Ed.): CPAIOR 2025, LNCS 15763, pp. 18–34, 2025.
https://doi.org/10.1007/978-3-031-95976-9_2

seafood is the largest private sector employer, where for many communities, fisheries are the primary contributor to the local economy [14]. However, fishery incomes can be highly variable, where low-income years can be devastating for small communities. For example, in 2022 a crash in the snow crab fishery led the island of St. Paul to declare a 'cultural, economic, and social emergency', with tax revenue no longer able to cover emergency medical services [18]. As some fisheries are negatively correlated, income variance can be reduced through participation in multiple fisheries. However, obtaining new permits, gear, and vocational skills pose a significant barrier to entry. In this work, we explore the potential to reduce community-level revenue variance for over 170 fishing communities in Alaska, formulating a new constrained resource allocation problem and modeling the potential benefits of the interventions of permit financing and vocational skills training. The problem was studied working closely with fisheries experts in both academia and the Alaskan fisheries management community.

Alaska is the largest seafood producer in the United States, accounting for over a third of the revenue from the $5.9 billion industry [8]. Alaskan fisheries are also extremely diverse, with over 50 active commercial fisheries and operations ranging from individual owner-operated businesses to large vertically integrated catcher-processor enterprises [14]. Beyond economic impacts, income variability can have negative effects on community structures and populations' physical and mental health [8]. While prior work suggests income variability can be reduced through diversification by participating in multiple fisheries [15,23], this has not considered the problem at the system-level, taking into account that Alaska's limited entry commercial permit system only allows a fixed amount of participation in each fishery. Further, experts and policy makers want to understand how to maximize the benefits of potential interventions, such as financing to help fisherman with limited collateral receive loans to purchase new permits and equipment or vocational training to develop the skills to participate in new fisheries.

In addition to being the largest fisheries system in the United States, Alaskan fisheries are extremely data-rich, with over 40 years of data on fishing permit ownership, sales, and revenue. This provides a unique opportunity to understand economic patterns and study the potential for interventions through data-driven approaches.

Prior Work. While prior work suggests that income variability can be reduced through fishery diversification, to our knowledge, we are the first work to optimize at the system-level. Retrospective analyses have found that fishery income volatility decreases with portfolio diversity at both the individual and community-level [15,23]. A number of strategies have been investigated to mitigate fisheries risk [12,22] and several recent papers have characterized the risk-return frontier for fishery portfolios [17,25]. However, these works do not optimize over the full system and thus do not take into account that there is a limited amount of catch that can be sustainably caught from each fishery. Such efforts are critical for both understanding the potential benefits of diversification as a practical fisheries risk management strategy and for identifying policy levers to support diversification.

Although the permits owned by each community can be seen as a portfolio, this problem differs from standard portfolio optimization problems in that we are optimizing portfolios for multiple individuals with constraints ensuring that all permits are allocated [16]. Thus, the problem is more closely related to resource allocation problems, in which a fixed amount of resources must be divided among groups or tasks to minimize a cost function [9]. Integer and mathematical programming methods have been applied to a variety of discrete resource allocation problems including proportional division for parliamentary systems, virtual machine placement in cloud infrastructure, and in multi-carrier communication systems [11,19,21]. However, minimizing a measure of variance complicates our problem by introducing a quadratic objective, where non-linear resource allocation problems have received less attention [6].

Paper Contribution. The contributions of our paper are as follows:

1. In collaboration with fisheries experts, we formulate a resource allocation problem to study the potential for reducing community-level income variability across Alaska through fishery diversification and model permit financing and vocational training interventions.
2. Using integer programming approaches, we are able to optimize over instances as large as the full state, including over 170 communities and over 10,000 individual permits from over 50 fisheries.
3. Our results show that an overall 30% average variance reduction is possible through financial incentives alone and this variance can be further reduced by over 75% by combining financial and vocational training interventions, revealing new opportunities for fisheries and optimization research.

Paper Organization. The remainder of the paper is organized as follows. Section 2 formally defines the optimization problem and presents a quadratic integer programming formulation and its linearization. Section 3 describes data processing and instance generation, the experimental setup, and experimental results. Section 4 discusses the optimization performance and implications of the solutions. Finally, Sect. 5 summarizes our contribution and outlines future research directions.

2 Modeling Framework

We now provide a more detailed description of the Alaska's commercial fishery permit system and then formally define the optimization problem. This analysis considers all commercial limited entry permits managed by the Alaskan Commercial Fisheries Entry Commission. Limited entry permits are owned by individuals and allow them to fish for a specific {species, location, and gear type} triplet combination. For example, a permit may allow an individual to fish for salmon in Cook Inlet using a drift gillnet, while different permits would be needed to fish for salmon in the same location using a set gillnet or in Prince

William Sound using the same gear. Permits of the same type are interchangeable, with the same potential for revenue. However, quotas vary from year to year, which – combined with factors such as changes in market price – can create highly variable revenue.

We define portfolios at the community-level using geographic locations, where a permit is considered part of a community's portfolio if it is registered to an address in that geographic location. While permits are owned by individuals, diversification and interventions often have impacts at the community-level. For example, fishing income is often reinvested into the community through the purchase of goods and services, leading to multiple individuals and businesses being impacted by low-income years. Communities are also an important source of knowledge, where skills can be transferred within a community.

Our goal is to determine an allocation of permits that minimizes community-level revenue variance, subject to feasibility constraints. First, we want to ensure that expected community revenues do not vary too much from their initial values. Second, Alaska is geographically vast, where the travel burden to use a permit can range from being able to fish within a community to requiring over 2,000 km of travel. Thus, we characterize a travel cost associated with each permit based on the location of fishing grounds and its community of origin. Finally, we want to understand the impact of two potential interventions: (i) financing, to allow the purchase of new permits and (ii) vocational training, to allow the usage of new permit types. While permit prices can be obtained from historical sales data, it is difficult to assign a financial cost to vocational skills training. Instead, we assign skills acquisition costs for the relative difficulty of acquiring the skills to use a permit assuming one previously fished another permit, which is then generalized to a community-level skills acquisition cost by taking the minimum skills acquisition cost across all permit types initially owned by a community. By assigning a vocational training budget, we can compare the relative impact of permit financing and vocational training investments.

2.1 Mixed Integer Quadratic Program Formulation

Let C and P be the sets of communities and permit types respectively, where a permit type is defined by a species-location-gear triplet (see Table 1 for all parameter definitions). For each permit $p \in P$, let $\text{Price}(p)$ denote its price, $\text{Rev}(p)$ denote its expected annual revenue, $\text{Var}(p)$ denote its variance, and $\text{Cov}(p, p')$ denote its covariance with $p' \in P$. For a community $c \in C$, and permit $p \in P$, let d_{cp} denote the distance that community c must travel to use permit p. Let $s : p_1, p_2 \rightarrow \mathbb{R}$ be a function representing the difficulty of obtaining the skills to transition from fishing permit p_1 to permit p_2 and P_c denote the set of permits initially owned by community c. We can then define the community-level vocational skills transition function, $t : c, p \rightarrow \mathbb{R}$ as

$$t(c, p) = \min_{p' \in P_c} s(p, p'). \tag{1}$$

Further, let x_{pc} be an integer decision variable representing the number of permits of type p in community c and x^0_{pc} be a constant for the number of

permits of type p initially in community c. We can define the community level expected revenue and variance as

$$\text{Rev}(c) = \sum_{p \in P} \text{Rev}(p) \cdot x_{cp}, \tag{2}$$

$$\text{Var}(c) = \sum_{p \in P} \left[\text{Var}(p) \cdot x_{cp}^2 + \sum_{p' \in P, p \neq p'} \text{Cov}(p, p') \cdot x_{cp} x_{cp'} \right]. \tag{3}$$

where $\text{Var}(c^0)$ and $\text{Rev}(c^0)$ denote the values for the initial permit configurations defined by the x_{pc}^0 values. Finally, let v_c be a non-negative continuous decision variable used to enforce the permit financing constraint. By bounding these variables below by the cost difference between a community's initial and optimized portfolios, we can limit the total value of permit purchases that need to be supported by financing.

Using these definitions, we can state the problem by the mixed integer quadratic program (MIQP)

$$\min \quad \frac{1}{|C|} \sum_{c \in C} \frac{\text{Var}(c)}{\text{Rev}^2(c^0)} \tag{4}$$

$$\text{s.t.} \quad \sum_{c \in C} x_{pc} = \sum_{c \in C} x_{pc}^0 \qquad \forall p \in P \tag{5}$$

$$\sum_{p \in P} x_{pc} = \sum_{p \in P} x_{pc}^0 \qquad \forall c \in C \tag{6}$$

$$\sum_{p \in P} \text{Rev}(p) \cdot x_{pc} \geq (1 - \phi) \cdot \text{Rev}(c^0) \qquad \forall c \in C \tag{7}$$

$$\sum_{p \in P} d_{pc} x_{pc} \leq (1 + \eta) \cdot \sum_{p \in P} d_{pc} x_{pc}^0 \qquad \forall c \in C \tag{8}$$

$$\sum_{p \in P} \text{Price}(p) \cdot (x_{pc}^0 - x_{pc}) \leq v_c \qquad \forall c \in C \tag{9}$$

$$\sum_{c \in C} v_c \leq \psi \tag{10}$$

$$\sum_{p \in P} \sum_{c \in C} t_{pc} x_{pc} \leq \rho \tag{11}$$

$$v_c \geq 0 \qquad \forall c \in C \tag{12}$$

$$x_{pc} \in \mathbb{Z}_{\geq 0} \qquad \forall p \in P, c \in C \tag{13}$$

We minimize the average community variance over squared revenue ratio as an approximation of the coefficient of variation (CV), which is preferable to the total variance as the impact of revenue variance is relative to a community's income. For example, a standard deviation in revenue of \$100,000/year is much more impactful for a community with an expected annual revenue of \$100,000 than one of \$1 million. Constraints (5) and (6) capture that the number of permits per fishery and community is fixed. Fixing the number of permits per community ensures roughly the same number of livelihoods remain in each community. However, we test relaxing this constraint. Constraints (7) and (8) ensure that community-level revenue and travel distance remain within fixed proportions of their initial values (ϕ and η, respectively). Constraints (9) and (10) ensure the total community-level change in permit portfolios requiring financing does not exceed ψ. Constraint (11) ensures that the total vocational skills training costs do not exceed ρ. Finally, constraints (12) and (13) define variables' characteristics.

2.2 Linearized Formulation

To avoid a quadratic objective, we can test a straight-forward linearization by considering each individual permit instead of permit type and introduce binary variables tracking whether pairs of permits are in the same community. Let P' be the set of all individual permits and extend the definitions of $\text{Price}(p), \text{Rev}(p), \text{Var}(p)$, and $\text{Cov}(p, p')$ to individual permits. Let y_{pc} be a binary decision variable for whether permit p is owned by community c and $w_{pp'c}$ be a binary decision variable for whether permits p and p' are both owned by community c. Further, extend the definitions of $\text{Rev}(c)$ and $\text{Var}(c)$ to individual permits, namely

$$\text{Rev}(c) = \textstyle\sum_{p \in P'} \text{Rev}(p) \cdot y_{pc}, \tag{14}$$

$$\text{Var}(c) = \textstyle\sum_{p \in P'} \left[\text{Var}(p) \cdot y_{pc} + \sum_{p' \in P', p' \neq p} \text{Cov}(p, p') \cdot w_{pp'c} \right]. \tag{15}$$

We can state the resulting mixed binary linear program (MBLP) as

$$\min \quad \frac{1}{|C|} \sum_{c \in C} \frac{\text{Var}(c)}{\text{Rev}^2(c^0)} \tag{16}$$

$$\text{s.t.} \quad w_{pp'c} \geq y_{pc} + y_{p'c} - 1 \qquad \forall c \in C, p, p' \in P' \tag{17}$$

$$\textstyle\sum_{c \in C} y_{pc} = \sum_{c \in C} y^0_{pc} \qquad \forall p \in P \tag{18}$$

$$\textstyle\sum_{p \in P'} y_{pc} = \sum_{p \in P'} y^0_{pc} \qquad \forall c \in C \tag{19}$$

$$\textstyle\sum_{p \in P'} \text{Rev}(p) \cdot y_{pc} \geq (1 - \phi) \cdot \text{Rev}(c^0) \qquad \forall c \in C \tag{20}$$

$$\textstyle\sum_{p \in P'} d_{pc} y_{pc} \leq (1 + \eta) \cdot \sum_{p \in P'} d_{pc} y^0_{pc} \qquad \forall c \in C \tag{21}$$

$$\textstyle\sum_{p \in P'} \text{Price}(p) \cdot (y^0_{pc} - y_{pc}) \leq v_c \qquad \forall c \in C \tag{22}$$

$$\textstyle\sum_{c \in C} v_c \leq \psi \tag{23}$$

$$\textstyle\sum_{p \in P'} \sum_{c \in C} t_{pc} y_{pc} \leq \rho \tag{24}$$

$$v_c \geq 0 \qquad \forall c \in C \tag{25}$$

$$y_{pc}, w_{pp'c} \in \{0, 1\} \qquad \forall p, p' \in P, c \in C \tag{26}$$

Constraint (17) connects the binary variables for whether pairs of permits are in each community, $w_{pp'c}$, to the binary variables for each permit being in each community, y_{pc}. The objective and other constraints remain the same, adapted to the binary variables. As discussed further in Sect. 4, this straight-forward linearization significantly increases the number of variables and constraints (with negative effects on the solvability of the model).

3 Experiments

The goal of the experiments is to both evaluate the optimization performance of the integer programming methods, in terms of both the optimality gap and objective, and to test that our modeling assumptions and interpretations are supported by the historical data. We test the following assumptions and hypotheses:

Table 1. Model parameters and sets

Sets	
P	Set of all permit types
P'	Set of all individual permits
C	Set of all communities
Parameters	
Price(p)	Purchase price of permit p
Rev(p)	Expected annual revenue of permit p
Var(p)	Annual variance of permit p
Cov(p, p')	Annual covariance between permits p and p'
Rev(c)	Expected annual revenue of community c, defined in (2)
Var(c)	Variance of community c, defined in (3)
d_{cp}	Travel distance for community c to use permit p
t_{cp}	Transition cost for community c to acquire permit p
ϕ	Maximum proportion of community expected revenue change
η	Maximum proportion of community travel distance change
ψ	Maximum cost of permit purchases requiring financing
ρ	Maximum value for vocational training costs across all communities

1. We assume vocational skill transition values capture the difficulty in transitioning between fisheries, so permit acquisitions with a high transition cost occur rarely in practice.
2. We hypothesize permit acquisitions have historically occurred that would incur a vocational skills cost in our model. We are interested in the number and magnitude of historical actions and how they compare to the distribution in the optimized solutions.
3. We hypothesize all combinations of revenue and variance increase and decrease occur in the historical data, including communities experiencing a year-to-year decrease in expected revenue and increase in variability.[1] We are interested in the distribution of expected revenue and variance changes in the historical data and how they compare to the optimized solutions.

3.1 Data and Instance Generation

Historical data for 2000–2023 was obtained from the Alaska Commercial Fisheries Entry Commission [3,4]. While over 450 permit types were issued over these

[1] For example, decreased community level income and increased variability could occur if an individual chose to sell a permit, decreasing revenue, while a lower cost permit was purchased that increased variance.

years, many have low levels of commercial activity or are not exclusively for commercial purposes, such as trials of new permit types and educational permits. To focus on active commercial fisheries, each year we included permits where at least 25% of all permit owners and a minimum of 10 permit holders reported revenue each year over the past five years. This reduced the average number of permit types to 52 per year while retaining an average of 96% of revenue.

Permit sale prices and expected revenue were obtained from CFEC Basic Information Tables (BIT), which provide aggregated financial information at the permit-level [3]. To maintain confidentiality, values are not reported if fewer than five permit owners report revenue for a given year. In the rare cases where values were needed for a permit with anonymized data, we imputed the missing values with the average revenue across all reported years. The raw data provide us with a time series to calculate expected revenue and variance.

As recent performance is of more importance to fisherman than long-term trends, we used exponentially weighted calculations for mean revenue and variance. These calculations require a single parameter, α, for kernel width which determines the amount of weight placed at the start of the time series. For $\alpha = 0$, we recover the mean and variance of the time series while for $\alpha = 1$ the expected revenue becomes that of the previous year with no variance. Consulting with fisheries experts, we chose $\alpha = 0.25$ to place $\sim$95% of weight on the previous five years.

We define communities as any geographic location with at least three registered permits in a given year, resulting in an average of 170 communities per year. Initial community portfolios were determined using the annual records in the CFEC's permit registration database [4]. Spatial layers were used to determine centroids for each community and fisheries region, and travel distances were calculated as the distance between centroids. Finally, quantitative transition difficulty scores in $\{0, \ldots, 10\}$ were assigned to pairs of permits using fishery expert ratings of the difficulty of transitioning between each component of the species, gear, and location attributes that define a fishing permit type.

In addition to creating instances based on all communities, we defined 14 small instances of 4–45 communities based on fishing regions and intermediate instances with 6–88 communities based on four larger geographic regions (see Fig. 1). Running the model without the vocational skills budget constraint, we found the optimal transition cost to be $\sim 10^6$. To understand the impact of different orders of magnitude of vocational skills budget, we tested vocational training budgets of 0 and 10^i for $i = 2, \ldots, 6$. We ran initial feasibility tests on the 14 small instances using the 2023 data and a vocational skills budget of 1000. For the larger regions, we tested instances for the five regions for six vocational skills budgets across 24 years, resulting in 720 unique instances.

3.2 Experimental Configuration

Both the MIQP and linearized BIP were implemented in Gurobi using gurobipy 11.0.3. Instances were given a 5 min time limit and an optimality tolerance of 10^{-3} on an Apple M1 processor with 8 GB or RAM running on macOS Sonoma

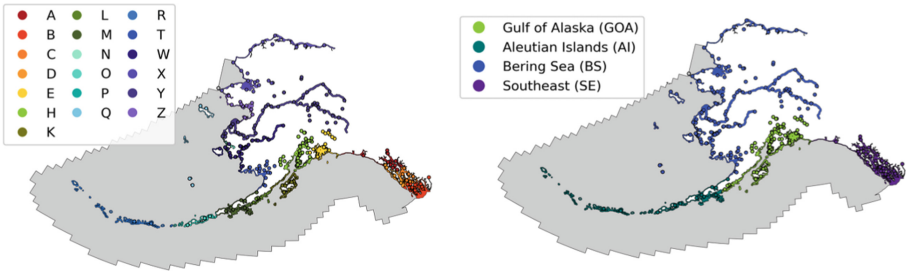


Fig. 1. Spatial visualization of instances, where colored polygons indicate regions and colored dots indicate communities associated with that instance. Left: Small spatial instances where communities are grouped based on fishing permit regions, where instances are defined by a set of communities and their initial permits. Right: Intermediate spatial instances based on geographic regions of Alaska.

14.5. Additional hour long runs were performed for vocational skills budgets of 0, 10^3, and 10^6 for the years of 2003, 2013, and 2023. For these runs, we used a maximum community-level travel distance increase of 50%, $\eta = 0.5$, maximum community-level revenue decrease of 15%, $\phi = 0.15$, and an exponential scaling with base four for the transition scaling function and relaxed the financing constraint, $\phi = \infty$. Additionally, we tested the sensitivity of the results to the maximum-community level distance and revenue change parameters (η and ϕ) and the vocational scaling function.

3.3 Results

On the small instances, the MIQP significantly outperformed the MBLP. The MIQP was able to solve all instances to optimality in under a minute, while the MBLP was only able to solve 6/14 in the 5 min time limit (see Table 2). On the remaining instances, the MBLP had multiple optimality gaps over 10% and was not able to instantiate the largest instance due to exceeding the 8GB of RAM allotted. Due to the poor performance of the MBLP, only the MIQP was tested on the larger instances.

For the larger instances, the MIQP was able to solve 50% of instances in the 5 min time limit, increasing to 64% with a 30 min time limit (see Table 3). For the five minute time limit, the average optimality gap ranged from 0.7 to 5.7% by region, generally increasing with instance size. As expected, the optimality gap also increased with the vocational training budget, increasing from 0.0% to 7.0%. Performance also varied by year, with instances increasing in difficulty in the 2010s from an average gap of around 3% up to 10%. Increasing to a 30 min limit, nearly two-thirds of instances were solved to optimality, including over half of the statewide instances, and reduced the average optimality gap by over 60% (see Table 3).

The optimization was able to reduce the average coefficient of variation by 34% without any vocational training investment (see Fig. 2). While there was

Table 2. Instance parameters and comparison of the MIQP and MBLP on the small spatial instances, where bold values indicate the best value when comparing the models. The MIQP was able to solve all instances to optimality in under a minute. In comparison, the MBLP was only able to solve 6/14 instances to optimality, with gaps as large as 19% on the remaining instances. Further, the MBLP was unable to instantiate the instance with the largest number of ternary variables due to reaching the memory limit. $|C|$: number of communities, $|P|$: number of permit types, $|P'|$: number of individual permits.

Instance	$\|C\|$	$\|P\|$	$\|P'\|$	$\|C\|\|P\|\|P\|$	Runtime (min)		Optimality Gap (%)	
					MIQP	MBLP	MIQP	MBLP
W	4	5	25	2.5×10^3	**0.00**	**0.00**	**0.0**	**0.0**
L	4	4	46	8.5×10^3	**0.00**	**0.00**	**0.0**	**0.0**
X	8	7	191	2.9×10^5	**0.00**	0.03	**0.0**	**0.0**
P	21	7	128	3.4×10^5	**0.00**	5.00	**0.0**	**0.0**
Z	7	7	226	3.5×10^5	**0.00**	5.00	**0.0**	**0.0**
M	9	5	237	5.0×10^5	**0.00**	4.09	**0.0**	**0.0**
T	8	7	636	3.2×10^6	**0.00**	5.00	**0.0**	1.1
E	26	5	485	6.1×10^6	**0.00**	5.00	**0.0**	10.8
K	28	7	698	1.4×10^7	**0.00**	5.00	**0.0**	19.0
B	26	7	806	1.7×10^7	**0.01**	5.00	**0.0**	7.2
D	33	4	858	2.4×10^7	**0.00**	5.00	**0.0**	2.9
A	35	8	840	2.5×10^7	**0.02**	5.00	**0.0**	10.0
C	32	5	1053	3.4×10^8	**0.65**	5.00	**0.0**	2.0
H	45	26	2762	3.5×10^8	**0.00**	–	**0.0**	–

Table 3. Instance parameters and performance of the MIQP on the large instances with runtimes of 5 (total of 720 instances) and 30 min (total of 45 instances). Gurobi was able to solve half of all instances in 5 min, increasing to nearly two-thirds in 30 min. For unsolved instances, gaps ranged from 0.6–4.0% increasing with instance size and vocational training budget. $|C|$: number of communities, $|P|$: number of permit types, $|P'|$: number of individual permits.

Region	$\|C\|$	$\|P\|$	$\|P'\|$	Solved Ratio		Runtime Ratio		Avg. Gap (%)	
				5	30	5	30	5	30
AI	6	9	139	1.00	1.00	0.00	0.00	0.0	0.0
BS	88	33	3597	0.54	0.67	0.44	0.38	1.0	0.6
GOA	47	45	3673	0.42	0.56	0.68	0.47	3.3	2.3
SE	28	41	4946	0.33	0.44	0.76	0.77	0.7	0.4
ALL	170	52	12378	0.22	0.56	0.92	0.90	5.7	4.0

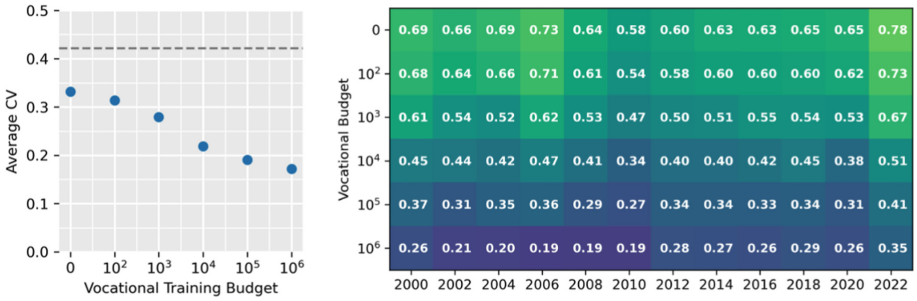


Fig. 2. Analysis of variance reduction. Left: Average community-level coefficient of variation (CV) in the optimized solutions for 2023 (blue points) compared to the average CV before optimization (gray line). Right: Matrix visualizing the proportion of the initial average CV of the optimized solutions by transition budget for even years from 2000–2023. Across years, with virtually no vocational skill investment the average CV can be reduced by over 30%. With moderate or intensive vocational skills interventions the average CV can be reduced by an average of 50% or 75% respectively. (Color figure online)

some variability across years, 19/24 years experienced at least a 30% reduction and all years achieved at least a 22% improvement. For the highest vocational skills budget, the average coefficient of variation was reduced by more than 75% of its initial value, with a 35–60% improvement for intermediate vocational training budgets.

We compared the historical data and the optimized solutions to test our hypotheses and modeling assumptions. First, we analyzed community-level revenue and variance changes to understand the distribution of impacts on communities. For the historical data, we considered year-to-year changes, while for the optimization we considered the difference between the initial and optimized solutions. Plotting the ratio of expected revenue versus the standard deviation in revenue by community, we can visualize the distribution of community outcomes (see Fig. 3). Averaging from 2001–2023, the optimized solutions increased the number of communities experiencing a variance decrease from 41% to 83–90%, increasing with the vocational skills budget, and increase the average magnitude of decrease from under 10% to 45–55%. Further, the optimization increased the number of communities experiencing the best outcome of increased expected revenue and decreased expected variability from 6% to 11–13% and decreased the proportion of communities experiencing the worst outcome of decreased expected revenue and increased variability from 3% to 1%. However, while historically the most wealthy communities remained near the origin in the optimized solutions they tend to end up in the first quadrant, experiencing an increase in revenue and variability.

Second, we analyzed the actions taken by each intervention. For the vocational skills budget, we can assign transition costs to historic data based on year-to-year permits transfers, determining a budget for historical actions. On

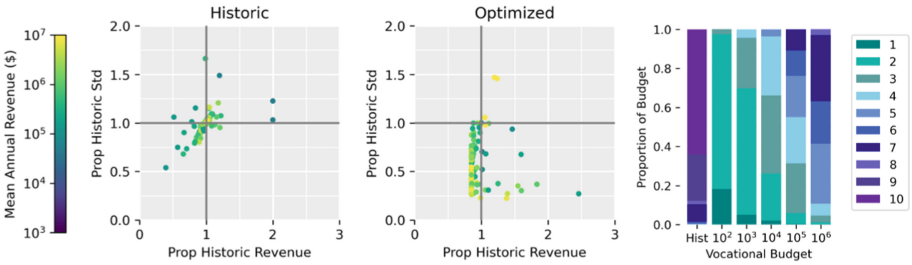


Fig. 3. Analysis of the optimized solutions. Left and center: Change in expected revenue and standard deviation in revenue by community from the historical data from 2022 to 2023 (left) and from the 2023 historical data to the optimized solution for a transition budget of 1,000 (center). The optimized solution results in more communities experiencing a variance decrease and larger magnitude decreases. Additionally, more communities experience higher revenue and lower variance and fewer experience lower revenue and higher variance. Right: Assigning vocational skills costs to historical transitions, we can compare the vocational skills budget that would have been required for historical actions to the optimization results. Historically, few high cost actions occurred but they dominated the vocational training budget. In contrast, the optimized solutions consist almost entirely of low (level 1–3) or mid-cost actions (level 4–6) which dominate both the number and cost of actions.

average, the transition budget needed for historical year-to-year changes was 37,000 units but only consisted of 41 actions, i.e., cases where permits were acquired at a transition cost. In contrast, the optimized solutions with a transition budget of 100 units resulted in an average of 12 actions and a transition budget of 10,000 units resulted in an average of 344 actions. While historically, only an average of 6 actions occurred per year of transition levels 5–10, due to the small number of total actions and exponential scaling of the transition scaling function, they dominate the transition budget (see Fig. 3). For the optimized solutions, at all budget levels over 70% of actions are level 1–3 (see Fig. 3). However, the cost is slowly dominated by the small proportion of higher level transitions. For permit financing, we looked at the magnitude and number of communities that would require some level of financing. While the total value of permits does not change with the optimization, we see community level changes to portfolio values. Communities with decreased portfolio value would obtain additional one-time revenue through the sale of their permits, while communities with increased portfolio value would require financing to help purchase new permits. In the optimization results, the total value of permits requiring financing ranged from $92–106 million, increasing with vocational training budget, with 28–40% of communities requiring some level of financing.

Finally, we tested the sensitivity of the results to the community-level revenue change and distance change constraints and the vocational skills scaling function. Fully relaxing the distance constraint only resulted in benefits at the highest vocational training budgets (see Fig. 4). Increasing or decreasing the maximum

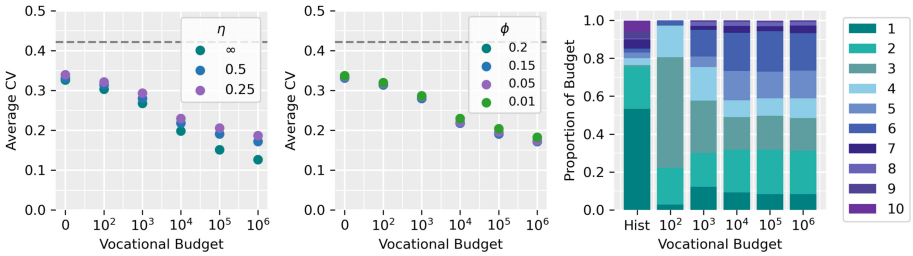


Fig. 4. Tests of model sensitivity. Results are shown for 2023, where parameters for main runs are shown in blue and the grey line shows the pre-optimization objective. Left: The model is not highly sensitive to maximum increase in travel distance (η), where fully relaxing the constraint only results in moderate improvements for the largest transition budgets. Center: The model is insensitive to the maximum community-level revenue decrease (ϕ), where objectives are nearly identical for the constraint varying from 20% to 1%. Right: Distribution of actions in the vocational budget with a quadratic scaling function. Changing the vocational scaling function does not change the distribution of actions for extreme budgets, but results in more mid-level (level 4–6) actions for intermediate budgets. (Color figure online)

community-level revenue change up to 20% or as low as 1% only resulted in very small changes to the objective, often within the optimality gap (see Fig. 4). Finally, changing the vocational skills scaling function does not change the objective value for the extreme budgets, it does change the distribution of actions. Namely, due to the slower scaling of the quadratic function, we see more intermediate level actions (level 4–6), however we continue to see very few of the highest cost actions (levels 8–10) (see Fig. 4).

4 Discussion

The poor performance and high memory demands of the MBLP arise from the ternary variables and corresponding constraints for every individual permit-permit-community combination. This leads to millions of variables and constraints even in small instances due to hundreds of individual permits and tens of communities. In contrast, the MIQP could solve some statewide instances with over 10,000 permits and 170 communities to optimality.

The MIQP was able to solve half the larger instances in 5 min and nearly two-thirds of instances in 30 min, with an average remaining gap of 1.5%. In practice, a runtime of 30 min would not be an issue, as a limited number of configurations would be tested to inform decision makers for periods of years. Further, gaps of a few percent would be tolerable, as solutions would not be implemented exactly but instead would be used by decision makers to understand the types and magnitudes of changes that could be most effective in lowering community-level variance and the potential magnitude of benefit from combinations of permit financing and vocational training interventions. The results are promising for

this application, showing potential community-level reduction of coefficient of variation of over 30% with permit financing interventions alone and improvements over 75% with combined financing and vocational skills investments. The consistent results across reference years suggests that the model is robust to initial conditions within the historical range and that Gurobi is able to provide good solutions. Further, the sensitivity analyses suggests the results of the model are relatively robust to changes in constraint thresholds and the vocational scaling function.

The analysis also supports our assumptions and hypotheses. The vocational skills budget analysis found that high transition cost actions occur rarely in practice and few transitions occur historically where there has not been concerted investment in vocational skills training. It makes sense that there is a background level of actions occurring that would incur a transition cost in our model, as there is limited capacity in the existing system to overcome these costs. For example, permit owners can move between communities or an individual working for an existing business can eventually acquire enough skills and capital to start their own business. However, the results of the optimization suggest that significant benefits could be achieved by facilitating additional vocational training. Key differences between the historical and optimized vocational skills budgets are the number and kinds of actions. Historically, there were very few cases in which permits were acquired at a vocational skill cost, though transitions occurred at all cost levels. In contrast, the optimized solutions resulted in significantly more actions but at lower budget levels. This suggests that significant diversification benefits can be achieved without the need for drastic changes in the fisheries that communities participate in.

The community-level analysis found that historically the distribution of year-to-year changes includes all outcomes of changes in revenue and variance, with the majority of communities experiencing a small improvement to one at the cost of the other. In addition to lowering income variation overall, the optimized solutions resulted in significantly more communities experiencing a variance reduction, with larger magnitude reductions, more communities experiencing the best outcome, and fewer communities experiencing the worst outcome. This is an appealing result for potential policy, as the vast majority of communities have the potential to benefit from diversification. While the distribution for the wealthiest communities shifted to experiencing higher expected revenue and income variability, this may be tolerated or even perceived as a benefit, as the larger annual revenue and diversification of income outside of fisheries not captured in this analysis may make the wealthiest communities more tolerant of income variability for the benefit of higher expected revenue.

As the first analysis at the system-level, we show there is not only potential for benefit to individuals or communities from diversification, but potential significant benefits for the entire system. The analysis suggests that over a 30% reduction in average income variability is possible through permit financing alone, with improved outcomes realized across the distribution of communities. While financing is an important management lever to facilitate fisheries adap-

tation, fishers' access to financing is limited because conventional commercial banks typically do not provide loans for fishing permits, gear, or boats. Available financing in Alaska is insufficient and restricted to niche lending programs including the public-private Alaska Commercial Fishing and Agriculture Bank and small community-based programs such as the non-profit Alaska Local Fish Fund program [1,2]. It is important to note that the cost of this intervention could be significantly smaller than the $92–106 community-level increase in portfolio values in the optimized solutions, as the interventions could include offering loans at reduced interest rates or with reduced collateral. Further, interventions of the magnitude of tens of millions are not unprecedented. For example, the US Department of Commerce allocated $40 million in financial disaster relief in response to the 2024 snow crab fishery crisis [13]. To date, vocational training has not commonly been viewed as a key policy lever in natural resource sustainability science, however the results of the optimization show that combining vocational training with permit financing can more than double the potential reduction in income variability. While the instances presented in this paper fully relax the permit financing constraint, the optimization approach presented here provides a flexible framework for decision makers to quantify cost-benefit trade-offs in financing and vocational training investments.

The results of the optimization motivate several directions for future work. First, the amount of permit financing and vocational skills training proposed by the optimized solutions suggests further research into the potential cost and scalability of these interventions. This information could be incorporated into future iterations of the optimization as constraints or allowing for a single financial budget to be allocated between permit financing and vocational skills interventions. However, the magnitude of the interventions in the optimized solutions also suggest that changes would realistically be implemented over time, motivating an alternative formulation with yearly intervention budgets. Further, ecosystems in Alaska have already began to undergo significant shifts due to climate change [7,20]. While these changes are captured retrospectively through the revenue and variance calculations, as we optimize over longer time periods, incorporating climate trends will be key to accurately characterizing income variability.

5 Conclusion

High income variability from fisheries has many negative impacts on Alaskan communities. Previous work suggests that community-level income variability can be reduced through diversification in fisheries participation, though prior work had not considered the effects at the system level. In collaboration with fisheries experts, we formulate a new constrained resource allocation problem to determine how to optimally allocate permits to minimize the average community-level coefficient of variation and model financing and vocational skills interventions. Testing integer programming methods on multiple regions over 20 years, we find that the model can solve instances up to the state-level to optimality, with over 170 communities and 10,000 permits.

The optimization reduced the average community-level coefficient of variation by over 30% through permit financing alone and up to 75% with both financing and vocational skills training interventions. Comparing to historical changes, the optimization not only resulted in better average values but improved the proportion of communities experiencing a variation reduction from 41% to up to 90% with both interventions, and increased the magnitude of the reduction from 10% to up to 55%. However, the level of permit financing and vocational training required for the optimized solutions significantly exceeds the capacity of the current system, motivating future work into optimization overtime and incorporating climate trends to accurately capture future income variability.

Acknowledgments. We would like to thank Adrianna Muir from The Nature Conservancy, and Chris Siddon from the Alaska Department of Fish and Game, and Brad Harris from Alaska Pacific University for their contributions in the development of the project. This work was funded through a joint Cornell Atkinson Center for Sustainability - The Nature Conservancy funding Innovation for Impact Grant.

Disclosure of Interests. The authors have no competing interests to declare.

References

1. Alaska Commercial Fishing and Agriculture Bank. Anchoarage, AK. https://www.cfabalaska.com/
2. Alaska Local Fish Fund. Sitka, AK. https://localfishfund.org/
3. Alaska Commercial Fisheries Entry Commission: Fishery statistics (2024). https://www.cfec.state.ak.us/fishery_statistics/permits.html
4. Alaska Commercial Fisheries Entry Commission: Public permit database (2024). https://www.cfec.state.ak.us/plook/#permits
5. Béné, C., et al.: Feeding 9 billion by 2050-putting fish back on the menu. Food Secur. **7**, 261–274 (2015)
6. Bretthauer, K.M., Shetty, B.: The nonlinear resource allocation problem. Oper. Res. **43**(4), 670–683 (1995)
7. C, S., J, H.: An overview of the economic status of Alaska's limited entry permit types. Technical report, State of Alaska Commercial Fisheries Entry Commission (2022)
8. Cody, R., Liddel, M., Pereira, V.: Fisheries of the United States 2022. Technical report, NOAA Fisheries Office of Science and Technology (2024).d https://www.fisheries.noaa.gov/resource/document/fisheries-united-states-2022
9. Du, D., Pardalos, P.M.: Handbook of Combinatorial Optimization, vol. 4. Springer (1998)
10. FAO: The state of the world's fisheries and aquaculture. Technical report, Food and Agriculture Organization of the United Nations (2020). https://openknowledge.fao.org/items/b752285b-b2ac-4983-92a9-fdb24e92312b
11. Gortzen, S., Schmeink, A.: Optimality of dual methods for discrete multiuser multicarrier resource allocation problems. IEEE Trans. Wireless Commun. **11**(10), 3810–3817 (2012)
12. Herrmann, M., Greenberg, J., Hamel, C., Geier, H.: Extending federal crop insurance programs to commercial fisheries: the case of Bristol Bay, Alaska, sockeye salmon. North Am. J. Fish. Manag. **24**(2), 352–366 (2004)

13. James, M.: U.S. department of commerce allocates $39.5 million in funding for Alaska fishery disaster (2024). https://www.noaa.gov/news-release/us-department-of-commerce-allocates-395-million-in-funding-for-alaska-fishery-disaster
14. Kasperski, S.: Aslaska seafood snapshot. Technical report, NOAA Fisheries Resource Ecology and Fisheries Management (2024). https://www.fisheries.noaa.gov/resource/outreach-materials/alaska-seafood-snapshot
15. Kasperski, S., Holland, D.S.: Income diversification and risk for fishermen. Proc. Natl. Acad. Sci. **110**(6), 2076–2081 (2013)
16. Kolm, P.N., Tütüncü, R., Fabozzi, F.J.: 60 years of portfolio optimization: practical challenges and current trends. Eur. J. Oper. Res. **234**(2), 356–371 (2014)
17. Lopetegui, I., del Valle, I.: An efficient portfolio approach towards ecosystem-based fisheries governance in EU. Fish. Res. **254**, 106427 (2022)
18. Maggie, N.: St. Paul government declares emergency in attempt to get ahead of looming crab crash (2022). https://alaskapublic.org/2022/11/09/st-paul-government-declares-emergency-in-attempt-to-get-ahead-of-looming-crab-crash/
19. Rezvani, M., Akbari, M.K., Javadi, B.: Resource allocation in cloud computing environments based on integer linear programming. Comput. J. **58**(2), 300–314 (2015)
20. Rooper, C.N., et al.: Predicted shifts of groundfish distribution in the Eastern Bering Sea under climate change, with implications for fish populations and fisheries management. ICES J. Mar. Sci. **78**(1), 220–234 (2021)
21. Rudek, R., Heppner, I.: Efficient algorithms for discrete resource allocation problems under degressively proportional constraints. Expert Syst. Appl. **149**, 113293 (2020)
22. Sethi, S.A.: Risk management for fisheries. Fish Fish. **11**(4), 341–365 (2010)
23. Sethi, S.A., Reimer, M., Knapp, G.: Alaskan fishing community revenues and the stabilizing role of fishing portfolios. Mar. Policy **48**, 134–141 (2014)
24. Teh, L.C., Sumaila, U.R.: Contribution of marine fisheries to worldwide employment. Fish Fish. **14**(1), 77–88 (2013)
25. Townsend, H., Link, J.S., DePiper, G., Brewster, L.R., Cadrin, S.X., Edwards, F.: Multispecies portfolios of us marine fisheries: ecosystem-based fisheries management reduces economic risk. Fisheries (2024)

Analyzing the Numerical Correctness of Branch-and-Bound Decisions for Mixed-Integer Programming

Alexander Hoen[1(✉)] and Ambros Gleixner[1,2]

[1] Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany
hoen@zib.de
[2] HTW Berlin, 10313 Berlin, Germany
gleixner@htw-berlin.de

Abstract. Most state-of-the-art branch-and-bound solvers for mixed-integer linear programming rely on limited-precision floating-point arithmetic and use numerical tolerances when reasoning about feasibility and optimality during their search. While the practical success of floating-point MIP solvers bears witness to their overall numerical robustness, it is well-known that numerically challenging input can lead them to produce incorrect results. Even when their final answer is correct, one critical question remains: Were the individual decisions taken during branch-and-bound justified, i.e., can they be verified in exact arithmetic? In this paper, we attempt a first such *a posteriori* analysis of a pure LP-based branch-and-bound solver by checking all intermediate decisions critical to the correctness of the result: accepting solutions as integer feasible, declaring the LP relaxation infeasible, and pruning subtrees as suboptimal. Our computational study in the academic MIP solver SCIP confirms the expectation that in the overwhelming majority of cases, all decisions are correct. When errors do occur on numerically challenging instances, they typically affect only a small, typically single-digit, amount of leaf nodes that would require further processing.

Keywords: Mixed integer programming · branch and bound · exact computation

1 Introduction

Mixed Integer Programming (MIP) solvers have evolved into highly sophisticated software capable of solving a growing number of large-scale and complex problems efficiently. They typically implement a branch-and-bound scheme based on a linear programming (LP) relaxation. For performance reasons, nearly all solvers utilize floating-point arithmetic in order to speed up computations and control memory requirements for representing values of rational numbers. These computational advantages of floating-point arithmetic, however, also imply limitations in precision: not all rational numbers can be represented exactly, leading to tiny round-off errors that can accumulate in magnitude over the course of

G. Tack (Ed.): CPAIOR 2025, LNCS 15763, pp. 35–50, 2025.
https://doi.org/10.1007/978-3-031-95976-9_3

the algorithm. Therefore, solvers introduce tolerances for feasibility and a zero tolerance for deciding equality between two numbers. Additionally, by default, commercial solvers operate with relative gaps between 0.01% and 0.0001%.

It is well documented that sometimes accumulated rounding errors can cause solvers to incorrectly declare infeasibility, feasibility, or optimality for suboptimal solutions [6,15,19,22]. Further, the MIPLIB 2017 instance collection labels 193 instances with problematic numerics with a tag "numerics", partly because different solvers reported inconsistent results during tests for the compilation process [12]. As a result, for applications that require absolute certainty, such as chip design verification [1], combinatorial auctions [24], or computational proofs in experimental mathematics [10], the level of trust provided by floating-point solvers is often not sufficient.

Nevertheless, it is widely accepted that the way that mature floating-point MIP solvers handle numerics is overall very robust. This judgment is not only due to the fact that for the vast majority of industrial MIP applications, it may be acceptable to work with solutions that are slightly suboptimal or slightly infeasible. The wide-spread use of floating-point MIP solvers and the fact that the general methodology of handling round-off errors in MIP solvers has not changed substantially over many years of development rather suggests that in the vast majority of cases, their answers may indeed be correct in an exact sense.

However, to the best of our knowledge, the literature holds no record of a systematic computational investigation of this fundamental methodological question: To what extent do approximate floating-point solvers provide numerically exact answers? More specifically, to what extent is the *reasoning* of a floating-point LP-based branch-and-bound solver, which is composed of a large number of critical decisions during the search, correct in a numerically exact sense? While studies exist that compare the final result of numeric and exact solvers, e.g., in [7], we are not aware of any analysis that looks deeper at the branch-and-bound process. This is particularly interesting for the large number of cases where the final answer of a floating-point solver is correct: Is this due to the fact that all individual decisions were correct, or because all incorrect decisions did not affect or compromise the final result?

In this paper, we try to give empirical answers to these questions by performing a first such *a posteriori* analysis of a pure LP-based branch-and-bound process. To this end, we inspect all leaves of the branch-and-bound tree and check if the leaf is pruned correctly because (a) its LP relaxation is infeasible, (b) its dual bound proves that the leaf does not contain improving solutions, or (c) the LP solution is primal feasible. To test the correctness of each decision, we apply a hierarchy of techniques from the literature such as safe bounding by directed rounding [17], rational reconstruction by continued fraction approximations [18,20,23,25], exact LU factorization of simplex bases, and, if necessary, exact LP solving [13,14]. We base our experiments on the open-source MIP solver SCIP [3] and use two curated test sets from the literature with and without numerical challenges [7]. Our code base is publicly available.[1]

[1] https://github.com/alexhoen/bnbanalyzer.

The paper is organized as follows. In Sect. 2, we briefly introduce the general concept of LP-based branch and bound, provide a classification of numerical errors, and survey the techniques to check the correctness of floating-point decisions. In Sect. 3, we present the results of our experiments and analyze the different errors encountered in the floating-point solver. In Sect. 4, we summarize our results and give an outlook on their implications for future research.

2 Numerical Errors in LP-Based Branch and Bound

A *mixed integer program* (MIP) is given in the form

$$\begin{aligned} \min\ & c^T x \\ \text{s.t.}\ & Ax \geq b, \\ & \ell \leq x \leq u, \\ & x_i \in \mathbb{Z} \ \text{ for all } i \in \mathcal{I}, \end{aligned}$$

where we assume rational input data $A \in \mathbb{Q}^{m \times n}$, $c \in \mathbb{Q}^n$, $u, \ell \in (\mathbb{Q} \cup \{\pm\infty\})^n$, and $b \in \mathbb{Q}^m$. The index set $\mathcal{I} \subseteq \{1, \ldots, n\}$ defines which decision variables are required to be integer.

MIPs are typically solved by variants of *LP-based branch and bound*, which dates back to [16] and still forms the backbone of today's most competitive MIP solvers. In Sect. 2.1, we describe briefly the procedure of an LP-based branch-and-bound algorithm. For a more detailed description we refer to [1]. In Sect. 2.3, we give an overview of the techniques we use to obtain an exact evaluation of a node that was solved in floating-point arithmetic. In Sect. 2.2, we categorize how numerics can impact the critical branch-and-bound decisions in a floating-point solver and how incorrect decisions affect the overall result.

2.1 Basics of the LP-Based Branch-and-Bound Algorithm

The first step in an LP-based branch-and-bound algorithm is to *relax* the problem by dropping all integrality constraints. The resulting LP relaxation can be solved by an LP solver. If the LP solution is *integer feasible*, i.e., all integer variables have an integral solution value, the problem is solved. If the LP relaxation is detected to be infeasible, then also the original MIP is infeasible. Otherwise, the algorithm *branches*: It chooses an integer variable x_i with a fractional value $\hat{x}_i$ in the solution of the LP relaxed and creates two new sub-problems by taking the original problem and adding the constraint $x_i \geq \lceil \hat{x}_i \rceil$ and $x_i \leq \lfloor \hat{x}_i \rfloor$, respectively. This ensures that the optimal solution is preserved in at least one of the two sub-problems and the current LP solution is no longer feasible in both sub-problems. This process of LP solving and branching is iterated recursively.

If during this process an integer feasible solution is found its objective value can be used to update the *primal bound*, which is the objective value of the best known solution encountered so far. This primal bound can be used to *prune*, i.e., remove from further consideration all open sub-problems for which the *dual*

bound, i.e., the objective value of the sub-problem's LP relaxation, is greater than or equal to the primal bound.

This recursive procedure produces a search tree with sub-problems as nodes. A node is called *leaf* if it is not further branched on because it is (a) LP infeasible, (b) it was pruned, or (c) the LP solution is integer feasible. Figure 1 presents an example of a branch-and-bound tree for a minimization problem. The solution of the relaxed LP at the node is displayed slightly above and to the right of the node, while its objective value is positioned on the left side. The leaf N_4 is integer feasible and appears to be also optimal. The leaf N_3 is LP infeasible and the node N_1 is cut off since branching on N_1 does not improve the solution of N_4 anymore.

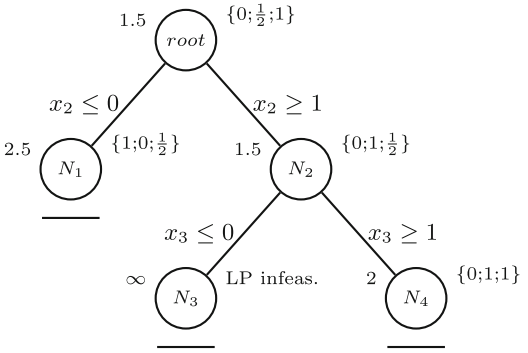


Fig. 1. An example for a branch-and-bound tree on a minimization problem.

Note that although the heuristic decision on which fractional variable to branch may be affected by numerical calculations, any sequence of branching decisions yields a correct branch-and-bound process. The same holds for the heuristic decision in which order to process open sub-problems: any order yields a correct branch-and-bound algorithm. By contrast, the decision of whether to remove a sub-problem from consideration is both affected by numerics and can critically affect the correctness of the final result. In the next section, we focus on these critical decisions.

2.2 Types of Numerical Errors in LP-Based Branch and Bound

In our analysis, we distinguish the following types of incorrect decision that can be taken by a floating-point solver due to numerical errors. All of these errors occur at a leaf of the branch-and-bound tree.

Solution Errors. By design, a floating-point solver may produce approximate solutions with slight violations of integrality and constraint feasibility. A simple post-processing step is to first round the value of all integer variables to the nearest integer and second, if continuous variables are present, to compute rational

values for these such that all constraints are satisfied exactly. If this is certifiably not possible, i.e., if the floating-point solver accepted an assignment for the integer variables as feasible that cannot be completed to an exactly feasible solution, we refer to this as a *solution error*.

We distinguish two types of solution errors. If the LP relaxation at this node is infeasible in exact arithmetic, then no solution was discarded by removing the current leaf. We refer to this error as a *weak solution error*. Otherwise, the primal solution of the LP relaxation does not satisfy integer feasibility, but the sub-problem below the current leaf node may still contain improving solutions. In this case, only a lower bound can be derived and we refer to this situation as a *strong solution error*.

Bound Errors. If a floating-point solver uses a lower bound to prune a node, this decision can be verified *a posteriori* by computing solving the LP relaxation exactly or computing a safe dual bound that is valid in exact arithmetic, by techniques described in Sect. 2.3. If such a safe bound is greater than or equal to the current primal bound, then the decision to prune the node was correct. Otherwise, if the exact objective value of the LP relaxation is below the primal bound, we label this decision as a *bound error*. Again, we distinguish two cases. If later during the solving process an exactly feasible solution is found that improves the primal bound and in hindsight justifies pruning the leaf, we call the error a *weak bound error*, otherwise a *strong bound error*.

Gap Errors. For the third type of error, consider the situation that a floating-point solver declares a node to be integer feasible and produces an approximate solution that can be converted to an exactly feasible solution after rounding the integer assignment. Although we record no solution error here, the decision to terminate the search below this node may still be incorrect if the objective value of the converted, exact solution does not match the dual bound derived by solving the LP relaxation exactly. We refer to this error as a *gap error*, and as above we distinguish between *weak gap errors* that are justified in hindsight and *strong gap errors*.

Infeasibility Errors. Finally, if the floating-point solver declares a leaf infeasible but there exists a solution to the LP relaxation that is exactly feasible we refer to this situation as an *infeasibility error*.
Even when some of the errors listed above occur, the overall result of the floating-point solver may still be correct. Specifically, weak bound and weak gap errors do not compromise the correctness of the final result since the corresponding decisions are justified in hindsight. We still record these errors, because we are interested not only in the correctness of the final result but in the correctness of the floating-point solver's reasoning that led to the result.

Also in the presence of strong bound or strong gap errors, the solution or at least its objective value z^* returned by the floating-point solver may still be optimal. Because our analysis produces a safe dual bound at all occurrences of errors outlined above, we can then take the minimum of all these dual bounds

and derive a global lower bound $\hat{z} < z^*$ for the true optimal objective value. This certifies that the true optimal objective value must lie in the interval $[\hat{z}, z^*]$. If infeasibility errors occur, i.e., if a leaf is removed although the LP relaxation is feasible in exact arithmetic, we can use the exact objective value of the LP relaxation as a valid dual bound.

Finally, note that in the presence of solution errors or gap errors, the floating-point solver will work with an incorrect, usually too optimistic primal bound during part of the search. When checking for bound errors, however, we always take the perspective of the floating-point solver, i.e., we treat the primal bound registered in the floating-point solver at that time as valid, and check whether the pruning decision is justified with respect to this primal bound. This helps us to separate more clearly and quantify more precisely the different types of incorrect reasoning in our computational analysis later.

2.3 Checking Numerical Correctness of Floating-Point Decisions

A straightforward way to evaluate whether and which of the errors described above occurs at a leaf node is to solve the LP relaxation once again with an exact LP solver and compare the results. For most but small instances, this approach is prohibitively slow. Hence, we use a hierarchy of methods that may be able to verify or refute the correctness of leaf decisions faster, described in the following.

Safe Bounding. Consider the dual of the LP relaxation,

$$\max \left\{ b^T y + \ell^T r^+ - u^T r^- : A^T y + r^+ - r^- = c,\, y, r^+, r^- \geq 0 \right\},$$

and assume we have an approximate dual solution $(\hat{y}, \hat{r}^+, \hat{r}^-)$ with dual residual error

$$\hat{\varepsilon} := c - A^T \hat{y} - \hat{r}^+ + \hat{r}^- \in \mathbb{Q}^n. \tag{1}$$

If we compute, in exact arithmetic,

$$r_i^+ := \hat{r}_i^+ + \max\{\hat{\varepsilon}_i, 0\} \text{ and } r_i^- := \hat{r}_i^- - \min\{\hat{\varepsilon}_i, 0\} \tag{2}$$

for all variables $i \in \{1, \ldots, n\}$, then $\{\hat{y}, r^+, r^-\}$ is an exactly feasible dual solution with objective value $b^T \hat{y} + \ell^T r^+ - u^T r^-$. This modification thus ensures dual feasibility at the cost of reducing the objective value by $\ell^T(\hat{r}^+ - r^+) + u^T(r^- - \hat{r}^-)$. The quality of the resulting safe dual bound is impacted by the variable bounds u, ℓ, which, ideally, should be as tight as possible to minimize the amount by which the approximate dual bound is weakened. If infinite variable bounds are used during the computation, the resulting safe bound becomes negative infinity. This technique is first explained in [17] in a version using interval arithmetic and discussed in more detail in [21]. Safe bounding can be applied analogously to validate a Farkas proof for infeasible LPs.

Rational Reconstruction. In the hope of recovering an exact rational solution value from an approximate floating-point value, we use continued fractions approximation, which can be computed by the extended Euclidean algorithm. This way, we can round floating-point values to a nearby rational number with a limited denominator. This method is applied to both primal and dual LP solutions, as well as to Farkas proofs. Originally, this technique is described and improved in [18,20,23,25]. We rely on an implementation in the exact LP solver SoPlex [13,14].

Factorization. If a simplex basis is available for the LP relaxation solution, we perform an exact rational LU factorization in order to compute the exact rational solution associated to the basis reached by the floating-point LP solver. Again, we use the implementation in SoPlex [13,14]. This step is only skipped for LP infeasible nodes since no basis can be obtained from the LP solver.

Exact LP Solving. The last and most expensive option to derive the most accurate lower bound on the objective is to call a round-off-error-free LP solver. The open-source solver SoPlex provides a configuration to perform such an exact rational solve [13,14]. We refer to this fallback as Exact-SoPlex. It is performed only if all techniques described above are not applicable or fail to decide the status of a leaf.

Exact LP solving is also used to test whether an approximate primal solution can be converted to an exact primal solution by fixing the integer variables to the rounded values of the floating-point solution and solving the LP over the continuous variables in rational arithmetic with Exact-SoPlex. For each primal solution value, it is ensured to lie within its lower and upper bounds.

3 Computational Study

The guiding questions for the design of our experiments are the following: Assuming the final result produced by a floating-point branch-and-bound solver is correct, can we verify that the also reasoning of the individual decisions of the solver is correct? If not, how many of the critical decisions would require re-evaluation, and which of the errors defined in Sect. 2.2 occur at which frequency? Furthermore, how effective are the different post-processing techniques outlined in Sect. 2.3. Before evaluating our experiments in Sect. 3.2, we first describe the setup and specify the software used in Sect. 3.1.

3.1 Experimental Setup

As branch-and-bound framework, we use the open-source MIP solver SCIP 9.3 [1,3] (hash: b7906251d5) and configure SCIP via parameters such that it mimics the behavior of the pure LP-based branch-and-bound algorithm described in [6]. We use the open-source solver SoPlex 8.0 (hash: 7b9bd461) both as the underlying floating-point LP solver in SCIP, and as an exact LP solver during our *a posteriori* analysis.

Preprocessing. Before passing the instances to SCIP and starting the actual branch-and-bound process, we apply simple presolving steps [2,4]. First, we perform some *model cleanup*, i.e., we delete redundant constraints and convert singleton rows to variable bounds. Further, we apply a limited form of *constraint propagation* in rational arithmetic in order to reduce the number of infinite bounds and help increase the success rate of safe dual bounding.

Event System. In order to track SCIP's branch-and-bound process, we register so-called events that inform us every time a leaf is reached. In detail, our implementation uses the following events. Every time SCIP finds a solution, it triggers the event BESTSOLFOUND. A NODEINFEASIBLE event is triggered both when the LP relaxation at the current leaf is deemed infeasible, or when a node is pruned. We access the LP solving status to differentiate whether the relaxed LP is infeasible or feasible and eventually pruned. Note that when SCIP creates new nodes it adds them to a queue and sets the local lower bound to the objective value of the LP relaxation of its parent. After a new solution is found, SCIP immediately removes all nodes from the queue whose lower bound is worse than the objective of the newly found solution. Each of these removed leaves triggers a NODEDELETE event. Finally, every integer-feasible node triggers a NODEFEASIBLE event.

All node events come with a list of branching decisions. Additionally for the NODEFEASIBLE and NODEINFEASIBLE events, also the primal and dual LP solution, respectively the Farkas proof [11], and the simplex basis are available. When the NODEDELETE event is triggered, we recompute the dual LP solution by solving the LP again in floating-point arithmetic with SoPlex. At the BESTSOLFOUND event, SCIP only provides the primal solution.

Test Sets. We consider two test sets curated in [6] for benchmarking an exact MIP solver. The FPEasy test set consists of 57 instances that were found easy by the floating-point solver SCIP. The NumDiff test set consists of 50 instances and was compiled to be numerically challenging due to large coefficient ranges or poor conditioning. For a detailed description we refer to [6], which also lists on which instances the results of floating-point SCIP deviate from the exact rational results.

To provide more stable results, we perform our experiments on each instance twice: one time on the original instance and another time after permuting the order of variables and constraints. For the NumDiff test set instance normalized-aim-200-1_6-ye we use aim-200 as an abbreviation.

The code base used for our experiments is publicly available at https://github.com/alexhoen/bnbanalyzer. All experiments were carried out on identical machines with Intel(R) Xeon(R) CPU E7-8880 v4 @ 2.20 GHz with a time limit of 10800 s for each instance, which includes both the solving time of the floating-point solver SCIP as well as the evaluation of the leaves.

3.2 Evaluation of Branch-and-Bound Decisions in SCIP

Table 1 provides a first overview of the results for the floating-point solver SCIP on both test sets FPEasy and NumDiff, summarizing how many instances were correctly solved and on how many SCIP returns an inexact solution or even an incorrect status. The columns "correct" state the number of instances where no errors occurred as defined in Sect. 2.2: All solutions are either exactly integer feasible or convertible to such, all infeasible leaves are infeasible in rational arithmetic, and any node pruning is also justified in exact arithmetic. By contrast, the columns "fails" state the number of instances where at least one such error occurred and solving finished within the time limit. The column "primal" reports the number of instances with a solution error (see Sect. 2.2), the column "dual" reports the number of instances with a bound, gap, or infeasibility error. Note that for the two MIPLIB-instances (30:70:4_5:0_95:100 and npmv07) SCIP timed out before reaching a leaf or producing a solution and therefore did not issue any event. Since no wrong decision was made these instances are labeled as correct.

Table 1. Aggregate statistics on instances with/without incorrect decisions in floating-point SCIP.

test set		instances	within time limit				time out	
			correct	fails	primal	dual	correct	fails
FPEasy	original	57	49	2	0	2	6	0
	permuted	57	49	2	0	2	6	0
NumDiff	original	50	11	20	12	9	14	5
	permuted	50	13	18	12	9	13	7

Regardless of the permutation, for 55 of the 57 instances of FPEasy, we could verify that SCIP followed a fully correct solving process. On 49 instances it terminated with an exact optimal solution, while on 6 instances SCIP hit the time limit. Only for two instances, we encountered dual fails. On one instance (vpm2) the result is still correct, on the other instance (dano3_4), the result is slightly suboptimal. This can also be seen in Table 5, where we compare the results of floating-point SCIP to the results of a numerically exact version of SCIP [6,8,9], on the instances where SCIP made at least one wrong decisions.

As expected, verifying the results of the NumDiff test set exhibits more errors. Only on 25 (original) and 27 (permuted) of the 50 instances, respectively, SCIP followed a fully correct solution process. On each seed, 14 of these instances hit the time limit, so only for 11 respectively 13 instances SCIP terminated with a verifiably exact optimal solution. Still, on both test sets, a notable amount of the floating-point solves exhibits no errors.

For a more detailed analysis, Tables 2, 3 and 4 include all instances labeled as "fail" in Table 1 and those instances for which SCIP produced a "correct"

result but made incorrect decisions during the solving process. All instances where SCIP made a wrong decision on FPEasy are listed in Table 2. Instances of NumDiff that encountered errors and are solved within the time limit are listed in Table 3. A complete overview NumDiff instances with wrong decisions where SCIP timed out are listed in Table 4. The column "leaves" lists the total number of processed leaves of the tree; the remaining columns report the number of leaves with errors according to the definition in Sect. 2.2, where "W" and "S" stands for weak and strong errors, respectively.

Table 2. Analysis of number of leaves with incorrect decisions on the FPEasy instances within the time limit.

Instance	Perm	leaves	Sol		Bound		Gap		Inf
			W	S	W	S	W	S	
dano3-4	0	29	0	0	0	0	0	1	0
	1	29	0	0	0	0	0	1	0
vpm2	0	688 018	0	0	5	34	0	0	0
	1	649 085	0	0	11	167	0	0	0

Notably, only the instances ns1859355 and vpm2 show a correct result despite wrong decisions during the solving process. On FPEasy the vast majority of instances in Table 2 are solved without wrong decisions. SCIP incorrectly evaluates nodes on only two instances, affecting one node in dano3-4 and 39 of 178 nodes in vpm2, respectively. Though 39 or 178 wrong decisions on vpm2 seems to be a notable amount of wrong decisions, compared to the overall evaluated nodes on the entire FPEasy test set the wrong decisions on these two instances make up a tiny portion.

As expected, on the numerically more challenging NumDiff test set, more wrong decisions are made. First, let us focus on the instances with dual fails. Besides the instances alu16_8, tkatTV5, and tkat3TV, all instances show at most two leaves with strong bound or gap errors. On the instances, tkat3*, ns1629327 and ns1859355, more weak and strong bound or gap errors appear, and alu10_9 produces 46 infeasibility errors. Nevertheless, these all make up only a small fraction of the total number of evaluated nodes.

On the alu*-instances (except for alu16_1), as well as on the instances dfn6_load, neos-1053591, neos-1062641, neos-1603965, ns1866531 the solutions generated by SCIP are slightly infeasible and can not be converted to exact solutions. Especially the alu instances are numerical difficult: Either the instance is infeasible or there is a huge gap between the floating-point solution and the solution of Exact-SCIP as can be seen in Table 5.

Table 3. Analysis of number of leaves with incorrect decisions on the NumDiff instances within the time limit.

			Sol		Bound		Gap		Inf
Instance	Perm	leaves	W	S	W	S	W	S	
alu10_1	0	883	2	0	0	0	0	0	0
alu10_7	0	811	0	1	0	0	0	0	0
	1	734	0	3	0	0	0	0	0
alu10_8	0	4 254	0	3	0	0	0	0	0
	1	4 486	1	3	0	0	0	0	0
alu10_9	0	6 665	1	3	0	0	0	0	0
	1	7 675	1	5	0	0	0	0	0
alu16_1	0	958 643	0	0	0	0	0	0	46
alu16_7	0	694	0	3	0	0	0	0	0
	1	1 307	0	4	1	1	0	0	0
alu16_8	0	TL	0	8	0	20	0	0	3
	1	219 362	0	3	0	2	0	0	0
alu16_9	0	199 225	0	4	0	0	0	0	0
	1	44 961	0	13	0	0	0	0	0
bernd2	0	31 081	0	4	0	0	0	0	0
	1	23 144	0	5	0	0	0	0	0
dfn6_load	0	11 210	2	6	9	0	0	0	0
	1	3 064	0	6	9	0	0	0	0
aim-200	0	7 414	0	0	0	0	0	0	1
	1	9 534	0	0	0	0	0	0	1
neos-1053591	0	225 381	0	2	0	0	0	0	0
	1	TL	1	1	0	0	0	0	0
neos-1062641	0	92	0	1	0	0	0	0	0
	1	46	0	1	0	0	0	0	0
neos-1603965	0/1	1	0	1	0	0	0	0	0
ns1629327	0	9 985	0	0	0	0	0	2	0
	1	11 788	0	0	0	2	0	2	0
ns1859355	0	12 581	0	0	0	0	1	0	0
	1	6 434	0	0	0	0	2	0	0
ns1866531	0/1	1	0	1	0	0	0	0	0
prodplan2	0	3	0	0	0	1	1	1	0
	1	27	0	0	3	0	2	1	0
tkat3K	0	4 367	0	0	11	1	0	0	0
	1	8 335	0	0	5	2	0	0	0
tkat3T	0	14 604	0	0	27	1	0	0	0
	1	5 219	0	0	4	4	0	0	0
tkat3TV	0	4 546	0	0	10	7	0	0	0
	1	19 032	0	0	28	5	0	0	0
tkatTV5	0	8 459	0	0	15	26	0	0	0
	1	18 999	0	0	1	17	0	0	0

Table 4. Analysis of number of leaves with incorrect decisions on the NumDiff instances with time out.

Instance	Perm	leaves	Sol		Bound		Gap		Inf
			W	S	W	S	W	S	
alu16_5	0	1 342 439	0	0	0	0	0	0	11
	1	280 574	0	0	0	0	0	0	875
dfn6fp_load	1	585	0	2	3	4	0	0	0
ns2080781	0	701 865	7	3	156	72	0	0	0
	1	52 258	0	4	0	0	0	0	0
ns1925218	1	22 556	0	0	0	0	0	0	1
prodplan1	0	36	0	0	0	1	0	0	0
ran14x18-	0	1 268 265	0	12	1	0	0	0	0
-disj-8	1	1 288 277	0	6	2	2	0	0	0
neos-799716	1	1 095	0	0	0	0	0	0	1

One natural attempt to reduce such solution errors is to call SCIP with a tightened primal feasibility tolerance. Per default, the feasibility tolerance is 10^{-6}. We repeated our experiments but tightened the tolerance parameter "numerics/feastol" to 10^{-9}. The results show a significant reduction in solution errors. On the instances alu10_1, alu10_8, alu10_9, bernd2, aim-200 and dfn6_load, SCIP finds an exactly integer-feasible solution, or they can be converted to such, or SCIP times out, but on the instances alu_16_7 or ns1859355, the floating-point solver still produces slightly infeasible solutions. On the FPEasy instance dano3_4 also all gap errors were removed, successfully closing the dual gap on this instance.

However, we also observe that the number of explored nodes is significantly increased. This does not only lead to increased solving time, but can also create more numerically challenging leaves, hence potentially increasing the absolute number of incorrect decisions. An example of this behavior is the instance blend2, for which the result of SCIP can no longer be verified after tightening the tolerance.

3.3 Analysis of Different Post-processing Techniques

Finally, let us have a look at the effectiveness and success rate of the different post-processing techniques described in Sect. 2.3. Table 6 reports aggregate results over all leaves of a complete run over each test set (column "leaves") and categorizes them based on the techniques that were successful for verification. The column "floating-point" lists the percentage of leaves that are verified automatically or through the use of safe bounding. The columns "Reconstruct", "Factorize", and "Exact" likewise represent the percentage for the correspond-

Table 5. Comparison of the results of floating-point SCIP and rational solutions. The exact results were generated with Exact-SCIP (hash: c7bd4be7b9) and a time limit of 18800 s.

Instance	SCIP		range of exact solution
	Seed 0	Seed 1	
dano3_4	576.435224722083	576.435224722083	576.4352247072
vpm2	13.75	13.75	13.75
alu10_1	86	inf	inf
alu10_7	83	83	[1243230.17385925,∞)
alu10_8	84	84	[169.4062227788,∞)
alu10_9	84	83	[1166.7499882579,3726773]
alu16_1	84	inf	inf
alu16_7	79	79	[315.4000244141,∞)
alu16_8	x	79	[2256.0481863315,∞)
alu16_9	79	79	[3801.0189752579,∞)
bernd2	11209.0573895658	11209.0573899187	113091.469015879
dfn6_load	3.743842258432	3.743842258432	[3.7683622392,4.4007262645]
aim-200	inf	inf	200
neos-1053591	-3662.9144	time limit	-3662.9144
neos-1062641	0	0	0
neos-1603965	619244367.662956	619244367.662956	[619246130.539662,∞)
ns1629327	-10.9803191329325	-10.9803191329325	-10.9803191329
ns1866531	0	0	10
prodplan2	-239399.435140992	-239399.4351411	-239399.435141407
tkat3K	4772818.1	4772818.1	4772818.1
tkat3T	5564891.75	5564891.75	5564891.75
tkat3TV	8388398.65	8388398.65	8388398.65
tkatTV5	28117644.225	28117644.225	28117644.225

ing methods. The row "unb" reports the numbers for all instances that contain variables with unbounded domains.

Surprisingly, 93.76% (FPEasy) to 99.69% (NumDiff) of the leaf decisions can be verified using floating-point arithmetic. Hence, the most successfull methods are also the ones that can be implemented most efficiently. In relative terms, the percentage of erroneous leaf decisions is small on all test sets.

Safe bounding requires bounded variables and can therefore more reliably be applied to problems with bounded variables. As a result, unbounded problems, listed in Table 6 in row "unb", have a significantly higher percentage of expensive Exact-SoPlex calls. On these unbounded problems, the success rate of the "floating-point" techniques arithmetic drops significantly to 65.18% on FPEasy and 94.89% on NumDiff. This leads to an increase in more expensive techniques; most notably, the number of Exact-SoPlex calls significantly increases from 2.85% to 29.89% on FPEasy and from 0.06% to 3.86% on NumDiff.

Table 6. Distribution on the different verification methods over all leaves on seed 0.

		leaves	floating-point	Reconstruct	Factorize	EXACT	error
FPEASY	all	19952937	93.76%	2.93%	0.46%	2.85%	0.00020%
	unb	424881	65.18%	0.15%	4.78%	29.89%	0.00024%
NUMDIFF	all	8389838	99.69%	0.0%	0.04%	0.27%	0.00543515%
	unb	71221	94.89%	0.02%	1.22%	3.86%	0.00140408%

4 Conclusion

Our experiments indicate that the overwhelming majority of the decisions made by the floating-point solver SCIP on both the test set FPEASY and even on the numerically more challenging test set NUMDIFF are correct even in rational arithmetic. For feasible instances, only a small, typically single-digit, number of nodes per instance can not be verified and would require further treatment to generate a proof of correctness of the floating-point calculation. Notably, most of the node decisions can be verified in fast floating-point arithmetic and do not require techniques that rely on more expensive rational arithmetic.

However, we also observe that sub-problems that are infeasible in exact arithmetic pose a special situation. Here, sometimes the floating-point solver can find solutions with slight violations and use them to prune nodes incorrectly in terms of rational arithmetic. This is particularly troubling for instances that are globally infeasible in exact arithmetic. We found that tightening the feasibility tolerance of the floating-point solver is not a reliable remedy against this issue.

Overall, these results are encouraging. On the one hand, they quantify the widely accepted belief that floating-point MIP solvers are generally numerically robust for well-behaved input data. On the other hand, they also suggest a path forward to using floating-point solvers as a grey box in order to solve MIPs exactly over the rational numbers. First, the experimental setup put forward in this paper can be used to generate partial certificates of optimality, which can even be verified for example in the VIPR format [5]. Second, these partial certificates can then be completed by continuing to explore sub-problems that were incorrectly discarded, either recursively with increased precision or by calling an exact MIP solver. For this to become competitive with the state of the art in exact MIP solving, our *a posteriori* verification of LP-based branch-and-bound needs to be extended to include more advanced techniques like presolving or cutting plane separation.

Acknowledgements. The work for this article has been partly conducted within the Research Campus MODAL funded by the German Federal Ministry of Education and Research (BMBF grant number 05M14ZAM).

References

1. Achterberg, T.: Constraint Integer Programming. Ph.D. thesis (2007). https://doi.org/10.14279/depositonce-1634
2. Achterberg, T., Bixby, R., Gu, Z., Rothberg, E., Weninger, D.: Presolve reductions in mixed integer programming. INFORMS J. Comput. **32** (2019). https://doi.org/10.1287/ijoc.2018.0857
3. Bolusani, S., et al.: The SCIP optimization suite 9.0 (2024). https://arxiv.org/abs/2402.17702
4. Brearley, A.L., Mitra, G., Williams, H.P.: Analysis of mathematical programming problems prior to applying the simplex algorithm. Math. Program. **8**(1), 54–83 (1975). https://doi.org/10.1007/BF01580428
5. Cheung, K., Gleixner, A., Steffy, D.E.: Verifying integer programming results. In: Eisenbrand, F., Koenemann, J. (eds.) IPCO 2017. LNCS, vol. 10328, pp. 148–160. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59250-3_13
6. Cook, W., Koch, T., Steffy, D.E., Wolter, K.: An exact rational mixed-integer programming solver. In: Günlük, O., Woeginger, G.J. (eds.) IPCO 2011. LNCS, vol. 6655, pp. 104–116. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20807-2_9
7. Cook, W., Koch, T., Steffy, D.E., Wolter, K.: A hybrid branch-and-bound approach for exact rational mixed-integer programming. Math. Program. Comput. **5**(3), 305–344 (2013). https://doi.org/10.1007/s12532-013-0055-6
8. Eifler, L., Gleixner, A.: A computational status update for exact rational mixed integer programming. Math. Program. (2022). https://doi.org/10.1007/s10107-021-01749-5
9. Eifler, L., Gleixner, A.: Safe and verified gomory mixed integer cuts in a rational MIP framework (2023). https://arxiv.org/abs/2303.12365
10. Eifler, L., Gleixner, A., Pulaj, J.: A safe computational framework for integer programming applied to chvátal's conjecture. ACM Trans. Math. Softw. **48**(2) (2022). https://doi.org/10.1145/3485630
11. Farkas, J.: Theorie der einfachen ungleichungen. Journal für die reine und angewandte Mathematik **124**, 1–27 (1902). http://eudml.org/doc/149129
12. Gleixner, A., et al.: Miplib 2017: data-driven compilation of the 6th mixed-integer programming library. Math. Program. Comput. **13**(3), 443–490 (2021). https://doi.org/10.1007/s12532-020-00194-3
13. Gleixner, A., Steffy, D.E.: Linear programming using limited-precision oracles. Math. Program. **183**(1–2), 525–554 (2019). https://doi.org/10.1007/s10107-019-01444-6
14. Gleixner, A.M., Steffy, D.E., Wolter, K.: Iterative refinement for linear programming. INFORMS J. Comput. **28**(3), 449–464 (2016)
15. Klotz, E.: Identification, assessment, and correction of ill-conditioning and numerical instability in linear and integer programs. In: Newman, A., Leung, J. (eds.) Bridging Data and Decisions, pp. 54–108. TutORials in Operations Research (2014). https://doi.org/10.1287/educ.2014.0130
16. Land, A.H., Doig, A.G.: An automatic method of solving discrete programming problems. Econometrica **28**(3), 497–520 (1960). http://www.jstor.org/stable/1910129
17. Neumaier, A., Shcherbina, O.: Safe bounds in linear and mixed-integer linear programming. Math. Program. **99**, 283–296 (2004). https://doi.org/10.1007/s10107-003-0433-3

18. Pan, V.Y.: Nearly optimal solution of rational linear systems of equations with symbolic lifting and numerical initialization. Comput. Math. Appl. **62**(4), 1685–1706 (2011). https://doi.org/10.1016/j.camwa.2011.06.006. https://www.sciencedirect.com/science/article/pii/S0898122111004743
19. Paxian, T., Biere, A.: Uncovering and classifying bugs in MaxSAT solvers through fuzzing and delta debugging. In: Järvisalo, M., Le Berre, D. (eds.) Proceedings of the 14th Internantional Workshop on Pragmatics of SAT Co-located with the 26th International Conference on Theory and Applicationas of Satisfiability Testing (SAT 2003), Alghero, Italy, 4 July 2023. CEUR Workshop Proceedings, vol. 3545, pp. 59–71. CEUR-WS.org (2023). http://ceur-ws.org/Vol-3545/paper5.pdf
20. Saunders, B.D., Wood, D.H., Youse, B.S.: Numeric-symbolic exact rational linear system solver. In: Proceedings of the 36th International Symposium on Symbolic and Algebraic Computation, ISSAC 2011, pp. 305–312. Association for Computing Machinery, New York (2011). https://doi.org/10.1145/1993886.1993932
21. Steffy, D., Wolter, K.: Valid linear programming bounds for exact mixed-integer programming. Technical Report 11-08, ZIB, Takustr. 7, 14195 Berlin (2011)
22. Steffy, D.E.: Topics in exact precision mathematical programming. Ph.D. thesis, Georgia Institute of Technology (2011). http://hdl.handle.net/1853/39639
23. Ursic, S., Patarra, C.: Exact solution of systems of linear equations with iterative methods. SIAM J. Algebraic Discrete Methods **4**(1), 111–115 (1983). https://doi.org/10.1137/0604014
24. Vries, S., Vohra, R.: Combinatorial auctions: a survey. INFORMS J. Comput. **15**, 284–309 (2003). https://doi.org/10.1287/ijoc.15.3.284.16077
25. Wan, Z.: An algorithm to solve integer linear systems exactly using numerical methods. J. Symb. Comput. **41**(6), 621–632 (2006). https://doi.org/10.1016/j.jsc.2005.11.001. https://www.sciencedirect.com/science/article/pii/S0747717105001653

LLMs for Cold-Start Cutting Plane Separator Configuration

Connor Lawless(✉), Yingxi Li(✉), Anders Wikum(✉), Madeleine Udell, and Ellen Vitercik

Management Science and Engineering, Stanford University, Stanford, USA
{lawlessc,yingxi,wikum,udell,vitercik}@stanford.edu

Abstract. Mixed integer linear programming (MILP) solvers ship with a staggering number of parameters that are challenging to select a priori for all but expert optimization users, but can have an outsized impact on the performance of the MILP solver. Existing machine learning (ML) approaches to configure solvers require training ML models by solving thousands of related MILP instances, generalize poorly to new problem sizes, and often require implementing complex ML pipelines and custom solver interfaces that can be difficult to integrate into existing optimization workflows. In this paper, we introduce a new LLM-based framework to configure which cutting plane separators to use for a given MILP problem with little to no training data based on characteristics of the instance, such as a natural language description of the problem and the associated LaTeX formulation. We augment these LLMs with descriptions of cutting plane separators available in a given solver, grounded by summarizing the existing research literature on separators. While individual solver configurations have a large variance in performance, we present a novel ensembling strategy that clusters and aggregates configurations to create a small portfolio of high-performing configurations. Our LLM-based methodology requires no custom solver interface, can find a high-performing configuration by solving only a small number of MILPs, and can generate the configuration with simple API calls that run in under a second. Numerical results show our approach is competitive with existing configuration approaches on a suite of classic combinatorial optimization problems and real-world datasets with only a fraction of the training data and computation time.

Keywords: Algorithm Configuration · Large Language Models · Cutting Planes · Integer Programming

1 Introduction

Mixed Integer Linear Programming (MILP) is a remarkably effective mathematical framework for discrete optimization that has been successfully deployed in a variety of industries, ranging from healthcare [1] to routing [16] and production planning [52]. Modern MILP solvers, such as Gurobi and SCIP, employ a number

G. Tack (Ed.): CPAIOR 2025, LNCS 15763, pp. 51–69, 2025.
https://doi.org/10.1007/978-3-031-95976-9_4

of parameters that control key algorithmic components including cutting planes, branching strategies, and the use and scheduling of primal heuristics. Configuring these parameters for good solver performance on a particular problem is notoriously difficult, even for optimization experts. MILP solvers are equipped with default parameter configurations that are typically tuned for good average performance over a broad class of potential optimization problems. These general-purpose parameter settings can often be improved substantially for specific instances or sets of instances [46,61]. Motivated by this opportunity, a flurry of recent work (see [14] for a comprehensive survey) has focused on tuning parameters for *families* of instances using machine learning (ML).

In this paper, we study the problem of configuring which cutting planes separators to use for a given MILP instance. Cutting planes (or cuts) play a pivotal role in the performance of MILP solvers—they have driven impressive 32× speedups in computation time for branch-and-cut solvers such as CPLEX [4] by strengthening linear relaxations at nodes in the branch-and-bound tree. Recent work has shown that better configuration of cutting plane separators alone can achieve up to 72% speedups over SCIP default settings on classic combinatorial optimization problems [46]. Existing ML-based approaches for cutting plane separator configuration, and IP configuration more broadly, typically involve training a ML model by solving hundreds, if not thousands, of related MILP instances multiple times. This effort requires access to both historical data from the same instance family and a hefty computational budget to run all the instances. Furthermore, these methods often employ custom solver interfaces that expose parts of a solver's internal machinery. This requirement complicates implementation and restricts use to less powerful, open-source solvers. Given the daunting complexity and computational cost of implementing existing ML approaches, in this paper, we study the following research question.

Can we configure problem-specific cutting plane separators with little to no training data, negligible computation time, and only access to existing solver APIs?

We call this problem the *cold-start* cutting plane separator configuration problem. Previous literature has highlighted some potential pathways to an affirmative answer. Bixby and Rothberg [15] remarked that ideal cutting plane methods would *"be able to recognize models to which it can be applied, introducing little or no overhead when the model does not fit the mold"*. Indeed, a configuration approach that can identify relevant model structure and turn on/off the associated separators could lead to better problem-specific configuration. Furthermore, previous studies on cutting plane separator configuration have validated their models by highlighting that the learned configurations match known results from literature [46], which suggests that existing cutting plane research could provide a good signal for configuration.

Based on these two observations, large language models (LLMs) offer a promising direction toward generating better problem-specific configurations without training. Recent work has highlighted that LLMs are able to identify and model MILP structures from natural language alone [6,43]. Moreover, LLMs

are trained on wide swaths of the internet [2], which may include existing work on cutting planes and mathematical programming. However, many challenges arise when using LLMs for configuration. First, each MILP solver implements its own (often proprietary) set of cutting plane separators, which can differ from others in name, implementation, or both [3,21]. As a result, our task is solver-dependent. Second, LLMs are known to be noisy and to hallucinate (i.e., generate factually incorrect text) [31], which is problematic given that solver performance is extremely sensitive to parameter choices (i.e., adding or removing certain separators). More generally, despite some research highlighting the performance of specific separators on some special problem structures, cutting plane selection more broadly is a notoriously difficult problem from both a theoretical and empirical perspective [24].

In this paper, we showcase the viability of LLMs for performing cold-start algorithm configuration based solely on a natural language description of the model. To the best of our knowledge, this is the first work to explore the use of LLMs for algorithm configuration. To reduce hallucinations and design solver-specific configurations, we augment base LLMs by providing solver-specific descriptions of cutting plane separators that have been generated by summarizing existing research literature. Because LLM outputs are stochastic, we introduce an ensembling technique that first generates a pool of candidate configurations, then clusters them using a k-median clustering algorithm to build a small representative set of potential configurations. We select from this set of candidate configurations using either a heuristic in settings with no data, or by testing the set on a small collection of training instances. Empirically, we demonstrate that our approach generates high-performing cutting plane configurations that are competitive with existing ML approaches at a fraction of the computational cost, and without the need for a custom solver interface.

2 Background and Related Work

A mixed-integer linear program is an optimization problem of the form:

$$\underset{x}{\operatorname{argmin}} \left\{ c^T x \mid Ax \leq b, x \in \mathbb{R}^{n-r} \times \mathbb{Z}^r \right\},$$

where x is a vector of decision variables, $c \in \mathbb{R}^n$ is a vector of objective coefficients, and $A \in \mathbb{R}^{m \times n}$ is a constraint coefficient matrix. The size of a MILP is characterized by the number of constraints (m) and variables (n) in the problem.

The standard algorithms for solving MILPs to optimality are variants of the branch-and-bound (B&B) algorithm [20]. At a high level, B&B recursively partitions the solution space, organizing this partition with a search tree (branch). It computes linear programming (LP) relaxation bounds along the way to prune sub-trees that provably cannot contain an optimal solution (bound). To enhance solution efficiency, valid linear inequalities that tighten the feasible region without excluding any feasible solutions, called *cutting planes* (or cuts), are introduced into the LP relaxations. This augmented approach, referred to as branch-and-cut, can significantly accelerate B&B by more quickly refining the search

space. However, generating cutting planes can be computationally expensive [15].

MILP solvers typically have a diverse set of separators that are used to generate cuts, ranging from general-purpose (e.g., Gomory, MIR, and Cover Cuts [20]) to specialized for specific constraint structures (e.g., knapsack inequalities). Each separator presents a different trade-off between performance improvement and computation time. Typically, separators are run in a prioritized order until a specified number of cuts have been generated. The generated cuts are then added to a cut pool that acts as a buffer to store the cuts and apply the most promising ones to a given B&B tree node. Well-chosen cutting plane configurations can both expedite cutting plane generation by deactivating computationally demanding or ineffective separators and help improve the quality of the cut pool.

The rest of this section reviews the literature most relevant to our work.

2.1 Algorithm Configuration

Our work is closely related to the rich literature on algorithm configuration [9,11], a field that seeks automatic methods to find the best settings for parameterized algorithms. Early work developed general-purpose configuration methods, with notable frameworks including ParamILS [35], SMAC [34], and GGA+ [7]. While these methods are effective in identifying a single configuration with strong performance across a dataset, they can struggle with instance heterogeneity. Instance-specific frameworks, which are customized to individual problem instances using machine learning (ML), were later developed to address this limitation. Prominent examples include ISAC [37], which clusters instances based on features to identify configurations, Hydra [60], which iteratively constructs portfolios of configurations, and PCIT [48], which employs advanced ML techniques for automated configuration tuning.

Early applications of ML to MILP solving emerged from these instance-specific algorithm configuration frameworks. A follow-up to Hydra, the Hydra-MIP framework [61], was specifically designed for parameter selection in MILP solvers. Hydra-MIP is trained over distributions of MILP datasets and, at the time, demonstrated performance improvements compared to the default configurations of CPLEX, showcasing the potential of automated parameter tuning to improve solver efficiency. However, training these parameter configuration models is computationally intensive: Hydra-MIP, for example, requires over 250,000 CPU days of runtime and 500 sample MILP instances. Our method distinguishes itself from classic configuration approaches in that it achieves significant performance improvements over solver defaults at little computational cost, and that it uses a novel input modality—natural language descriptions—to develop the configurations.

2.2 Machine Learning for Combinatorial Optimization

The existing body of research on ML for combinatorial optimization (CO) can be broadly categorized into two approaches: (1) integrating ML models into

existing MILP solvers to enhance their performance, and (2) training ML models to directly make decisions that constitute part or all of the solution [23,36,38,42,63]. Our work aligns closely with the first category.

In recent years, for example, there has been an explosion of work on using ML to tune various search heuristics within the B&B algorithm [27,39,40] to accelerate MILP solutions. The use of ML for cutting plane selection has also garnered significant attention, with ML guiding cut generation [10,12,25,29], selecting cuts from a generated pool [51,55], or configuring separators that generate high-quality pools of cuts [46]. Among these, the work by [46] is particularly relevant to ours, proposing a neural contextual bandit model that predicts solver-specific cutting plane configurations for various problem distributions. Their model is trained on MILP problems sampled from the same distribution, requires solving thousands of related MILP instances during model training, and leverages a custom SCIP solver interface to change the configuration at different depths of the B&B tree.

In contrast to the work of [46], we focus on the problem of selecting an *instance-agnostic configuration* (i.e., one configuration to use for all instances coming from a given problem distribution). We do so for two practical reasons. First, learning instance-specific configurations often requires training ML models to select a configuration from a portfolio of potential configurations [46,61] based on high-dimensional instance-specific features (e.g., a bipartite graph representation of the problem). These approaches also require a large computational budget to train the ML model, requiring thousands of MILP solves. In resource-constrained settings, there are often not enough instances to effectively train such models nor sufficient computational resources to support the training process. Second, deep learning-based configuration algorithms for instance-specific inference often require additional computational resources and memory, which makes them impractical to deploy in some settings. Our work also differs from [46] in that we configure separators once rather than at each level of the B&B tree, as this approach does not require any custom solver interface. In brief, our approach distinguishes itself by requiring little to no training data—only a natural language description of the problem—and by being compatible with existing solver APIs.

2.3 LLMs and Optimization

Recent advances in large language models (LLMs) have led to impressive performance in tasks across a number of domains [18,64,65]. However, LLMs are known to hallucinate, generating seemingly plausible but factually incorrect text (see [67] and [31] for comprehensive surveys). They are also sensitive to prompts: semantically similar prompts can result in significantly different outcomes [41]. Recent literature focuses on addressing this problem through prompt optimization [44,47,53] and augmentation of LLM using interactive tools [50].

Despite these shortcomings, the ability of LLMs to leverage natural language points us to new possibilities for optimization. Recent work has explored LLMs

for optimization modeling [5,6,8,54,56,57,59,62]. Building on these capabilities, LLMs have powered chatbots that allow users to interact with an underlying optimization model in the context of supply chain management [45], meeting scheduling [43], and debugging infeasible models [17]. Our work differs in that we use LLMs to improve the efficiency of solving an optimization problem described in natural language rather than formulating the MILP. Most similar to our work, a line of research has investigated the use of LLMs to tune hyperparameters [22,49,66]. These approaches employ LLMs as *optimizers* to iteratively refine hyperparameter selection and require repeatedly solving MILP instances. In contrast, our work focuses on the cold-start problem where very few MILP instances need to be solved, leverages existing domain knowledge in optimization to improve the LLM performance on this task, and introduces novel strategies to ensemble noisy LLM outputs.

3 Methodology

In this paper, we study a variant of the cutting plane separator configuration problem introduced in [46]. The input to the problem consists of a MILP solver with M different separator algorithms (which take as input a MILP and return valid cutting planes) and S possible settings for each separator. For example, Gurobi supports $M = 21$ different separator algorithms, each with $S = 3$ possible settings (disable, use, and use aggressively), for a total of $\mathcal{O} = |S|^M$ candidate configurations. The goal of the separator configuration problem is to find the best setting of the separators for a given MILP.

We assume access to a natural language description of the MILP that includes a short text description of the problem as well as its corresponding LaTeX formulation. Let $\mathcal{X}$ represent the unknown distribution of MILP instances corresponding to the natural language description. We focus on the problem of selecting an *instance-agnostic configuration* $\mathcal{C}$ (i.e., one configuration to use for all instances coming from $\mathcal{X}$) in the resource-constrained setting, where we have a small training dataset $\mathcal{K}_{\text{val}}$ sampled from $\mathcal{X}$ and want to generate a high-performing configuration under a minimal computational budget. We refer to the setting where $\mathcal{K}_{\text{val}} = \emptyset$ as the *cold-start configuration problem*. The performance of a separator configuration for a particular solver can be measured by a number of metrics. Following the setup of [46], we use relative improvement in solve time over the default solver configuration—a detailed description of this metric can be found in our experimental setup in Sect. 4.1.

Our approach uses LLMs to generate promising candidate separator configurations, exploiting problem-specific context and prior knowledge encoded in the LLM. Figure 1 summarizes our separator configuration procedure. In Sect. 3.1, we detail how we leverage LLMs to generate candidate separator configurations based on natural language descriptions of the optimization problem and available cutting plane separators. In Sect. 3.2, we show how we ensemble a pool of candidate configurations into a single final configuration via k-medoids clustering and evaluate on a small training dataset of instances.

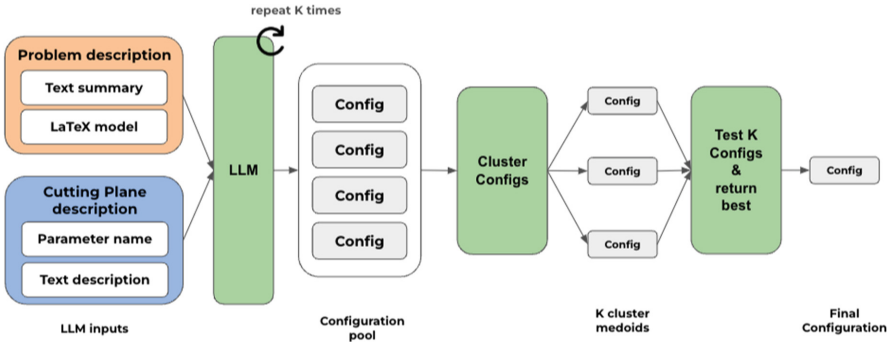


Fig. 1. Overview of cutting plane separator configuration algorithm.

3.1 Generating Configurations

To generate high-performing separator configurations from LLMs, we construct a prompt that includes both problem-specific and solver-specific information (see Fig. 2 for a sample prompt). The prompt includes a text description of the optimization problem as well as its corresponding LaTeX formulation. For the models evaluated in this paper, this natural language description was generated using a LLM to summarize either a relevant textbook section or research paper that introduced the model, and then checked for accuracy by an optimization expert. Although a natural language description is an additional requirement compared to existing algorithm configuration approaches, which typically rely on access to the model directly as a `.mps` or `.lp` file, the optimization user often already has a text description of the problem, or can generate one easily, in most practical applications. Later in Sect. 4.3, we show that our approach is robust to the type of information included about the model and performs comparably when given a text summary or LaTeX model alone.

Each MILP solver has a different set of separators available, each with its own solver-specific parameter names, and so we limit LLM hallucinations by including a solver-specific description of available cutting plane separators in the prompt. These descriptions include the specific parameter name as well as a natural language description of the separators. To generate these descriptions, we use an LLM to summarize existing research papers that introduce the different separators, producing concise descriptions that specify the types of problem structure the cutting planes can be used on (e.g., *Implied Bound Cuts can be used in settings where there is a logical implication between binary and continuous variables*). We further augment these sources with online resources, such as slides and survey papers, and expert knowledge from the integer programming academic community. An inherent limitation of our approach is that for closed-source solvers (e.g., Gurobi), it is impossible to know exactly what implementation and separator procedure is used for a given family of cutting planes. Consequently, these descriptions offer only a broad summary. Nonetheless, these initial descriptions perform well in practice and can only be improved by richer

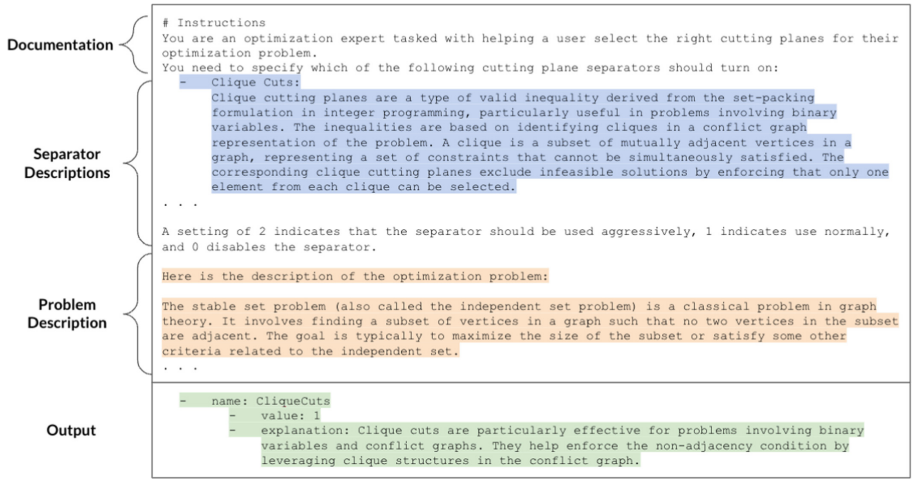


Fig. 2. Sample prompt for generating a separator configuration. Blue (orange) lines indicate solver-specific (problem-specific) information input into the prompt. Green lines indicate a sample LLM output. In the interest of space, only a subset of the full cutting plane and problem descriptions are included. (Color figure online)

information about the underlying separator algorithms. For open-source solvers, additional information such as code implementations is publicly available and thus can provide even more context to a LLM-generated summary of the separators. While we do not exploit this information in our descriptions, effectively leveraging this additional context, as well as other information sources more broadly, is an important direction for future research.

3.2 Ensembling Configurations

LLMs have stochastic outputs: re-running the same prompt multiple times can produce different outcomes. Motivated by similar approaches [13,19], we exploit this feature by sampling M candidate separator configurations using the LLM and then *ensembling* them together into one final configuration. Initial experiments showed that separators used in LLM-generated configurations were correlated. Hence we cluster the pool of configurations into k clusters using a k-medoids clustering algorithm to find structurally distinct configurations. We use k-medoids instead of k-means to ensure that the selected configuration is a configuration that was output by the LLM. To select a final configuration from the k medoids of the generated clusters, we evaluate each medoid on the validation dataset and select the configuration with the best performance, as measured by the median improvement over the solver's default configuration. Note that the number of clusters k can, and should, be chosen to match the computational

budget of the application. Larger values of k correspond to trying more candidate configurations and are expect to increase the performance of the algorithm on the validation dataset. In the cold-start setting, without any related MILP instances, we select the medoid of the largest cluster as a heuristic.

4 Experiments

This section is organized in two parts. In the first, we demonstrate the effectiveness of our LLM-based configuration algorithm. We call our algorithm LLM(k), where k denotes the number of clusters generated during the ensembling process described in Sect. 3.2. For our experiments we generate 100 configurations to form the configuration pool, and use GPT-4o [33] as the LLM. Section 4.1 details our experimental setup, while Sect. 4.2 evaluates the performance of LLM(k) relative to baselines. The second part (Sect. 4.3) consists of ablation studies that validate our hypothesis that LLMs can reason about separator configuration. All code for our experiments is available at https://github.com/OptiMUS-optimization-modeling/LLM-Solver-Configuration.

4.1 Setup

Data. We evaluate our method on both standard MILP benchmarks from the paper by Tang et al. [55] and Ecole [27], as well as real-world MILP distributions from the Distributional MIPLIB library [32]: *Load Balancing* and the *Middle-Mile Consolidation Network Design* problem. *Load Balancing*, featured in the ML4CO competition [26] and categorized as *hard* in the Distributional MIPLIB library, is a large-scale job-scheduling problem with side constraints that ensure robustness to machine failure. The *Middle-Mile Consolidation Network Design* problem [28] determines a minimum-cost allocation of capacity on a transportation network involving vendors, fulfillment centers, and last-mile delivery centers. Table 1 provides an overview of the MILP families used in our experiment. For each, we maintain a small validation set $\mathcal{K}_{\text{val}}$ and a larger evaluation set $\mathcal{K}_{\text{eval}}$ of 30 and 100 MILP instances, respectively. In our expeirments, we solve all MILP instances to optimality with no time limits, with the exception of the *Load Balancing* instances which we solve to a 10% optimality gap.

Evaluation. Following the setup of [46], we measure performance of a separator configuration for a particular solver by its relative solve time improvement over that solver's default configuration. Explicitly, the (mean) relative solve time improvement (δ) of a separator configuration $\mathcal{C}$ on a MILP instance $x \in \mathcal{K}$ is

$$\delta(x, \mathcal{C}) = 100 \cdot \frac{t_x^{\mathcal{C}} - t_x}{t_x},$$

where t_x and $t_x^{\mathcal{C}}$ are the mean solves time of instance I under the default separator configuration and $\mathcal{C}$, respectively. Empirically, we estimate these means

Table 1. Families of MILPs considered

Name	# vars / # constrs	SCIP solve time (s)	Source
Binary Packing	300/300	1.76	[55]
Capacitated Facility Location	100/100	20.11	[27]
Combinatorial Auction	100/500	2.18	[27]
Maximum Independent Set	500/1088	1.916	[27]
Max Cut	54/134	0.49	[55]
Packing	60/60	12.25	[55]
Set Cover	500/250	1.26	[27]
Load Balancing	64340/61000	14.68[1]	[26]
Middle-Mile Consolidation Network Design (MM)	569/248	1.33	[28]

by taking the average solve time across 10 MILP solves for each configuration, under different random seeds.

To mitigate the effects of increased CPU times from parallel processing, the solve times we report for SCIP are wall times (s), while solve times for Gurobi are measured in work units. A work unit corresponds roughly to one second, but it is a deterministic unit of measurement for the same instance and hardware, regardless of parallelization.

Baselines. The baselines we consider include the baselines and methods for configuration proposed in [46]. For each MILP family: (1) PRUNING turns off any cutting planes that were not used while solving the validation instances $\mathcal{K}_{\text{val}}$ with default settings. (2) SEARCH(d), the instance-agnostic configuration method from [46], samples d candidate configurations uniformly at random, then applies the one with the best median performance on the validation set $\mathcal{K}_{\text{val}}$. We refer to d as the depth of the search. Note that we implicitly also compare to default separator settings in Gurobi and SCIP, as our performance metric measures the relative time improvement over default.

Hardware. All MILPs are solved on a distributed compute cluster whose nodes are each equipped with two 64-core AMD EPYC 7763 CPUs, for a total of 256 threads. We fully parallelize when evaluating Gurobi separator configurations, and impose a limit of 4 threads per process when evaluating configurations for SCIP. This restriction is standard in the ML for integer programming community (see, e.g. [30]) to limit the impact of thread competition on solve time.

4.2 Performance

Tables 2 and 3 report solve time improvements for separator configurations generated by our LLM-based algorithm and baselines compared to SCIP and Gurobi defaults, respectively. For each method, we report the median and interquartile range of the improvements $\{\delta(x, \mathcal{C}) : x \in \mathcal{K}_{\text{eval}}\}$ over the evaluation set

Table 2. Median (higher is better) and IQR of relative solve time improvements (%) over default SCIP separator configuration.

	SCIP configuration				
Problem	Pruning	Search(5)	Search(500)	LLM(0)	LLM(5)
Bin. Pack.	1.33 (7.78)	9.23 (28.37)	39.3 (27.72)	16.76 (23.79)	38.35 (27.38)
Cap. Fac.	−0.64 (14.68)	9.57 (24.42)	2.72 (3.09)	7.61 (17.19)	26.12 (30.88)
Comb. Auc.	1.96 (14.89)	58.1 (29.69)	64.01 (49.66)	21.06 (31.67)	63.59 (49.74)
Ind. Set	2.07 (22.42)	26.95 (46.08)	67.01 (22.7)	21.6 (49.37)	71.95 (21.95)
Max. Cut	−2.18 (5.17)	17.72 (29.47)	69.63 (12.43)	71.43 (11.8)	71.01 (11.73)
Pack.	15.87 (47.68)	−13.81 (71.34)	24.49 (42.57)	15.09 (40.53)	25.51 (42.27)
Set Cov.	6.62 (13.6)	−10.04 (37.5)	61.08 (22.56)	61.72 (21.96)	61.74 (22.39)
Load Bal.	0.08 (2.51)	−150.01 (0.01)	−50.02 (0.03)	0.0 (23.43)	6.37 (13.38)
MM	−0.12 (6.05)	−8.83 (91.58)	50.03 (45.82)	−6.52 (43.54)	53.3 (47.42)

Table 3. Median (higher is better) and IQR of relative solve time improvements (%) over default Gurobi separator configuration.

	Gurobi configuration				
Problem	Pruning	Search(5)	Search(500)	LLM(0)	LLM(5)
Bin. Pack.	0.53 (0.72)	0.12 (2.34)	12.94 (22.68)	4.31 (5.56)	4.31 (5.56)
Cap. Fac.	−1.82 (12.75)	−0.16 (15.12)	8.37 (29.93)	6.05 (45.41)	6.05 (45.41)
Comb. Auc.	0.39 (7.44)	−1.59 (26.33)	1.51 (25.45)	−1.05 (32.49)	0.64 (26.39)
Ind. Set	1.83 (1.83)	−13.97 (11.66)	7.87 (35.58)	6.29 (39.72)	6.33 (39.75)
Max. Cut	2.3 (0.93)	16.84 (15.91)	19.84 (14.92)	10.17 (6.18)	30.14 (15.0)
Pack.	5.16 (16.71)	−36.57 (27.27)	5.88 (20.36)	2.92 (17.37)	2.92 (17.37)
Set Cov.	1.32 (1.64)	−2.33 (49.28)	2.06 (40.74)	1.77 (82.83)	6.1 (43.31)
Load Bal.	1.29 (2.09)	−0.8 (6.05)	1.2 (15.22)	0.41 (5.51)	0.81 (5.6)
MM	−3.65 (16.01)	−14.68 (20.12)	−2.06 (17.21)	−154.73 (4.78)	4.3 (18.0)

$\mathcal{K}_{\text{eval}}$. Large IQRs reflect the heterogeneity of MILP instances in our evaluation datasets, which has been observed in prior work [46,58]. A horizontal line divides MILPs whose generators come from standard benchmarks vs. real-world MILPs. Figure 4 summarizes the results of Tables 2 and 3. It shows the distribution of improvements and highlights the trade-off between number of MILPs solved and performance for each configuration method.

LLM(5) improves runtime significantly (6–71%) over the SCIP default configuration and more modestly (0.6–30%) over the Gurobi default. Moreover, LLM-based separator configuration performs well both for problem families that are well-documented (e.g., independent set) and also for non-standard MILP formulations (e.g., Load Bal.), which are likely to be out-of-distribution. In the cold-start setting, our approach LLM(0), which does not solve any MILP instance,

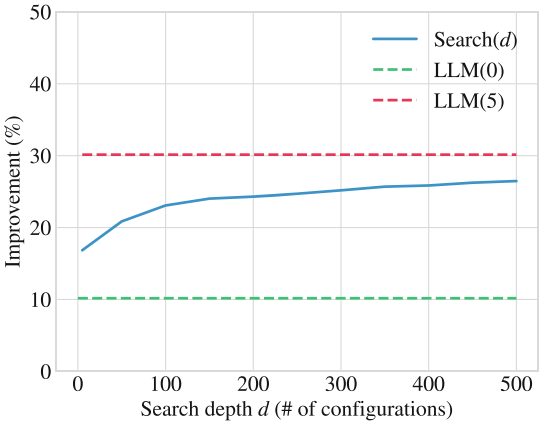


Fig. 3. Median performance gain for the SEARCH(d) procedure as a function of the number of candidate configurations, as measured on Max Cut instances. Computing SEARCH(d) requires solving 300d MILPs, while LLM(0) and LLM(5) solve 0 and 1,500 MILPs, respectively.

outperforms SCIP and Gurobi default settings in all but two problem families. Compared to existing methods, we consistently outperform the PRUNING heuristic and are often able to match the improvement of the computationally intensive SEARCH(500) procedure at a fraction of the cost: while LLM(5) can be computed by testing only 5 configurations on $\mathcal{K}_{val}$, SEARCH(500) requires trying 500 configurations for a total of 150,000 MILP solves. Figure 4 shows that our approach yields a new point on the Pareto frontier trading off computation time vs performance. At the same computational budget of five configurations, our approach LLM(5) outperforms SEARCH(5) on all problem families. Figure 3 shows the trade-off between performance and computational cost for SEARCH(d) in terms of search depth for one problem family. Lowering the search depth reduces the number of MILP solves needed but also degrades performance.

4.3 Ablations

To gauge the importance of different elements of our approach on the configuration performance, we run a sequence of ablations on a representative sub-sample of our evaluation datasets, the results of which are included in Table 4.

First, we validate that our evaluation includes settings where cutting planes are important to the performance of the MILP solver. On two of the four datasets, disabling all cutting plane separators reduces performance compared to the SCIP default, while LLM(5) improves over the default. This observation confirms that LLM(5) can find non-trivial cutting plane configurations when cutting planes can improve solver performance.

However, disabling all cutting planes has performance comparable to LLM(5) on the other two datasets. This observation highlights that in settings where

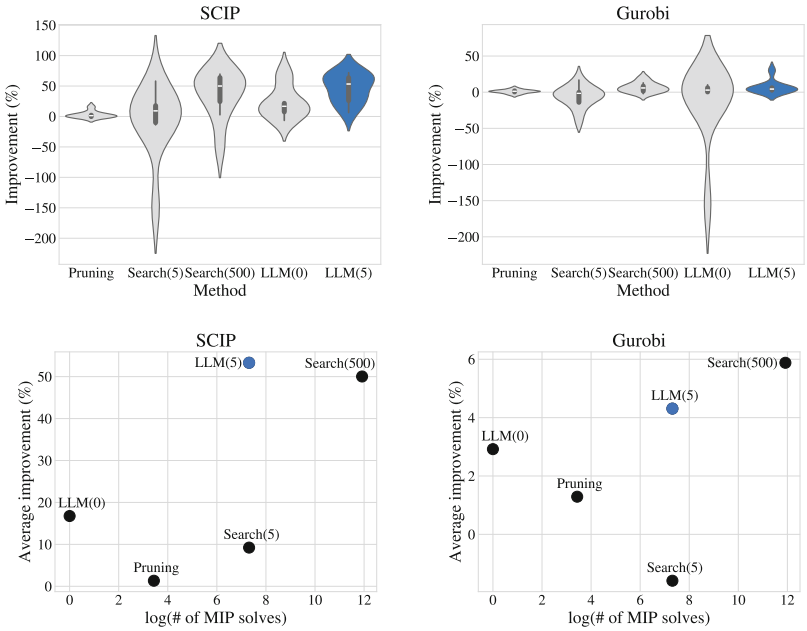


Fig. 4. Distribution of median performance gain over evaluation datasets (higher is better) for all algorithms on both solvers. Note that LLM(5) is able to achieve comparable performance to Search(500) at 1% of the computational effort.

cutting planes do not help performance, our approach finds lightweight cutting plane configurations that do not substantially slow down the solver.

We also validate that our LLM pipeline responds to the given problem description, rather than simply generating generic high-performing configurations. To test this hypothesis, we generate configurations using a random problem description that differs from the description of the problem we evaluate on. Across all datasets, using a mismatched problem description leads to worse performance.

Next, we evaluate the impact of different elements of the prompt used to generate the separator configurations, including (1) the text description of separators available, (2) the text description of the optimization model, and (3) the LaTeX description of the optimization model. Replacing the text descriptions with just the name of the cutting plane (e.g., *Clique Cutting Planes*) leads to comparable performance on half of the datasets but a notable decrease in performance for both binary packing and middle-mile consolidation network design distributions. The latter are both settings where less well-known cutting plane families (e.g., Implied Bound and Flow Cover Cuts) are used in the final configuration, indicating that providing summarized information about the separators is important for separators that are underrepresented in the LLM's training data.

Both independent set and maximum cut problems employ well-studied separator families (e.g., clique cuts) which lends further credence to this claim.

Table 4. Median (higher is better) and IQR of relative solve time improvements (%) over default SCIP separator configuration for variants of our algorithm.

	Ind. Set	**Max. Cut**	**Bin. Pack.**	**MM**
LLM(5)	71.95 (5.88)	71.01 (2.81)	38.35 (7.99)	53.3 (12.71)
Disable Cutting Planes	−14.96 (62.16)	71.25 (11.8)	30.43 (26.55)	−150 (86.12)
Different Problem Descr.	23.4 (12.77)	1.01 (8.49)	16.07 (7.48)	−17.51 (13.14)
Prompt Components				
No Sep. Text Descr.	72.27 (5.82)	71.49 (2.76)	16.85 (9.52)	9.29 (11.5)
No LaTeX Model	72.45 (5.89)	71.02 (2.87)	14.55 (10.81)	50.64 (11.38)
No Text Descr.	41.37 (12.14)	54.7 (4.52)	18.15 (9.18)	−7.4 (17.8)
Ensemble Strategies				
Average Configuration	20.65 (13.43)	71.24 (2.77)	17.52 (9.5)	−11.08 (11.22)
Mode Configuration	21.08 (13.43)	71.44 (2.79)	18.11 (9.31)	−12.63 (20.11)
Smallest Configuration	20.83 (13.46)	70.91 (2.86)	17.42 (9.44)	−4.74 (17.75)
$\mathcal{K}_{\text{eval}}$ **Size**				
$\lvert\mathcal{K}_{\text{eval}}\rvert = 1$	71.95 (5.88)	71.01 (2.81)	16.86 (7.39)	−23.27 (23.0)
$\lvert\mathcal{K}_{\text{eval}}\rvert = 5$	71.95 (5.88)	71.01 (2.81)	5.18 (5.06)	9.5 (11.32)
$\lvert\mathcal{K}_{\text{eval}}\rvert = 20$	71.95 (5.88)	71.01 (2.81)	38.35 (7.99)	53.3 (12.71)

We also investigate the impact of different ensembling strategies over the same pool of candidate LLM configurations. Across all four datasets, ensembling the pool of configurations by averaging (i.e., rounding the average number of times each cutting plane is used over configuration pool), selecting the most common configuration, and taking the smallest configuration (i.e., the configuration that turns on the fewest cutting planes) yield similar performance to our cold-start approach that selects the medoid of the largest cluster. However, selecting the medoid with the best performance on the validation dataset $\mathcal{K}_{\text{val}}$ leads to a significant improvement in performance over the other ensembling methods. Finally, we test the impact of the size of the validation dataset $\mathcal{K}_{\text{val}}$ on the performance of the approach. As expected, adding more validation instances leads to better performance for half of the problem families, as small validation datasets provide a very noisy signal of the performance of a configuration on the evaluation dataset. However, both independent set and maximum cut problems yield the same performance with only a single training instance.

5 Conclusion

In this paper, we introduced a LLM-based framework for cutting plane separator configuration in MILP solvers. To the best of our knowledge, this work is

the first to explore the use of LLMs for algorithm configuration. Our framework leverages LLMs to generate individual separator configurations based on natural language descriptions of both the optimization problem and cutting plane separators. We then apply a k-medoids clustering algorithm to consolidate the generated configurations into a single robust final configuration. Empirically, our framework demonstrates competitive performance improvements when benchmarked against state-of-the-art instance-agnostic methods for solver configuration. Importantly, it achieves these improvements while requiring significantly fewer computational resources. Additionally, our approach maintains strong performance on non-standard MILP formulations arising from real-world applications. We believe this is an intriguing demonstration of the potential for LLMs to augment traditional data used in existing ML methods with natural language data. This capability opens new avenues for integrating unstructured, descriptive information into algorithm configuration pipelines, enabling richer and more adaptive decision-making processes.

We consider this work an important first step towards LLM-based algorithm configuration for MILP solvers. Below, we outline several key challenges and potential directions for future exploration.

Instance-Specific Separator Configuration: While this work focuses on generating instance-agnostic separator configurations, tailoring configurations to specific instances could further improve performance. Combining LLM-generated configurations with existing portfolio-based algorithm configuration models like Hydra-MIP [61] or L2Sep [46] is an interesting direction for future research.

LLMs for Algorithm Configuration: This work addresses MILP solver separator configuration, but future work could investigate the efficacy of LLMs for broader configuration tasks like heuristic selection and scheduling, or parameter tuning in frameworks like constraint programming or heuristic algorithms.

Smooth Trade-Offs Between Computation and Performance: Our framework provides significant computational savings compared to previous approaches. However, more work remains to refine the trade-off between computational cost and performance. For example, adaptive methods that dynamically adjust clustering parameters or the number of configurations generated could be developed to better balance resource usage and solver efficiency for specific use cases.

Fine-Tuning with More Domain Knowledge: Domain-specific knowledge could improve the quality of LLM-generated configurations. Generic LLMs, trained on broad internet data, are adept at addressing problems that are widely studied and discussed. However, their performance may falter in specialized domains due to a lack of fine-grained expertise. Fine-tuning LLMs using curated cutting plane research or other MILP-specific studies could help reduce noise in their outputs, align their recommendations more closely with domain best practices, and ultimately improve solver performance.

Acknowledgements. MU and CL gratefully acknowledge support from the National Science Foundation (NSF) Award IIS-2233762, the Office of Naval Research (ONR) Awards N000142212825, N000142412306, and N000142312203, IBM, and the Alfred P. Sloan Foundation. EV gratefully acknowledges support from CCF-2338226. AW was supported in part by a National Defense Science & Engineering Graduate (NDSEG) fellowship.

References

1. Abdalkareem, Z.A., Amir, A., Al-Betar, M.A., Ekhan, P., Hammouri, A.I.: Healthcare scheduling in optimization context: a review. Heal. Technol. **11**, 445–469 (2021)
2. Achiam, J., et al.: GPT-4 technical report. arXiv preprint arXiv:2303.08774 (2023)
3. Achterberg, T.: What's new in Gurobi 9.0. Webinar Talk (2019). https://www.gurobi.com/wp-content/uploads/2019/12/Gurobi-90-Overview-Webinar-Slides-1.pdf
4. Achterberg, T., Wunderling, R.: Mixed integer programming: analyzing 12 years of progress. In: Facets of Combinatorial Optimization: Festschrift for Martin grötschel, pp. 449–481. Springer (2013)
5. AhmadiTeshnizi, A., Gao, W., Udell, M.: OptiMUS: optimization modeling using MIP solvers and large language models. arXiv preprint arXiv:2310.06116 (2023)
6. AhmadiTeshnizi, A., Gao, W., Udell, M.: OptiMUS: scalable optimization modeling with (MI)LP solvers and large language models. arXiv preprint arXiv:2402.10172 (2024)
7. Ansótegui, C., Malitsky, Y., Samulowitz, H., Sellmann, M., Tierney, K., et al.: Model-based genetic algorithms for algorithm configuration. In: IJCAI, pp. 733–739 (2015)
8. Astorga, N., Liu, T., Xiao, Y., van der Schaar, M.: Autoformulation of mathematical optimization models using LLMs. arXiv preprint arXiv:2411.01679 (2024)
9. Balcan, M.F., DeBlasio, D., Dick, T., Kingsford, C., Sandholm, T., Vitercik, E.: How much data is sufficient to learn high-performing algorithms? Generalization guarantees for data-driven algorithm design. In: Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing, pp. 919–932 (2021)
10. Balcan, M.F., Prasad, S., Sandholm, T., Vitercik, E.: Sample complexity of tree search configuration: cutting planes and beyond. In: Conference on Neural Information Processing Systems (NeurIPS) (2021)
11. Balcan, M.F., Sandholm, T., Vitercik, E.: Generalization in portfolio-based algorithm selection. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35-14, pp. 12225–12232 (2021)
12. Balcan, M.F.F., Prasad, S., Sandholm, T., Vitercik, E.: Structural analysis of branch-and-cut and the learnability of Gomory mixed integer cuts. In: Conference on Neural Information Processing Systems (NeurIPS), vol. 35, pp. 33890–33903 (2022)
13. Bansal, H., Hosseini, A., Agarwal, R., Tran, V.Q., Kazemi, M.: Smaller, weaker, yet better: training LLM reasoners via compute-optimal sampling. arXiv preprint arXiv:2408.16737 (2024)
14. Bengio, Y., Lodi, A., Prouvost, A.: Machine learning for combinatorial optimization: a methodological tour d'horizon. Eur. J. Oper. Res. **290**(2), 405–421 (2021)

15. Bixby, R., Rothberg, E.: Progress in computational mixed integer programming-a look back from the other side of the tipping point. Ann. Oper. Res. **149**(1), 37 (2007)
16. Caceres-Cruz, J., Arias, P., Guimarans, D., Riera, D., Juan, A.A.: Rich vehicle routing problem: survey. ACM Comput. Surv. **47**(2) (2014)
17. Chen, H., Constante-Flores, G.E., Li, C.: Diagnosing infeasible optimization problems using large language models. arXiv preprint arXiv:2308.12923 (2023)
18. Chen, X., Lin, M., Schärli, N., Zhou, D.: Teaching large language models to self-debug. arXiv preprint arXiv:2304.05128 (2023)
19. Cobbe, K., et al.: Training verifiers to solve math word problems. arXiv preprint arXiv:2110.14168 (2021)
20. Conforti, M., Cornuejols, G., Zambelli, G.: Integer Programming. Springer (2014)
21. CPLEX User's Manual: IBM ILOG CPLEX optimization studio. Version **12**(1987–2018), 1 (1987)
22. Custode, L.L., Caraffini, F., Yaman, A., Iacca, G.: An investigation on the use of large language models for hyperparameter tuning in evolutionary algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, pp. 1838–1845 (2024)
23. Deudon, M., Cournut, P., Lacoste, A., Adulyasak, Y., Rousseau, L.M.: Learning heuristics for the TSP by policy gradient. In: Integration of AI and OR Techniques in Constraint Programming (2018)
24. Dey, S.S., Molinaro, M.: Theoretical challenges towards cutting-plane selection. Math. Program. **170**, 237–266 (2018)
25. Deza, A., Khalil, E.B., Fan, Z., Zhou, Z., Zhang, Y.: Learn2Aggregate: supervised generation of Chvatal-Gomory cuts using graph neural networks. arXiv preprint arXiv:2409.06559 (2024)
26. Gasse, M., et al.: The machine learning for combinatorial optimization competition (ML4CO): results and insights. In: NeurIPS 2021 Competitions and Demonstrations Track, pp. 220–231. PMLR (2022)
27. Gasse, M., Chételat, D., Ferroni, N., Charlin, L., Lodi, A.: Exact combinatorial optimization with graph convolutional neural networks. In: Advances in Neural Information Processing Systems, vol. 32 (2019)
28. Greening, L.M., Dahan, M., Erera, A.L.: Lead-time-constrained middle-mile consolidation network design with fixed origins and destinations. Transp. Res. Part B: Methodol. **174**, 102782 (2023)
29. Guaje, O., Deza, A., Kazachkov, A.M., Khalil, E.B.: Machine learning for optimization-based separation: the case of mixed-integer rounding cuts. arXiv preprint arXiv:2408.08449 (2024)
30. Gupta, P., et al.: Lookback for learning to branch. arXiv preprint arXiv:2206.14987 (2022)
31. Huang, L., et al.: A survey on hallucination in large language models: principles, taxonomy, challenges, and open questions. ACM Trans. Inf. Syst. (2023)
32. Huang, W., Huang, T., Ferber, A.M., Dilkina, B.: Distributional MIPLIB: a multi-domain library for advancing ml-guided MILP methods (2024). https://www.arxiv.org/abs/2406.06954
33. Hurst, A., et al.: GPT-4o system card. arXiv preprint arXiv:2410.21276 (2024)
34. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, 17–21 January 2011, Selected Papers 5, pp. 507–523. Springer (2011)

35. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: an automatic algorithm configuration framework. J. Artif. Intell. Res. **36**, 267–306 (2009)
36. Jiang, Y., Cao, Z., Wu, Y., Song, W., Zhang, J.: Ensemble-based deep reinforcement learning for vehicle routing problems under distribution shift. In: Advances in Neural Information Processing Systems, vol. 36 (2024)
37. Kadioglu, S., Malitsky, Y., Sellmann, M., Tierney, K.: ISAC–instance-specific algorithm configuration. In: ECAI 2010, pp. 751–756. IOS Press (2010)
38. Khalil, E., Dai, H., Zhang, Y., Dilkina, B., Song, L.: Learning combinatorial optimization algorithms over graphs. In: Advances in Neural Information Processing Systems, vol. 30 (2017)
39. Khalil, E., Le Bodic, P., Song, L., Nemhauser, G., Dilkina, B.: Learning to branch in mixed integer programming. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 30-1 (2016)
40. Khalil, E.B., Morris, C., Lodi, A.: MIP-GNN: a data-driven framework for guiding combinatorial solvers. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 36-9, pp. 10219–10227 (2022)
41. Kojima, T., Gu, S.S., Reid, M., Matsuo, Y., Iwasawa, Y.: Large language models are zero-shot reasoners. In: Advances in Neural Information Processing Systems, vol. 35, pp. 22199–22213 (2022)
42. Kool, W., van Hoof, H., Welling, M.: Attention, learn to solve routing problems! In: International Conference on Learning Representations (2018)
43. Lawless, C., et al.: "I want it that way": enabling interactive decision support using large language models and constraint programming. ACM Trans. Interact. Intell. Syst. **14**(3), 1–33 (2024)
44. Lester, B., Al-Rfou, R., Constant, N.: The power of scale for parameter-efficient prompt tuning. arXiv preprint arXiv:2104.08691 (2021)
45. Li, B., Mellou, K., Zhang, B., Pathuri, J., Menache, I.: Large language models for supply chain optimization. arXiv preprint arXiv:2307.03875 (2023)
46. Li, S., Ouyang, W., Paulus, M., Wu, C.: Learning to configure separators in branch-and-cut. In: Advances in Neural Information Processing Systems, vol. 36 (2024)
47. Li, X.L., Liang, P.: Prefix-tuning: optimizing continuous prompts for generation. arXiv preprint arXiv:2101.00190 (2021)
48. Liu, S., Tang, K., Yao, X.: Automatic construction of parallel portfolios via explicit instance grouping. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33-01, pp. 1560–1567 (2019)
49. Mahammadli, K., Ertekin, S.: Sequential large language model-based hyper-parameter optimization. arXiv preprint arXiv:2410.20302 (2024)
50. Nakano, R., et al.: WebGPT: browser-assisted question-answering with human feedback. arXiv preprint arXiv:2112.09332 (2021)
51. Paulus, M.B., Zarpellon, G., Krause, A., Charlin, L., Maddison, C.: Learning to cut by looking ahead: cutting plane selection via imitation learning. In: International Conference on Machine Learning, pp. 17584–17600. PMLR (2022)
52. Pochet, Y., Wolsey, L.A.: Production Planning by Mixed Integer Programming, vol. 149. Springer (2006)
53. Pryzant, R., Iter, D., Li, J., Lee, Y.T., Zhu, C., Zeng, M.: Automatic prompt optimization with "gradient descent" and beam search. arXiv preprint arXiv:2305.03495 (2023)
54. Ramamonjison, R., et al.: Augmenting operations research with auto-formulation of optimization models from problem descriptions. arXiv preprint arXiv:2209.15565 (2022)

55. Tang, Y., Agrawal, S., Faenza, Y.: Reinforcement learning for integer programming: learning to cut. In: International Conference on Machine Learning, pp. 9367–9376. PMLR (2020)
56. Tang, Z., et al.: ORLM: training large language models for optimization modeling (2024). https://arxiv.org/abs/2405.17743
57. Tsouros, D., Verhaeghe, H., Kadıoğlu, S., Guns, T.: Holy grail 2.0: from natural language to constraint models. arXiv preprint arXiv:2308.01589 (2023)
58. Wang, Z., et al.: Learning cut selection for mixed-integer linear programming via hierarchical sequence model. In: The Eleventh International Conference on Learning Representations (2023)
59. Xiao, Z., et al.: Chain-of-experts: when LLMs meet complex operations research problems. In: The Twelfth International Conference on Learning Representations (2023)
60. Xu, L., Hoos, H., Leyton-Brown, K.: Hydra: automatically configuring algorithms for portfolio-based selection. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 24-1, pp. 210–216 (2010)
61. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Hydra-MIP: automated algorithm configuration and selection for mixed integer programming. In: RCRA Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion at the International Joint Conference on Artificial Intelligence (IJCAI), pp. 16–30 (2011)
62. Yang, Z., et al: OptiBench meets ReSocratic: measure and improve LLMs for optimization modeling. arXiv preprint arXiv:2407.09887 (2024)
63. Ye, H., Wang, J., Cao, Z., Liang, H., Li, Y.: DeepACO: neural-enhanced ant systems for combinatorial optimization. In: Advances in Neural Information Processing Systems, vol. 36 (2024)
64. Yuan, A., Coenen, A., Reif, E., Ippolito, D.: WordCraft: story writing with large language models. In: Proceedings of the 27th International Conference on Intelligent User Interfaces, pp. 841–852 (2022)
65. Zhang, B., Haddow, B., Birch, A.: Prompting large language model for machine translation: a case study. In: International Conference on Machine Learning, pp. 41092–41110. PMLR (2023)
66. Zhang, M.R., Desai, N., Bae, J., Lorraine, J., Ba, J.: Using large language models for hyperparameter optimization. arXiv preprint arXiv:2312.04528 (2023)
67. Zhang, Y., et al.: Siren's song in the AI ocean: a survey on hallucination in large language models. arXiv preprint arXiv:2309.01219 (2023)

A Dynamic Programming Approach for the Job Sequencing and Tool Switching Problem

Emma Legrand[1(✉)], Vianney Coppé[3], Daniele Catanzaro[2], and Pierre Schaus[1]

[1] Department of Computing Science Engineering, Université Catholique de Louvain, Place Sainte Barbe 2, 1348 Louvain-la-Neuve, Belgium
{emma.legrand,pierre.schaus}@uclouvain.be

[2] Center for Operations Research and Econometrics, Université Catholique de Louvain, Voie du Roman Pays 34, 1348 Louvain-la-Neuve, Belgium
daniele.catanzaro@uclouvain.be

[3] Hexaly, 251 Boulevard Pereire, 75017 Paris, France

Abstract. We present a new dynamic programming-based exact solution algorithm for the *Job Sequencing and Tool Switching Problem* (JS-TSP), a combinatorial optimization problem originating from manufacturing systems and encompassing the Traveling Salesman Problem as a special case. We propose a new family of lower bounds for the optimal solution to the problem, which are provably tighter than existing bounds in the literature and enhance both solution quality and pruning efficiency. We propose the use of $A*$ and its anytime variants to explore the solution space of the problem as well as a specific data structure, called *FreeTools*, both to keep track of the state information and to compute incremental costs throughout the implicit search efficiently. Extensive computational experiments show that the presented approach brings significant performance improvements over state-of-the-art methods for the JS-TSP, including branch-and-bound and integer linear programming formulations.

Keywords: combinatorial optimization · job sequencing · tool switching · branch and bound · A* · dynamic programming

1 Introduction

Consider a flexible machine $\mathcal{M}$ that can be configured with any subset of maximum c tools from a given set $T = \{0, 1, 2, \ldots, m-1\}$ of $m \geq 3$ tools. We refer to c as the *capacity* of $\mathcal{M}$. In addition, consider a set $J = \{1, 2, \ldots n\}$ of $n \geq 3$ jobs, that must be processed sequentially on $\mathcal{M}$. Each job $j \in J$ requires a subset $t(j) \subseteq T$ of tools and is such that $|t(j)| \leq c$. Whenever a job must be processed on $\mathcal{M}$, one needs to decide which tool, from among those currently present on $\mathcal{M}$, is going to stay and which, instead, needs to be replaced. This

G. Tack (Ed.): CPAIOR 2025, LNCS 15763, pp. 70–85, 2025.
https://doi.org/10.1007/978-3-031-95976-9_5

operation is commonly referred to as *tool switching*. For example, by referring to the job sequence reported in Table 1, two tool switches are involved in the transition from job 3 to job 4, in particular, tools 6 and 8 are going to stay in $\mathcal{M}$ while tools 1 and 5 must be replaced by tools 0 and 4. The *Job Sequencing and Tool Switching Problem* (JS-TSP) consists of determining the job sequence that minimizes the total number of tool switches required to process the jobs in J. For example, Table 2 shows the job sequence that minimizes the overall number of tool switches for the instance of the JS-TSP shown in Table 1. The optimal value for this instance is equal to 11. The tools underlined are those that give rise to a tool switch, while the circled tools have the special property of being not immediately necessary to process a specific job j in the sequence, but needed for processing the jobs subsequent to j. This property, firstly introduced by [16], gives rise to a greedy algorithm, commonly referred to as the *Keep Tool Needed Soonest* (KTNS) algorithm, that computes in polynomial-time the optimal tool switches for a fixed job sequence.

Table 1. Example of an instance with 9 tools and 6 jobs with a capacity of 4.

Jobs	1	2	3	4	5	6
Tools	0	1	1	0	5	3
	1	3	5	4	6	5
	2	4	6	6	7	
			8	8		
Magazine capacity: 4						

Table 2. Example of an optimal solution for the instance 1 for which the optimal value is equal to 11.

Jobs	1	2	4	3	5	6
Tools	$\underline{0}$	⓪	0	$\underline{1}$	6	$\underline{3}$
	$\underline{1}$	1	$\underline{6}$	6	$\underline{7}$	5
	$\underline{2}$	$\underline{3}$	$\underline{8}$	8	5	
		$\underline{4}$	4	$\underline{5}$		
Magazine capacity: 4						

The first applications of the JS-TSP appeared in the literature already in 1966 [2]. The problem, however, was formalized only in 1987 by Tang and Denardo [16], who first proved also its general $\mathcal{NP}$-hardness. Specifically, the authors observed that if the start and stop states of the job processing sequence are represented by a dummy job having index 0 and $t(0) = \emptyset$, then any instance of the JS-TSP that satisfies $|t(j)| = c$, for all $j \in J \setminus \{0\}$, can be seen as an instance of the (symmetric) *Traveling Salesman Problem* (TSP) [1,8] in which the weight associated with the edge (i, j) of the graph is equal to $|t(j)| - |t(i) \cap t(j)|$. Tang and Denardo also proposed the first *Integer Linear Programming* (ILP) formulation for the problem known in the literature, which unfortunately proved to be rather disappointing in practice, mostly due to the poor lower bound provided by its linear programming relaxation [13]. This fact motivated the scientific community to search for improved exact solution algorithms of practical use. Laporte et al. [13] proposed the first ILP formulation able to improve upon Tang and Denardo's one as well as a *Branch-and-Bound* (B&B) algorithm based

on dynamic programming. The ILP formulation solved only instances of JS-TSP containing 10 jobs and tools within the reference time of 1 h. Ghiani et al. [9] proposed an alternative (nonlinear) formulation for the problem able to improve upon Laporte et al.'s one when the ratio between the minimum number of tools required by jobs and the magazine capacity is between 60% and 90%. Bessiere et al. [4] studied a similar problem using Constraint Programming, and Catanzaro et al. [5] proposed new ILP formulations, with the best outperforming earlier ones but struggling to solve instances with 15 jobs and 20 tools within the reference time. At present, the state-of-the-art exact solution algorithm for the JS-TSP is still Laporte et al.'s B&B algorithm, which proves capable of solving instances of the problem having up to 25 jobs and 25 tools within the reference time. This method combines a depth-first recursive enumeration of job sequences with combinatorial lower bounds on the optimal solution to the problem, which proved generally poorer than the ones proposed by Catanzaro et al. [5] but that are very fast to compute. Here, we extend the result discussed in [13] by:

- introducing a new family of combinatorial lower bounds for the optimal solution to the JS-TSP, which are provably tighter than those described in [13];
- proposing the use of A* and its anytime variant to explore the solution space of the problem; in particular, we introduce a specific data structure, called *FreeTools*, to keep track of the state information during the implicit search and to compute the incremental cost of partial job sequences efficiently.

Extensive computational experiments show that the solution algorithm presented in the next sections significantly outperforms state-of-the-art methods for the JS-TSP, including the B&B and ILP formulations discussed in [5,13].

The article is organized as follows. In Sect. 2, we briefly review, for the sake of completeness, Laporte et al.'s B&B algorithm, the KTNS algorithm, and known lower bounds for the optimal solution to the problem; in addition, we present a new lower bound which is provably tighter than the previous ones. In Sect. 3, we introduce an alternative version of the KTNS algorithm, called *Lazy KTNS*, and a specific representation of a state of the search space. The Lazy KTNS allows to improve the efficiency of computing the incremental cost of partial job sequences throughout the implicit search, while the specific state representation allows to avoid the exploration of redundant nodes in the search space. In Sect. 4, we analyze the results obtained when comparing the performance of the new exact solution algorithm versus state-of-the-art methods on a set of benchmark instances of the problem, including Laporte et al.'s B&B algorithm and the ILP formulations discussed above. Finally, in Sect. 5, we provide some preliminary conclusions and outline possible future research directions.

2 Brief Recalls from the Literature on the JS-TSP

Starting from an initially empty job sequence S, Laporte et al.'s B&B algorithm [13] enumerates all the permutations of J by augmenting S in a depth-first fashion, i.e., by appending one job at a time to S at each node of the search

space. Thus, each node of the search space is defined by the sequence S. Let z^* denote the optimal solution to an instance of the JS-TSP and, for a given partial sequence S, let R denote the subset of the remaining jobs to process, i.e., $R = J \setminus S$. Then, Laporte et al.'s B&B algorithm attempts to prune a generic node of the search space by computing a lower bound $lb = \hat{z} + \tilde{z}$ for z^*, where $\hat{z}$ is a lower bound on the number of tool switches associated with S and $\tilde{z}$ is a lower bound on the number of tool switches associated with R, respectively. The choice of the next job to appended to S is determined by the following simple heuristic: if S is empty, the job must have (i) the greatest number of tools, and (ii) these tools must figure from among the most frequently used ones. If $S \neq \emptyset$, the next job is the one sharing the most tools with the last job appended.

Algorithm 1: Evaluate a job sequence S according to the *Keep Tool Needed Soonest* (KTNS) policy.

```
1  Function KTNS
       Input  : S → a (partial) job sequence encoded as a list of integers
       Output: cost→ an integer encoding the cost of S
       // computes next[i][t] ← min{{j ∈ [i..|S|] | t ∈ t(S[j])} ∪ {|S| + 1}}
2      next[i][t] ← |S| + 1   ∀(i, t) ∈ [1..|J|] × [1..|S|] ;
3      for i from |S| − 1 to 1 do
4          for t ∈ T do
5              if t ∈ t(S[i]) then
6                  next[i][t] ← i ;
7              else
8                  next[i][t] ← next[i + 1][t] ;
9      t_1 ← t(S[1]) ;                    // tools currently set on the machine
10     cost ← |t_1| ;
11     for i from 2 to |S| do
12         t_2 ← t(S[i]);
13         cost ← cost + |t_2 \ t_1| ;
14         t_1 ← t_1 ∪ t_2;
15         while |t_1| > c do
16             t ← arg max_{k∈t_1} {next[i][k]} ;
17             t_1 ← t_1 \ {t};
18     return cost ;
```

Given a (partial) job sequence S, one can compute $\hat{z}$ by using the KTNS algorithm outlined in Algorithm 1. Specifically, the KTNS algorithm relies on the upcoming jobs to decide the current configuration of $\mathcal{M}$. The algorithm first initializes a matrix that computes, for each tool and each position, the soonest next position in the sequence where this tool appears (see lines 2–8 of Algorithm 1). Then, given the set of tools currently set in $\mathcal{M}$, the algorithm

starts with the tools required by the first job $S[1]$. New tools required by the next job induce a cost of 1; and after adding the tools of the next job, the tools less urgently required in the future are removed based on their values in the precomputed `next` table to keep the capacity constraint satisfied. The time complexity of the KTNS algorithm is $\mathcal{O}(n^2 \cdot m)$.

2.1 Lower Bounds on the Optimal Solution to the JS-TSP

We present now three possible combinatorial lower bounds on z^* that can be obtained by specifying a way to compute $\tilde{z}$. We consider in particular three alternative bounds, denoted by $\tilde{z}_1$, $\tilde{z}_2$, and $\tilde{z}_3$, the first two of which can be derived by [6] and [13], respectively, while the third one is new. We start by observing that if $\mathcal{M}$ is set up with c tools at a given state of the job processing sequence, then any tool t which is not currently present in $\mathcal{M}$ and that is needed to process any of the remaining jobs in $R = J \setminus S$ will necessarily entail a tool switch. The following lower bound captures this rationale:

$$\tilde{z}_1 = \left| \bigcup_{i \in R} t(i) \right| - \min \left\{ c, \left| \bigcup_{j \in S} t(j) \right| \right\}. \tag{1}$$

Example 1. Consider the JS-TSP instance shown in Fig. 1 and assume that $\mathcal{M}$ has already processed the job sequence $S = \langle 1, 3, 4 \rangle$. Consider the set $R = J \setminus S = \{2, 5, 6\}$. Then,

$$\tilde{z}_1 = \left| \bigcup_{i \in R} t(i) \right| - \min \left\{ c, \left| \bigcup_{j \in S} t(j) \right| \right\} = 6 - \min\{4, 7\} = 2.$$

The second lower bound, i.e., $\tilde{z}_2$, is obtained by considering a complete undirected weighted graph $G = (V, E)$ having $V = R$ and edge weight $w_{ij} = \max\{|t(i) \cup t(j)| - c, 0\}$ for each $e = (i, j) \in E$. Let $T = (V, E_T)$ denote a *Minimum Spanning Tree* (MST) of G, and let l denote the last job in S. The length of this spanning tree plus the cost of the cheapest edge to connect it to l is the lower-bound $\tilde{z}_2$:

$$\tilde{z}_2 = \sum_{(i,j) \in E_T} w_{ij} + \min_{i \in R} w_{li}. \tag{2}$$

Proof. A lower bound on the cost induced by scheduling j immediately after i is given by: $w_{ij} = |t(j) \setminus t(i)| - (c - |t(i)|)$. Here, $|t(j) \setminus t(i)|$ represents the new tools required for j, but up to $c - |t(i)|$ of these tools may already have been set due to the previous jobs scheduled before i. Simplifying, this formula yields $w_{ij} = |t(j) \setminus t(i)| + |t(i)| - c = |t(i) \cup t(j)| - c$. Note that this cost is symmetric, as $w_{ij} = w_{ji}$. Now, observe that the length of the shortest Hamiltonian path on G constitutes a lower bound for z^*. In turn, the length of the MST constitutes a lower bound on the length of the shortest Hamiltonian path as the MST relaxes the degree constraint on the internal vertices of the Hamiltonian path. Thus, adding the cost of the cheapest edge connecting the last job of S to any vertex of T provides a lower bound for z^*. □

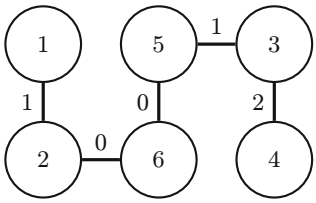


Fig. 1. Example of a Minimum Spanning Tree at the root node.

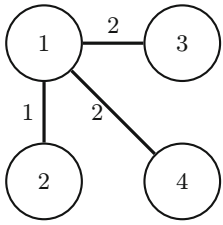


Fig. 2. Example of a Minimum Spanning Tree where $R = \{1, 2, 3, 4\}$.

It is worth noting here that, in general, it is not possible to determine which lower bound between $\tilde{z}_1$ and $\tilde{z}_2$ is the tightest, as shown in the following example.

Example 2. Consider again the JS-TSP instance shown in Fig. 1 and assume that $\mathcal{M}$ has not processed any job yet (i.e., $S = \emptyset$, $R = J$). In this case, we have that $\tilde{z}_1 = 9 - \min\{4, 0\} = 9$. To compute $\tilde{z}_2$, it is sufficient to compute the length of the MST for $G = (V, E)$ having $V = J$ with $\min_{i \in R} w_{li} = 0$ as there is no processed job. For example, a MST T for G is shown in Fig. 1 and its length is equal to $w_T = 1 + 0 + 0 + 1 + 2 = 4$. Hence, we have that $\tilde{z}_1 > \tilde{z}_2$. Now, consider the case in which $S = \langle 5, 6 \rangle$ and $R = \{1, 2, 3, 4\}$. Then, we have that $\tilde{z}_1 = 8 - 4 = 4$. By considering the MST for $G = (V, E)$, with $V = R$, shown in Fig. 2, we have that $w_T = 1 + 2 + 2 = 5$, $\min_{i \in R} w_{li} = w_{62} = 0$ and that $\tilde{z}_2 = 5$. In this case, $\tilde{z}_1 < \tilde{z}_2$ holds.

A third lower bound $\tilde{z}_3$ can be obtained by combining the rationale at the core of $\tilde{z}_1$ and $\tilde{z}_2$. As for $\tilde{z}_2$, an MST $T = (V, E_T)$ is first computed on the graph of the remaining jobs with the same edge weights. Let $\{e_1, \ldots, e_{k-1}\} \subseteq E_T$ denote a set of $k - 1$ edges of the MST. Removing these edges induces partitioning the remaining jobs R into k connected components denoted as $C_1, \ldots, C_k$. Then, the following quantity

$$\tilde{z}_3 = \sum_{i=1}^{k-1} w_{e_i} + \sum_{i=1}^{k} \max\left(0, |\cup_{j \in C_i} t(j)| - c\right) + \min_{i \in R} w_{li} \tag{3}$$

is a lower bound for z^*. The proof of the validity of $\tilde{z}_3$ is omitted here. Still, it can be easily derived from that of $\tilde{z}_2$ by aggregating the nodes in a component and by replacing the length of a spanning tree in each component with $\tilde{z}_1$. Intuitively, the proof of this bound follows the reasoning used in establishing the second lower bound. In this case, the key difference is that an MST node may represent a set of jobs. Since $\tilde{z}_1$ is a lower bound for a set of jobs, it can also be used as a lower bound on the cost associated with this node.

Note that when $k = |R| - 1$, each node corresponds to a component and therefore $\tilde{z}_3 = \tilde{z}_2$ since all the edges of the spanning tree contribute to the cost. When $k = 1$, there is only one component R and $\tilde{z}_3 = \tilde{z}_1 + \min_{i \in R} w_{li}$. The lower bound $\tilde{z}_3$ is a family of bounds since (i) several minimum spanning trees might

Table 3. Example of the computation of the lower bound defined in Eq. (4) for the MST in Fig. 2.

C_i	C_j	$\tilde{z}_3(C_1) + \tilde{z}_3(C_2) + w_e$	$\lvert \cup_{j \in (C_1 \cup C_2)} t(j) \rvert - c$	$\tilde{z}_3(C_i \cup C_j)$
1	2	$0 + 0 + 1 = 1$	1	1
$\{1, 2\}$	3	$1 + 0 + 2 = 3$	4	4
$\{1, 2, 3\}$	4	$4 + 0 + 2 = 6$	4	6

exist for G and (ii) up to $2^{|R|-1}$ components exist for a given minimum spanning tree. Since we cannot easily guess the best combination of spanning tree and components to get the tightest possible value for $\tilde{z}_3$, we use a heuristic to define the component along the computation of the Kruskal algorithm without the computational overhead. At each step, Kruskal adds an edge e with weight w_e and thus connects two components C_1 and C_2. Let $\tilde{z}_3(C)$ denote the contribution to the bound for a component C. Then, we can set

$$\tilde{z}_3(C_1 \cup C_2) = \max(\tilde{z}_3(C_1) + \tilde{z}_3(C_2) + w_e, \lvert \cup_{j \in (C_1 \cup C_2)} t(j) \rvert - c). \tag{4}$$

By doing so, the bound $\tilde{z}_3$ that we compute is at least as tight as $\tilde{z}_2$ and is obtained with the same time complexity, assuming constant-time computation for union and size operations on sets. This assumption holds for most challenging open instances with fewer than 64 tools, which can be represented as a bitset within a single memory word.

Example 3. Consider the instance of the JS-TSP shown in Fig. 1 and the MST shown in Fig. 2. Assume that $\mathcal{M}$ has already processed (in this order of appearance) the set of jobs $S = \langle 5, 6 \rangle$ inducing an overall number of tool switches equal to $\hat{z} = 4$. To compute $\tilde{z}_3$, we apply recursively Eq. (4). The result of this recursion is shown in Table 3 and the value obtained is $\tilde{z}_3 = 6 > \tilde{z}_2 = 5 > \tilde{z}_1 = 4$.

3 A Graph Search Algorithm

In this section, we present a new dynamic programming-based exact solution algorithm for the JS-TSP that exploits both strategies to reduce the number of nodes explored in the search space and a lazy version of the KTNS algorithm that allows to compute incrementally the cost of a partial sequence so as to keep track of the set of tools that can be reused for free later in the sequence.

3.1 A Preliminary Observation on the Search Space

The number of nodes explored in the search space can be significantly reduced by identifying, during the implicit search, partial sequences that lead to equivalent solutions. The next lemma provides an example of how this situation may occur.

Lemma 1. *Consider two non-empty sequences S_1 and S_2 (i) defined on the same set of jobs (S_1 is a permutation of S_2), (ii) such that their last job is identical $last(S_1) = last(S_2)$ and such that (iii) this last job requires c tools, then any sequence on $R = J \setminus S_1 = J \setminus S_2$ induces the same overall cost to complete either S_1 or S_2.*

The example provided in Lemma 1 is unlikely to occur frequently in practice, as more challenging instances typically do not involve jobs that saturate a machine's capacity. Nonetheless, it may be valuable to explore more complex state representations that facilitate efficient testing of whether two partial sequences—defined on the same subset of jobs and ending with the same job—can lead to equivalent solutions.

Example 4. Consider three jobs: $t(1) = \{1\}$, $t(2) = \{2\}$, and $t(3) = \{3, 4\}$, out of n jobs to be scheduled on a machine with capacity $c = 4$. Now, take the two partial sequences $S_1 = \langle 1, 2, 3\rangle$ and $S_2 = \langle 2, 1, 3\rangle$. It is easy to see that the same sub-sequence can be used to complete S_1 and S_2 optimally with the remaining $n-3$ jobs, as the tools $\{1, 2\}$ can remain on the machine at step 3 in both cases.

Building on this idea, we present a lazy version of the KTNS algorithm in the next section.

3.2 A Lazy Version of the KTNS Algorithm

The *lazy version of KTNS* (Lazy-KTNS for short) computes incrementally the cost of a partial sequence and keeps track of the set of tools that can be reused for free later in the sequence. The decision to keep an unnecessary tool on the machine at each step is deferred until it eventually becomes required on a further step. If the KTNS algorithm can be seen as a forward-looking greedy, the Lazy-KTNS algorithm can be seen as a backward-looking one.

Lazy-KTNS algorithm uses a specific data structure—called *FreeTools* and presented in Algorithm 2—to collect the set of tools that can be used "for free". Using a tool from this set means deciding it has remained on the machine since its last appearance in the partial sequence until the current time slot, as the standard KTNS algorithm would have done.

The *FreeTools* data structure encodes cardinality constraints on the set of active tools, coming from the limited capacity c of the machine. It is represented as a sequence of sets of tools $\langle F_1, \ldots, F_{size}\rangle$ with $size < c$. The generic F_k represents the set of tools removed from $\mathcal{M}$ at a previous step when the capacity slack was exactly k. Furthermore, the invariant on this sequence is that the sets are pairwise disjoint, and the tools in F_k were set on the machine at a time slot after the tools in $F_1, \ldots, F_{k-1}$. The implicit *FreeTools* semantic of the cardinality constraints on the tools that can be re-used freely reads as the conjunction of the following constraints:

- At most one tool from F_1 can be used for free, and
- At most two tools from $F_1 \cup F_2$ can be used for free,

Algorithm 2: Class FreeTools

```
1  F[k] : set ← ∅ , ∀ k ∈ [1..c − 1] ;
2  size ← 0
3  Method contains(t: int): int
4      foreach k from 1 to size − 1 do
5          if t ∈ F[k] then
6              return k;
7      return −1;
8  Method add(k: int, free: set): int
9      F[k] ← F[k] ∪ free ;
10     size ← max(size, k + 1) ;
11 Method remove(k: int, t: int): int
12     F[k] ← F[k] \ {t} ;
13     F[k − 1] ← F[k − 1] ∪ F[k] ;
14     foreach l from k to size − 1 do
15         F[l] ← F[l + 1] ;
16     F[size − 1] ← ∅ ;
17     size ← size − 1;
18 Method removeFrom(k: int): int
19     foreach l from k + 1 to size − 2 do
20         F[k] ← F[k] ∪ F[l] ;
21         F[l] ← ∅ ;
22     size ← k + 1;
```

– ...
– At most $size$ tools from $F_1 \cup \cdots \cup F_{size}$ can be used for free.

To ensure that this property is maintained, a tool used for free must call the method `remove`. When calling this method for a tool t present in F_k, the subsequent sets are left-shifted to reflect the update on the cardinality constraint, as can be seen in the body of the `remove` method.

Algorithm 3 outlines the Lazy-KTNS algorithm. Similarly to its non-lazy version described in Algorithm 1, Lazy-KTNS computes the optimal cost for a fixed partial job sequence S. In addition, Lazy-KTNS makes heavy use of the $FreeTools$ data structure. The tools in $t(i) \setminus t(i-1)$ are the new ones to be scheduled. Each such tool t can be used for free if it appears in $FreeTools$; if not, it must incur a cost of 1. In case it appears in $FreeTools$, then $FreeTools$ must be updated to lazily decide to keep a tool previously on $\mathcal{M}$ from a past job till step i. For example, suppose that we found it in F_k. Then, the slack is reduced by 1 for all the jobs scheduled after. This amounts to shifting all the sets after F_k by one position to the left since those tools were added afterward. This is the operation `remove` in Algorithm 2. The tools that can be used later for free

Algorithm 3: Evaluate a job sequence S using Lazy-KTNS

```
1  Function Lazy-KTNS
       Input  : S → a (partial) job sequence encoded as a list of integers
       Output: cost→ an integer encoding the cost of S
       Data   : next→ a |J| × |S| matrix that keeps track of which tool is needed
                first in the current sequence S
                t1, t2 → support sets of integers used to encode subsets of tools
2      cost ← |t(S[1])| ;
3      F ← new FreeTools() ;
4      for i from 2 to |S| do
5          new ← t(S[i]) \ t(S[i − 1]) ;    // tools not present on the machine
6          for t ∈ new do
7              k ← F.contains(t) ;
8              if k > 0 then          // can t be reused from previous jobs?
9                  F.remove(k, t) ;
10             else
11                 cost ← cost + 1 ;
12         slack ← c − |t(S[i])| ;
13         if slack > 0 then              // slots available to the machine?
14             if slack < F.size then
15                 F.removeFrom(slack) ;
16             if new ≠ ∅ then
17                 F.add(slack, t(S[i − 1]) \ t(S[i])) ;
18         else
19             F.reset() ;
20     return cost ;
```

in F must still be added. The quantity *slack* denotes the number of remaining slots $c - |t(S[i])|$. The tools that are not directly reused in $t(S[i-1]) \setminus t(S[i])$ are the ones that can possibly remain set for later use. By calling `add`, those are stored in the internal set F_{slack} of *FreeTools* to represent that *slack* of them could have stayed set. Also, all the internal sets F_k with $k > slack$ (if any) are added to F_{slack} and then deleted by calling the method `removeFrom`.

The time complexity of the Lazy-KTNS algorithm for processing a partial sequence of size $|S|$ is $\mathcal{O}(|S| \cdot c^2)$. Specifically, for each new job, at most c new tools are added. Each method of the *FreeTools* data structure executes in $\mathcal{O}(c)$, assuming bitset representations for the internal sets of tools and a number of tools that allows constant-time operations on the bitset operations. When the Lazy-KTNS algorithm is implemented in Laporte et al.'s B&B algorithm in place of the usual KTNS algorithm, the time complexity down a branch can be reduced to $\mathcal{O}(n \cdot c^2)$ from $\mathcal{O}(n^2 \cdot m)$ thanks to the reuse of the state of `F`.

Example 5. Consider again the JS-TSP instance shown in Fig. 1 and assume that $\mathcal{M}$ has processed a partial sequence $S = \langle 4, 2, 6 \rangle$ and the next job is 5. The current state of the *FreeTools* data structure is $[\{0, 6, 8\}, \{1, 4\}, \{\}]$. After adding the job 5, as F contains the tool 6, it is taken for free, and all sets are shifted by one to the left (line 9). So $\{0, 8\}$ can no longer be taken for free. All other tools induce a cost (line 11). Then, any tools that were previously on the machine but are no longer there are added to F, as there is still an empty space on the machine (line 17). The new current state of *FreeTools* is $[\{1, 3, 4\}, \{\}, \{\}]$.

3.3 On the Equivalence Between States

FreeTools can help to characterize the equivalence between two partial sequences.

Theorem 1. *Consider two non-empty partial sequences S_1 and S_2 such that:*

1. *S_1 and S_2 are defined on the same set of jobs (S_1 is a permutation of S_2),*
2. *their last job is identical, i.e., $last(S_1) = last(S_2)$,*
3. *the Lazy-KTNS algorithm applied to these sequences results in an identical* FreeTools *state,*

then any sequence S_{tail} on $R = J \setminus S_1 = J \setminus S_2$ induces the same cost to complete S_1 or S_2.

Proof. Let $i = |S_1| = |S_2|$, the length of the sequences. Let S_{tail} denote the sequence completing R. By examining the implementation of Algorithm 3 and assuming its correctness, we observe that the decisions taken by the algorithm for the concatenated sequences $S_1 \cdot S_{tail}$ and $S_2 \cdot S_{tail}$ are identical starting from index $i + 1$. This fact holds because these decisions depend only on the previous job $S[i - 1]$ and the state of the free tools at that point. By assumption, both $S[i - 1]$ and the *FreeTools* state are the same for S_1 and S_2. Therefore, the cost to complete either sequence is identical. □.

Example 6. Consider the JS-TSP instance shown in Fig. 1. Let denote $S_1 = \langle 2, 6, 1 \rangle$ and $S_2 = \langle 6, 2, 1 \rangle$, two sets of jobs that $\mathcal{M}$ have already processed (in this order of appearance). These two sets are defined on the same set of jobs, and the last job is identical: 1. The *FreeTools* state for each one is $[\{3, 4, 5\}, \{\}, \{\}]$. Thus, the cost to complete either sequence is identical.

3.4 A Graph Search Algorithm Based on A*

The $A*$ graph search algorithm [11] can be applied to solve instances of the JS-TSP by using a state representation that includes the *FreeTools* data structure, the set of remaining jobs to be scheduled, and the last scheduled job. By Theorem 1, equivalent partial sequences can be identified and merged, thereby reducing redundant exploration of the search space. In this framework, each state corresponds to a node in the graph, each arc is labeled with the job to be appended and its associated additional cost, and each path represents a sequence

Table 4. Dataset Properties.

	Dataset A	Dataset B
n	$\{8, 9, 15, 20, 25\}$	$\{10, 15\}$
m	$\{15, 20, 25\}$	$\{10, 20\}$
c	$\{5, 10, 15, 20\}$	$\{4, 5, 6, 7, 8, 10, 12\}$
min	$\{2, 3, 5, 10, 15\}$	$\{2\}$
max	$\{5, 10, 15, 20\}$	$\{4, 6\}$

of scheduled jobs. The objective is to find the shortest path in this layered graph, ending at a goal node that is reached when all jobs have been scheduled. The admissible heuristic $h(S)$ used by $A*$ is the lower-bound $\tilde{z}_3$. The cost-so-far value $g(S)$ represents the length (or cost) of the shortest path to the current node. The $A*$ algorithm systematically explores the search space by expanding states in the order of their estimated total cost $f(S) = g(S) + h(S)$. The first complete solution encountered is guaranteed to be optimal. In our experiments, we tested several anytime variants of $A*$ that have been successfully applied to other combinatorial optimization problems (see, for instance, [7,12,15]). These variants include *Iterative Beam Search* (IBS) [14], *Anytime Weighted* $A*$ (AWA) [18], and *Anytime Column Search* (ACS) [17]. Additionally, we evaluated the decision diagram-based B&B approach [3] and the DDO solver of [10]. However, this direction was abandoned because the bounds obtained by merging states were significantly weaker compared to the specialized bounds.

4 Computational Experiments

In this section, we report on the results of applying the graph search algorithm to two benchmark datasets. Dataset A is used to compare our approach with the approach described in [13] as the dataset used in the article. Dataset B is used to compare our approach with that of [5]. Table 4 summarizes the most important characteristics of each dataset that may impact both the time complexity and difficulty of solving the problem to optimality. In particular, an instance is defined by n representing the number of jobs, m the number of tools, c the capacity, and min and max, respectively, the minimum and maximum number of tools contained in a job. Each instance of one of these datasets corresponds to a specific combination of these parameters, but not all combinations exist in the dataset, and 10 instances have the same combination.

The computational experiments reported in this section run an Intel(R) Xeon(R) CPU E5-2687W with 128 GB of RAM, with a timeout set to 3600 s. The implementation was done in Java and executed with Java 8. The ILP model [5] is modeled in Mosel and solved with FICO Xpress v9.5.0. The source code and instances are available[1].

[1] https://github.com/emmalegrand/JS-TSP.

Our experiments were driven by several key questions:

- What is the impact of the different lower bounds $\tilde{z}_1$, $\tilde{z}_2$, $\tilde{z}_3$ (on the cost of the remaining jobs) on the performance of a B&B algorithm that uses the KTNS algorithm to compute the lower bound $\hat{z}$ for the partial sequence?
- After selecting the best bound based on the previous answer, what is the impact of replacing the KTNS algorithm with the Lazy-KTNS algorithm on the execution time of the B&B algorithm?
- What is the impact of using an A* like graph search algorithm instead of a tree search?
- How do the B&B algorithm with Lazy-KTNS and the graph search algorithms compare to the state-of-the-art ILP formulation?

In the next subsections, we address each of these questions.

4.1 Testing the Lower Bounds

This experiment explores the impact of the lower bounds $\tilde{z}_1$, $\tilde{z}_2$, and $\tilde{z}_3$ on the performance of a B&B algorithm that uses the KTNS algorithm to compute the lower bound $\hat{z}$ for a partial sequence. The implementation using $\max(\tilde{z}_1, \tilde{z}_2)$ replicates Laporte et al.'s B&B algorithm [13]. In this test, we considered instances with a maximum of 15 jobs and focused on the time required to solve them optimally. In Fig. 3, we observe that the performance of the B&B algorithm based on lower bound $\tilde{z}_3$ overcomes the others. The algorithm can optimally solve 100% of the instances with up to 15 jobs in at most 3600 s. Lower bound $\tilde{z}_1$ also achieves decent results but cannot reach 100%. The second lower bound, $\tilde{z}_2$, is not strong enough to prune a significant number of nodes in the tree, thus failing to solve around 30% of instances within the time limit. Nevertheless, in some cases, it proves stronger than $\tilde{z}_1$, which experimentally supports the idea of using the maximum of the two lower bounds as suggested in [13].

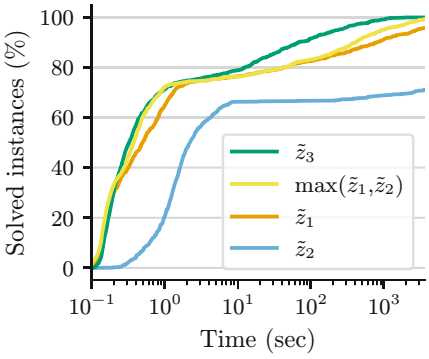


Fig. 3. Solved instances on dataset A regarding the time with $n = \{8, 9, 15\}$ for all lower bound defined in Sect. 2.1.

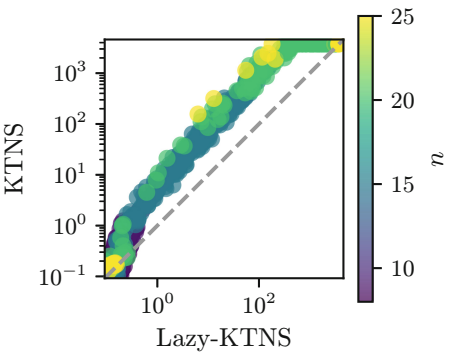


Fig. 4. Comparison of B&B with Lazy-KTNS and KTNS algorithms, regarding the time. Each dot represents an instance of the dataset A.

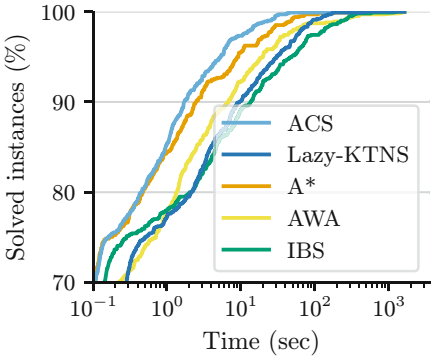


Fig. 5. Solved instances on dataset A regarding the time with $n = \{8, 9, 15\}$.

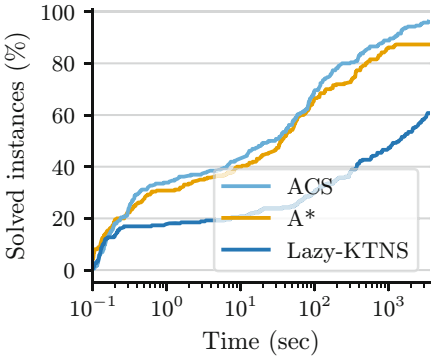


Fig. 6. Solved instances on dataset A regarding the time with $n = \{20, 25\}$.

4.2 Lazy-KTNS Algorithm vs KTNS Algorithm

Recall from Sect. 3 that the complexity of Lazy-KTNS algorithm, $\mathcal{O}(n \cdot c^2)$ is lower than $\mathcal{O}(n^2 \cdot m)$ because, in general, c^2 is less or equal than $n \cdot m$. This experiment proposes to compare KTNS and the Lazy-KTNS to compute the lower-bound $\hat{z}$ on the partial sequence while using the lower bound $\tilde{z}_3$ for the remaining jobs. The search spaces are thus identical; only the time spent in each node can vary. As can be observed on the right of Fig. 4 illustrating the time to find and prove the optimal solution for each instance with the two versions, replacing KTNS with its lazy version speeds up the execution time for most of the instances, and up to an order of magnitude for the larger values of n.

4.3 B&B vs Graph Search

As explained in Theorem 1, two non-empty sequences S_1 and S_2 can lead to the same state information. Therefore, graph search, which avoids recomputing similar states, can be used instead of a traditional tree search. This experiment compares our best B&B tree search, denoted as the Lazy-KTNS algorithm, with various A* graph search variants: the standard A* [11], *Anytime Column Search* (ACS) [17], *Iterative Beam Search* (IBS) [14], and *Anytime Weighted A** (AWA) [18]. All these versions utilize the lower bound $\tilde{z}_3$ for the remaining jobs.

As shown in Fig. 5 and Fig. 6, Lazy-KTNS takes longer to find and prove the optimal solution, compared to its graph search alternatives. This is especially visible for larger instances in Fig. 6, where Lazy-KTNS solves only 60% of the instances, while A* and ACS solve 80–90% of the instances. Figure 7 compares the number of explored nodes for each instance by Lazy-KTNS and ACS. It is easy to observe that the number of nodes explored in ACS is lower than in B&B for most instances. These results highlight that the gain from employing a state-caching strategy to avoid exploring similar states can be substantial.

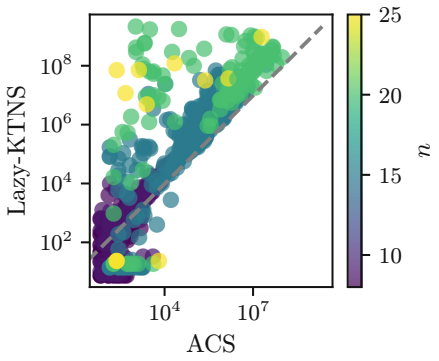


Fig. 7. Comparison of B&B with Lazy-KTNS and ACS, on dataset A, regarding the number of explored nodes.

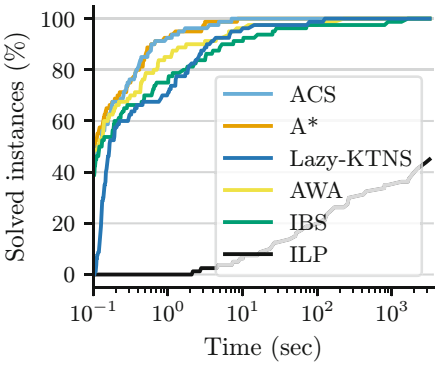


Fig. 8. Solved instances on dataset B regarding the time.

4.4 Comparison with ILP Formulations

In this final experiment, we compare the different approaches with the best ILP formulation from [5]. As expected, and consistent with the findings in [5], Fig. 8 demonstrates that the ILP formulation struggles to solve instances with 15 jobs within the 3600 s timeout. In contrast, the proposed approaches successfully solve these instances, regardless of the search algorithm used.

5 Conclusion

In this article, we proposed a novel approach for solving the JS-TSP. Our contributions include introducing a new family of lower bounds, denoted as $\tilde{z}_3$, which are provably tighter than existing bounds, and the development of the Lazy-KTNS algorithm, a computationally efficient variant of the KTNS algorithm. Additionally, we demonstrated the advantages of reformulating the traditional tree-based B&B algorithm into a graph search problem, leveraging the state-caching capabilities of modern search algorithms such as A* and its anytime variants. Our source code is publicly available, providing reproducible experiments for anyone interested in comparing their approach with ours.

For future work, we aim to explore alternative instantiations of the lower bound $\tilde{z}_3$ and compare them with the greedy one based on spanning tree construction. Additionally, we plan to investigate how a constraint programming solver performs on this problem and whether global constraints could be used or designed to solve it more efficiently. We believe the KTNS strategy could inspire strong constraint programming models for the JS-TSP.

References

1. Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J.: The Traveling Salesman Problem: A Compuational Study. Princeton University Press, Princeton (2006)
2. Belady, L.A.: A study of replacement algorithms for virtual storage computers. IBM Syst. J. **5**, 78–101 (1966)
3. Bergman, D., Cire, A.A., Van Hoeve, W.J., Hooker, J.N.: Discrete optimization with decision diagrams. INFORMS J. Comput. **28**(1), 47–66 (2016)
4. Bessiere, C., Hebrard, E., Ménard, M.-A., Quimper, C.-G., Walsh, T.: Buffered resource constraint: algorithms and complexity. In: Simonis, H. (ed.) CPAIOR 2014. LNCS, vol. 8451, pp. 318–333. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07046-9_23
5. Catanzaro, D., Gouveia, L., Labbé, M.: Improved integer linear programming formulations for the job sequencing and tool switching problem. Eur. J. Oper. Res. **244**(3), 766–777 (2015)
6. Crama, Y., Oerlemans, A.G., Spieksma, F.C., Crama, Y., Oerlemans, A.G., Spieksma, F.C.: Minimizing the Number of Tool Switches on a Flexible Machine. Springer (1996)
7. Fontaine, R., Dibangoye, J., Solnon, C.: Exact and anytime approach for solving the time dependent traveling salesman problem with time windows. Eur. J. Oper. Res. **311**(3), 833–844 (2023)
8. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, New York (2003)
9. Ghiani, G., Grieco, A., Guerriero, E.: Solving the job sequencing and tool switching problem as a nonlinear least cost Hamiltonian cycle problem. Netw.: Int. J. **55**(4), 379–385 (2010)
10. Gillard, X., Schaus, P., Coppé, V.: Ddo, a generic and efficient framework for MDD-based optimization. In: International Joint Conference on Artificial Intelligence (IJCAI-20) (2014)
11. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Syst. Sci. Cybern. **4**(2), 100–107 (1968)
12. Kuroiwa, R., Beck, J.C.: Solving domain-independent dynamic programming problems with anytime heuristic search. In: Proceedings of the International Conference on Automated Planning and Scheduling, vol. 33, pp. 245–253 (2023)
13. Laporte, G., Salazar-Gonzalez, J.J., Semet, F.: Exact algorithms for the job sequencing and tool switching problem. IIE Trans. **36**(1), 37–45 (2004)
14. Libralesso, L., Bouhassoun, A.M., Cambazard, H., Jost, V.: Tree search for the sequential ordering problem. In: ECAI 2020, pp. 459–465. IOS Press (2020)
15. Libralesso, L., Focke, P.A., Secardin, A., Jost, V.: Iterative beam search algorithms for the permutation flowshop. Eur. J. Oper. Res. **301**(1), 217–234 (2022)
16. Tang, C.S., Denardo, E.V.: Models arising from a flexible manufacturing machine, part I: minimization of the number of tool switches. Oper. Res. **36**(5), 767–777 (1987)
17. Vadlamudi, S.G., Gaurav, P., Aine, S., Chakrabarti, P.P.: Anytime column search. In: Thielscher, M., Zhang, D. (eds.) AI 2012. LNCS (LNAI), vol. 7691, pp. 254–265. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35101-3_22
18. Zhou, R., Hansen, E.A.: Multiple sequence alignment using anytime a*. In: AAAI/IAAI, pp. 975–977 (2002)

Time-Dependent Orienteering for High Altitude UAVs to Monitor Greenhouse Gases:
Mixed Integer Programming vs. Monte Carlo Tree Search

Richard Levinson[1,2(✉)], Vinay Ravindra[1,3], Jeremy Frank[1], and Meghan Saephan[1]

[1] NASA Ames Research Center, Mountain View, USA
{rich.levinson,vinay.ravindra,jeremy.d.frank,
meghan.saephan}@nasa.gov
[2] KBR Wyle, Inc., Houston, USA
[3] Bay Area Environmental Research Institute, Mountain View, USA

Abstract. Novel Earth observing systems aim to leverage newly developed Uncrewed Aerial Vehicles (UAVs) to provide higher resolution spatio-temporal data than is currently available through satellite, crewed flights, or in-situ sensors, for many climate science applications. The Intelligent Long Endurance Earth Observing System (ILEOS) provides science activity planning for obtaining higher quality spatio-temporal data from UAVs. ILEOS fuses coarse-grained data available from satellites and other sources to identify high value science targets, and produces flight plans for high altitude long endurance (HALE) UAVs, choosing which targets to observe to maximize science return. ILEOS uses a variant of the Orienteering Problem with time-dependent science reward values, with novel extensions to model vehicle turn times and no fly zones. Two solution methods are presented: Mixed Integer Linear Programming (MILP) and Monte Carlo Tree Search (MCTS). Experiments are conducted to compare their performance on six real-world scenarios. Results show that MCTS often outperforms a leading MILP solver.

Keywords: Mixed Integer Linear Programming · Monte Carlo Tree Search · Heuristic Search · Orienteering Problem · Earth Science

1 Science Problem and Application

Many satellites that survey climate-relevant gases have relatively coarse temporal or spatial resolution. Crewed aircraft campaigns are available, but often miss ephemeral events between observations. High Altitude Long Endurance (HALE) Uncrewed Aerial Vehicles (UAVs) are an emerging technology with the potential to provide longer observations with finer resolution sensors, thus enabling these platforms to be integrated with satellites and other in-situ measurement systems. The Intelligent Long-Endurance Observing System (ILEOS) provides HALE UAV science activity planning to obtain high spatio-temporal resolution

G. Tack (Ed.): CPAIOR 2025, LNCS 15763, pp. 86–102, 2025.
https://doi.org/10.1007/978-3-031-95976-9_6

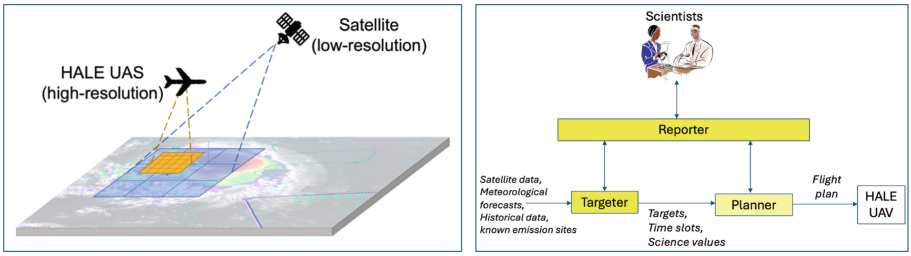


Fig. 1. ILEOS Concept of Operations (left) and Software Architecture (right)

data [18]. Figure 1 shows the ILEOS concept of operations (left) and three software components (right). The *Targeter* leverages science domain knowledge to fuse coarse-grained satellite images and forecast data to identify targets for the UAV to visit. Targets are mapped to science reward values, which may be time dependent. The *Planner* uses the science values to produce a UAV flight plan, choosing which targets to visit and when to visit them to maximize the rewards, without violating flight constraints (e.g., speed, turning radius). The *Reporter* provides the user interface for scientists to interact with the Targeter and Planner. ILEOS is a mixed-initiative decision support system. Users may repeatedly modify inputs and rerun the planner several times per day. To meet this cadence, the planner must produce high-quality results within minutes to hours.

This paper presents two real-world cases: (1) tracking nitrogen oxides (NO_x), a class of gasses including NO and NO_2, and (2) tracking methane (CH_4). The Targeter calculates science values for each target and timeslot, specific to each case, and is designed by climate scientists. For example, our NO_x case science values are based on meteorological forecasts (clouds, aerosol, NO_2 forecasts), historical NO_2 satellite observations, and known emissions from oil platforms and shipping lanes. In the NO_x case, we track ∼36,000 potential targets, spread out over ∼350,000 km^2. There are too many targets to visit them all. In one day, the UAV can fly only ∼850 km, and may only be able to visit ∼50 targets. We present experiments that establish a baseline for solution feasibility and quality in this novel Earth science application, and answer questions such as how many targets can be visited, what is the maximum achievable science value, how quickly can high quality solutions be produced, and what drivers affect solver performance and solution quality?

2 Planning Problem

2.1 Planner Input and Output

The planner chooses which targets to visit, and when to visit them to maximize science rewards. **The planner's input** is a set of targets and their rewards. Figure 2 shows targets off the coast of Galveston TX. The square tiles are targets with colors reflecting their science values. Colors indicate a continuous spectrum of values, from red (highest) through blue (medium) to white (low value). The red lines are shipping lanes, and the gold dots are oil platforms. Nominally, each

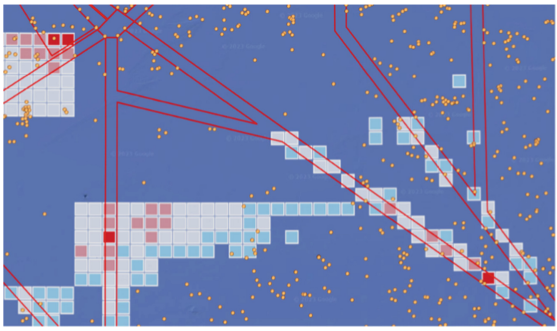


Fig. 2. Target value squares (red = high value, blue = medium, white = low), shipping lanes (red lines), and oil platforms (gold dots) (Color figure online)

target has 3 timeslots corresponding to morning, afternoon and evening, because the meteorological data was sampled for those periods, but any target may have any number of timeslots. **The planner's output** is a plan to visit as many high-value targets as possible within the mission duration. Figure 3 (left) shows a plan visualized in NASA's Mission Tools Suite [15] (similar to Google Earth). Potential targets (squares) with darker colors indicate higher reward values.

2.2 The Orienteering Problem with Time Dependent Values

The Orienteering Problem (OP) involves planning a tour to visit a subset of potential locations, considering travel time and an overall tour duration [3]. A reward is often associated with each visited location. The objective is to maximize those rewards by visiting as many high-reward locations as possible given the travel times and maximum tour duration. The OP has been widely studied and there are several time-dependent variants. Gavalas et al. [2] present a solution for tourism planning, where each target has limited visiting hours when rewards are available, and the reward does not change over time. Yu et al. [24] focus on service time dependent OP, where the reward increases the longer one remains at the target. We are interested in a variant where target rewards change throughout the day, so the reward for visiting a location depends on arrival time.

We model this as an orienteering problem with time-dependent reward values [22]. We adapted a specific formulation known as the Multi-Profit Orienteering Problem (MPOP) [8,9], which may delay arrival at some locations in order to arrive when rewards are highest. In the original MPOP model, each target is a <lat, lon> point and there is no service time at the target. We present novel MPOP extensions to (a) track the service time required to scan a scene, and (b) track the time required to turn from one scene to another, based on the previous location, the current location and next location and (c) enforce no fly zones.

2.3 Motion Planning Extensions

We have added a primitive form of motion planning to calculate flight trajectories based on the UAV's flight dynamics (speed and turn radius), and the sensor

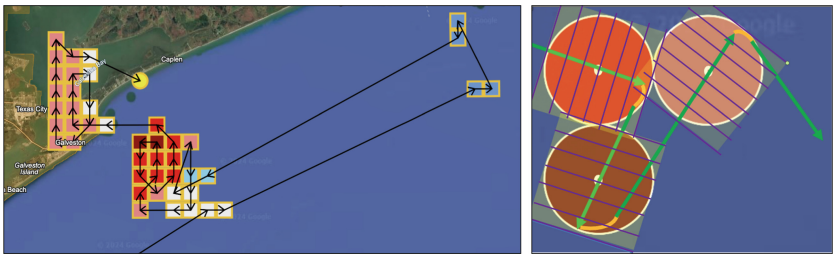


Fig. 3. UAV Flight Plan visualization (left, darker squares have higher value), and Flight plan for visiting and scanning 3 scenes (right) (Color figure online)

footprint. The model uses performance specifications for the Swift Engineering HALE UAV [21], which can fly at ∼60K ft altitude, with speed ∼60 mph, with ∼10K ft turn radius. It remains in flight for weeks using solar power, and flies in extended air space (>50K ft), high enough to fly line of sight without traffic control, and it can loiter around one location. The sensor payload is assumed to be the Compact Hyperspectral Air Pollution Sensor-Demonstrator (CHAPS-D) instrument, an in-development instrument [20].

Figure 3 (right) shows a flight plan for visiting 3 *scenes*. A *scene* is the area scanned by the sensor (reddish or brownish circles) around a target location (darker is higher value). The UAV may enter a scene from any direction so square scenes are not all in the same orientation. Instead of squares, we consider circles, with a diameter equal to the sensor footprint width (4.5 km), and the center (white dot) is the target <lat,lon>. The UAV flies a straight path between scenes and directly over the target in the center of each scene (green arrow).

Scanning: Observing a target involves flying directly over it and collecting 150 scans, each 4.5 km × 30 m, resulting in a 4.5 km^2 square image. In Fig. 3 (right), the scanning area is shown as a light yellow box. Scanning begins when the UAV enters a scene (enters the 4.5 km diameter circle centered on the target pt). The purple lines across each scene indicate the scan lines. Each scan is a 4.5 km × 30 m strip perpendicular to the UAV's direction of flight. The UAV continues scanning while it flies directly over target. Scanning ends when the UAV exits the scene (exits the circle centered on the target).

Turning: When the UAV reaches the edge of the scene and stops scanning, it begins turning towards the next target. This is shown as the gold arc between each pair of green arrows in Fig. 3 (right). The UAV's turning radius is nearly the same as the scene radius, so an entire 180° turn fits in one scene circle. The target in the center of the scene (white dot) becomes a virtual pivot point for the UAV's turn, before it continues flying directly to the next scene (green arrow). The algorithm calculates the UAV position after the turn and uses that to adjust the travel time to the next scene accordingly.

Turn time is a function of vertex triples rather than pairs. The turn magnitude is pre-calculated for all possible target sequence triples (i, j, k), based

Fig. 4. Without no-fly zones (left), UAV can fly the edge between any two targets (colored tiles). With no fly zones (right), edges are removed which cross the zone.

on the change in the bearing angle of turn required at the middle point j (the change in direction between the edge from i to j, and the edge from j to k). Half the scan time already included in the travel time to the center of the scene, so we only add half of the scan time to account for scanning the other half of the scene, past the center point. We define *sceneTime* as the time spent at a scene:

`sceneTime = scanTime/2 + turnTime`, where `scanTime = sceneWidth/speed`.

No-Fly Zones: Figure 4 shows how we model no-fly zones as polygons where the UAV cannot fly. They mutate the initial fully-connected graph by *removing vertices (targets) and edges (paths between vertices)*. Permanently restricted areas, and targets with high winds, are specified as polygons. Any target with predicted wind speed exceeding a threshold is considered a no-fly zone. We create <lat, lon> polygons around those targets and add them to a set of no-fly polygons. Pre-processing removes edges which intersect with no-fly polygons. The UAV can only fly on the remaining edges, so will not fly into the zone.

3 Solution Methods

We compare two methods: Mixed Integer Linear Program (MILP) vs. a novel method which combines Python with Monte Carlo Tree Search (MCTS). Both solutions include the same inputs and model elements described above, and the same objective calculation. The MILP and MCTS models both take the following input parameters:

Parameters
V = the set of available observation target locations (vertices)
D_i = the set of timeslots for viewing target (vertex) i, $i \in V$
$r_{i,d}$ = reward for visiting target (vertex) i during its d^{th} time slot, $i \in V$
N = the number of targets = $|V|$
T_{max} = mission duration (nominally 9 hrs, daylight is required for our sensor)

3.1 Mixed Integer Linear Program Formulation (MILP)

The Orienteering problem for time-dependent values starts with a fully connected graph containing a set of vertices composed of N target locations. MILP models for this problem require two additional vertices, *Start* (vertex 0) and *End*

(vertex N+1). The plan is modeled as a sequence of edges between vertices, thus the Start and End vertices must be specified *a priori*. Each visited vertex must have a predecessor and successor (flow conservation), *except* the Start and End locations. We set Start to 0.5 km north of the target with the highest value at the start of the day, and End to 0.5 km north of the target with the highest value at the end of the day. It takes less than 20 s to fly the extra 0.5 km on each end. The UAV remains in flight for weeks and can fly overnight to the start location for the next day.

Decision Variables
$x_{i,j} = 1$ if vertex i is followed by vertex j, otherwise 0 (binary variable)
$y_{i,d} = 1$ if vertex i is visited during its d^{th} time slot, otherwise 0 (binary variable)
$w_{i,j,k} = 1$ if vertex i is followed by j, and j is followed by k, otherwise 0 (binary)
s_i = arrival time at vertex i, $\quad 0 \leq s_i \leq T_{max}$ (continuous variable)

Objective: Maximize the sum of rewards collected for all visited vertices

$$Maximize \sum_{i=1}^{N} \sum_{d=1}^{|D_i|} r_{i,d}\, y_{i,d} \tag{1}$$

Constraints

$$\sum_{j=1}^{N+1} x_{0,j} = \sum_{i=0}^{N} x_{i,N+1} = 1 \tag{2}$$

$$\sum_{i=0}^{N} x_{i,j} = \sum_{k=1}^{N+1} x_{j,k} = \sum_{d=1}^{|D_j|} y_{j,d}, \quad \forall j = 1, ..., N \tag{3}$$

$$\sum_{d=1}^{|D_i|} y_{i,d} \leq 1, \quad \forall i = 1, ..., N \tag{4}$$

$$L_{i,d} - M(1 - y_{i,d}) \leq s_i \leq U_{i,d} + M(1 - y_{i,d}), \quad \forall i = 1, ..., N \ \ \forall d = 1, ...|D_i| \tag{5}$$

where $L_{i,d}$ and $U_{i,d}$ are the lower and upper bounds of vertex i's d^{th} timeslot.

$$s_{N+1} \leq T_{max} \tag{6}$$

$$s_j + [\texttt{scan/2} + \texttt{turn(i,j,k)} + \texttt{travel(j',k)}] \leq M(1 - w_{i,j,k}) + s_k \tag{7}$$

$$\forall i = 0, .., N \qquad \forall j = 1, .., N \qquad \forall k = 1, .., N + 1$$

$$\sum_{i=0}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n+1} [\texttt{scan/2} + \texttt{turn(i,j,k)} + \texttt{travel(j',k)}]\, w_{i,j,k} \leq T_{max} \tag{8}$$

$$w_{i,j,k} \leq x_{i,j}, \qquad \forall i = 0,..,N; j = 1,..,N; k = 1,..,N+1 \tag{9}$$

$$w_{i,j,k} \leq x_{j,k}, \qquad \forall i = 0,..,N; j = 1,..,N; k = 1,..,N+1 \tag{10}$$

$$x_{i,j} + x_{j,k} - 1 \leq w_{i,j,k}, \qquad \forall i = 0,..,N; j = 1,..,N; k = 1,..,N+1 \tag{11}$$

Constraints (2) say the Start vertex is exited once, and the End vertex is entered once. Constraints (3) say the number of times vertex j is entered equals the number of times it is exited, which equals the number of timeslots available to visit j. Constraints (4) say each vertex is assigned to at most 1 timeslot. Constraints (5) are conditional constraints, activated only when binary indicator variable $y_{i,d} = 1$, where M is a "Big-M" large constant. Constraints (5) say **IF** vertex i is assigned to timeslot d, **THEN** the vertex i's visit must start within the lower and upper bounds of timeslot d. Constraints (6) ensure the UAV arrives at the End vertex (N+1) before the mission ends.

Constraints (7) enforce the minimum temporal distance between vertices as conditional constraints, activated only when $x_{i,j} = x_{j,k} = 1$, with Big M. Constraints (7) say: **IF** (i is followed by j) and (j is followed by k), **THEN** startTime(j) + `scan/2` + `turn`(i, j, k) + `travel`(j', k) $\leq$ startTime(k). The terms `scan`, `turn` and `travel` are scan, turn and travel times which are calculated in advance and appear as constants in the model. The j' in the term `travel(j',k)` of (7) is the *adjusted position* of j (after flying over the center to the far side of the scene and completing the turn). `travel(j',k)` is the travel time from that adjusted position to the next target k. This position is a function of where the turn begins (2.25 km past the center of the scene), the turn radius, and the turn arc length. Figure 3 (right) shows the turn starts where the gold arc meets the head of a green arrow, and the turn ends where the gold arc meets the tail of the next green arrow. Constraints (8) ensure the sum of all scan + turn + travel times in the plan does not exceed the mission duration T_{max}. Constraints 9, 10, and 11 bind $w_{i,j,k}$ to 1 if and only if i is followed by j, followed by k. Novel extensions to model turn times include the new decision variable $w_{i,j,k}$ to represent a sequence of target triples, and related constraints (7) through (11).

3.2 Monte Carlo Tree Search (MCTS)

We now present a novel solution combining Python with Monte Carlo Tree Search (MCTS), using a nondeterministic programming system called *Propel* [11,14]. In Propel, Python is extended with two types of choice points: `chooseValue` (nondeterministic assignment statement) and `chooseTask` (nondeterministic subroutine call), which specify places in the code where a choice must be made. These choice points define the search space which MCTS explores. This is *nondeterministic* Python because each time `chooseValue` is executed, a different choice may be made. MCTS automatically chooses which value to assign, or which subroutine to call, rather than a programmer choosing it in advance.

Code Walkthrough: Figure 5 shows `createIleosPlan`, the Python program with choice points used for our experiments. The *chooseValue* statements (red boxes)

```
def createIleosPlan(self):
    time = 0        # Initialize mission time
    self.plan = []  # Used by objective fn to calculate rollout score
    currentTarget = None   # First target has not been chosen yet
    openTargets = copy(self.vertices)
    while time <= self.maxTime and openTargets:
        # Choose target: There must be a graph edge beween current and next targets (nodes)
        # Valid next targets must have edge to current target after removing no fly zone edges
        validTargets = self.getValidNextTargets(currentTarget, openTargets)
        nextTarget = self.planner.chooseValue(validTargets, self.sortTargetsByMaxValueAndDistance)
        openTargets.remove(nextTarget)
        time = self.getETA(time, self.plan, nextTarget)
        if time <= self.maxTime:
            timeslots = self.getValidTimeslots(target, time)
            if timeslots:
                # Choose time slot
                slot = self.planner.chooseValue(timeslots)
                time = max(time, slot.start)
                self.plan.append({"time": time, "target": target, "slot": slot})
                currentTarget = nextTarget # update current UAV location for next iteration
```

Fig. 5. Python code with choice points that define the MCTS search space

are choice points defining the MCTS search space. The code works as follows: First, the time and plan are initialized, and the variable `openTargets` is initialized to the list of all targets. *While time remains in the mission, repeatedly*: *Choose* a valid next target from the set of open targets. A next target is valid only if it is connected by an edge to the current target after no-fly zone targets and edges have been removed (determined by `getValidNextTargets`). Then, estimate the arrival time: `ETA = currentTime + scanTime/2 + turnTime + travelTime to next target`.

Finally, *choose* the timeslot for visiting the target, only considering time slots that end after the ETA (determined by `getValidTimeslots`). After choosing the target and time slot, state variables `time` and `self.plan` are updated to reflect the arrival time at the next target. If the timeslot starts after the current time, the planner delays arrival until the time window begins. This delay is implemented by the line `time = max(time, slot.start)`, which advances the time to the timeslot's start time, if it is in the future.

MCTS Tree and Stages: Figure 6 shows an example MCTS tree for this application, and the standard four MCTS stages: *Select, Expand, Simulate,* and *Update* [1]. Every MCTS iteration proceeds through these four stages. The method `createIleosPlan` (Fig. 5) is executed from beginning to end on each MCTS iteration, and the iteration ends when it exits. The *Select* stage chooses which leaf of the search tree to expand, *Expand* adds a new child node to the selected leaf, *Simulate* (also called a *rollout*) completes the remaining execution of `createIleosPlan` (generates a full plan), using a *rollout policy* to make selections at choice points. Simulation ends when `createIleosPlan` exits, and then the *Update* stage begins. During the *Update* stage, the objective function (plan score) is calculated and used to update the node learning statistics (the tree policy). Each node in Fig. 6 represents the program state of `createIleosPlan` when the choice point is executed. Each edge represents a choice from that state. Nodes

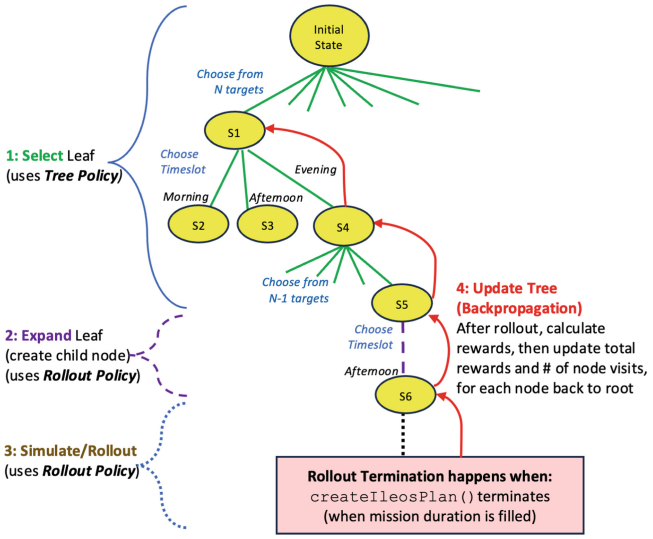


Fig. 6. MCTS Tree and Four MCTS Stages

and edges are only maintained for choices near the root, and are not required for the rollout choices made during the simulation stage. For our application, the state includes the variables `time` and `self.plan`, which are used to calculate the iteration score and by the rollout policy.

During the MCTS *select* stage, nodes are selected for expansion based on the Upper Confidence Bound For Trees (UCT) equation [10]. UCT drives the MCTS *tree policy* that is used to select the next leaf node to be expanded. The UCT value for each leaf node, j, is calculated by equation (12) below. The leaf with the highest UCT value is selected.

$$UCT_j = X_j + C * \sqrt{ln(n)/n_j} \tag{12}$$

where: X_j is the average score for node j, n is the number of times node j's parent has been visited, and n_j is the number of times node j has been visited. The constant C controls the trade between *exploring* new choices vs. *exploiting* choices which previously worked well. The default value for C is $\sqrt{2}$, which is $\sim$1.41, but it is usually set empirically. We found setting it to 0.08 produced the highest scores, which biases UCT to favor previously successful choices.

Python/MCTS Integration: The *chooseValue* statements define branches in the MCTS search space, and Propel manages the MCTS stage sequence [14]. The method `createIleosPlan` (Fig. 5) is executed from beginning to end on each MCTS iteration. Each iteration (each execution of `createIleosPlan`) starts in *Select* stage, and uses the UCT-based *Tree Policy* to select a leaf node to expand. If the selected leaf is not the root, then Propel *replays* the sequence of prior choices from the root to the leaf in order to reset the Python program to

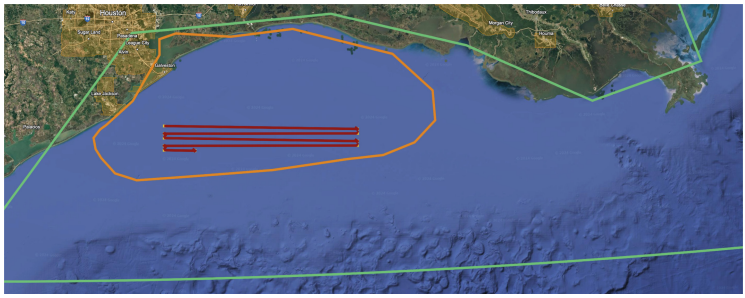


Fig. 7. The region of interest is the green outline. In one day, the UAV can fly either the distance of the orange perimeter OR fly the distance of the red lawnmower pattern. (Color figure online)

the state it was in when the leaf was created [14]. The second choice on each iteration is the *Expand* stage, using the rollout policy to add one child node to the selected leaf. All remaining choices are in the *Simulate* stage, using the rollout policy until the method terminates, without adding any new nodes to the tree. When `createIleosPlan` exits, the *Update* stage begins, which calculates the objective score and uses it to update the MCTS learning statistics. The objective calculation is the same as the MILP objective: the sum of rewards for all (target, slot) pairs (extracted from variable `self.plan`).

Rollout Policy: `chooseValue` takes an optional argument for a local heuristic to sort the choices based on the current state. This functions as the MCTS *rollout policy*, which is used to make choices during the *Expand* and *Simulate* stages. If no heuristic is passed to `chooseValue`, then choices are randomly selected. Random often provides a wider range of training experiences. Our experiments use a blend of random and greedy. In Fig. 5, the first call to `chooseValue` specifies a *greedy heuristic* function `sortTargetsByMaxValueAndDistance`, which sorts targets based on the target value and it's distance from the current location. This heuristic subtracts a *distance penalty* from the target's maximum reward, based on the distance from the current location to the chosen next one. The distance penalty reduces the large distance value, to be closer in magnitude to the target value it's subtracted from. The heuristic value, h, of a target is:

$$h(target) = maxValue(target) - distancePenalty * distanceTo(target)$$

4 Evaluation

Scaling performance to visit as many targets as possible is a challenge. One option to reduce model size is to create a smaller region of interest. Figure 7 shows the scale of our target selection problem. The green polygon shows *part of* the region of interest for our NO_x case. This green area is $\sim$215,000 km^2, and

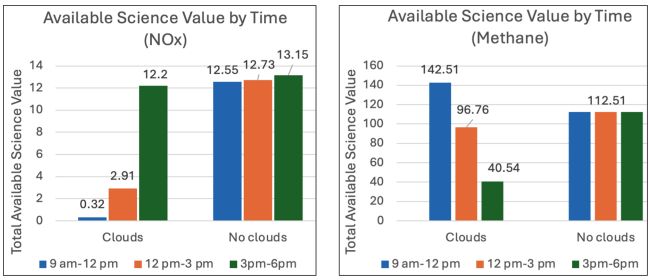


Fig. 8. Time dependent target values for the NO_x (left) and Methane (right) cases

does not even cover the whole area of potential targets. The HALE can only cover a small fraction of this region in a given day. It can fly ∼850 km in 9 h, covering an area of 850 km × 4.5 km = 3,825 km^2. If a smaller region (polygon) of interest is provided, the planner restricts targets to be within that smaller region, reducing the number of targets, and limiting the maximum distance between targets. The orange polygon in Fig. 7 has a perimeter of 850 km, and contains an area of 40,285 km^2. The red lawnmower pattern is also 850 km long. In one day, the UAV can either fly the distance of the orange perimeter or the distance of the red lawnmower pattern.

Time-dependent Values: Figure 8 shows how target values change over time. It shows the total science reward available per time slot for the NO_x case (left), and the Methane case (right). In the NO_x case with clouds, most of the targets are not visible until the last time slot and their maximum reward is after 3 pm. The UAV sensor requires visibility, so if the cloud coverage exceeds a specified threshold (30% in our application), then the target value is set to 0. With clouds, it's difficult to fit many targets clustered into only a 3-hour timeslot. The *methane case* is shown on the right in Fig. 8. With clouds, most rewards are in the morning and afternoon. Without clouds, the rewards remain the same all day, with a total value of 112.51. The methane case values are higher than the NO_x case because it uses a different value calculation. This clustering can skew the data. If it is cloudy in the morning then most of the science value might be clustered in the last 3 h of the day, making it hard to visit all of the targets within the mission duration. Although it is an artifact of our real world data, this clustering is like an experiment case with a short 3-hour mission duration. The tests without clouds have science values more evenly distributed across time, and are designed to evaluate planner behavior when data is not clustered.

4.1 Experiment Design

All our tests used real world data. For each of our two use cases, NO_x in the Gulf, and methane in Alaska, we evaluated two variants: with and without clouds (Fig. 8). The experiments were designed to answer key questions for our appli-

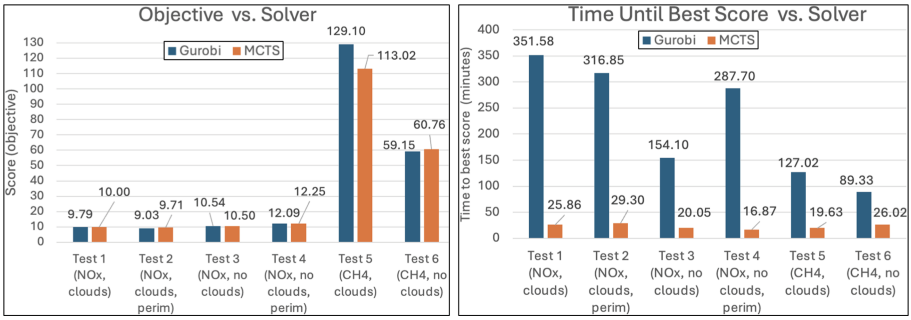


Fig. 9. Objective vs. Solver (left) and Time to best score vs. Solver (right)

cation: How many targets can be visited, what is the maximum science reward which can be collected, and how long does it take to produce high-quality plans?

Computing Environment: All experiments ran on a 2023 MacBook Pro, with an M3 Max chip, 36 GB Memory with 14 cores, using Python 3.8. Our MILP solver was Gurobi version 11.0.3 [5].

Test Scenarios: Experiments included the six test scenarios shown in Figs. 9 and 10. Tests 1–4 are NO_x cases, and Test 5 and 6 are methane (CH_4) cases. Test 1 is the NO_x case with clouds. It considers all targets within the large green polygon shown in Fig. 7 (∼23,000 targets with non-zero rewards). Test 2 is also NO_x with clouds, but only includes targets within the orange "perimeter" polygon shown in Fig. 7, which contains 1,881 targets. Test 3 is the NO_x case without clouds. It includes the same targets as Test 1 (all targets in the large green polygon). Test 4 is NO_x without clouds, but only includes targets within the orange "perimeter", like Test 2, and includes the same number of targets as Test 2. Test 5 is the methane (CH_4) case with clouds, which has 4,070 targets with non-zero rewards. Test 6 is the methane case without clouds. This is a degenerate test case where all timeslots have the same values (Fig. 8), and serves to evaluate how the planners behave when values are not time-dependent. Timeslots with the same value are merged, so there is only one 9-hour timeslot for this case, which has 6,010 targets with non-zero rewards.

Including all targets in these tests produces models so large the MILP solver produces very poor solutions (MIP Gap $> 150\%$) even after many hours. To make it tractable, all tests consider only the 50 highest reward targets for that test. For example, there are 1,881 targets inside the orange perimeter for tests 2 and 4. The planner considers only the 50 highest reward targets from that set of 1,881. This 50 target filter was chosen empirically after many MCTS and MILP runs never produced a plan with more than 45 targets, until one outlier test for this paper covered all 50 targets (test 4). Even with 50 targets, the problem size is huge. The number of permutations for visiting 50 targets is $50! = 3.04 \times 10^{64}$.

We ran each test on 2 solvers: MILP using the formulation described in Sect. 3.1, and the MCTS system described in Sect. 3.2. Both solvers leverage

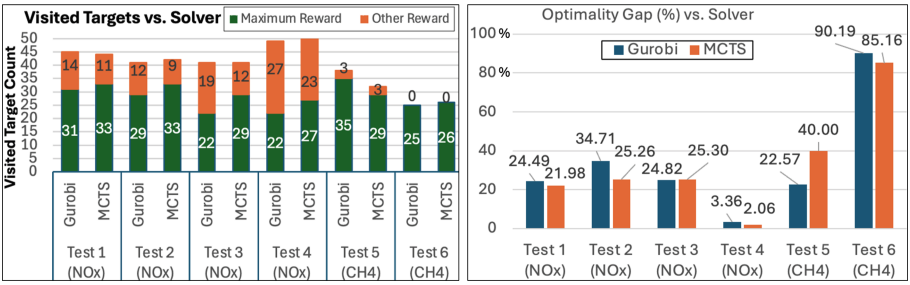


Fig. 10. Targets vs. Solver (left) and Optimality Gap vs. Solver (right)

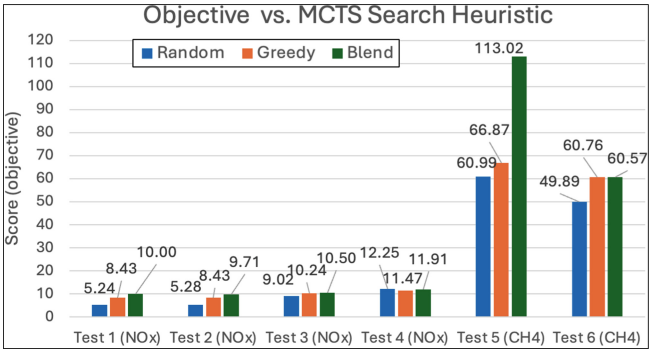


Fig. 11. Comparison of MCTS local heuristics (rollout policies)

the MacBook's 14 CPU cores. Parallel MCTS uses *Root Parallelization*, where parallel processes manage independent MCTS trees, and do not share learning statistics [19]. Our tests ran 11 parallel MCTS processes, each process performing 15 million iterations, for a total of 165 million iterations, taking ∼3.5 h. The best score for each process was collected and the best score among all processes is shown in our results. Parallel MILP: Gurobi uses all available CPU's, but details are a mostly a black box, and parallelism varies throughout the solving process. Upon startup, Gurobi reports: `14 physical cores...using up to 14 threads`. See [4] for more information about parallel processing in Gurobi.

4.2 Results

Figure 9 (left) shows the best objective scores produced by Gurobi and MCTS. MCTS is very competitive with Gurobi, achieving the highest score on tests 1, 2, 4 and 6, and it nearly tied Gurobi on test 3. The only test where Gurobi clearly outperformed MCTS is 5. All MILP tests timed out at 6 h without optimal solutions, and all MCTS tests ran for 15 million iterations (∼3.5 h). However, their best scores were achieved long before those limits were reached. Figure 9 (right) shows the time taken (minutes) to reach the best scores shown on the

left. MCTS reaches its high score in less than 30 min in all tests, while Gurobi takes much longer. Figure 9 shows MCTS produced higher or similar scores in 30 min compared to Gurobi scores produced in 2 or 3 h. Gurobi spends time proving optimality, estimating the upper bound, pre-solving to simplify the math constraints, and other functions, which MCTS does not do.

Figure 10 (left) shows the total number of targets visited, and the number of targets visited at their maximum reward time (called *maximum reward targets*). The only test where *all 50 targets are visited* was Test 4 with MCTS. The total number of targets visited is similar between MCTS and Gurobi on all tests except test 5 where Gurobi visited 38 targets vs. MCTS's 32. MCTS has more maximum reward targets than Gurobi on all tests except test 5.

Optimality Gap: MILP solvers produce a mathematically proven best bound on the score, even when they cannot find a solution with that score. Ours is a maximization problem, so the best bound is an *upper bound*. The *optimality gap* quantifies the percentage difference between best score and that upper bound: `MILP gap = (MILP upper Bound - MILP score)/MILP score`. MCTS does not provide any optimality estimate, but we can use the MILP upper bound to estimate MCTS optimality. The MILP and MCTS solutions have the *same upper bound* because they maximize the same objective and use the same target and reward inputs. Thus we can calculate the MCTS gap as:

`MCTS gap = (MILP upper Bound - MCTS score)/MCTS score`.

Figure 10 (right) compares the optimality gap for Gurobi vs. MCTS, using the upper bound determined by Gurobi to calculate the MCTS gap. MCTS has a smaller gap (closer to MILP upper bound) on all tests except 3 and 5. Test 4 has an MCTS gap of 2.06% and a MILP gap of 3.38%. Test 5, methane *with clouds*, has a higher score than methane *without clouds* (test 6). This may be because (a) the maximum rewards in test 5 are much larger, and (b) the gap on test 6 is nearly 100%, more than double the gap on test 5 (for both MILP and MCTS). More solve time on test 6 may increase the score as the gap gets smaller.

MCTS local Heuristics (Rollout Policies): Figure 11 shows MCTS performance with different local heuristics (rollout policies) for choosing targets. We ran each test with 3 options: 100 % *Random* choices vs. 100% *Greedy* choices vs. *Blend* of both. The Blend leveraged the 11 parallel MCTS processes, where each process used a different blend of greedy vs. random choices. Each process was assigned a different *random choice percentage* which specified how often the greedy choice was ignored in favor of a random choice. Process 1 used 0% random choices (all greedy), then the random percentage incremented by 10% for each process until the eleventh process used 100% random choices (all random). The *Blend* achieved higher scores on tests 1, 2, 3, and 5. *Random* achieved the high score on test 4, and *Greedy* scored slightly higher than the Blend on Test 6.

Analysis of MILP Results: Key factors affecting MILP performance are: the number of targets (vertices), the travel time between targets (edge distances),

the distribution of target values over time, and the mission duration (plan horizon). Analysis of MCTS results: Fig. 11 shows that a blend of greedy and random choices generally worked best. We also learned that lowering the UCT exploration weighting factor, C, from the standard $\sqrt{2}$ to 0.08, a major decrease favoring exploitation, was important for increasing scores. One reason MCTS can do 15 million iterations so quickly is the shallow search tree. Since only ∼50 targets fit into the 9 h horizon, the main loop in `createIleosPlan` (Fig. 5) repeats only ∼50 times before exiting. The branching factor when choosing a target starts with 50, then decreases by one on each iteration. Including the perimeter likely contributed to achieving a high water mark of visiting all 50 targets in Test 4 by limiting the distance between targets.

5 Related Work

Kim et al. [8,9] developed a specialized MPOP algorithm which combined Dynamic Programming with branch and bound methods designed specifically for MPOP, aiming to outperform a commercial MILP solver, and they use public benchmark data for evaluation. In contrast, we are using the commercial Mixed Integer Program (MIP) optimizer, Gurobi, to solve real-world problem instances.

Yu et al. [24] perform orienteering with service-time dependent rewards, where rewards increase the longer one remains at the target. They include extensions for multiple UAVs, and a constraint that all paths start and end at the same place. Their aim was to develop a Hybrid Artificial Bee Colony algorithm to outperform a MILP solver on standardized benchmarks, whereas our aim to use Gurobi to solve real-world problems with real data.

Pham et al. [17] present routing methods where costs depend on triples of nodes successively traversed for the Quadratic Traveling Salesman problem, resulting in a quadratic objective function. They have no rewards, and minimize cost (plan duration). All targets must be visited, and the Start and End locations must be the same. They have a quadratic objective function, but ours is linear. We pre-compute the cost of each triple before constructing the MILP model, and use those costs as linear coefficients. We compile away the need for quadratic constraints at the cost of increased number of linear constraints.

Wang et al. [23] present work on path planning for solar-powered HALE platforms, addressing complex engineering issues to produce plans for automatic execution which is outside the scope of our work. Our UAV will be controlled by a team of Swift engineers on the ground. We will provide feasible plans, but Swift operators will smooth out trajectories and ensure safe operations.

Kiam et al. [7] present work aimed at integrating as much quantitative motion planning into a PDDL planner as possible, but a PDDL planner would not be appropriate for this NP-hard problem which does not require causal reasoning.

Patra et al. [16] present the most similar work to Propel, with a planner that (a) reasons about *procedural models which contain loops and conditionals*, and (b) uses MCTS to make decisions at choice points. Their *operational models*

require knowledge of and access to a specialized system called the Reactive Acting Engine, a derivative of the Procedural Reasoning System (PRS) [6], while Propel only requires knowledge of and access to Python, and is more expressive.

Levinson has presented prior versions of Propel, adding the nondeterministic `chooseValue` to C^{++} [13], and LISP [12], instead of Python, but without MCTS.

6 Conclusion

We presented a novel application of the Orienteering Problem with time dependent values and extensions for turning times and no-fly zones. We established a baseline understanding of solution quality for emerging mission capabilities using HALE UAV. Results show high-quality solutions can be obtained within 30 min, and that for our application, ∼50 out of ∼20,000 targets with non-zero rewards can be visited within a day. MCTS generally outperformed MILP on our tests and offers more options for search control. Although the MILP problem is too large to solve optimally, we showed the upper bound calculated by MILP can be used to quantify the MCTS optimality gap. This yields confidence that our quick-turnaround MCTS plans remain high quality.

Acknowledgments. This work is supported by NASA's Earth Science Technology Office Advanced Information Systems Technology (AIST) Program. We thank Kate Bartlett for important contributions to the ILEOS project including work described in this paper.

Disclosure of Interests. The authors have no competing interests.

References

1. Coulom, R.: Efficient selectivity and backup operators in Monte-Carlo tree search. In: International Conference on Computers and Games, pp. 72–83. Springer (2006)
2. Gavalas, D., Konstantopoulos, C., Mastakas, K., Pantziou, G., Vathis, N.: Heuristics for the time dependent team orienteering problem: application to tourist route planning. Comput. Oper. Res. **62**, 36–50 (2015)
3. Golden, B., Levy, L., Vohra, R.: The orienteering problem. In: Naval Research Logistics, vol. 34, pp. 307–318 (1987)
4. Gurobi Optimization, LLC: How to exploit parallelism in linear and mixed integer programming (2020). https://www.gurobi.com/wp-content/uploads/How-to-Exploit-Parallelism-in-Linear-and-Mixed-Integer-Programming.pdf
5. Gurobi Optimization, LLC: Gurobi optimization (2024). https://www.gurobi.com
6. Ingrand, F.F., Chatila, R., Alami, R., Robert, F.: PRS: a high level supervision and control language for autonomous mobile robots. In: Proceedings of IEEE International Conference on Robotics and Automation, vol. 1, pp. 43–49. IEEE (1996)
7. Kiam, J., Scala, E., Javega, M., Schulte, A.: An AI-based mission planning for haps in a time-varying environment. In: 30th International Conference on Automated Planning and Scheduling (ICAPS) (2020)
8. Kim, H., Kim, B.I.: Hybrid dynamic programming with bounding algorithm for the multi-profit orienteering problem. Eur. J. Oper. Res. **303**(2), 550–566 (2022)

9. Kim, H., Kim, B.I., Noh, D.J.: The multi-profit orienteering problem. Comput. Industr. Eng. **149**, 106808 (2020)
10. Kocsis, L., Szepesvári, C.: Bandit based Monte-Carlo planning. In: European Conference on Machine Learning, pp. 282–293. Springer (2006)
11. Levinson, R.: Integrated planning, execution and goal reasoning for python. In: The 30th International Conference on Automated Planning and Scheduling (ICAPS 2020), Workshop on Integrated Execution and Goal Reasoning (IntEx/GR) (2020)
12. Levinson, R.: A general programming language for unified planning and control. In: Artificial Intelligence, Special Issue on Planning and Scheduling, vol. 76 (1995)
13. Levinson, R.: Unified planning and execution for autonomous software repair. In: The 15th International Conference on Automated Planning and Scheduling (ICAPS 2005), Workshop on Plan Execution (2005)
14. Levinson, R.: Monte Carlo tree search for integrated planning, learning, and execution in nondeterministic python. In: The 34th International Conference on Automated Planning and Scheduling (ICAPS 2024), Workshop on Bridging the Gap between Planning and Reinforcement Learning (PRL) (2024)
15. National Aeronautics and Space Administration: Airborne science mission tool suite (2024). https://airbornescience.nasa.gov/content/ASP_Mission_Tools_Suite
16. Patra, S., Mason, J., Ghallab, M., Nau, D., Traverso, P.: Deliberative acting, planning and learning with hierarchical operational models. In: Artificial Intelligence, vol. 299 (2021)
17. Pham, Q.A., Lau, H.C., Hà, M.H., Vu, L.: An efficient hybrid genetic algorithm for the quadratic traveling salesman problem. In: The 33rd International Conference on Automated Planning and Scheduling (ICAPS 2023) (2023)
18. Saephan, M., et al.: ILEOS: a novel intelligent observing system enabled by high altitude long endurance uncrewed aerial systems. In: AIAA Aviation Forum and ASCEND 2024 (2024)
19. Steinmetz, E., Gini, M.: More trees or larger trees: parallelizing Monte Carlo tree search. IEEE Trans. Games **13**(3), 315–320 (2020)
20. Swartz, W.H., et al.: CHAPS: a sustainable approach to targeted air pollution observation from small satellites. In: Sensors, Systems, and Next-Generation Satellites XXV, vol. 11858, pp. 274–282. SPIE (2021)
21. Swift Engineering: High altitude long endurance (HALE) unmanned aerial system (UAS) (2024). https://www.swiftengineering.com/r-and-d/hale-haps-uas
22. Vincent, F.Y., Jewpanya, P., Lin, S.W., Redi, A.P.: Team orienteering problem with time windows and time-dependent scores. Comput. Industr. Eng. **127**, 213–224 (2019)
23. Wang, X., Yang, Y., Wu, D., Zhang, Z., Ma, X.: Mission-oriented 3D path planning for high-altitude long-endurance solar-powered UAVs with optimal energy management. IEEE Access **8** (2020)
24. Yu, Q., Fang, K., Zhu, N., Ma, S.: A matheuristic approach to the orienteering problem with service time dependent profits. Eur. J. Oper. Res. **273**(2), 488–503 (2019)

A Column Generation Heuristic for Multi-depot Electric Bus Scheduling

Yoann Sabatier Montanaro[1], Thomas Jacquet[1], Quentin Cappart[1,3(✉)], and Guy Desaulniers[1,2]

[1] Polytechnique Montréal, Montreal, Canada
{yoann.montanaro,thomas.jacquet,quentin.cappart,
guy.desaulniers}@polymtl.ca
[2] GERAD, Montreal, Canada
[3] CIRRELT, Montreal, Canada

Abstract. In public transit, the *multi-depot electric vehicle scheduling problem* (MDEVSP) involves assigning a fleet of electric buses to a set of timetabled trips while addressing constraints related to battery recharging. A common approach to solving this problem leverages column generation, wherein a master problem identifies a base solution, and subproblems generate additional schedules to enhance it. These subproblems are often modeled as shortest path problems on a graph, where nodes represent trips and potential recharging opportunities. Prior works have used time-space networks with dynamic selection of recharging opportunities. However, this network type introduces practical limitations, such as the inability to enforce ad-hoc constraints on trip sequences. Recently, Gerbaux et al. (2025) proposed a machine-learning-based method to accelerate the resolution of the subproblems by heuristically reducing their size. While effective, this approach raises concerns in industrial applications, including data privacy for customers and compliance with legal regulations. In this context, we propose a novel approach to model the subproblems within a column-generation-based algorithm for the MDEVSP. Our contributions are as follows: (1) A new graph formulation that preselects recharging opportunities, offering greater flexibility in trip assignment and enabling the integration of ad-hoc constraints; (2) A constructive meta-heuristic, based on a *greedy randomized adaptive search procedure*, to reduce the subproblem size without relying on machine learning or historical data; (3) Additional pruning rules to further reduce the subproblem size. Experimental results, conducted on hundreds of realistic instances derived from real bus lines in Montreal, show that our approach achieves comparable performance to the state-of-the-art method by Gerbaux et al. (2025), while avoiding the use of machine learning.

Keywords: Electric bus scheduling · Column generation · Heuristics · Network reduction

1 Introduction

Climate change presents significant environmental and societal challenges. The transportation sector contributes approximately 14% of global greenhouse gas

G. Tack (Ed.): CPAIOR 2025, LNCS 15763, pp. 103–118, 2025.
https://doi.org/10.1007/978-3-031-95976-9_7

(GHG) emissions [17], playing a major role in urban air pollution and carbon emissions, particularly from diesel-powered public transit systems. Transitioning to electric buses is a critical step toward achieving climate goals, such as *providing affordable and clean energy*, one of the 17 sustainable development goals outlined by the United Nations [1]. Electric buses reduce fossil fuel dependence, lower emissions, and enhance urban air quality, making them an essential component of sustainable urban transportation. A persistent challenge in public transportation is the *vehicle scheduling problem* (VSP), which involves optimizing the allocation of buses to timetabled trips while minimizing operating costs [3,9]. A VSP solution is expressed as a set of *bus schedules*, each of them corresponding to a sequence of trips covered by a bus over, e.g., a day. Integrating electric buses into transit fleets further complicates this task due to additional constraints such as battery management, charging station availability, and limited range [14,15]. The *multi-depot electric vehicle scheduling problem* (MDEVSP) extends the VSP by incorporating multiple depots and electric buses, introducing further complexity in managing operations under charging constraints, where charging times are non-linear [13]. Given that real-world instances often involve timetables with thousands of trips, the development of scalable and efficient solution methods is essential for practical implementation.

Column generation has become a widely used solution approach for solving large-scale VSPs [5,7,12,16], including the MDEVSP [18]. This algorithm decomposes the problem into two components: a master problem which assigns buses to trips, and subproblems, which are formulated as shortest path problems on a time-space network [11] and propose bus schedules to include in the master problem. By iteratively refining the subset of bus schedules considered, column generation efficiently produces high-quality solutions. However, despite its effectiveness, the method faces challenges with the complexity of the MDEVSP, particularly due to the large size of the network in the subproblems. As network sizes increase, computation times grow, potentially rendering the approach impractical for real-world transport operators. To address this computational challenge, Gerbaux et al. [10] recently proposed a learning-based heuristic to reduce the size of the subproblems by predicting which arcs are likely to belong to an optimal solution. While this approach improves computational efficiency, it presents two significant challenges for practical use. First, the time-space network structure employed in their method limits the ability to impose ad-hoc constraints, such as ensuring that a specific trip is never followed by another due to driver restrictions. This flexibility is critical for our industrial partner, GIRO Inc., a leader in developing optimization software for public transit systems. Second, the reliance on machine learning introduces practical issues, including performance degradation on out-of-distribution instances, concerns about customer data privacy, and compliance with legal regulations, such as the *general data protection regulation* (GDPR) in Europe.

Based on this context, this paper addresses both identified issues while aiming to maintain the computational efficiency of Gerbaux et al. [10]. Our contributions are as follows: (1) We propose a new formulation for the subproblem with

precomputed recharging opportunities, allowing trip sequencing constraints to be enforced directly; (2) We develop a constructive meta-heuristic based on the *greedy randomized adaptive search procedure* (GRASP) [8], enabling subproblem size reduction without relying on machine learning or historical data; (3) We introduce additional arc reduction methods based on *trip-to-trip* arc costs and on *relevant recharging options* to further simplify the subproblem. Our approach offers an additional advantage by providing multiple solutions, enabling a trade-off between solution quality and computation time through a Pareto front. This flexibility allows decision-makers to select the most appropriate solution based on their specific requirements. We conduct experiments on hundreds of realistic instances derived from actual bus lines in Montreal, encompassing a range of complexities with up to 3,000 trips and 80 electric buses. The results demonstrate that our matheuristic achieves performance comparable to or better than the state-of-the-art method of Gerbaux et al. [10], while avoiding the use of machine learning.

2 Problem Statement and Current Solution Process

The MDEVSP consists in scheduling a fleet of electric buses to cover a set $T = \{t_1, t_2, \ldots, t_N\}$ of N trips, where each trip $t \in T$ has a departure time q_t^D, an arrival time q_t^E, a departure location l_t^D, and an arrival location l_t^E. The buses are housed in a set D of depots, with each depot $d \in D$ having v_d buses. Each bus has a battery capacity of $\overline{\sigma}$, starts the day fully charged, and returns to its starting depot at the end of the day. Buses can recharge at charging stations in a set H, where station $h \in H$ has u_h chargers available. Recharging follows a piecewise-linear function [13], defined as $\sigma^F = f(\sigma^I, \Lambda)$, where σ^I is the initial *state-of-charge* (SoC), Λ is the charging time, and σ^F is the final SoC. The function f is concave and piecewise-linear, representing different charging rates evolving across time, as shown in [10]. To handle charging sessions, time is discretized into a set $P = \{p_1, p_2, \ldots, p_M\}$ of M consecutive intervals of δ minutes. Each recharging session must be performed over a number of consecutive periods in P. A recharging event will never yield a SoC exceeding battery capacity, but the bus will occupy the charger for all reserved recharging periods.

The objective is to assign trips to buses to minimize operational costs while satisfying some constraints. Each bus follows a schedule starting and ending at the same depot, with battery levels remaining within $[\underline{\sigma}, \overline{\sigma}]$. Buses can return to any depot for a break, provided they stay for at least γ minutes. To prevent excessive idle time, the time between consecutive trips t_1 and t_2 is constrained by $q_{t_2}^D - q_{t_1}^E \leq \beta$. Each trip must be covered exactly once, and the number of buses used at each depot cannot exceed its fleet size. At any time, the number of buses charging cannot exceed the available spots at the charging stations. The operational costs include fixed and variable costs. A fixed cost c^F is incurred for each bus used in the solution. Variable costs include a waiting cost c^W per time unit for idle buses outside the depot, a deadhead cost c^T per time unit for travel without passengers, a penalty c^R for returning to the depot, and charging costs

consisting of a fixed cost c_F^H for initiating a recharge and a variable cost c_V^H per time unit spent at a recharge station.

2.1 Mathematical Formulation

The MDEVSP can be formulated as a *mixed integer linear program*. Let S_d be the set of feasible schedules departing from depot $d \in D$. For each schedule $s \in S_d$, we define the following notation: c_s represents the total cost of schedule s; e_s^t is a binary parameter indicating whether schedule s covers trip t; $g_s^{p,h}$ is a binary parameter indicating whether schedule s includes a recharge at station h during period p; and x_s is a binary variable equal to 1 if schedule s is used in the solution. The formulation is as follows:

$$\min \quad \sum_{d \in D} \sum_{s \in S_d} c_s x_s \tag{1}$$

$$\text{subject to:} \quad \sum_{d \in D} \sum_{s \in S_d} e_s^t x_s = 1 \qquad \forall t \in T \tag{2}$$

$$\sum_{s \in S_d} x_s \leq v_d \qquad \forall d \in D \tag{3}$$

$$\sum_{d \in D} \sum_{s \in S_d} g_s^{p,h} x_s \leq u_h \qquad \forall p \in P, \forall h \in H \tag{4}$$

$$x_s \in \{0, 1\} \qquad \forall d \in D, \forall s \in S_d. \tag{5}$$

The objective function (1) minimizes total operational costs. Constraints (2) ensure that every trip is covered exactly once. Constraints (3) restrict the number of buses used from each depot to its fleet size. Constraints (4) limit simultaneous recharging to the available number of chargers at each station. Finally, constraints (5) enforce binary requirements on the bus schedule variables.

2.2 Learning-Based Column Generation (Gerbaux et al. [10])

This section describes the learning-based column generation heuristic proposed by Gerbaux et al. [10] to solve the MDEVSP. Column generation iteratively solves the linear relaxation of (1)–(4), called the *master problem*, by generating a restricted subset of feasible schedules. At each iteration, subproblems are solved to identify new schedules with negative reduced costs. An integer solution is obtained by applying a diving heuristic that iteratively fixes to 1 the variable x_s with the largest fractional value and re-starts column generation until reaching an integer solution. Our focus is on how the subproblems are modeled and solved, as this is the key area where our contributions provide advancements.

Column Generation. Column generation is an iterative algorithm. At iteration ℓ, the master problem restricted to a subset of its schedules S_d^ℓ for each depot, called the *restricted master problem* (RMP), is first solved to obtain a

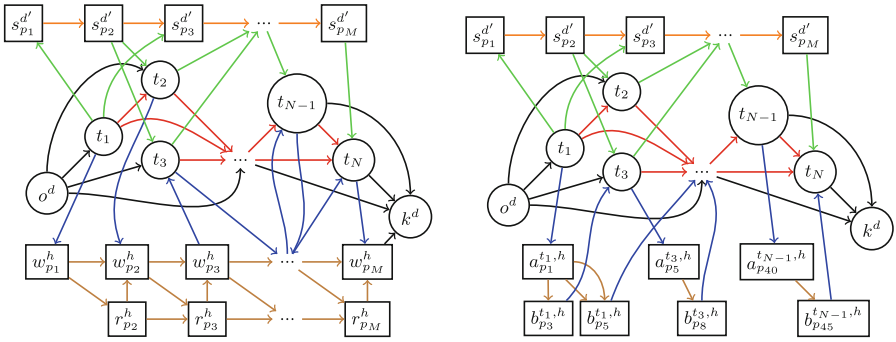
primal solution and a vector of dual variables $(\pi^\ell, \eta^\ell, \nu^\ell)$ corresponding to the trip coverage constraints (2), the charging station capacity constraints (4), and the depot capacity constraints (3), respectively. Then, subproblems are solved to find new schedules with negative reduced costs. The reduced cost of a schedule s is given by:

$$\bar{c}_s^\ell = c_s - \sum_{t \in T} \pi_t^\ell e_s^t - \eta_d^\ell - \sum_{p \in P} \sum_{h \in H} \nu_{p,h}^\ell g_s^{p,h}. \tag{6}$$

If no schedule with a negative reduced cost is found, the RMP solution is also optimal for the master problem. Otherwise, the new schedules are added to the restricted set, and the process repeats until convergence. For the MDEVSP, there is one subproblem per depot, and it can be modeled as a shortest path problem with resource constraints on a time-space network [11], where feasible paths with a negative cost represent promising schedules to add to the RMP.

Solving a Subproblem. The subproblem for depot $d \in D$ can be defined on an acyclic network $\mathcal{G}_d = (\mathcal{V}_d, \mathcal{A}_d)$. The network used by Gerbaux et al. [10] is depicted in Fig. 1a, where h and d' represent arbitrary recharging station and depot, respectively. The node set $\mathcal{V}_d$ includes trip nodes (t), depot nodes (o^d and k^d), recharging nodes (r_p^h, for recharging at station h during period p), and waiting nodes (w_p^h for waiting at station h and $s_p^{d'}$ for waiting at depot d' during period p). There are six types of arcs in set $\mathcal{A}_d$. They are defined as follows:

1. **Pull-out and pull-in arcs** (in black). For each trip $t \in T$, there is a *pull-out arc* (o_d, t), where o_d is the depot node representing the start of the day at depot d. Similarly, there is a *pull-in arc* (t, k_d), where k_d represents the end of the day at depot d. Additionally, for each station h, another *pull-in* arc $(w_{p_M}^h, k_d)$ allows the schedule to end with a recharging event before returning to the depot.
2. **Trip-to-trip arcs** (in red). For any two distinct trips $t_i, t_j \in T$, a *trip-to-trip arc* (t_i, t_j) is created if trip t_j is allowed to follow trip t_i. This is determined by checking if the time difference between the end of t_i and the start of t_j satisfies the constraint $\rho_{l_{t_i}^E, l_{t_j}^D} \leq q_{t_j}^D - q_{t_i}^E \leq \beta$, where $\rho_{i,j}$ is the travel time between any two points i and j in the network (e.g., depots, stations, terminals).
3. **Depot-return arcs** (in green). For each trip and depot, *from-depot arcs* $(s_{p_L}^d, t)$ and *to-depot arcs* $(t, s_{p_E}^d)$ are defined, ensuring that the time constraints for returning to or departing from the depot are satisfied. Here, p_L is the latest period such that the bus can leave the depot and reach trip t and p_E is the earliest period at which the bus can arrive at the depot after completing trip t.
4. **Connection-with-recharge arcs** (in blue). For each charging station $h \in H$, *connection-with-recharge arcs* connect trips to charging operations. There are *from-station arcs* $(w_{p_L}^h, t)$ and *to-station arcs* $(t, w_{p_E}^h)$, where p_L and p_E are defined as above but with respect to station h instead of depot d.
5. **Waiting-at-depot arcs** (in orange). Each depot d has *waiting-at-depot arcs* $(s_{p_i}^d, s_{p_{i+1}}^d)$ between consecutive periods p_i and p_{i+1}.



(a) Network $\mathcal{G}_d$ used by Gerbaux et al. [10]. (b) Proposed network $\mathcal{G}_d^{\text{new}}$.

Fig. 1. Comparison of the network of Gerbaux et al. [10] with ours for modeling the column generation subproblem associated with depot d. (Color figure online)

6. **Recharging arcs** (in brown). Each recharging station has *waiting arcs* $(w^h_{p_i}, w^h_{p_{i+1}})$ and *recharging arcs* $(r^h_{p_i}, r^h_{p_{i+1}})$ between two consecutive periods. It also has $(w^h_{p_i}, r^h_{p_{i+1}})$ and $(r^h_{p_i}, w^h_{p_i})$ arcs to start and to end the recharge, respectively.

Directly from the schedule reduced cost Definition (6), the cost $\overline{c}^{\ell}_{i,j}$ of each arc $(i,j) \in \mathcal{A}_d$ is computed as follows:

$$\overline{c}^{\ell}_{i,j} = \begin{cases} c_{i,j} - (\eta^{\ell}_d + \pi^{\ell}_j) & \text{if } i = o_d \text{ and } j \text{ is a trip} \\ c_{i,j} - \pi^{\ell}_j & \text{if } i,j \text{ are trips} \\ c_{i,j} - \nu^{\ell}_{p,h} & \text{if } j \text{ is recharging} \\ c_{i,j} & \text{otherwise.} \end{cases} \tag{7}$$

The real cost $c_{i,j}$ is computed based on the arc type and the previously defined costs for waiting, deadheading, recharging, etc. Every feasible schedule in S_d corresponds to a path from o^d to k^d in $\mathcal{G}_d$. However, there are paths that do not represent a feasible schedule as the battery autonomy is not taken into account into the network structure. Gerbaux et al. [10] embed these constraints as resources and solve the subproblem using a labeling algorithm. We refer to this paper for a full description of the resources, the labels, and the labeling algorithm.

Learning-Based Network Reduction. A major drawback of this formulation is that the resulting network can become prohibitively large, making it impractical for real-world applications. One way to address this challenge is to reduce the network size by considering only a subset $\hat{\mathcal{A}}_d \subset \mathcal{A}_d$ of arcs that are likely to be part of an optimal solution. Gerbaux et al. [10] tackled this issue using a learning-based approach that combines a constructive heuristic with a graph neural network, an architecture demonstrating significant potential in

combinatorial optimization [4]. Their method achieved a notable reduction in computation time while maintaining a solution quality close to that of the full problem formulation.

3 Reducing the Network Size Without Machine Learning

Although the approach of Gerbaux et al. [10] showed a strong performance, it suffers from several limitations. First, the reliance on historical data for training introduces practical challenges, such as ensuring customer data privacy and the need to frequently update the model with the latest data. Second, the time-space network used (Fig. 1a) restricts the ability to efficiently impose certain constraints, such as forbidding a specific trip to follow another due to driver regulations. As a concrete example, it is not possible to prevent trip t_1 from being followed by trip t_3 in network $\mathcal{G}_d$ without adding an additional resource in the subproblem formulation as there is a path from t_1 to t_3 through the waiting and recharging nodes, e.g., $(t_1, w^h_{p_1}, r^h_{p_2}, r^h_{p_3}, w^h_{p_3}, t_3)$. Note that removing arc $(w^h_{p_3}, t_3)$ is also not desirable as it would prevent any path to t_3 coming from a recharging node. This flexibility to prevent sequence constraints is crucial for adhering to operational requirements in real-world applications. To address these challenges, we enhance the approach of Gerbaux et al. [10] through two key mechanisms: (1) a redesigned network for modeling the subproblems, enabling the integration of additional constraints on trip sequences, and (2) a constructive meta-heuristic based on the *greedy randomized adaptive search procedure* (GRASP) to reduce the subproblem size without relying on machine learning or historical data.

3.1 Redesigning the Subproblem's Network (First Contribution)

In this section, we propose a new acyclic network $\mathcal{G}^{\mathsf{new}}_d = (\mathcal{V}^{\mathsf{new}}_d, \mathcal{A}^{\mathsf{new}}_d)$ to model the subproblem related to depot $d \in D$ (see Fig. 1b). Node set $\mathcal{V}^{\mathsf{new}}_d$ still includes trip nodes (t), depot nodes (o^d and k^d), and waiting-at-a-depot nodes (s^d_p), which remain unchanged. However, recharging nodes and waiting-at-a-station nodes are replaced by two new node types: an *in-station* node $a^{t,h}_p$ and an *out-station* node $b^{t,h}_p$. They respectively represent entering and exiting a recharge station h at period p, immediately after completing trip t. These nodes are specific to each trip. Returning to our previous example, we can now safely forbid trip t_3 from following t_1 by removing the arc $(b^{t_1,h}_p, t_3)$, without forbidding the sequence t_2 to t_3, as t_2 now has its own in-station and out-station nodes. Note that we could also apply a similar modeling for a return to the depot but the transit companies often allow all trip sequences when the trips are separated by a return to the depot because they offer the possibility to change the driver.

The second modification we provide to the network is how the recharging arcs (in brown) are defined. First, we limit these arcs to transitions $(a^{t,h}_p, b^{t,h}_{p'})$ such that $p' > p$. Let $\Delta_{p,p'} = p' - p$ be the number of periods between an *in-station* node $a^{t,h}_p$ and an *out-station* node $b^{t,h}_{p'}$. As a recharging in consecutive periods must occur when a bus visits a recharging station, there are $\frac{\Delta_{p,p'}(\Delta_{p,p'}+1)}{2}$

possible recharging options, namely, one for each pair of recharge starting period and duration. Each option is represented by a specific arc between a pair of nodes $(a_p^{t,h}, b_{p'}^{t,h})$. All the other arcs from the initial network $\mathcal{G}_d$ remain unchanged.

3.2 Reducing the Network's Size (Second Contribution)

Unfortunately, including all possible recharging options for each trip results in an extremely large network that is impractical to use. As a concrete example, in our easiest instance sets (with 687 trips on average and 1 recharging station), the new network (Fig. 1b) contains roughly 59 million arcs, compared to only 20,000 arcs in the initial network (Fig. 1a). This drastic increase in size necessitates a significant reduction in the network's scale. We propose three strategies to address this issue: (1) selecting only a subset of relevant recharging options, (2) employing a heuristic to identify the most promising arcs to keep, (3) filtering *trip-to-trip* arcs based on their deaheading travel time.

Strategy 1: Selecting Relevant Recharging Options. Intuitively, many recharging (or waiting) options can be safely discarded. For instance, it is unlikely that a bus will recharge a large number of periods (e.g., more than 2 h) in the middle of the day. Therefore, we can prune arcs between two nodes $a_p^{t,h}$ and $b_{p'}^{t,h}$ that correspond to such unlikely scenarios. To address this, we introduce five pruning rules. They are as follows:

1. Buses can only recharge for at least ϵ_1 periods and at most ϵ_2 periods after completing a trip. Similarly, buses can only wait for at least ϵ_3 periods and at most ϵ_4 periods. The values ϵ_1, ϵ_2, ϵ_3, and ϵ_4 are hyper-parameters that must be set either by the operators based on their expert knowledge of the specific situation to solve, or by using aggregated statistics on network usage.
2. Waiting is allowed only after recharging and not before. The rationale behind this rule is that a bus that needs to recharge should prioritize this action and wait afterwards before performing the next trip.
3. We forbid accessing a recharging station h from a trip t if the destination of t is too far from it. Formally, let L be the set of all possible trip terminals, $\alpha_1 \in [0, 1]$ be a hyper-parameter that controls how tolerant we are in accepting an arc, and recall that $\rho_{i,j}$ is the travel time between locations i and j. A *connection-with-recharge* arc $(t, a_p^{t,h})$ is preserved only if

$$\rho_{l_t^F,h} \leq \min_{\ell \in L} \rho_{\ell,h} + \alpha_1 (\max_{\ell \in L} \rho_{\ell,h} - \min_{\ell \in L} \rho_{\ell,h}). \tag{8}$$

4. Similarly, we also forbid access to a trip t from a station h if its starting location is too far from it. Arcs $(b_p^{t,h}, t)$ are then kept only if

$$\rho_{h,l_t^D} \leq \min_{\ell \in L} \rho_{h,\ell} + \alpha_1 (\max_{\ell \in L} \rho_{h,\ell} - \min_{\ell \in L} \rho_{h,\ell}). \tag{9}$$

5. Let τ_t be the number of remaining recharging options after a trip t. We keep only $\theta \cdot \tau_t$ of these options, where $\theta \in [0, 1]$. This selection is performed using a weighted probability distribution. Formally, let O be the set of charging options and W_o the number of waiting periods for the option $o \in O$. The weight of each option is defined as $\kappa_o = \frac{1}{W_o+1}, \forall o \in O$. This weighting scheme ensures that options with shorter waiting times are prioritized, thereby reducing the probability of long waiting times. Arcs leading to other recharging options are then removed from the network.

Strategy 2: Selecting Promising Sequences of Trips. Although the previous mechanism allows for a drastic reduction in the number of arcs, it only operates on arcs involving a recharging station. Consequently, we also propose a method to prune other arcs in the network, particularly the arcs linking trips (in red in Fig. 1b), which are numerous. To achieve this, we introduce a constructive and randomized heuristic (e.g., a GRASP without a local search component) to select only arcs that are likely to be part of the optimal solution. The local search is omitted as we want this procedure to operate quickly. It is worth noting that this idea is similar to what was proposed by Gerbaux et al. [10] but does not require the use of machine learning. Briefly, the principle of our method is to generate diverse schedules by executing a randomized constructive heuristic K times. Once completed, we select the union of the arcs obtained at each iteration and combine them with those obtained by the heuristic of Gerbaux et al. [10], but without the learning component. This arc selection is performed a priori and, during the proposed column generation heuristic, the subproblems are solved using a labeling algorithm on the corresponding reduced networks.

Algorithm 1 summarizes this process. We operate on the complete network $(\mathcal{V}^{\mathsf{new}}, \mathcal{A}^{\mathsf{new}}) = (\bigcup_{d \in D} \mathcal{V}_d^{\mathsf{new}}, \bigcup_{d \in D} \mathcal{A}_d^{\mathsf{new}})$ (line 6). The set $\mathcal{A}^{\mathsf{TT}}$ stores the trip-to-trip arcs that have been used in the schedules generated throughout the process. The first main loop (lines 8–33) executes K iterations of the randomized constructive heuristic. Within each iteration, we aim to cover all trips, which are stored in the dynamic set T' (line 9). While there are still uncovered trips (lines 10–33), we initiate a new schedule for a bus starting at a depot d (lines 11–14). This depot is selected randomly among those not yet used. Once all depots have been utilized, the selection restarts randomly from the set of all depots. The first trip to cover is chosen randomly from the μ uncovered trips with the earliest starting times. This trip is marked as covered (line 14).

The next steps mainly consist in sequentially adding activities to the schedule until the final destination k^d is reached (lines 15–33). There are four alternatives for this process, depending on the current location of the bus and its current SoC. These alternatives are:

1. *Feasible trips can be covered (lines 17–21).* From the current position (i.e., a node n), we prioritize uncovered trips; if none are available, we may select a covered trip. The feasibility of a trip t is determined by (i) the existence of an arc $(n, t) \in \mathcal{A}^{\mathsf{new}}$, (ii) sufficient SoC to complete trip t and travel from l_t^E to the nearest depot d'. If such a trip exists, the next trip is selected randomly

Algorithm 1: Randomized constructive heuristic for arc selection.

▷ **Pre:** T is the set of all trips to cover, sorted by earliest start time.
▷ **Pre:** α_2 is the percentage of trips to consider when extending a partial path.
▷ **Pre:** K is the number of iterations of the randomized constructive heuristic.
▷ **Post:** $\mathcal{A}^{\mathsf{grasp}}$, a subset of arcs containing schedules covering all the trips.

$(\mathcal{V}^{\mathsf{new}}, \mathcal{A}^{\mathsf{new}}) := (\bigcup_{d \in D} \mathcal{V}_d^{\mathsf{new}}, \bigcup_{d \in D} \mathcal{A}_d^{\mathsf{new}})$
$\mathcal{A}^{\mathsf{TT}} := \emptyset$ ▷ *Set that will store all used trip-to-trips arcs*
for i **from** 1 **to** K **do**
  $T' := T$ ▷ *Dynamic set containing all trips to cover*
  while $T' \neq \emptyset$ **do**
    $d := \mathsf{getNextDepot}()$
    $n := o^d$ ▷ *Day starts for a bus at depot d*
    $n := \mathsf{getRandomTrip}(n, T', \mu)$ ▷ *Selection among the μ earliest trips*
    $T' := T' \setminus \{n\}$
    while $n \neq k^d$ **do**
      $\mathcal{F} := \{t \in T \mid (n,t) \in \mathcal{A}^{\mathsf{new}} \ \wedge \ \mathsf{isFeasible}(n,t)\}$
      if $\mathcal{F} \neq \emptyset$ **then**
        $t := \Phi(n, \mathcal{F}, \alpha_2)$ ▷ *Randomized selection, Eq. (10)*
        $\mathcal{A}^{\mathsf{TT}} := \mathcal{A}^{\mathsf{TT}} \cup \{(n,t)\}$ ▷ *The trip-to-trip arc is added to $\mathcal{A}^{\mathsf{TT}}$*
        $n := t$ ▷ *Update the current node*
        $T' := T' \setminus \{n\}$
      else if $\mathsf{getSoC}(n) \leq \lambda$ **then**
        $h := \mathsf{getClosestRechargingStation}(n)$
        $a := \mathsf{getStationNode}(n, h)$ ▷ *Node $a_p^{n,h}$ in the subproblem*
        $\mathcal{B} := \{b \in \mathcal{V}^{\mathsf{new}} \mid (a,b) \in \mathcal{A}^{\mathsf{new}}\}$
        $b := \Phi(a, \mathcal{B}, 1)$ ▷ *Node $b_{p'}^{n,h}$ in the subproblem*
        $\mathcal{H} := \{t \in T \mid (b,t) \in \mathcal{A}^{\mathsf{new}} \ \wedge \ \mathsf{isFeasible}(b,t)\}$
        $n := \Phi(b, \mathcal{H}, \alpha_2)$ ▷ *Randomized selection, Eq. (10)*
        $T' := T' \setminus \{n\}$
      else if $\exists\ (n,s) \in \mathcal{A}^{\mathsf{new}} \mid \mathsf{isDepotWaitingNode}(s)$ **then**
        $n := \mathsf{getClosestDepot}(n)$
      else
        $n := k^d$
$\mathcal{A}^{\mathsf{grasp}} := \mathcal{A}^{\mathsf{TT}} \cup \left(\mathcal{A}^{\mathsf{new}} \setminus (T \times T)\right)$
return $\mathcal{A}^{\mathsf{grasp}}$

from a subset of the best feasible ones. The quality of an arc (n,t) is defined by its cost $c_{n,t}$. The random selection of the trip following n among the set of feasible candidates $\mathcal{F}$ is determined by

$$\Phi(n, \mathcal{F}, \alpha_2) \sim \mathsf{Uniform}\Big\{t \in \mathcal{F} \ \Big| \ c_{n,t} \leq c^{\mathsf{min}} + \alpha_2(c^{\mathsf{max}} - c^{\mathsf{min}})\Big\}, \tag{10}$$

where $c^{\min}$ is the cost of the best arc, $c^{\max}$ is the cost of the worst arc, and $\alpha_2 \in [0,1]$ is a hyper-parameter controlling the candidates' quality.

2. *The SoC is low (lines 22–29).* When the bus SoC falls below a threshold λ, it is directed to the nearest recharging station (i.e., a node $a_p^{n,h}$ in the network). The recharging option (i.e., an arc $(a_p^{n,h}, b_{p'}^{n,h})$) is selected randomly from the remaining options after the filtering performed in *Strategy 1*. This is similar to using Eq. (10) with $\alpha_2 = 1$. After recharging, the next trip is selected randomly among the best ones using the same equation.
3. *No feasible trips to cover at the moment (lines 30–31).* If no feasible trip is available, the bus waits at the nearest depot and proceeds to the next iteration of the loop.
4. *The day is over (lines 32–33).* The bus returns to its home depot k^d. If there are still uncovered trips, the outer loop starts a new iteration (lines 15–33).

When the K iterations are completed, the final arc set $\mathcal{A}^{\mathsf{grasp}}$ is obtained by replacing all trip-to-trip arcs by those in set $\mathcal{A}^{\mathsf{TT}}$ (line 34). Note that the execution time of this algorithm is negligible compared to the column generation algorithm executed subsequently. We empirically set the values $\mu = 10$, $\lambda = 50\%$, and $\alpha_2 = 0.1$.

Strategy 3: Filtering Trip-to-Trip Arcs. Alongside the selection proposed in Algorithm 1, we provide a complementary method for reducing the number of trip-to-trip arcs (in red in Fig. 1b). Let again $\mathcal{A}^{\mathsf{new}} = \bigcup_{d \in D} \mathcal{A}_d^{\mathsf{new}}$ be the set of all arcs. Additionally, let $(t, t') \in \mathcal{A}^{\mathsf{new}}$ be a trip-to-trip arc. Briefly, the filtering rule consists in keeping only the best arcs in terms of the value $\rho_{l_t^E, l_{t'}^D}$, i.e., the deaheading travel time required to start a new trip t' after having completed the trip t. The selection follows the same idea as in Eq. (10), where $\alpha_3 \in [0,1]$ is a hyper-parameter controlling the tolerance we want in the selection. The result is the arc set

$$\mathcal{A}^{\mathsf{filtered}} = \mathcal{A}^{\mathsf{new}} \setminus \bigcup_{t \in T} \left\{ (t,t') \in \mathcal{A}_t^{\mathsf{TT}} \mid \rho_{l_t^E, l_{t'}^D} > \rho_t^{\min} + \alpha_3 (\rho_t^{\max} - \rho_t^{\min}) \right\}, \quad (11)$$

where $\mathcal{A}_t^{\mathsf{TT}} = \{(t,t') \in \mathcal{A}^{\mathsf{new}} \mid t' \in T\}$ is the set of trip-to-trip arcs in $\mathcal{A}^{\mathsf{new}}$ originating from trip t, and $\rho_t^{\min}$ (resp., $\rho_t^{\max}$) is the travel time of the best (resp., worst) of its arcs.

4 Computational Experiments

The goal of the experiments is to compare our matheuristic with the learning-based approach proposed by Gerbaux et al. [10]. To achieve this, we reused the same instances, derived from Montreal public transit timetables [2], which are publicly available at https://www.gerad.ca/~guyd/mdevsp.html. These instances are categorized into seven subsets, labeled A through G, to represent a range of problem dimensions and configurations. The subsets exhibit an average

Table 1. Main characteristics of the test instances.

Subset	Network	$\|D\|$	$\|H\|$	No. Trips Avg (Min-Max)	v_d	u_h	No. Instances (tuning/validation/test)
A	1	1	1	687 (568–893)	60	4	500 (350/75/75)
B	1	1	2	687 (568–893)	60	2	500 (350/75/75)
C	1	2	2	687 (568–893)	30	2	500 (350/75/75)
D	1	1	1	687 (568–893)	60	4	500 (350/75/75)
E	2	2	2	787 (669–880)	50	2	500 (350/75/75)
F	2	2	2	1336 (1152–1474)	70	3	200 (140/30/30)
G	2	2	2	2600 (2374–3115)	200	5	50 (35/7/8)

Table 2. Values of some MDEVSP parameters.

$c^F = 1000$	$c^R = 30$	$\underline{\sigma} = 0$	$\beta = 45$
$c^W = 2$	$c_F^H = 30$	$\overline{\sigma} = 100$	$\gamma = 30$
$c^T = 4$	$c_V^H = 2$	$\delta = 15$	

trip count ranging from 687 to 2600 and involve a fleet size ranging from 50 to 200 electric buses. Subsets A to D are based on a first network with four bus lines, characterized by decentralized depots and recharging stations, while the network of subsets E to G features eight lines and more centralized facilities. The core characteristics of each subset are summarized in Table 1. The values of some problem parameters are given in Table 2 (see [10] for more details). Our method is calibrated on 70% of the instances and validated on 15% of the instances. The final results reported hereafter are evaluated on the remaining 15% of the instances, consistent with the evaluation protocol of Gerbaux et al. [10]. The experiments were conducted on a machine equipped with a 12th Gen Intel Core i7-12700 processor (20 cores). The CG heuristic was implemented using the Gencol library [6].

4.1 Benchmarked Column Generation Heuristics

In our experiments, we compare the following heuristics.

1. cgBasic *(baseline)*. The basic column generation heuristic, which relies on the networks $\mathcal{G}_d$ (Fig. 1a) without any arc selection for reducing their size. This method typically provides the best results in terms of solution quality, but is prohibitive in terms of computation time.
2. cgHBD *(Gerbaux et al. [10])*. The machine-learning-based column generation heuristic of Gerbaux et al. [10] (see Sect. 2.2).
3. cgStrategy1 *(ablation)*. The basic column generation heuristic improved with the selection of relevant recharging options (*Strategy 1*) and the new networks $\mathcal{G}_d^{\text{new}}$ (Fig. 1b). The six hyper-parameters $(\epsilon_1, \epsilon_2, \epsilon_3, \epsilon_4, \alpha_1, \theta)$ have been set by manual tuning to yield configuration $(2, 4, 0, 0, 0.3, 0.3)$ for instances A to D, and $(2, 4, 0, 2, 0.3, 0.3)$ for instances E to G.

4. cgStrategy2 *(ablation)*. The column generation heuristic improved only with the arc selection provided by *Strategy 2* with $K = 5$ iterations. The networks $\mathcal{G}_d$ are used.
5. cgFull(K, α_3, ω) *(our complete approach)*. The complete matheuristic combining all our contributions (*Strategies 1, 2, and 3*) using the new networks $\mathcal{G}_d^{\text{new}}$. There are three hyper-parameters that will vary with our experiments: the number of iterations K in Algorithm 1, the tolerance α_3 in Eq. (11), and the decision to take either the union ($\mathcal{A}^{\text{grasp}} \cup \mathsf{A}^{\text{filtered}}$) or the intersection ($\mathcal{A}^{\text{grasp}} \cap \mathsf{A}^{\text{filtered}}$) with the arc subsets obtained with Strategies 2 and 3 ($\omega \in \{\cup, \cap\}$). Intuitively, these options give different trade-offs between computation time and solution quality.

4.2 Results: Trade-Off Between Quality and Computation Time

The main goal of our matheuristic (cgFull) is to reduce the computation time while not significantly degrading the cost of the solution compared to the basic column generation heuristic (cgBasic). Let $K \in \{5, 10\}$, $\alpha_3 \in \{0.1, 0.2, 0.3, 0.5, 1\}$, and $\omega \in \{\cup, \cap\}$ define the range of the hyper-parameters for cgFull. The trade-offs between solution quality and computation time are visualized in Fig. 2 as a Pareto front. In this figure, each point (square or circle) indicates the average results obtained by one configuration of cgFull on all test instances. Configurations with both $\alpha_3 = 1$ and $\omega = \cup$ are not included, as they correspond to cgStrategy1. Also, results with a cost difference exceeding 3% are excluded, as they deteriorate the solution quality too much. From this experiment, we selected three configurations offering different trade-offs:

1. cgFull(5, 1, $\cap$): Prioritizes time reduction, ideal for rapid decision-making scenarios.
2. cgFull(5, 0.3, $\cup$): Minimizes cost difference, suited for cost-sensitive applications.
3. cgFull(10, 0.5, $\cup$): Serves as a compromise between the first two.

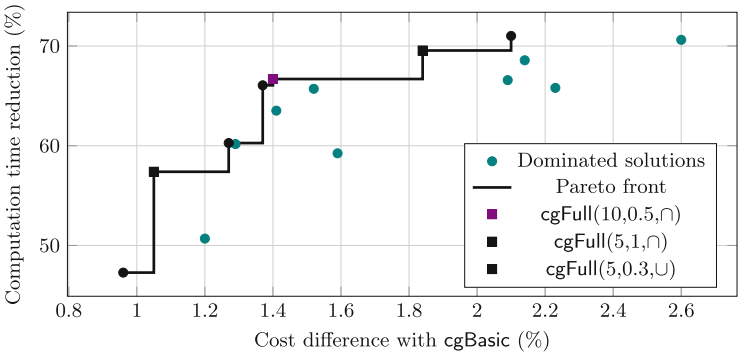


Fig. 2. Pareto front for cgFull.

Table 3. Average results for the instance subsets A to G.

Subset	Algorithm	Time (sec.)	Time Red (%)			Cost Diff (%)			Ratio	No. vehicles	No. arcs
			min	avg	max	min	avg	max			
A	cgBasic	347	–	–	–	–	–	–	–	41.9	20,695
	cgHBD	–	60.1	**70.8**	79.3	−1.7	1.4	6.9	50.6	42.2	3,850
	cgStrategy1	231	18.9	33.4	99.5	−1.3	**1.1**	23.4	30.4	41.8	20,801
	cgStrategy2	113	54.4	66.3	75.6	0.2	1.7	5.2	39.0	42.0	4,672
	cgFull(5,0.3,∪)	140	49.0	58.7	66.8	−1.3	**1.1**	4.8	**53.4**	42.0	8,712
	cgFull(10,0.5,∩)	116	56.6	65.7	72.5	−1.1	1.4	4.5	46.9	42.0	7,032
	cgFull(5,1,∩)	106	57.6	68.5	76.7	−0.7	1.8	5.3	36.1	42.1	6,770
B	cgBasic	430	–	–	–	–	–	–	–	42.0	22,224
	cgHBD	–	57.9	**72.0**	87.1	−1.6	2.1	6.3	34.3	42.4	4,137
	cgStrategy1	266	25.2	38.2	51.4	−2.1	**1.1**	3.3	34.7	42.0	22,647
	cgStrategy2	141	58.8	66.2	74.1	−1.4	1.4	4.4	**47.3**	42.2	5,139
	cgFull(5,0.3,∪)	192	43.1	54.4	64.9	−2.1	1.2	3.5	45.3	42.3	10,722
	cgFull(10,0.5,∩)	150	53.2	63.8	74.0	−2.0	1.7	5.1	37.5	42.4	9,124
	cgFull(5,1,∩)	135	54.7	67.2	75.0	−1.3	1.8	4.3	37.3	42.4	8,860
C	cgBasic	455	–	–	–	–	–	–	–	41.7	25,609
	cgHBD	–	69.6	**74.4**	80.0	−1.8	2.1	6.2	35.4	42.1	5,643
	cgStrategy1	284	29.4	38.4	50.0	−2.3	**0.5**	3.6	76.8	41.7	26,097
	cgStrategy2	139	60.6	68.5	76.4	−1.5	1.6	5.8	42.8	41.8	6,822
	cgFull(5,0.3,∪)	203	3.9	55.7	64.8	−1.9	0.7	2.9	**79.6**	41.7	13,734
	cgFull(10,0.5,∩)	156	56.3	65.2	72.8	−1.1	1.1	4.6	59.3	42.0	12,138
	cgFull(5,1,∩)	146	60.0	67.3	74.1	−1.3	1.1	3.6	61.2	41.9	11,886
D	cgBasic	372	–	–	–	–	–	–	–	41.9	19,726
	cgHBD	–	73.0	**87.5**	96.3	−0.8	1.6	4.7	**54.7**	42.3	3,866
	cgStrategy1	250	17.4	32.5	43.1	−3.3	**0.7**	3.3	46.4	41.9	21,280
	cgStrategy2	117	52.9	67.0	77.2	−2.1	1.4	8.2	47.9	42.1	4,715
	cgFull(5,0.3,∪)	166	−15.3	53.3	63.8	−3.2	1.2	3.8	44.4	42.2	8,847
	cgFull(10,0.5,∩)	121	57.0	66.2	73.7	−1.4	1.5	5.1	44.1	42.2	7,138
	cgFull(5,1,∩)	109	61.4	69.3	77.6	−2.9	1.6	4.7	43.3	42.1	6,833
E	cgBasic	491	–	–	–	–	–	–	–	29.1	23,904
	cgHBD	–	51.7	**58.6**	65.9	-1.6	3.7	8.3	**15.8**	30.3	6,371
	cgStrategy1	427	2.0	13.0	21.3	−1.4	**2.1**	5.2	6.2	29.3	26,234
	cgStrategy2	205	50.1	58.2	66.6	0.8	4.8	9.0	12.1	30.7	7,204
	cgFull(5,0.3,∪)	323	24.5	34.2	45.9	−0.2	2.9	6.6	11.8	30.4	16,116
	cgFull(10,0.5,∩)	284	29.7	41.9	49.0	0.1	3.7	7.9	11.3	30.7	14,996
	cgFull(5,1,∩)	274	33.5	43.9	53.0	0.4	4.5	7.1	9.8	30.9	14,784
F	cgBasic	2147	–	–	–	–	–	–	–	45.3	55,709
	cgHBD	–	60.0	65.7	70.9	0.8	3.2	7.1	**20.5**	46.8	10,021
	cgStrategy1	2158	−10.4	−0.6	6.8	0.6	**2.3**	4.0	−0.3	45.6	65,341
	cgStrategy2	730	61.3	**65.8**	69.4	2.7	5.2	8.0	–	47.6	11,230
	cgFull(5,0.3,∪)	1470	23.7	31.4	39.1	0.6	3.2	5.8	9.8	47.5	34,913
	cgFull(10,0.5,∩)	1132	42.7	47.1	50.9	2.1	4.0	6.1	11.8	47.8	30,678
	cgFull(5,1,∩)	1056	44.6	50.7	56.2	1.4	4.6	7.1	11.0	48.0	30,049
G	cgBasic	17001	–	–	–	–	–	–	–	84.5	167,520
	cgHBD	–	31.5	52.5	62.1	3.3	4.2	6.9	12.5	87.1	17,948
	cgStrategy1	17708	−14.2	−4.7	8.4	1.3	**1.8**	2.4	−2.6	84.1	199,395
	cgStrategy2	5217	58.6	**68.8**	75.0	3.8	5.8	6.9	11.9	88.3	20,763
	cgFull(5,0.3,∪)	10343	27.6	38.7	44.8	1.7	2.5	3.9	**15.5**	86.4	92,381
	cgFull(10,0.5,∩)	7204	48.9	57.1	65.1	3.1	3.8	4.9	15.0	87.4	75,556
	cgFull(5,1,∩)	6746	51.1	60.0	66.5	3.9	4.5	5.6	13.3	87.6	73,592

4.3 Results: Performance of cgFull

The results comparing cgFull with the other column generation heuristics are summarized in Table 3. The absolute computation time of cgHBD has not been reported, as the results were obtained using a different hardware. However, we present the percentage of time reduction relative to cgBasic, as reported in [10]. The *Ratio* column shows the ratio between time reduction and cost difference, where higher values are desirable. The meaning of the other columns is straightforward. For each instance subset, we highlight in bold the best average time reduction, cost difference, and ratio.

Focusing solely on the average computation time, we observe that cgHBD often achieves the greatest reduction. However, this comes at the cost of a larger deterioration in solution quality. Furthermore, the performance of this approach degrades on subsets F and G, likely due to the learned model's difficulty in generalizing to larger instances. In terms of solution quality, cgStrategy1 delivers the best results but offers only modest time reductions (less than 40% on subsets A to E) and almost no improvement on the largest instances. On the other hand, cgStrategy2 provides balanced results but, as it is based on the time-space networks $\mathcal{G}_d$, it does not leverage the increased flexibility of the proposed networks $\mathcal{G}_d^{\text{new}}$. Interestingly, the variants of our complete matheuristic (cgFull) also deliver balanced results while benefiting from the enhanced flexibility offered by the proposed networks.

5 Conclusion

This paper proposed a new column generation heuristic for solving the MDEVSP, which is of great interest to public transit companies. Three key contributions were made: (1) a new formulation for the subproblem with precomputed recharging opportunities, enabling trip sequencing constraints to be enforced directly, (2) a randomized constructive meta-heuristic to reduce subproblem size without relying on machine learning or historical data, and (3) a secondary arc reduction method based on expert rules to further simplify the subproblem. Experiments showed an average computation time reduction of 63% for small instances with cost increases under 1.4%, and 45% for large instances with a 3.7% average cost increase. These results highlight the method's practicality and efficiency in real-world scenarios. This framework provides a flexible and robust tool for managing large-scale networks, meeting operational constraints without compromising solution quality. Consequently, it supports the adoption of electric buses, promoting sustainable urban mobility. While our filtering rules are heuristic in nature, they align with business needs of our partner Giro Inc. Future work could explore alternative heuristics and extend our approach to other electric vehicle routing problems.

Acknowledgments. This work was funded by the Natural Sciences and Research Council of Canada and GIRO Inc., whose support has been instrumental in the advancement of this research. We thank the team at GIRO for their valuable insights and collaboration throughout the project.

References

1. Arora, N.K., Mishra, I.: United nations sustainable development goals 2030 and environmental sustainability: race against time. Environ. Sustain. **2**(4), 339–342 (2019)
2. Brasseur, J.: Accélération d'une méthode d'agrégation dynamique de contraintes par apprentissage automatique pour le problème de construction d'horaires de conducteurs d'autobus. Master's thesis, Polytechnique Montréal (2022)
3. Bunte, S., Kliewer, N.: An overview on vehicle scheduling models. Public Transp. **1**(4), 299–317 (2009)
4. Cappart, Q., Chételat, D., Khalil, E.B., Lodi, A., Morris, C., Veličković, P.: Combinatorial optimization and reasoning with graph neural networks. J. Mach. Learn. Res. **24**(130), 1–61 (2023)
5. Costa, L., Contardo, C., Desaulniers, G.: Exact branch-price-and-cut algorithms for vehicle routing. Transp. Sci. **53**, 946–985 (2018)
6. Desrosiers, J.: GENCOL: Une équipe et un logiciel d'optimisation. Groupe d'études et de recherche en analyse des décisions (2010)
7. Feillet, D.: A tutorial on column generation and branch-and-price for vehicle routing problems. 4OR **8**(4), 407–424 (2010)
8. Feo, T.A., Resende, M.G.: A probabilistic heuristic for a computationally difficult set covering problem. Oper. Res. Lett. **8**(2), 67–71 (1989)
9. Foster, B.A., Ryan, D.M.: An integer programming approach to the vehicle scheduling problem. J. Oper. Res. Soc. **27**(2), 367–384 (1976)
10. Gerbaux, J., Desaulniers, G., Cappart, Q.: A machine-learning-based column generation heuristic for electric bus scheduling. Comput. Oper. Res. **173**, 106848 (2025)
11. Irnich, S., Desaulniers, G.: Shortest path problems with resource constraints. In: Column generation, pp. 33–65. Springer (2005)
12. Massen, F., Deville, Y., Van Hentenryck, P.: Pheromone-based heuristic column generation for vehicle routing problems with black box feasibility. In: Beldiceanu, N., Jussien, N., Pinson, É. (eds.) CPAIOR 2012. LNCS, vol. 7298, pp. 260–274. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29828-8_17
13. Montoya, A., Guéret, C., Mendoza, J.E., Villegas, J.G.: The electric vehicle routing problem with nonlinear charging function. Transp. Res. Part B: Methodol. **103**, 87–110 (2017)
14. Pasha, J., et al.: Electric vehicle scheduling: state of the art, critical challenges, and future research opportunities. J. Industr. Inf. Integr. **38**, 100561 (2024)
15. Perumal, S.S., Lusby, R.M., Larsen, J.: Electric bus planning & scheduling: a review of related problems and methodologies. Eur. J. Oper. Res. **301**(2), 395–413 (2022)
16. Taillard, É.D.: A heuristic column generation method for the heterogeneous fleet VRP. RAIRO-Oper. Res. **33**(1), 1–14 (1999)
17. United States Environmental Protection Agency: Global greenhouse gas emissions overview (2023). https://www.epa.gov/ghgemissions/global-greenhouse-gas-overview. Accessed 01 Nov 2024
18. Wu, W., Lin, Y., Liu, R., Jin, W.: The multi-depot electric vehicle scheduling problem with power grid characteristics. Transp. Res. Part B: Methodol. **155**, 322–347 (2022)

On the Efficiency of Algebraic Simplex Algorithms for Solving MDPs

Dibyangshu Mukherjee(✉) and Shivaram Kalyanakrishnan

Indian Institute of Technology Bombay, Mumbai 400076, India
{dbnshu,shivaram}@cse.iitb.ac.in

Abstract. Markov Decision Problems (MDPs) are a widely-used abstraction of sequential decision-making. A *policy* specifies an action to take from each state of the MDP. Every MDP has an *optimal* policy, which maximises the expected long-term reward from each starting state. A well-known approach to compute an optimal policy for a given MDP is by solving an induced Linear Program (LP). This paper establishes the computational efficiency of a family of "Algebraic Simplex" (AS) algorithms for solving these induced LPs. Unlike geometric methods that rely on quantities such as "value", "gain", or "flux", AS algorithms query policies sequentially for the *discrete set* of locally-improving state-action pairs. We provide upper bounds on the complexity of AS algorithms in three settings: (1) when there are many more states than actions, (2) when there are many more actions than states, and (3) when transitions are deterministic. For all three cases, we furnish AS algorithms with running-time upper bounds that are within a polynomial factor of the tightest known yet across all algorithms. These results demonstrate that the use of geometric information and random access memory do not contribute substantively to the established efficiency of state-of-the-art algorithms.

Keywords: Markov Decision Process · Linear Programming · Simplex · Policy Iteration · Abstract Models

1 Introduction

Markov Decision Problems (MDPs) [32] are a well-studied and widely-applied formalism of long-term decision making under uncertainty. The most common approaches to solve MDPs—such as value iteration, policy iteration, and linear programming—were originally proposed as early as 7–8 decades back. The last 3–4 decades have also witnessed several results published on the computational complexity of solving MDPs, and of specific algorithms. Since MDPs can be solved through induced linear programs, the emergence of poly-time algorithms for linear programming [21,22] has implied that the number of arithmetic operations required to solve any MDP is at most a polynomial in the number of bits used to encode the MDP. In the other direction, it has been shown that solving MDPs under common reward structures such as finite horizon, infinite horizon with discounted rewards, and infinite horizon with average reward, is *P*-complete [30].

G. Tack (Ed.): CPAIOR 2025, LNCS 15763, pp. 119–136, 2025.
https://doi.org/10.1007/978-3-031-95976-9_8

An alternative measure of computational complexity arises from the "real RAM" or "infinite precision arithmetic" model, under which any arithmetic operation on real-valued operands can be performed in constant time, regardless of the bit-size [29]. This perspective furnishes running-time bounds that depend only on the number of states and actions in the input MDP: in particular the bounds do not depend on the bit-size of real-valued parameters such as the transition probabilities, rewards, and discount factor. It remains unknown if general MDPs can be solved in *strongly polynomial* time: that is, by using at most $\text{poly}(n, k)$ real RAM operations, where $n \geq 2$ denotes the number of states, and $k \geq 2$ denotes the number of actions in the input MDP.

In this paper, we consider the complexity of solving MDPs under the real RAM computational model. Concretely, we take long-term rewards to be infinite discounted sums. We review three algorithms for this setting, chosen because they provide the tightest upper bounds currently known.

1. MDPs with $n \geq 2$ states and $k = 2$ actions induce acyclic unique sink orientations (AUSOs) of n-dimensional hypercubes [36], which the Random Facet algorithm [12] can solve using $\text{poly}(n) \cdot O(e^{2\sqrt{n}})$ operations in expectation. For $k \geq 2$, the upper bound can be generalised to $\text{poly}(n, k) \cdot e^{O(\sqrt{n \log nk})}$ [18,27]. For any fixed $k > 2$, this upper bound has the slowest (sub-exponential) growth rate in n. Let us denote the generalised version of the Random Facet algorithm $\mathcal{L}_1$.
2. Clarkson [7] proposed a randomised recursive algorithm to solve linear programs with a large number of constraints. Combined with the Random Facet algorithm it is possible to solve LPs induced by n-state, k-action MDPs with a guaranteed running-time of $O(n^3 k + e^{O(\sqrt{n \log n})})$ expected operations. This upper bound is only linear in k, and hence tighter than the one for $\mathcal{L}_1$ when $k \gg n$ (albeit looser when $n \gg k$). For our purposes, denote by $\mathcal{L}_2$ the algorithm of Clarkson.
3. An interesting sub-class of MDPs, called Deterministic MDPs (DMDPs) are those in which all transitions are deterministic. It is well known that DMDPs can be solved in strongly polynomial time—that is, using $\text{poly}(n, k)$ operations. Of the several algorithms that do enjoy polynomial running time on DMDPs [1,25,31], denote the one of Post and Ye [31] as $\mathcal{L}_3$. This algorithm implements the "max gain" simplex procedure on an LP induced by the input MDP.

From a distance, $\mathcal{L}_1$, $\mathcal{L}_2$, and $\mathcal{L}_3$ appear quite different in the way they consume and manipulate information from the input MDP. $\mathcal{L}_1$ and $\mathcal{L}_2$ operate on the set of constraints of an induced LP, eliminating constraints by random testing. Critical to the analysis of $\mathcal{L}_3$ is its choice of the single "max-gain" action for each policy update; other Simplex variants are known with even exponential lower bounds on their running time [2]. In this paper, we demonstrate that in spite of their apparent disparity, $\mathcal{L}_1$, $\mathcal{L}_2$, and $\mathcal{L}_3$ can all be implemented (albeit with minor modifications) as "Algebraic Simplex" (AS) algorithms. In the remainder of this section, we provide an informal introduction to this class of algorithms. Formal definitions follow in Sects. 2 and 3.

1.1 Algebraic Simplex Family of Algorithms

In a commonly-used formulation of an LP from an MDP, vertices of the feasible polytope of the LP are in 1-1 correspondence with (deterministic) policies of the MDP [31].

In this feasible polytope, edges connect any two policies that differ in their action for exactly one state. Simplex algorithms begin at some vertex of the LP, and repeatedly update to a neighbouring vertex that improves the objective function. In general, Simplex algorithms can use any available information—such as the real-valued objective function (or "gain") along each edge, the geometric coordinates of the vertex, and so on—for selecting the next vertex at any iteration. We denote as *Algebraic Simplex* (AS) algorithms the subclass of Simplex algorithms that are only provided at each iteration the *set* of neighbouring vertices that improve the objective function (equivalently the set of improving state-action pairs). Algorithms from the AS family do not use any geometric information; for example, $\mathcal{L}_3$ [31], which implements the "max-gain" pivot rule, does not belong to the AS family. Note that standard policy iteration algorithms [16] that "jump" in the space of policies (equivalently, which switch multiple state-action pars at each iteration) are also not from the AS family.

The withholding of geometric information would appear rather restrictive. As we demonstrate in Sect. 2.2, the states values (that is, the expected long-term rewards) of only $\text{poly}(n, k)$ policies are needed to construct the LP induced by an MDP—and this LP, in turn, determines the set of optimal policies for the MDP. Hence, from an information-theoretic standpoint, only $\text{poly}(n, k)$ queries of the MDP are needed if each query—of a policy—can return the state values under that policy. On the other hand, if querying a policy only returns its discrete, locally-improving set of policies, how many queries are needed? And how much further limiting is the constraint of having to move only to some neighbouring vertex in the LP's feasible polytope?

Our contribution is to illustrate the surprising finding that algorithms from the AS family *match* the known upper bounds for $\mathcal{L}_1$, $\mathcal{L}_2$, and $\mathcal{L}_3$ up to a $\text{poly}(n, k)$ factor. In Sect. 4, we propose an implementation of $\mathcal{L}_1$ as a member of AS, denoting our variant $\mathcal{L}'_1$. Similarly, in Sect. 5, we implement $\mathcal{L}_2$ an AS algorithm $\mathcal{L}'_2$. For DMDPs, we propose an AS algorithm $\mathcal{L}'_3$, which is inspired by $\mathcal{L}_3$, but different. We show in Sect. 6 that $\mathcal{L}'_3$ is also strongly polynomial for DMDPs. Not surprisingly, a key element of AS algorithms $\mathcal{L}'_1$, $\mathcal{L}'_2$, and $\mathcal{L}'_3$ is the effective use of memory, which, nonetheless, remains polynomially-sized in the number of states and actions.

MDPs are at the heart of planning and reinforcement learning [28,35]—topics pursued by the artificial intelligence community. The theoretical analysis of MDPs (and their induced LPs) has been of interest within operations research [31,38] and combinatorial optimisation [33,36]. Our results add to this literature. We begin with formal definitions of MDPs (Sect. 2) and the AS family (Sect. 3).

2 MDPs and Induced LPs

A **Markov Decision Problem** (**MDP**) models the interaction between an agent and a stochastic environment that the agent navigates sequentially. An MDP $M = (S, A, T, R, \gamma)$ consists of a set of states S, a set of actions A, a transition probability function T, a reward function R, and a discount factor $\gamma \in [0, 1)$. We assume S and A are finite, with $|S| = n \geq 2$ and $|A| = k \geq 2$. If an agent chooses action $a \in A$ from state $s \in S$, then the probability of transitioning to state s' is denoted by $T(s, a, s')$, and the associated reward is given by $R(s, a) \in \mathbb{R}$.

Formally, the agent's behaviour is encoded as a **policy**, which is a mapping from the set of states to the set of actions. For a policy $\pi : S \to A$, the **value function** V^π characterises the expected long term reward while adhering to π. It can be recursively defined using the Bellman Equations for $s \in S$ as follows:

$$V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') \cdot V^\pi(s').$$

We define two natural partial orders, $\geq$ and $\succ$, on the set of policies Π. For policies $\pi, \pi' \in \Pi$, we define $\pi \geq \pi' \iff (V^\pi(s) \geq V^{\pi'}(s),\ \forall s \in S)$. Also, $\pi \succ \pi' \iff (\pi \geq \pi'$ and $\exists s \in S$ such that $V^\pi(s) > V^{\pi'}(s))$. It is well known that there exists a policy $\pi^\star \in \Pi$ which maximises the values of all states: that is, $V^{\pi^\star}(s) \geq V^\pi(s),\ \forall s \in S, \forall \pi \in \Pi$ (equivalently, $\pi^\star \geq \pi$ for all $\pi \in \Pi$) [3]. To simplify exposition, we assume that the optimum is unique. Thus, given an MDP M, the task is to compute the optimal policy for M.

In addition to value vectors, some other quantities are useful while designing and analysing algorithms for MDPs. The **action value function** Q^π of policy π gives for each $(s, a) \in S \times A$ the expected total discounted reward obtained by taking action a from state s and following policy π afterwards. We can write it as:

$$Q^\pi(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') \cdot V^\pi(s').$$

The **flux** x^π of a policy π represents the total discounted number of times each $(s, a) \in S \times A$ is visited when the MDP starts simultaneously in all states and follows the Markov chain T^π, which is defined by taking action $a = \pi(s)$ at each state s, discounting by γ at each step. If an action a is unused by π, then $x^\pi(s, a) = 0$; otherwise, for $s \in S$,

$$x^\pi(s, \pi(s)) = 1 + \gamma \sum_{s' \in S} T(s', \pi(s'), s) \cdot x^\pi(s', \pi(s')).$$

The **gain** of $a \in A$ for state $s \in S$ with respect to a policy π is defined as: $G^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$.

2.1 LPs and Induced LPs

A linear programming problem is defined by a set of linear inequality constraints H and an objective vector $c \in \mathbb{R}^d$, where each constraint $h \in H$ has the form $a^T x \leq b$ with $a \in \mathbb{R}^d$, $b \in \mathbb{R}$, and $x \in \mathbb{R}^d$.

Let $\mathcal{P}(H)$ be the polyhedron defined by H. The objective is to find a point $x \in \mathcal{P}(H)$ that minimises $c^T x$. We define the **value** of H, denoted as v_H, as the lexicographically smallest point in $\mathcal{P}(H)$. If $\mathcal{P}(H) = \emptyset$, then $v_H = \infty$, and if $\mathcal{P}(H) \neq \emptyset$ but contains no minimum, then $v_H = -\infty$.

A set $B \subseteq H$ with $|B| = d$ is called a (feasible) **basis** if there exists a unique solution x_B. The point x_B corresponds to a vertex of the polyhedron defined by H. The value of a basis B is given by $v_B = c^T x_B$, and for any proper subset $B' \subset B$, it follows that $v_{B'} < v_B$. If $v_H > -\infty$, a basis of H is a minimal subset $B \subseteq H$ such that $v_B = v_H$. A constraint $h \in H$ is said to be **violated** by H if $v_H < v_{H \cup \{h\}}$.

The problem of finding an optimal policy of an MDP can be formulated as a linear program (LP). In this framework, the solution to the primal LP yields the optimal values associated with the MDP, while the solutions to the dual LP encapsulate the flux of the optimal state-action pairs. The primal and dual LPs [24] are given below.

$$\textbf{Primal:} \quad \text{Minimise: } \sum_{s' \in S} V(s')$$

$$\text{Subject to: } V(s) \geq R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V(s'), \qquad \forall s \in S, \forall a \in A.$$

$$\textbf{Dual:} \quad \text{Maximise: } \sum_{s \in S} \sum_{a \in A} R(s,a) \cdot x(s,a)$$

$$\text{Subject to: } \sum_{a \in A} x(s,a) = 1 + \gamma \sum_{s' \in S} \sum_{a \in A} T(s',a,s) \cdot x(s',a), \quad \forall s \in S.$$

The constraints of the induced primal LP correspond bijectively to the state-action pairs of the MDP, resulting in a total of n variables and $m = nk$ constraints. Similarly, there exists a bijection between the policies of the MDP and the vertices of the induced dual LP [31]. Our work primarily consists of the induced primal LP, although the reader may find useful to conceptualise certain ideas through the dual.

The **LP digraph** of a Markov Decision Problem (MDP) is a directed graph where each vertex corresponds to a policy (feasible basis of the induced dual LP), and a directed edge exists between two vertices if one policy can be obtained from the other by improving a single state-action pair.

2.2 A Polynomial-Time Extraction of MDP Information

Below, we present a procedure that uses a polynomial number of evaluations to extract all the necessary information to solve an MDP. This is achieved by utilising value functions, which serve as geometric coordinates. In contrast, the LP Digraph represents a much more restricted setting where such extraction is not feasible.

Consider the primal linear programming formulation of an MDP M, where each constraint corresponds to a state-action pair. For state s and action a, the hyperplane is:

$$H_{s,a} = \left\{ V \in \mathbb{R}^n \;\middle|\; V(s) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V(s') \right\}$$

With n states and k actions per state, there are n sets of k hyperplanes, resulting in a total of nk hyperplanes. Since a policy π prescribes exactly one action per state, its value can be represented in $\mathbb{R}^n$ as: $V^\pi = \bigcap_{s \in S} H_{s,\pi(s)}$, where the intersection captures the value function V^π, satisfying all constraints associated with π. For a given state-action pair (s,a), we define the set of policies: $\Pi_{s,a} = \{\pi \mid \pi(s) = a\}$. Clearly, $|\Pi_{s,a}| = k^{n-1}$ and for $\pi \in \Pi_{s,a}$, $V^\pi \in H_{s,a}$.

At each step, the algorithm selects an arbitrary subset $P_{s,a}$ of n policies from $\Pi_{s,a}$ and evaluates their value functions, yielding their coordinates in $\mathbb{R}^n$. Since n non-collinear points in $\mathbb{R}^n$ uniquely determine a hyperplane, these evaluations suffice to

construct the hyperplane $H_{s,a}$. By repeating this process for all state-action pairs, the algorithm constructs all nk hyperplanes using at most n^2k policy evaluations.

With this information extracted from the MDP, the optimal value function $V^\star$ can be solved as: $V^\star = \max_{\pi \in \Pi} \left\{ \bigcap_{s \in S} H_{s,\pi(s)} \right\}$ without any further access to the MDP.

This highlights how value functions encode rich structural information about the MDP, enabling efficient computation of optimal policies. However, our approach circumvents the use for such continuos data, relying instead on discrete inputs—improving state-action pairs—to solve the MDP.

2.3 Simplex and Policy Iteration

The Simplex method operates through single-step transitions, referred to as pivots, shifting from one feasible vertex to an adjacent feasible vertex with a strictly higher objective on the LP-digraph. This iterative process persists until the algorithm converges to an optimum, which corresponds to a vertex with the highest objective.

The choice of pivot rule is key in differentiating Simplex methods and determining their complexity. While the original pivot rule proposed by Dantzig [9] remains the most commonly employed in practice, Klee and Minty [23] demonstrated that this rule can necessitate an exponential number of steps to converge. Numerous other pivot rules have subsequently been observed to exhibit similar behaviour [10,14,17].

On the other hand, Kalai [18] introduced a randomised algorithm akin to the Simplex method, which achieved a subexponential bound of $2^{O(\sqrt{n \log m})}$ operations for solving an LP with n variables and m constraints. Matousek, Sharir, and Welzl [27] established an identical bound for the dual version algorithm. When combined with Clarkson's algorithm [7], their bound resolves to $O(n^2 m + e^{O(\sqrt{n \log n})})$. These bounds stand as the tightest known for solving LPs, without any derandomisations currently known.

Policy iteration (PI) [16] can be viewed as a generalised version of the Simplex method, where multiple simultaneous pivots or "jumps" are possible. It is a commonly used algorithm for solving Markov Decision Problems (MDPs). For a policy π, define the set of **improving state-actions pairs**, J^π, as follows:

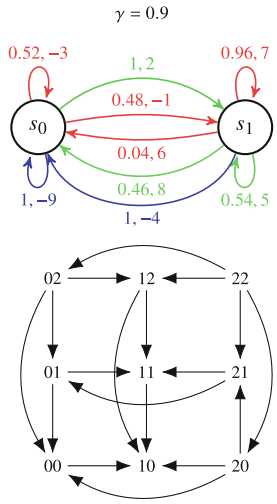
$$J^\pi = \{(s,a) \in S \times A \mid Q^\pi(s,a) > V^\pi(s)\}.$$

Policy Iteration algorithms rely on the following Policy Improvement Theorem, which states that if a policy has no improving state-action pair, it is already optimal. Otherwise, switching any improving pair yields a policy with a higher value. Formally,

Theorem 1. *For $\pi \in \Pi$: If $J^\pi = \emptyset$ then $\pi = \pi^\star$ otherwise for $(s_0, a_0) \in J^\pi$, define π' such that $\pi'(s_0) = a_0$ and $\pi'(s) = \pi(s)$ for all $s \in S,\ s \neq s_0$. Then $\pi' \succ \pi$.*

Starting from an initial policy, PI involves iteratively navigating through evaluation and improvement steps until reaching an optimal policy. Policy evaluation only needs $\text{poly}(n,k)$ arithmetic operations. For general MDPs the best known upper bounds on the number of iterations are exponential in the number of states n [19,26]. In case of Deterministic MDPs, Post and Ye [31] obtained the tightest known bounds of $O(n^5 k^2 \log^2 n)$ iterations for the max-gain simplex method.

Figure 1 illustrates an MDP with 2 states and 3 actions. The table lists the nine possible policies, their corresponding value functions, and the improving state-action pairs. The LP-digraph shows the possible simplex trajectories. Consider two simplex procedures starting with the initial policy $\pi_0 = 22$. One takes the following path: $22 \to 02 \to 12 \to 11 \to 10$, requiring 5 policy evaluations. The other takes a shorter trajectory: $22 \to 20 \to 10$, reducing the number of evaluations to 3.



π	V^π	J^π
0, 0	48.05, 63.89	{(1, 0)}
0, 1	18.05, 26.95	{(0, 0), (1, 1)}
0, 2	−26.31, −27.68	{(0, 0), (0, 1), (1, 2)}
1, 0	63.08, 67.87	∅
1, 1	47.87, 50.97	{(1, 0)}
1, 2	−8.42, −11.57	{(1, 0), (1, 1)}
2, 0	−90, 27.35	{(0, 0), (1, 0), (2, 1)}
2, 1	−90, 60.07	{(0, 1), (1, 1)}
2, 2	−90, −85	{(0, 2), (1, 2), (2, 0), (2, 1)}

Fig. 1. A 2-state, 3-action MDP and its LP digraph. The red, green, and blue edges correspond to actions 0, 1, and 2 respectively, and the tuples represent the transition probability and the rewards. The corresponding table contains values and improving state-action pairs of policies. (Color figure online)

3 Algebraic Simplex Computation Model

We introduce a new coordinate-free combinatorial model of computation for solving MDPs, called Algebraic Simplex (**AS**). AS differs from standard PI algorithms in two key aspects: its algebraic property and its utilisation of a polynomial-sized memory.

Starting from an initial policy π_0, at each iteration i of a standard PI algorithm, a mapping is proposed from the action values of the policy to a subset of improving pairs. This mapping can be expressed as: $Q^{\pi_i} \mapsto U \in J^{\pi_i}$ where U is the set of state-action pairs the algorithm chooses to switch to get to π_{i+1}. In contrast an AS algorithm implements a mapping $(J^{\pi_i}, \mathcal{M}_i) \mapsto (j \in J^{\pi_i}, \mathcal{M}_{i+1})$ where j is a (single) state-action pair selected for switching, and $\mathcal{M}_i, \mathcal{M}_{i+1}$ are bit vectors of polynomial size. Additionally, the computational complexity of each step is polynomially bounded in n and k. For randomised PI/AS algorithms, the mapping is stochastic (we can equivalently assume that the input includes a random seed). A significant aspect of AS is that its memory set

lacks access to any geometric information of the policies. The policy evaluation step is replaced by a query which returns the set of improving state-action pairs available from the current policy. Figure 2 illustrates the AS model. We review the literature on LPs along the key dimensions that distinguish AS.

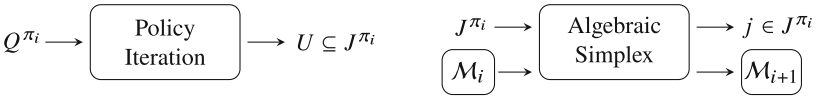


Fig. 2. An illustration of the AS model along with the standard PI model. The incoming/outgoing edges represent input/output to the algorithm.

3.1 Usage of Geometry, Randomness, Memory, and Jumping to Solve LPs

There has been a longstanding pursuit for a strongly polynomial algorithm for linear programming. While polynomial-time algorithms such as the ellipsoid method [22] and Karmarkar's algorithm [21] exist, their running time depends on the bit representation of the input. Notably, strongly polynomial bounds for linear programming have been attainable when the geometry of the feasible region is favourable.

Consider the Geometric Random-Edge algorithm [11], a variant of the Random-Edge pivot rule [13] for the simplex method. This algorithm applies when the constraint matrix A satisfies a geometric property introduced by Brunsch and Röglin [5]: the sine of the angle of a row of A to a hyperplane spanned by $n-1$ other rows of A is at least δ. Under this condition, the expected running time is $\text{poly}(n, m, 1/\delta)$.

Policy iteration (PI) algorithms can leverage geometry, memory, and jump operations. An example is Geometric Policy Iteration (GPI), which follows its own distinct value function update scheme. The Line Theorem [8] states that policies differing in a single state correspond to the same line segment in the value function polytope. GPI exploits this structure by preventing updates along the same line segment, achieving performance that matches the best-known bounds for solving MDPs (see Table 1).

Another example that uses memory and jump operations is the Fibonacci-Seesaw algorithm [36], which recursively solves acyclic unique sink orientations (AUSOs) by exploring antipodal vertices. This approach extends to solving k-action MDPs using recursive AUSOs [15], a generalisation of AUSOs, in $\text{poly}(n, k) \cdot k^{0.68n}$ iterations.

Randomisation indeed plays a crucial role in complementing geometric approaches. For instance, consider Megiddo's multidimensional search algorithm for LPs [29], which uses geometry to achieve a deterministic algorithm with a bound linear in m but doubly exponential in n. Clarkson's algorithm can also be derandomised [6] to obtain a deterministic algorithm that uses epsilon nets, a geometric property. Without geometry, deterministic algorithms for MDPs currently only achieve bounds exponential in n.

We summarise the features and upper bounds of various algorithms in Table 1. In the following sections, we introduce algorithms based on our AS model. These algorithms begin with an arbitrary policy π_0 and a suitably initialised memory $\mathcal{M}$. At each iteration,

they compute the next policy and memory state using the function *next* until the optimal policy is reached.

Table 1. Algorithms and their key component features across four primary categories. It is not apparent from their original descriptions that $\mathcal{L}_1$ [27] and $\mathcal{L}_2$ [7] do not need geometry and jumping. By proposing $\mathcal{L}'_1$ and $\mathcal{L}'_2$ as replacements, we bring out this aspect explicitly.

Algorithm	Running-time Upper Bound	Geometry	Randomness	Memory	Jump
Geometric RE [11]	$poly(n, m, 1/\delta)$	✓	✓	×	×
HPI [38]	$poly(n, k, 1/1-\gamma)$	✓	×	×	✓
Simplex [38]	$poly(n, k, 1/1-\gamma)$	✓	×	×	×
GPI [37]	$poly(n, k, 1/1-\gamma)$	✓	×	✓	✓
Megiddo [29]	$poly(m) \cdot exp(exp((n))$	✓	×	✓	✓
D-Clarkson [6]	$poly(m) \cdot exp(n)$	✓	×	✓	×
RSPI [20]	$poly(n, k) \cdot exp(n)$	✓	✓	×	×
FS [15,36]	$poly(n, k) \cdot exp(n)$	×	×	✓	✓
$\mathcal{L}_1$ [27]	$poly(n, k) \cdot exp(\sqrt{n \log nk})$	×	✓	✓	×
$\mathcal{L}_2$ [7,27]	$poly(n, k) \cdot exp(\sqrt{n \log n})$	×	✓	✓	×
$\mathcal{L}_3$ [31]	$poly(n, k)$	✓	×	×	×
$\mathcal{L}'_1$ (this paper)	$poly(n, k) \cdot exp(\sqrt{n \log nk})$	×	✓	✓	×
$\mathcal{L}'_2$ (this paper)	$poly(n, k) \cdot exp(\sqrt{n \log n})$	×	✓	✓	×
$\mathcal{L}'_3$ (this paper)	$poly(n, k)$	×	×	✓	×

4 $\mathcal{L}_1$: Random-Facet

The **Random-Facet** (R-F) [27] algorithm for solving LPs proceeds as follows. Given a set H of m constraints, the algorithm selects a constraint h uniformly at random from $H \setminus C$, where C is a given input basis, initially arbitrary. Subsequently, it computes a basis B for $H \setminus h$ recursively. If h is *not* violated by B, then B serves as a basis for H, marking the completion of the algorithm. However, if h is violated by B, then h is added to B and the recursion continues. Pseudocode is given in Algorithm 1. The algorithm simplifies in the case of combinatorial cubes [12].

4.1 Algorithm: $\mathcal{L}'_1$

Recall that in MDPs, each policy corresponds to a basis in the dual LP. We implement R-F on an MDP M as follows. Starting from a policy π, the algorithm selects a state-action pair (s, a) uniformly at random that is not used by π. It then recursively solves the MDP without (s, a) (denoted M') to obtain an optimal policy π' for M'. If (s, a) is not an improving pair for π' in M, then π' is necessarily optimal for M. Otherwise, the

algorithm switches state s with action a in π', yielding a new policy π'', and repeats the procedure from π''.

Algorithm 2 shows pseudocode for an implementation of R-F as an AS algorithm. Memory is implemented as a stack of state-action pairs, initialised as empty. Let P denote the set of all state-action pairs in the MDP, and let P^π represent the state-action pairs associated with a policy π, formally defined as $P^\pi = \{(s, \pi(s)) \mid s \in S\}$. Starting from an arbitrary policy, the algorithm iterates. At each step, the stack, of size at most $m - n$, is populated with state-action pairs uniformly sampled from a subset of P. These pairs are then removed from the stack sequentially until an improving pair is found. When such a pair is identified, the algorithm switches to a new policy using the improving pair and restarts with the updated policy and memory stack. The operation $\texttt{switch}(\pi, (s, a))$ modifies policy π by assigning action a to state s.

4.2 Analysis

We revisit the analysis of $\mathcal{L}_1$ [27] to establish the complexity of $\mathcal{L}_1'$.

Theorem 2. *Starting from an arbitrary policy π_0, the expected number of calls to next-$\mathcal{L}_1'$ is upper-bounded by: $\text{poly}(n, m) \cdot e^{O(\sqrt{n \log m})}$.*

Proof. Within a call to *next*, a state-action pair p is popped from the stack, and the process continues until p is improving for the current policy π. Let v_p denote the optimal value of the MDP without the state-action pair p, i.e., $v_p = v_{P \setminus p}$. Then, p is improving for π if and only if $v_p > v_{P^\pi}$. Clearly, if $p \in P^{\pi^\star}$, then $v_p \geq v_{P^\pi}$ for all $\pi \in \Pi$.

Consider an ordering of state-action pairs $P = (p_1, p_2, \ldots, p_m)$ such that

$$v_{p_1} \leq v_{p_2} \leq \cdots \leq v_{p_{n-\theta}} < v_{p_{n-\theta+1}} \leq \cdots \leq v_{p_m}.$$

Define a state-action pair p as *fixed* for π if $v_p < v_{P^\pi}$. Under this ordering, if $v_{p_{n-\theta}} < v_{P^\pi}$, then exactly $n-\theta$ pairs are fixed in π. Consequently, after transitioning from policy π to π', only θ state-action pairs from $\{p_{n-\theta+1}, \ldots, p_m\}$ can lead to further switches.

Since the algorithm selects pairs uniformly at random from $P \setminus P^\pi$, each has a probability of $\frac{1}{m-n}$ of being chosen. Let $T_\theta(m)$ denote the expected number of switches performed by algorithm $\mathcal{L}_1'$ with m state-action pairs, $n - \theta$ of which are fixed.

The recurrence relation for $T_\theta(m)$ is given by:

$$T_\theta(0) = 0, \quad 0 \leq \theta \leq n;$$

$$T_\theta(m) \leq T_\theta(m-1) + 1 + \frac{1}{m-n} \sum_{i=1}^{\min\{m-n, \theta\}} T_{\theta-i}(m).$$

As stated in the theorem, this recurrence satisfies [27]:

$$T_n(m) \leq \text{poly}(n, m) \cdot e^{O(\sqrt{n \log m})}$$

Algorithm 1 Random Facet ($\mathcal{L}_1$) [27]

```
1: procedure Φ_R(H, C)
2:     if H = C then
3:         return C
4:     else
5:         Pick h ∈ H \ C uniformly at
6:                                  random
7:         B ← Φ_R(H \ h, C)        ▷ 1st call
8:         if h is violated by B then
9:             return Φ_R(H, basis(B, h))
                                    ▷ 2nd call
10:        else
11:            return B
```

Algorithm 2 "Next" Function under $\mathcal{L}'_1$

```
1: procedure next_L'_1(π, M)
2:     U ← P \ M ∪ P^π
3:     while U ≠ ∅ do
4:         Pick u = (s, a) ∈ U uniformly
                                  at random
5:         Push (u, M)
6:         U ← U \ {u}
7:     repeat
8:         q ← Pop(M)
9:     until q ∈ J^π
10:    π' ← switch(π, q)
11:    return (π', M)
```

5 $\mathcal{L}_2$: Las Vegas

In this section, we study the scenario where the MDP possesses a large set of actions as compared to the set of states. We devise our AS algorithm, denoted $\mathcal{L}'_2$, drawing inspiration from a "Las Vegas" algorithm (denoted $\mathcal{L}_2$) proposed by Clarkson [7].

Consider an LP with many more constraints than variables. The key idea of $\mathcal{L}_2$ is to exclude redundant constraints from the LP and find the optimal basis quickly. The algorithm does this by building a set of constraints $W^\star$ that will eventually contain the optimal basis. It proceeds recursively in phases, adding a constraint set W to $W^\star$ at each phase. The set W satisfies two properties: its size remains below a threshold, and it includes at least one constraint from the optimal basis. The recursion base case is solved using the Simplex method. The pseudocode for $\mathcal{L}_2$ is given in Algorithm 3.

5.1 Algorithm: $\mathcal{L}'_2$

Our algorithm $\mathcal{L}'_2$ adapts the procedure $\mathcal{L}_2$ for solving MDPs, albeit with a few key modifications. At each stage, it solves a sub-MDP defined by a set of state-action pairs, storing them in memory until the size of the MDP drops below a threshold. It then solves the reduced MDP using $\mathcal{L}'_1$. Unlike $\mathcal{L}_2$, $\mathcal{L}'_2$ follows a simplex procedure. It initialises the set $W^\star$ with P^{π_0}, where π_0 is the initial policy. At every iteration, it adds an improving state-action pair to $W^\star$. This ensures that the algorithm follows a sequence of policies $\pi_0, \pi_1, \ldots, \pi_i$, such that $\pi_{i+1} \succ \pi_i$.

Pseudocode for $\mathcal{L}'_2$ is given in Algorithm 4. Note that the memory operates on sets, so the Push and Pop operations are performed with sets of pairs rather than individual elements. The memory has two components: $\mathcal{M}^0$, which stores the state-action pairs representing the sub-mdp and $\mathcal{M}^1$, which contains the set $W^\star$. Initially, $\mathcal{M}^0$ holds all state-action pairs of the MDP, while $\mathcal{M}^1$ starts with pairs representing the initial policy.

5.2 Analysis

We extend Clarkson's analysis to $\mathcal{L}_2'$ as follows.

Theorem 3. *Starting from an arbitrary policy π_0, the expected number of calls to next-$\mathcal{L}_2'$ is upper-bounded by: $O\left(n^2m + e^{O(\sqrt{n \log n})}\right)$.*

Proof. The algorithm involves populating the memory $\mathcal{M} = \left(\mathcal{M}^0, \mathcal{M}^1\right)$ with sets of state-action pairs. It can be shown that if the set W is non-empty, it must contain a state-action pair belonging to the optimal policy $\pi^\star$ [7]. Consequently, the size of $\mathcal{M}^1$ is augmented at most $n + 1$ times.

A key result is that the expected size of W cannot exceed $\frac{mn}{|R|}$ [7]. Given that $|R| = n\sqrt{m}$, we have $E[|W|] \leq \sqrt{m}$. By Markov's inequality, $P(|W| > 2\sqrt{m}) \leq 0.5$. We define a success at an iteration as the event $|W| \leq 2\sqrt{m}$ and a failure otherwise. During a success, $2\sqrt{m}$ state-action pairs are added to $W^\star$, while only one pair is added during a failure. Hence, the size of $\mathcal{M}^1$ increases by at most $\sqrt{m} + 1$ in expectation. Given that there are at most $n + 1$ increments, the total size of $\mathcal{M}^1$ is bounded by $2n\sqrt{m}$. Since each element of $\mathcal{M}^0$ (beyond initialisation) is a union of R and $\mathcal{M}^1$, the size of $\mathcal{M}^0$ is bounded by $3n\sqrt{m}$.

Let $T(m)$ denote the expected time to solve an MDP with n states and m state-action pairs. As the algorithm requires solving at most $2(n+1)$ MDPs with at most $3n\sqrt{m}$ pairs, we obtain [7]:

$$T(m) \leq 2(n + 1)T(3n\sqrt{m}) + O(n^2 m).$$

For $m \leq 9n^2$, substituting $T(3n\sqrt{m})$ with our $\mathcal{L}_1'$ bound yields the desired result:

$$T(m) \leq O\left(n^2 m + e^{O(\sqrt{n \log n})}\right).$$

6 $\mathcal{L}_3$: Max-Gain

The Max-gain Simplex algorithm (here denoted $\mathcal{L}_3$) operates as follows: starting from an initial policy, it iteratively switches to a new policy by selecting a state-action pair with the highest gain. Specifically, at each iteration, it updates the policy π by replacing the action at state s_m with a_m, where: $(s_m, a_m) \in \arg\max_{(s,a)} G^\pi(s, a)$. This process transitions to the next policy in the sequence. Post and Ye [31] showed that $\mathcal{L}_3$ is strongly polynomial on Deterministic MDPs (DMDPs). Below, we summarise their analysis.

Algorithm 3 Las Vegas ($\mathcal{L}_2$) [7]

procedure $\Psi_r(H$: set of constraints)
  $W^\star \leftarrow \varnothing$; $C_n \leftarrow 9n^2$
  if $|H| \leq C_n$ **then**
    return $\Psi_s^\star(H)$ ▷ Simplex
  repeat
    Choose $R \subset H \setminus W^\star$ uniformly at random, $|R| = n\sqrt{m}$
    $x^\star \leftarrow \Psi_r^\star(R \cup W^\star)$
    $W \leftarrow \{h \in H \mid x^\star \text{ violates } h\}$
    if $|W| \leq 2\sqrt{m}$ **then**
      $W^\star \leftarrow W^\star \cup W$
  until $W = \emptyset$
  return $x^\star$

Algorithm 4 "Next" Function under $\mathcal{L}_2'$

procedure $next\_\mathcal{L}_2'(\pi, \mathcal{M})$
  $U \leftarrow \mathbf{Pop}(\mathcal{M}^0)$; $W^\star \leftarrow \mathbf{Pop}(\mathcal{M}^1)$
  Push$(U, \mathcal{M}^0)$
  while $|U| > 9n^2$ **do**
    Pick $R \subset U \setminus W^\star$ uniformly at random, $|R| = n\sqrt{|U|}$
    $U \leftarrow R \cup W^\star$
    Push $(U, \mathcal{M}^0)$
  Curr $\leftarrow \mathbf{Pop}(\mathcal{M}^0)$
  $\pi' \leftarrow \mathcal{L}_1'(\pi, \text{Curr})$
  repeat
    Prv $\leftarrow \mathbf{Pop}(\mathcal{M}^0)$
    $W \leftarrow J^{\pi'} \cap$ Prv
  until $W \neq \emptyset$
  Push(Prv, $\mathcal{M}^0$)
  if $|W| < 2\sqrt{|\text{Prv}|}$ **then**
    $W^\star \leftarrow W \cup W^\star$
  else
    Pick a $w \in W$
    $W^\star \leftarrow w \cup W^\star$
  Push $(W^\star, \mathcal{M}^1)$
  return $(\pi', \mathcal{M})$

6.1 Analysis of $\mathcal{L}_3$

A DMDP can be represented as a weighted directed multigraph, where the vertices correspond to states and the edges represent actions. A policy corresponds to a subgraph in which each vertex has exactly one outgoing edge. As a result, every policy can be visualised as a collection of disjoint paths that lead to corresponding disjoint cycles. The analysis leverages the concept of flux within these paths and cycles.

Fix some policy $\pi \in \Pi$. If an action of π resides on a path, it can only be utilised once, contributing a unit of flux per state. In contrast, if an action is part of a cycle, each state within the cycle contributes a total flux of $1/(1-\gamma)$ to the cycle. Therefore, the flux bounds can be expressed for $s \in S$ as: $1 \leq x^\pi(s, a_p) \leq n$ and $\frac{1}{1-\gamma} \leq x^\pi(s, a_c) \leq \frac{n}{1-\gamma}$, where a_p and a_c denote actions on a path and in a cycle, respectively. A key aspect of the analysis is that an action update on a path contributes to optimising the paths leading to their respective cycles, whereas an update on a cycle results in a significant improvement in the overall objective. The progress is quantified as follows.

Suppose the algorithm pivots with a path update from policy π to π' and subsequently to π'', where π'' represents a policy in which no further path updates are possible. Post and Ye show that:

$$\sum_s \{U(\pi'', s) - U(\pi', s)\} \leq \left(1 - \frac{1}{n^2}\right) \sum_s \{U(\pi', s) - U(\pi, s)\}. \tag{1}$$

On the other hand if π and π' were policies where π' creates a new cycle we have:

$$\sum_s \left\{U(\pi^\star, s) - U(\pi', s)\right\} \leq \left(1 - \frac{1}{n}\right) \sum_s \left\{U(\pi^\star, s) - U(\pi, s)\right\} \tag{2}$$

where $U(\pi, s) = R(s, \pi(s)) \cdot x^\pi(s, \pi(s))$. We obtain the two lemmas below using (1) and (2), respectively [31].

Lemma 1. *Let π be a policy. After $O(n^2 \log n)$ iterations starting from π, either $\mathcal{L}_3$ finishes, a new cycle is created, a cycle is broken, or some action in π never appears in a policy again until a new cycle is created.*

Lemma 2. *Let π be a policy. Starting from π, after $O(n \log n)$ iterations in which a new cycle is created, some action in π is either eliminated from cycles for the remainder of $\mathcal{L}_3$ or entirely eliminated from policies for the remainder of the algorithm.*

Lemmas 1 and 2 combine to provide the required result:

Theorem 4. *$\mathcal{L}_3$ converges in at most $O(n^5 k^2 \log^2 n)$ iterations on DMDPs.*

6.2 Algorithm: $\mathcal{L}_3'$

Since gain is a real-valued geometric quantity, $\mathcal{L}_3$ alone does not satisfy the AS property. We design a strongly polynomial-time AS algorithm, $\mathcal{L}_3'$, using the analysis of $\mathcal{L}_3$.

Let π and $\overline{\pi}$ be two policies such that $\overline{\pi} \succ \pi$. Starting from π, we define a procedure $\Phi(\pi, \overline{\pi})$ to identify a policy π' satisfying $\pi' \succeq \overline{\pi}$. Define $D = J^\pi \cap P^{\overline{\pi}}$. At each step, an element p is arbitrarily selected from D, and π is updated by switching p, yielding a new policy π'. The procedure then continues with π' replacing π.

Since this follows the simplex procedure, any selected p cannot reappear in a subsequent D, ensuring that $|D|$ decreases by one at each step. Given that $|D| \leq n$, the process terminates in at most n steps.

We define our AS algorithm based on the procedure Φ as follows: Let Θ^π be the set of policies which result after switching a state-action pair from J^π. At each step, we select $\theta \in \Theta^\pi$ and apply Φ to obtain a policy π' that dominates θ. Iterating over all elements of Θ^π yields a policy π'' that dominates every element in Θ^π.

Since the policy θ_m, obtained via a max-gain switch from π, is guaranteed to be in Θ^π, it follows that $\pi'' \succeq \theta_m$. The process then restarts from π'' and repeats until optimality is reached. The pseudocode for the *next* procedure is provided in Algorithm 5. We use memory $\mathcal{M}$ to store Θ^π, initialising it as an empty set.

Algorithm 5 "Next" Function under $\mathcal{L}_3'$

procedure $next\_\mathcal{L}_3'(\pi, \mathcal{M})$
  repeat
    if $\mathcal{M} = \emptyset$ **then**
      for $p \in J^\pi$ **do**
        Push $(\texttt{switch}(\pi, p), \mathcal{M})$
    $\overline{\pi} \leftarrow$ **Pop** $(\mathcal{M})$
  until $J^\pi \cap P^{\overline{\pi}} \neq \emptyset$
  return $\Phi(\pi, \overline{\pi}, \mathcal{M})$

Algorithm Helper Function: Φ

procedure $\Phi(\pi, \overline{\pi}, \mathcal{M})$
  Pick $p \in J^\pi \cap P^{\overline{\pi}}$
  $\pi' \leftarrow \texttt{switch}(\pi, p)$
  if $J^{\pi'} \cap P^{\overline{\pi}} \neq \emptyset$ **then**
    Push$(\overline{\pi}, \mathcal{M})$
  return $(\pi', \mathcal{M})$

6.3 Analysis

Starting from a policy π, we define the operator η, which returns a policy dominating every element of Θ^π. Consequently, the policies in the trajectory of $\mathcal{L}_3'$ can be structured into two layers: i) Those forming the sequence $\mathcal{X} = (\pi, \eta(\pi), \eta^2(\pi), \dots)$, and ii) those appearing between successive elements of $\mathcal{X}$. As established earlier, $\eta^{i+1}(\pi) \geq \theta_m$, where θ_m is the policy obtained by selecting the action with the highest gain from $\eta^i(\pi)$. The following two lemmas help establish a correspondence between the trajectory of policies in $\mathcal{X}$ and those visited by $\mathcal{L}_3$.

Lemma 3. *Let π, π', and π'' be three adjacent elements in the sequence $\mathcal{X}$. Suppose the switches from π to π' and from π' to π'' do not introduce any new cycles, and no further path updates are possible from π''. Then, π, π', and π'' satisfy Eq.* (1).

Proof. Let $\Delta = \max_{(s,a)} G^\pi(s, a)$ be the highest gain and let A_c denote the set of actions belonging to cycles in π. Since π'' does not create a new cycle, $G^\pi(s, a) = 0$ for all $a \in A_c$. Define the operator $\Psi(\pi', \pi) = \sum_s (U(\pi', s) - U(\pi, s))$. Now we have:

$$\Psi(\pi'', \pi') = \Psi(\pi'', \pi) - \Psi(\pi', \pi) \tag{3}$$

Let $\bar{\pi}'$ be the policy which results by switching the action with the highest gain from π. Since $\pi' \geq \bar{\pi}'$, we have: $\Psi(\pi', \pi) \geq \Psi(\bar{\pi}', \pi)$. Since policy $\bar{\pi}'$ must have at least 1 unit of flux on the action with gain Δ, $\Psi(\pi', \pi) \geq \Psi(\bar{\pi}', \pi) \geq \Delta$. Replacing $\Psi(\pi', \pi)$ with Δ in Eq. (3) we obtain: $\Psi(\pi'', \pi') \leq \Psi(\pi'', \pi) - \Delta$. Recall $x^\pi(s, a_P) \leq n$ for all π and a_P therefore the total flux on paths is bounded by n^2 which gives $\Psi(\pi'', \pi) \leq n^2\Delta$. Thus we have: $\Psi(\pi'', \pi') \leq \Psi(\pi'', \pi) - \Delta$ and $\Delta \geq \Psi(\pi'', \pi)/n^2$. Therefore: $\Psi(\pi'', \pi') \leq (1 - 1/n^2)\Psi(\pi'', \pi)$.

Lemma 4. *Suppose π and π' be adjacent policies of a sequence $\mathcal{X}$ such that π' creates a new cycle. Then π and π' satisfy Eq.* (2).

Proof. We have:

$$\Psi(\pi^\star, \pi') = \Psi(\pi^\star, \pi) - \Psi(\pi', \pi) \tag{4}$$

Again let $\bar{\pi}'$ be the policy which results by switching the action with the highest gain from π and let a be an action which forms a cycle. Since $x^\pi(s, a_C) \geq 1/(1 - \gamma)$,

switching a will result in at least $1/(1-\gamma)$ units of flux through a. Since $\pi' \geq \bar{\pi}'$, we have: $\Psi(\pi', \pi) \geq \Delta/(1-\gamma)$. Replacing in Eq. (4), we obtain: $\Psi(\pi^\star, \pi') \leq \Psi(\pi^\star, \pi) - \Delta/(1-\gamma)$. Since the total flux on an MDP is bounded by $\frac{n}{1-\gamma}$, we have: $\Psi(\pi^\star, \pi) \leq n\Delta/(1-\gamma)$. Therefore we get: $\Psi(\pi^\star, \pi') \leq (1-1/n)\Psi(\pi^\star, \pi)$.

Lemmas 1 and 2 hold for the policies in the sequence $\mathcal{X}$ as a direct consequence of Lemmas 3 and 4, leading analogously to the following result:

Theorem 5. *For a given policy π, the length of the sequence $\mathcal{X}$ and the number of iterations in $\mathcal{L}_3$ have the same asymptotic order.*

Consider a policy $\pi \in \mathcal{X}$. It is clear that: $|\Theta^\pi| \leq n(k-1)$. Now, let θ_1 and θ_2 be two consecutive policies in the trajectory of $\mathcal{L}_3'$ such that both $\theta_1, \theta_2 \in \Theta^\pi$. Between θ_1 and θ_2, there can be at most n policies in $\mathcal{L}_3'$. Consequently, the total number of policies in a trajectory of $\mathcal{L}_3'$ is bounded by: $n^2(k-1) \cdot O(n^5 k^2 \log^2 n) = O(n^7 k^3 \log^2 n)$.

7 Summary

We have introduced a coordinate-free computation model to explore the role of geometry in MDP planning. By analysing three algorithms renowned for their efficiency in solving LPs and MDPs and adapting them to our algebraic and jump-free model we have uncovered intriguing insights. These findings are striking given the substantial advantages geometry often confers on algorithms. To the best of our knowledge, this is a novel investigation in the MDP literature. Our results prompt an important question: do geometry and jumping provide only a polynomial advantage in solving MDPs, or can they lead to significantly tighter upper bounds than currently established?

Our coordinate-free AS framework has promising applications. It provides a stable and efficient method for computing Blackwell optimal policies [4], mitigating numerical instabilities arising from dependence on geometric properties. Furthermore, our study suggests that the polytope structure of MDPs remains underutilised in algorithm design. By harnessing this structure—much like LP algorithms exploit polytopes—it may be possible to develop fundamentally more efficient MDP solvers. Additionally, extending AS to generalisations like Stochastic Games [34] could yield new insights into computing solutions such as Nash Equilibria.

Acknowledgements. The authors thank Sundar Vishwanathan and Supratik Chakraborty for providing useful comments.

References

1. Andersson, D., Vorobyov, S.: Fast Algorithms for Monotonic Discounted Linear Programs with Two Variables per Inequality. Preprint NI06019-LAA, Isaac Newton Institute for Mathematical Sciences, Cambridge, UK (2006)
2. Ashutosh, K., Consul, S., Dedhia, B., Khirwadkar, P., Shah, S., Kalyanakrishnan, S.: Lower bounds for policy iteration on multi-action MDPs. In: 59th IEEE Conference on Decision and Control, CDC 2020, pp. 1744–1749. IEEE (2020)

3. Bellman, R.: Dynamic Programming. Princeton University Press, Princeton (1957)
4. Blackwell, D.: Discrete dynamic programming. Ann. Math. Stat. 719–726 (1962)
5. Brunsch, T., Röglin, H.: Finding short paths on polytopes by the shadow vertex algorithm. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) ICALP 2013. LNCS, vol. 7965, pp. 279–290. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39206-1_24
6. Chan, T.M.: Improved deterministic algorithms for linear programming in low dimensions. ACM Trans. Algorithms **14**(3), 30:1–30:10 (2018)
7. Clarkson, K.L.: Las Vegas algorithms for linear and integer programming when the dimension is small. J. ACM **42**(2), 488–499 (1995)
8. Dadashi, R., Bellemare, M.G., Taïga, A.A., Roux, N.L., Schuurmans, D.: The value function polytope in reinforcement learning. In: Proceedings of the 36th International Conference on Machine Learning, ICML 2019, USA, pp. 1486–1495. PMLR (2019)
9. Dantzig, G.B.: Linear Programming and Extensions. Princeton University Press (1963)
10. Disser, Y., Friedmann, O., Hopp, A.V.: An exponential lower bound for Zadeh's pivot rule. Math. Program. **199**(1), 865–936 (2023)
11. Eisenbrand, F., Vempala, S.S.: Geometric random edge. Math. Program. **164**(1–2), 325–339 (2017)
12. Gärtner, B.: The random-facet simplex algorithm on combinatorial cubes. Random Struct. Algorithms **20**(3), 353–381 (2002)
13. Gärtner, B., Kaibel, V.: Two new bounds on the random-edge simplex algorithm. SIAM J. Discret. Math. **21**(1), 178–190 (2007)
14. Goldfarb, D., Sit, W.Y.: Worst case behavior of the steepest edge simplex method. Discret. Appl. Math. **1**(4), 277–285 (1979)
15. Gupta, A., Kalyanakrishnan, S.: Improved strong worst-case upper bounds for MDP planning. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, pp. 1788–1794. ijcai.org (2017)
16. Howard, R.A.: Dynamic Programming and Markov Processes. MIT Press (1960)
17. Jeroslow, R.: The simplex algorithm with the pivot rule of maximizing criterion improvement. Discret. Math. **4**(4), 367–377 (1973)
18. Kalai, G.: A subexponential randomized simplex algorithm (extended abstract). In: Proceedings of the 24th Annual ACM Symposium on Theory of Computing, pp. 475–482. ACM (1992)
19. Kalyanakrishnan, S., Mall, U., Goyal, R.: Batch-switching policy iteration. In: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, pp. 3147–3153. IJCAI/AAAI Press (2016)
20. Kalyanakrishnan, S., Misra, N., Gopalan, A.: Randomised procedures for initialising and switching actions in policy iteration. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, pp. 3145–3151. AAAI Press (2016)
21. Karmarkar, N.: A new polynomial-time algorithm for linear programming. In: Proceedings of the 16th Annual ACM Symposium on Theory of Computing, pp. 302–311. ACM (1984)
22. Khachiyan, L.G.: Polynomial algorithms in linear programming. USSR Comput. Math. Math. Phys. **20**(1), 53–72 (1980)
23. Klee, V., Minty, G.J.: How good is the simplex algorithm. Inequalities **3**(3), 159–175 (1972)
24. Littman, M.L., Dean, T.L., Kaelbling, L.P.: On the complexity of solving Markov decision problems. In: UAI 1995: Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence, pp. 394–402. Morgan Kaufmann (1995)
25. Madani, O., Thorup, M., Zwick, U.: Discounted deterministic Markov decision processes and discounted all-pairs shortest paths. ACM Trans. Algorithms **6**(2), 33:1–33:25 (2010)

26. Mansour, Y., Singh, S.: On the complexity of policy iteration. In: UAI 1999: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, pp. 401–408. Morgan Kaufmann (1999)
27. Matousek, J., Sharir, M., Welzl, E.: A subexponential bound for linear programming. Algorithmica **16**(4/5), 498–516 (1996)
28. Mausam, Kolobov, A.: Planning with Markov Decision Processes: An AI Perspective. Morgan & Claypool (2012)
29. Megiddo, N.: Linear programming in linear time when the dimension is fixed. J. ACM **31**(1), 114–127 (1984)
30. Papadimitriou, C.H., Tsitsiklis, J.N.: The complexity of Markov decision processes. Math. Oper. Res. **12**(3), 441–450 (1987)
31. Post, I., Ye, Y.: The simplex method is strongly polynomial for deterministic Markov decision processes. In: Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, pp. 1465–1473. SIAM (2013)
32. Puterman, M.L.: Markov Decision Processes. Wiley (1994)
33. Schurr, I., Szabó, T.: Jumping doesn't help in abstract cubes. In: Jünger, M., Kaibel, V. (eds.) IPCO 2005. LNCS, vol. 3509, pp. 225–235. Springer, Heidelberg (2005). https://doi.org/10.1007/11496915_17
34. Shapley, L.S.: Stochastic games. Proc. Natl. Acad. Sci. **39**(10), 1095–1100 (1953)
35. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press (1998)
36. Szabó, T., Welzl, E.: Unique sink orientations of cubes. In: 42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, pp. 547–555. IEEE Computer Society (2001)
37. Wu, Y., Loera, J.A.D.: Geometric policy iteration for Markov decision processes. In: KDD 2022: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, pp. 2070–2078. ACM (2022)
38. Ye, Y.: The simplex and policy-iteration methods are strongly polynomial for the Markov decision problem with a fixed discount rate. Math. Oper. Res. **36**(4), 593–603 (2011)

Reinforcement Learning-Based Heuristics to Guide Domain-Independent Dynamic Programming

Minori Narita[1(✉)], Ryo Kuroiwa[2], and J. Christopher Beck[1]

[1] University of Toronto, 5 King's College Road, Toronto, ON, Canada
minori.narita@mail.utoronto.ca, jcb@mie.utoronto.ca
[2] National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, Japan
kuroiwa@nii.ac.jp

Abstract. Domain-Independent Dynamic Programming (DIDP) is a state-space search paradigm based on dynamic programming for combinatorial optimization. In its current implementation, DIDP guides the search using user-defined dual bounds. Reinforcement learning (RL) is increasingly being applied to combinatorial optimization problems and shares several key structures with DP, being represented by the Bellman equation and state-based transition systems. We propose using reinforcement learning to obtain a heuristic function to guide the search in DIDP. We develop two RL-based guidance approaches: value-based guidance using Deep Q-Networks and policy-based guidance using Proximal Policy Optimization. Our experiments indicate that RL-based guidance significantly outperforms standard DIDP and problem-specific greedy heuristics with the same number of node expansions. Further, despite longer node evaluation times, RL guidance achieves better run-time performance than standard DIDP on three of four benchmark domains.

Keywords: Dynamic Programming · Reinforcement Learning · Deep Learning · Machine Learning · Optimization

1 Introduction

Domain-Independent Dynamic Programming (DIDP) is a state-space search paradigm based on dynamic programming (DP) and heuristic state-space search [17,18]. Previous work has shown DIDP to be competitive with Constraint Programming (CP) and Mixed-Integer Programming (MIP) on a number of benchmark problem classes in combinatorial optimization. Current DIDP solvers guide search with an f-value computed at each state, where $f(s) = g(s) + h(s)$; $g(s)$ is the path cost to the current state s and $h(s)$ is a heuristic that estimates the cost from s to a base state. In its current implementation, DIDP uses user-defined dual bounds as the h-value. However, such dual bounds may not always be very informative and a stronger heuristic guidance could improve solver performance.

G. Tack (Ed.): CPAIOR 2025, LNCS 15763, pp. 137–154, 2025.
https://doi.org/10.1007/978-3-031-95976-9_9

Reinforcement learning (RL) has achieved remarkable performance in fields such as control tasks and games [23,24,31,33], and is increasingly being applied to combinatorial optimization [6,8,9,25,38]. The goal of the RL agent is to learn an optimal policy for the given task through trial-and-error interactions with an environment described as a Markov Decision Process (MDP) [2,14]. RL shares several key structures with DP, being represented by the Bellman equation and state-based transition systems. Cappart et al. [8] formulated optimization problems as dynamic programs to bridge an RL model and a CP model, enabling RL-based guidance for variable selection in CP solvers. However, their framework restricts CP formulations to be compatible with the DP model, which can limit performance. In contrast, RL-guided DIDP leverages the alignment between RL and DP models, which may offer a natural integration of RL into exact solvers.

In this paper, we investigate two ways to guide the search in DIDP using RL, value-based and policy-based guidance, and evaluate them on four combinatorial optimization problems. The key contributions of this paper are as follows:

- We introduce two approaches – value-based guidance and policy-based guidance – that effectively direct the search in DIDP;
- We demonstrate that an RL model can be systematically mapped from a DIDP model, establishing a basis for automated mapping in future work;
- Our experimental evaluation shows that DIDP with RL guidance outperforms DIDP based on node expansions and, to a lesser extent, on run-time.

2 Background

In this section, we describe the two foundations of our work: Domain-Independent Dynamic Programming (DIDP) and reinforcement learning (RL).

2.1 DIDP

In DIDP, the user defines a DP model in the Dynamic Programming Description Language (DyPDL) and the model is solved by a solver. While different solving approaches are possible, thus far existing solvers are based on heuristic search.

DyPDL. DyPDL is a solver-independent formalism to define a DP model [19] represented as a tuple $\langle \mathcal{V}, s_0, \mathcal{T}, \mathcal{B}, \mathcal{C} \rangle$, where $\mathcal{V} = \{v_1, ..., v_n\}$ is the set of state variables, s_0 is the target state, $\mathcal{T}$ is the set of transitions, $\mathcal{B}$ is the set of *base cases*, and $\mathcal{C}$ is the set of state constraints. Each state variable $v_i \in \mathcal{V}$ is either an element, a set, or a number, and has a domain $\mathcal{D}_i$. A state s is a complete assignment to the state variables, represented by a tuple $\langle d_1, ..., d_n \rangle \in \mathcal{D}$ where $\mathcal{D}$ is the cartesian product of $\mathcal{D}_1$ ... $\mathcal{D}_n$. We denote $s[v_i]$ as the value of the v_i in state s. A target state s_0 is the initial state in the transition system, i.e., the state for which the optimal value is to be computed. State constraints $\mathcal{C}$ are conditions on state variables that must be satisfied by all valid states. A base

case $\langle \mathcal{C}_B, \mathsf{cost}_B \rangle \in \mathcal{B}$ is a set of conditions $\mathcal{C}_B$ to terminate the transitions and the associated cost function cost_B. A state that satisfies $\mathcal{C} \cup \mathcal{C}_B$ is called a base state. A transition $\tau \in \mathcal{T}$ is a 4-tuple $\langle\ \mathsf{eff}_\tau,\ \mathsf{cost}_\tau,\ \mathsf{pre}_\tau,\ \mathsf{forced}_\tau \rangle$. The effect $\mathsf{eff}_\tau : \mathcal{D}_i \rightarrow \mathcal{D}_i$ is a function that maps a value of a state variable v to another value. A state transition returns a successor state $s[\![\tau]\!]$ by applying τ to each state variable in s, i.e., $s[\![\tau]\!][v_i] = \mathsf{eff}_\tau[v_i](s), \forall v_i \in \mathcal{V}$. A numeric cost $\mathsf{cost}_\tau(s)$ is associated with each transition τ from a state s. Preconditions pre_τ are conditions on state variables, and τ is *applicable* in a state s only if all preconditions are satisfied, denoted by $s \models \mathsf{pre}_\tau$. The flag $\mathsf{forced}_\tau \in \{\bot, \top\}$ is a boolean value; if forced transitions are applicable at state s, then the first defined one is executed and all other forced and non-forced transitions are ignored.

Let $x = \{x_1, ..., x_m\}$ be a sequence of transitions for a DyPDL model. Then, x is a *solution* to the model if the sequence starts from s_0 and ends at a base state. For minimization problems, the cost of a solution is $\sum_{i=0}^{m-1} \mathsf{cost}_{x_{i+1}}(s_i) + \min_{\{B | B \in \mathcal{B}; s_m \models \mathcal{C}_B\}} \mathsf{cost}_B(s_m)$, where s_i is the state resulting from applying the first i transitions of the solution from s_0. For maximization, min is replaced with max. We can represent a DyPDL model by a recursive equation called a Bellman equation [4]. The Bellman equation $V(s)$ returns the optimal cost starting from state s where $V(s) = \infty$ (or $V(s) = -\infty$ for maximization) if there does not exist a base state reachable from s. For the minimization (maximization) problem, $\eta(s)$ is a dual bound function iff $\eta(s) \leq V(s)$ $(\eta(s) \geq V(s))$, $\forall s \in \mathcal{D} \wedge s \models \mathcal{C}$.

State-Based Heuristic Search. Kuroiwa and Beck [19] implement seven heuristic search algorithms for DIDP based on the literature. Starting from s_0, each algorithm *expands* states, generating a successor state $s' = s[\![\tau]\!]$ for each transition applicable in s. At each successor state, the f-value is computed as $f(s') = g(s') + h(s')$, where $g(s')$ is the path cost from s_0 to s', and $h(s')$ is the heuristic estimate of the cost from s' to a base state. Given a sequence of transitions $x = \{x_1, ..., x_j\}$ to reach a state s' $(= s_j)$ from s_0, the path cost for s' is defined as $g(s') = \sum_{i=0}^{j-1} \mathsf{cost}_{x_{i+1}}(s_i)$. User-defined dual bounds $\eta(s)$ and state constraints $\mathcal{C}$ are used for pruning. Let $\bar{\zeta}$ be the primal bound. Then, we can prune the node s if $g(s) + \eta(s) \geq \bar{\zeta}$ (for maximization, $g(s) + \eta(s) \leq \bar{\zeta}$). By default, $h(s') = \eta(s')$. For formal details, see Kuroiwa and Beck [19].

In Sect. 4, we used three search algorithms: complete anytime beam search (CABS), anytime column progressive search (ACPS), and anytime pack progressive search (APPS). Algorithm details are in Appendix A in the arXiv version.[1]

2.2 Reinforcement Learning

Reinforcement learning (RL) [35] is a framework for learning to achieve a goal from interaction with the environment. In RL, the agent operates based on a Markov Decision Process (MDP) [28], defined as a 4-tuple $\langle \mathrm{S}, \mathrm{A}, \mathrm{T}, \mathrm{R} \rangle$, where S is a set of states, A is a set of actions, $\mathrm{T} : \mathrm{S} \times \mathrm{A} \rightarrow \mathrm{S}$ is the transition function,

[1] https://arxiv.org/abs/2503.16371.

and $\mathrm{R} : \mathrm{S} \times \mathrm{A} \to \mathbb{R}$ is the reward function. For simplicity, we use the notation s to represent a state in the MDP as well as a state in the DP. The initial state s_0 is sampled from an initial state distribution ρ_0: $\mathbb{R} \to \mathrm{S}$. At each time-step, the agent performs an action $a \in \mathrm{A}$ at current state s, which brings the agent to the next state s' and gives a reward $r = \mathrm{R}(s, a)$. We assume a deterministic transition function, so $\mathrm{T}(s, a)$ returns the next state s'. An episode terminates when the agent reaches a terminal state. The goal of the RL agent is to learn an optimal policy π, that maximizes the expected total reward $\sum_{t=0}^{\infty} \gamma^t r_t$, given the initial state distribution ρ_0, where γ is a discount factor ($0 \leq \gamma \leq 1$) and t is a time-step. A policy $\pi : \mathrm{S} \times \mathrm{A} \to [0, 1]$ is a conditional distribution over actions given the state, indicating the likelihood of the agent choosing an action. Policy-based methods like TRPO [30] and PPO [31] directly optimize the policy through policy gradient methods. Value-based methods such as DQN [24] learn a value function from exploration. For instance, DQN learns an estimated Q-value function $Q^\pi(s, a)$, the expected return for selecting action a at state s if the agent follows policy π afterwards.

RL is increasingly being applied to solve combinatorial optimization problems. Early approaches used recurrent neural networks to learn constructive heuristics for generating solutions [5]. However, graph neural networks (GNNs) have become more prevalent [9,16,25] as they are size-agnostic and permutation-invariant. A limitation of end-to-end RL methods is the challenge of managing constraints, as well as the lack of a systematic way to improve the obtained solutions, unlike exact methods such as CP and MIP [7,8].

Significant progress has recently been made in combining search with learned heuristics to address these limitations [8,11,15,34]. Kool et al. [15] formulated routing problems as a DP problem and guided the search using a learned heat map of edge weights. Cappart et al. [8] formulated optimization problems as DP models to bridge an RL model and a CP model, enabling RL-based guidance for variable selection in CP solvers.

Guiding search using deep reinforcement learning has also been explored in AI planning. Orseau et al. [26,27] proposed learning Q-value functions and policies to weight nodes in best-first search to solve two-dimensional single-agent problems like Sokoban. DeepCubeA [1] tackled the Rubik's Cube by learning a heuristic function through approximated value iteration and applying it in batch-weighted A* search. Lastly, Gehring et al. [12] leveraged domain-independent heuristic functions as dense reward generators to train RL agents, and used the RL-based heuristics to improve search efficiency for classical planning problems.

3 RL-Based Search Guidance for DIDP

Given the similarities in the state-based formulations of DIDP and RL, it is natural to investigate the guidance of search in DIDP based on a heuristic function learned with RL. Each component of an MDP can be systematically derived from the corresponding component in a DIDP model. For instance, the set of states in the DIDP model matches precisely with the state space in the MDP

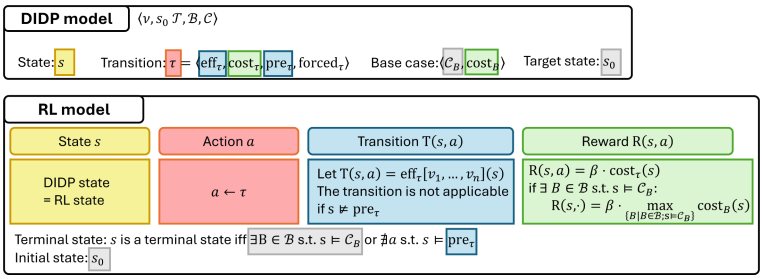


Fig. 1. Mapping from a DIDP model to an RL model for maximization problems. State constraints $\mathcal{C}$ and forced_τ are not mapped to the RL model.

and transitions in the DIDP model have a 1-to-1 mapping to actions in the MDP. An RL agent trained on the mapped MDP can then serve as a heuristic function for computing f-values in the DIDP search.

In this paper, we develop a scheme to convert a DIDP model to an RL model for a given combinatorial optimization problem. Figure 1 provides an overview of how an RL model is mapped from a DIDP model. Each component in the MDP is derived from the DIDP model as follows:

- `State` $s \in \mathrm{S}$: The same state space as in the DIDP model.
- `Action` $a \in \mathrm{A}$: 1-to-1 mapping from transition $\tau \in \mathcal{T}$, i.e., $\tau \mapsto a$.
- `Transition function` $\mathrm{T}(s,a)$: mapped from pre_τ and eff_τ in each $\tau \in \mathcal{T}$. For a state s, the next state reached by taking an action a is obtained by applying eff_τ to each state variable in s, i.e., $\mathrm{T}(s,a) = \mathsf{eff}_\tau[v_1, ..., v_n](s)$. The transition is not applicable if the preconditions are not met, i.e., $s \not\models \mathsf{pre}_\tau$. Thus, the mapping includes the masking of non-applicable actions as is common in RL models with invalid actions [8,13,33].
- `Reward function` $\mathrm{R}(s,a)$ corresponds to the transition cost incurred by applying τ at state s, $\mathsf{cost}_\tau(s)$. To improve the stability of RL training, the reward is scaled by a hyperparameter β. For a minimization problem, the reward has to be negated to make the RL task a maximization problem, i.e., $\mathrm{R}(s,a) = -\beta \cdot \mathsf{cost}_\tau(s)$. If s is the base state, cost_B is applied instead.

A state s is a terminal state if s satisfies at least one base case, i.e., $\exists \mathrm{B} \in \mathcal{B}$ s.t. $s \models \mathcal{C}_\mathrm{B}$, or there is no action that satisfies preconditions, i.e., $\nexists\, a$ s.t. $s \models \mathsf{pre}_\tau$ where a is mapped from τ. The initial state in RL is the target state in the DIDP model s_0. The state constraints $\mathcal{C}$ are not mapped to the RL model.[2]

Including domain knowledge and introducing auxiliary rewards are often crucial for successful RL training [21,32]. In this paper, we focus on leveraging the structural relationships between DIDP and RL models to systematically build

[2] While state constraints $\mathcal{C}$ can be mapped to action masking, the implications of action masking in RL remain underexplored [13]. Thus, the investigation of integrating $\mathcal{C}$ into the MDP is left for future work.

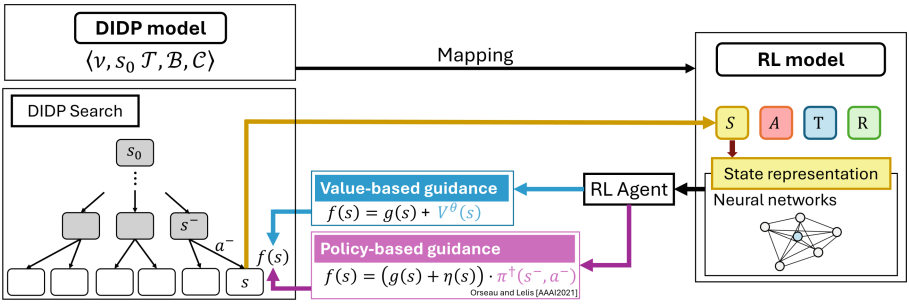


Fig. 2. Value-based guidance and policy-based guidance for DIDP. The equations for computing f-values in the figure are for maximization problems.

RL models for search guidance. Future work will examine automating this mapping and incorporating problem-specific structures into the mapping process.

3.1 Value-Based Guidance

The value-based guidance approach directly uses the value function approximated by neural network parameters θ, $V^\theta(s)$, as a heuristic function, where $h(s) = V^\theta(s)$. $V^\theta(s)$ is an estimated total reward from s to a terminal state, which aligns with the definition of $h(s)$, the estimated total cost from s to a base state. For minimization problems, the value function is negated, i.e., $h(s) = -V^\theta(s)$. Figure 2 shows the overview of this approach. First, the MDP is derived from the DIDP model, as described above. Then, the RL agent is trained in the mapped MDP with a value-based RL algorithm (e.g., DQN). During search, every time a node is expanded, the f-value is calculated for all its successor states s by $f(s) = g(s) + V^\theta(s)$; the f-values are then used to determine the priority of the state in the DIDP framework.

Details of the computation for each successor node s are as follows. First, path cost $g(s)$ is calculated by $g(s) = g(s^-) + \mathsf{cost}_\tau(s^-)$, where s^- is the parent node of s and τ is a transition that transforms s^- to s. To align the scale of $g(s)$ with that of $V^\theta(s)$, the scaling factor β is used; thus, $g(s) = g(s^-) + \beta \cdot \mathsf{cost}_\tau(s^-)$. The heuristic value is computed by calling a neural network prediction, i.e., $V^\theta(s) = \mathcal{F}(s;\theta)$. We use DQN as a value-based RL algorithm, so the output is the Q-values for each action a for the input state s. $V^\theta(s)$ is obtained by $V^\theta(s) = \max_{a \in A'} Q^\theta(s,a)$, where A' is the set of all applicable actions in s.

3.2 Policy-Based Guidance

The policy-based guidance approach uses the same MDP as value-based guidance, but is trained with a policy-based RL algorithm, such as PPO [31]. The algorithm learns a policy $\pi(s,a)$, a probability distribution over actions for a given state. $\pi(s,a)$ is then used to weight the original f-value to prioritize the expansion of the nodes that are deemed promising by the policy.

Details of the computation at each successor node generation are as follows. First, path cost $g(s)$ is calculated by $g(s) = g(s^-) + \mathsf{cost}_\tau(s^-)$. Unlike in value-based guidance, there is no need to scale the path cost $g(s)$, as the policy has a fixed scale of $[0, 1]$ and is only used to weight the original f-values. Then, the policy $\pi(s^-, a^-)$ is obtained by calling the neural network and obtain the a^-th output, i.e., $\pi(s^-, a^-) = \mathcal{F}(s^-; \theta)[a^-]$.[3] Our approach uses the accumulated probabilities up to state s from the root node, i.e., $\pi^\dagger(s^-, a^-) = \pi(s_0, a_0)\pi(s_1, a_1)...\pi(s^-, a^-)$, to take all the previous decisions up to s into consideration [27]. Therefore, the f-value is computed as $f(s) = (g(s) + \eta(s)) \cdot \pi^\dagger(s^-, a^-)$, where $\eta(s)$ is the dual bound at s. A promising action with a high probability in the policy will have a higher f-value (for maximization), thereby making the corresponding state more likely to be expanded next. For minimization problems, we divide $g(s)+\eta(s)$ by $\pi^\dagger(s^-, a^-)$ instead, i.e., $f(s) = (g(s) + \eta(s))/\pi^\dagger(s^-, a^-)$, so that promising actions yield lower f-values.

3.3 State Representation

The state representation needs to be able to handle instances of different sizes and to be invariant to input permutations [8]. Hence, we used a graph attention network (GAT) [36] as a state representation for routing problems to leverage the natural graph structure of these domains, and a set transformer [20] or Deep Sets [37] for packing problems. The details of the neural network architecture for each problem used in this paper appear in Appendix B in the arXiv version. The embedding obtained by the neural network can then be used as an input to a fully-connected network. For DQN, the network outputs Q-values for each action a for state s, so the output layer is of size $|A|$. For PPO, two separate networks for the actor and the critic are used. The critic network outputs a single value, representing the estimated value of the state, while the actor network applies a softmax activation function to its final layer to output action probabilities $\pi(s, \cdot)$. The outputs are processed as described in Sects. 3.1 and 3.2.

4 Experiments

We evaluated our methods on four combinatorial optimization problems: Traveling Salesperson Problem (TSP), TSP with Time Windows (TSPTW), 0-1 Knapsack, and Portfolio Optimization.[4] To assess the quality of RL guidance, the solution quality per node expansion was evaluated for different guidance methods. Our two RL-based guidance methods ($h =$ DQN and $\pi =$ PPO) were compared with dual-bound guidance (default DIDP implementation), uniform cost search (i.e., $h = 0$ for all states), and greedy heuristic guidance.

The greedy heuristic-based h-value at a state is equal to the cost of the path to a base case found by rolling-out the greedy heuristic from that state. Greedy

[3] In our implementation, the neural network is called only once when s^- is expanded. Each element of $\pi(s^-)$ is assigned to the corresponding successor through indexing.

[4] All code are available at https://github.com/minori5214/rl-guided-didp.

heuristics exploit domain-specific knowledge from outside the DP model, and thus serve as a baseline to evaluate how RL guidance competes with hand-crafted heuristics. The definitions of the greedy heuristics for each problem domain and further details are in Appendix C in the arXiv version.

We also evaluated the solution quality after a one hour run-time to compare the performance of our approach with baseline methods. The results include performance from MIP (Gurobi), pure CP (CPLEX CP Optimizer), RL-guided CP (BaB-DQN and RBS-PPO [8]), and sampling-based heuristic methods (dual-bound, greedy, DQN, PPO). Sampling is done by applying a softmax function to the h-values of all successor states to get action probabilities and choosing the next state probabilistically. The number of samples is set to 1280, following Kool et al. [16]. The memory limit for each approach is set to 8 GB.

Evaluation metric: At a given node expansion limit l, the gap is calculated by $gap = |x(i, m, l) - best(i)|/best(i) \times 100$, where $x(i, m, l)$ is the solution cost of method m for instance i up to the node expansion limit l, and $best(i)$ is the best known solution for i. The best known solution includes results from all the approaches, including baselines, within the time limit. If no feasible solution is found within the given node expansion limit, a fixed value of 100 [%] is used.

Training Process: Although the network architectures are size-agnostic, we trained DQN and PPO for each problem size and domain, as examining the scalability of neural networks is not our main focus. Training begins with randomly generating an instance from a fixed distribution using an instance generator. The agent then explores the instance following the current policy π until the agent reaches the terminal state with the experiences stored in the replay buffer. The network parameters θ are updated using the experiences sampled from the replay buffer. The training time is limited to 72 h. Details of the network architectures and hyperparameters are in Appendix B in the arXiv version.

4.1 Problem Domains

To evaluate our approach, we chose routing problems (TSP and TSPTW) and packing problems (0-1 Knapsack and Portfolio Optimization).

TSP. In the Traveling Salesperson Problem (TSP) [10], a set of customers $N = \{0, ..., n\}$ is given, and a solution is a tour starting from the depot ($i = 0$) and returning to the depot, visiting each customer exactly once. Visiting customer j from i incurs the travel time $c_{ij} \geq 0$. TSP instances are generated by removing time window constraints from the TSPTW instances used below.

DIDP Model: For TSP, a state is a tuple of variables $\langle U, i \rangle$ where U is the set of unvisited customers and i is the current location. In this model, one customer is visited at each transition. The minimum possible travel time to visit customer j is $c_j^{\text{in}} = \min_{k \in N \setminus \{j\}} c_{kj}$, and the minimum travel time from j is

$c_j^{\text{out}} = \min_{k \in N \setminus \{j\}} c_{jk}$. The DIDP model is represented by the following Bellman equation, adapted from the TSPTW model defined below.

$$\text{compute } V(N \setminus \{0\}, 0) \tag{1}$$

$$V(U,i) = \begin{cases} c_{i0} & \text{if } U = \emptyset \\ \min_{j \in U} c_{ij} + V(U \setminus \{j\}, j) & \text{if } U \neq \emptyset \end{cases} \tag{2}$$

$$V(U,i) \geq \max \left\{ \sum_{j \in U \cup \{0\}} c_j^{\text{in}}, \sum_{j \in U \cup \{i\}} c_j^{\text{out}} \right\} \tag{3}$$

Expression (1) declares that the optimal cost is the cost to visit all customers ($U = N \setminus \{0\}$) starting from the depot ($i = 0$). The second line of Eq. (2) corresponds to visiting customer j from i; then, j is removed from U and the current location i is updated to j. The first line of Eq. (2) is the base case, where all customers are visited ($U = \emptyset$) and the recursion ends. Equation (3) represents two dual bounds.

RL Model: The MDP for this DIDP model is defined as follows:

`State` s: $\langle U, i \rangle$
`Action` $a = j \in U$
`Transition function` $\mathrm{T}(s,a)$: $\mathrm{T}(\langle U, i \rangle, j) = \langle U \setminus j, j \rangle$
`Reward function` $\mathrm{R}(s,a)$: $\mathrm{R}(\langle U, i \rangle, j) = \beta \cdot (-c_{ij})$

The reward function is the negative distance between the current location i and the next customer j. The scaling factor is $\beta = 0.001$. Dual bounds are not used in the MDP.

TSPTW. In TSP with Time Windows (TSPTW) [29], the visit to customer i must be within a time window $[a_i, b_i]$. If customer i is visited before a_i, the salesperson has to wait until a_i. The instances were generated in the same way as Cappart et al. [8], but the maximal time window length allowed is set $W = 100$, and the maximal gap between two consecutive time windows is set $G = 1000$ to make the instances more challenging.

DIDP Model: A state is a tuple $\langle U, i, t \rangle$ where t is the current time. The set of customers that can be visited next is $U' = \{j \in U \mid t + c_{ij} \leq b_j\}$. The DIDP model is represented by the following Bellman equation [19]:

$$\text{compute } V(N \setminus \{0\}, 0, 0) \tag{4}$$

$$V(U,i,t) = \begin{cases} c_{i0} & \text{if } U = \emptyset \\ \min_{j \in U'} c_{ij} + V(U \setminus \{j\}, j, \max(t + c_{ij}, a_j)) & \text{if } U \neq \emptyset \end{cases} \tag{5}$$

$$V(U,i,t) = \infty \qquad \text{if } \exists j \in U, t + c_{ij}^* > b_j \tag{6}$$

$$V(U,i,t) \leq V(U,i,t') \qquad \text{if } t \leq t' \tag{7}$$

$$V(U,i,t) \geq \max \left\{ \sum_{j \in U \cup \{0\}} c_j^{\text{in}}, \sum_{j \in U \cup \{i\}} c_j^{\text{out}} \right\} \tag{8}$$

In the second line of Eq. (5), time t is updated to $\max(t+c_{ij}, a_j)$. Equation (6) is a state constraint that sets the value of a state to be infinity if there exists a customer j that cannot be visited by the deadline b_j even if we take the shortest path with distance c^*_{ij}. Inequality (7) is a dominance relationship; if other state variables are the same in two states, then a state having smaller t always leads to a better solution. Equation (8) represents two dual bounds.

RL Model: The MDP for this DIDP model is defined as follows:

`State` s: $\langle U, i, t \rangle$
`Action` $a = j \in U$
`Transition function` $\mathrm{T}(s, a)$: $\mathrm{T}(\langle U, i, t \rangle, j) = \langle U \setminus j, j, \max(t + c_{ij}, a_j) \rangle$
`Reward function` $\mathrm{R}(s, a)$: $\mathrm{R}(\langle U, i, t \rangle, j) = \beta \cdot (|\mathrm{UB}_{cost}| - c_{ij})$

UB_{cost} is a strict upper bound on the reward of any solution for this problem domain to ensure that the RL agent has the incentive to find feasible solutions first and then to find the best ones. UB_{cost} is not included in the mapping in Fig. 1, but this reward structure is originally introduced in Cappart et al. [8] and helps improve the RL training. The scaling factor is set to $\beta = 0.001$.

0-1 Knapsack. In the 0-1 Knapsack Problem [22], a set of items $N = \{0, ..., n-1\}$ with weights w_i and profits p_i for $i \in N$ and a knapsack with budget B are given. The objective is to maximize the total profit of the items in the knapsack. The items are sorted in descending order of the profit ratio (p_i/w_i). The instance distribution is taken from the "Hard" instances in Cappart et al. [8], where profits and weights are strongly correlated.

DIDP Model: A DIDP state is a tuple $\langle x, i \rangle$, where x is the current total weight and i represents the current item index. The DIDP model is based on Kuroiwa and Beck [19] with the remaining budget replaced by the current total weight x:

$$\text{compute } V(0,0) \tag{9}$$

$$V(x,i) = \begin{cases} \max\{p_i + V(x+w_i, i+1), V(x, i+1)\} & \text{if } i < n \wedge x + w_i \leq B \\ V(x, i+1) & \text{if } i < n \wedge x + w_i > B \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

$$V(x,i) \leq \min \left\{ \sum_{j=i}^{n-1} p_j, \max_{j \in \{i..n-1\}} \left(\frac{p_j}{w_j} \right) \cdot (B - x) \right\}. \tag{11}$$

Expression (9) declares that the optimal cost is the cost to consider all items starting from the first item ($i = 0$) with the current total weight $x = 0$. The first line of Eq. (10) corresponds to considering item i; if i is taken (the first term), then x is updated to $x + w_i$ and the item index is updated to $i + 1$. If i is not taken (the second term), x remains the same. The second line of Eq. (10) indicates that i cannot be taken if doing so exceeds budget B. The third line is

the base case; when all items are visited ($i \geq n$), then the recursion terminates. Equation (11) represents two dual bounds.

RL Model: The MDP for the RL agent is as follows:

`State` s: $\langle x, i \rangle$
`Action` $a \in \{0, 1\}$: whether to take the item i or not.
`Transition function` $\mathrm{T}(s, a)$: $\mathrm{T}(\langle x, i \rangle, a) = \langle aw_i + x, i + 1 \rangle$
`Reward function` $\mathrm{R}(s, a)$: $\mathrm{R}(\langle x, i \rangle, a) = \beta(ap_i)$

The RL state is same as the DP state, and the action set is binary: 0 indicates the item is not selected, while 1 means it is selected. The transition function matches the effect of a transition in the DIDP model. The state variable x is updated to $x + aw_i$, i.e., w_i is added to x if the item is selected ($a = 1$). Also, the item index i is incremented by 1. The reward function corresponds to p_i, the profit of item i. The scaling factor is set to $\beta = 0.0001$.

Portfolio Optimization. In the 4-moments portfolio optimization problem [3], a set of investments $N = \{0, ..., n-1\}$, each with a specific cost (w_i), expected return (μ_i), standard deviation (σ_i), skewness (γ_i), and kurtosis (κ_i), and the budget B are given. The goal is to find a portfolio with a maximum return as specified by the objective function (Eq. (12)). Each financial characteristic is weighted (λ_1, λ_2, λ_3, and λ_4). The instance distribution is taken from Cappart et al. [8].

DIDP Model: A state is a tuple $\langle x, i, Y \rangle$, where x is the current total weight, i is the current item index, and Y is a set of investments up to i, $\{0, ..., i-1\}$. The objective function value up to item i is defined as follows:

$$\nu(Y) = \lambda_1 \sum_{j \in Y} \mu_j - \lambda_2 \sqrt[2]{\sum_{j \in Y} \sigma_j^2} + \lambda_3 \sqrt[3]{\sum_{j \in Y} \gamma_h^3} - \lambda_4 \sqrt[4]{\sum_{j \in Y} \kappa_j^4}. \tag{12}$$

The transition cost is the difference between the objective value of the current state ($\nu(Y)$) and that of the successor state, i.e., $\nu(Y \cup \{i+1\}) - \nu(Y)$. The DIDP model is expressed as follows:

$$\text{compute } V(0, 0, \emptyset) \tag{13}$$

$$V(x, i, Y) = \begin{cases} \max(\nu(Y \cup \{i\}) - \nu(Y) + V(x + w_i, i + 1, Y \cup \{i\}), \\ \qquad V(x, i + 1, Y)) & \text{if } i < n \wedge x + w_i \leq B \\ V(x, i + 1, Y) & \text{if } i < n \wedge x + w_i > B \\ 0 & \text{otherwise.} \end{cases} \tag{14}$$

$$V(x, i, Y) \leq \min \left(\lambda_1 \sum_{j=i}^{n-1} \mu_j + \lambda_3 \sqrt[3]{\sum_{j=i}^{n-1} \gamma_j^3}, \max_{j \in \overline{Y}} K_j \cdot (B - x) \right) \tag{15}$$

where $K_j = \left(\frac{\lambda_1 \mu_j + \lambda_3 \sqrt[3]{\gamma_j^3}}{w_j} \right)$ and $\overline{Y} = \{i..n-1\}$. In the first line of Eq. (14), if item i is taken, Y is updated to $Y \cup \{i\}$. The second and third lines of Eq. (14) are the same as Eq. (10) except that a state includes Y as well. Equation (15) are two dual bounds. Proofs for the dual bounds are in Appendix D in the arXiv version.

RL Model: The MDP for the RL agent is as follows:

`State` s: $\langle x, i, Y \rangle$
`Action`: $a \in \{0, 1\}$
`Transition function` $\mathrm{T}(s, a)$:
$T(\langle x, i, Y \rangle, 1) = \langle w_i + x, i + 1, Y \cup \{i\} \rangle$
$T(\langle x, i, Y \rangle, 0) = \langle x, i + 1, Y \rangle$
`Reward function` $\mathrm{R}(s, a)$: $R(\langle x, i, Y \rangle, a) = a \cdot (\nu(Y \cup \{i\}) - \nu(Y))$

In the transition function, Y is updated to $Y \cup \{i\}$ if i is taken. The scaling factor is set to $\beta = 0.0001$.

5 Results

Figure 3 shows the solution quality per node expansion with different heuristic guidance for each problem and DIDP search algorithm.

TSP. The plots highlight the strong performance of PPO guidance (red) compared to other heuristics, including dual-bound (blue) and greedy heuristic (green), across all three solvers. DQN also outperforms the default dual-bound guidance except for APPS, though it falls short of the problem-specific greedy heuristic. Greedy heuristic guidance significantly outperformed dual-bound guidance.

TSPTW. The solid yellow and red lines (denoted as "RL = tsptw") represent the performance of DIDP guided by the RL agent trained in the TSPTW environment. The performance of these DIDP was significantly worse than that of dual-bound and greedy heuristic guidance. In fact, their performance was even worse than $h = 0$. Given these results, we experimented with using the TSP RL model to guide the search (dotted lines), which significantly outperformed DIDP guided by the TSPTW RL model but achieved about the same performance as other heuristic guidance methods, including $h = 0$.

0-1 Knapsack. All the heuristic guidance quickly achieved solutions with gaps of less than 1%, although dual-bound guidance (blue) exhibited slightly worse performance compared to the others. During training, the policies generated by DQN and PPO rapidly converged to the best-ratio heuristic (greedy heuristic), which explains the similar behavior across these three guidance methods.

Portfolio Optimization. The plots highlight that PPO guidance (red) significantly outperforms other guidance, including the problem-specific greedy guidance (green). DQN guidance (yellow) also surpassed the dual-bound guidance (blue) and was competitive with the greedy heuristic.

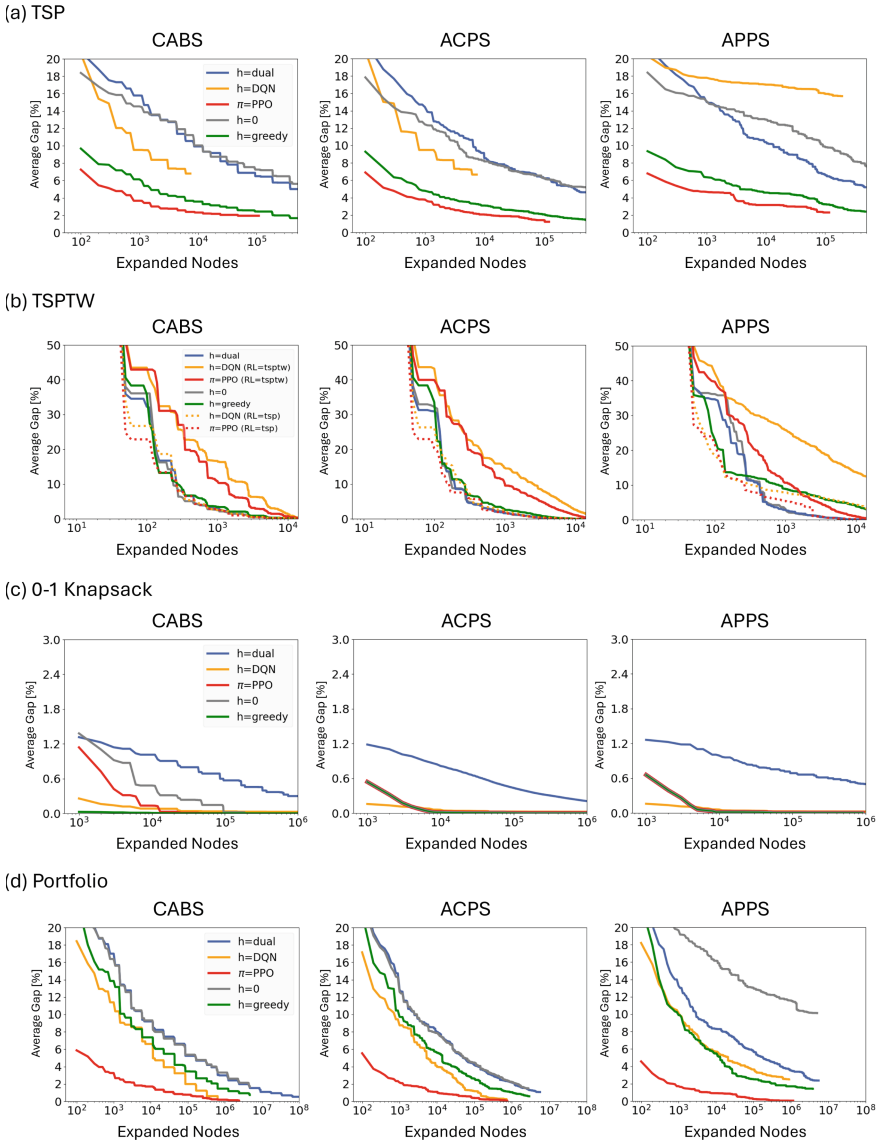


Fig. 3. Results of applying heuristics to guide DIDP, averaged over 40 instances (20 each for small and medium sizes). Small instances have $n = 20$ and medium instances have $n = 50$, except for 0-1 Knapsack ($n = 50$ small, $n = 100$ medium).

Table 1 shows the performance of the different methods. DIDP performed best in TSPTW, MIP (Gurobi) in TSP and Knapsack, and CP Optimizer for Portfolio. DIDP guided by PPO outperforms the dual-bound guidance using the same solver in terms of average gap across all problem domains except for

TSP with $n = 50$. As CABS with dual-bound guidance is the best performing solver [19], these results suggest that PPO guidance has surpassed the current state-of-the-art for DIDP in these problems. DIDP guided by DQN also outperformed dual-bound in several settings, though it falls short in TSP.

RL guidance takes orders of magnitude more time for per node expansion due to the call to the neural network prediction. Despite this bottleneck, PPO guidance achieves better performance than the baselines at the time limit. Compared to Cappart et al. [8], the DQN-guided approach in our framework achieves significantly higher performance. For instance, in Portfolio, BaB-DQN achieves an average gap of 10.8%, while CABS ($h = \text{DQN}$) achieves a substantially lower gap of 0.77%. Similarly, DIDP with $h = \text{DQN}$ outperforms BaB-DQN in TSPTW, likely because the base DIDP model substantially outperforms the base CP model. PPO guidance exhibits a similar trend, showing notable improvements over RBS-PPO in TSPTW and slightly better results in Portfolio (e.g., 0.50% for RBS-PPO compared to 0.19% for CABS ($\pi = \text{PPO}$)). However, in TSP, RBS-PPO shows slightly better performance than DIDP guided by PPO.

Table 1 also compares the performance of heuristic sampling against baseline methods. In TSP, PPO clearly outperformed other heuristics, achieving average gaps of 0.26% for $n = 20$ and 3.85% for $n = 50$. In TSPTW, DQN and PPO heuristics were relatively effective in finding feasible solutions (e.g., achieving feasibility in 16 out of 20 instances for $n = 20$), but their ability in optimizing the solution cost is poor (average gaps of 35.12% for DQN and 25.64% for PPO for $n = 20$). For Knapsack, the performance across heuristics was similar, although the dual-bound heuristic was slightly worse. In Portfolio, PPO showed strong standalone performance, being only 0.75% worse than the best-known solution for $n = 50$. The average gap for DQN (3.63% for $n = 20$ and 8.09% for $n = 50$) is comparable to that of the greedy heuristic (5.22% for $n = 20$ and 6.84% for $n = 50$), while the dual-bound heuristic performed considerably worse (4.19% for $n = 20$ and 18.46% for $n = 50$).

6 Discussion

When is RL Guidance Helpful, and to What Extent? RL guidance is most impactful when heuristic quality plays a critical role in solution quality. For instance, in TSPTW, the DIDP model prunes many states by time windows. In such cases, the primary role of heuristic guidance in DIDP appears to be minimizing solution costs rather than ensuring feasibility. In contrast, TSP and Portfolio DP models lack such pruning mechanisms, making heuristic quality a more critical factor in improving the solution quality.

When the performance of DIDP appears to depend primarily on heuristic guidance, the effectiveness of guidance aligns with the performance of the heuristic in sampling-based approaches. For example, PPO guidance consistently outperforms dual-bound guidance because the PPO heuristic is better at driving the search towards high quality solution, as shown in Table 1. Dual-bounds are admissible and thus effective at de-prioritizing unpromising decisions, but may not necessarily guide the search towards more promising solutions.

Table 1. Comparison of results with baseline methods. Values represent averages over 20 instances. The lowest average gap for each problem and size n is underlined. The symbol * indicates that optimality was proven for all 20 instances. In the DIDP results, values are highlighted in bold if the corresponding method achieves a better average gap than dual-bound guidance using the same solver. For TSP with $n = 50$, sampling with DQN timed-out before completing 1280 samples. "-" denotes reaching either time or memory limits. "t.o." indicates that all 20 instances reached the time limit.

			TSP				TSPTW						0-1 Knapsack				Portfolio			
			n=20		n=50		n=20			n=50			n-50		n=100		n=20		n=50	
Type	Name		Gap	Time	Gap	Time	Feas.	Gap	Time	Feas.	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time
CP	CP Optimizer		0.00	25	2.75	t.o.	20	12.04	t.o.	20	32.32	t.o.	0.00	t.o.	0.02	t.o.	0.00*	<1	0.00	t.o.
	BaB-DQN		3.77	t.o.	18.46	t.o.	20	0.00*	216	20	40.83	t.o.	0.00	t.o.	0.00	t.o.	0.00*	1270	10.18	t.o.
	RBS-PPO		0.12	t.o.	1.17	t.o.	20	0.65	t.o.	20	33.82	t.o.	0.00	t.o.	0.00	t.o.	0.00*	487	0.50	t.o.
MIP	Gurobi		0.00*	3	0.03	t.o.	20	0.00*	<1	20	0.00*	119	0.00*	<1	0.00*	<1	-	-	-	-
Heuristics	DQN		2.29	1251	(20.14)	(18239)	16	35.12	1405	0	-	-	0.02	44	0.04	121	3.63	36	8.09	132
	PPO		0.26	249	3.85	1783	20	25.64	423	20	46.93	2050	0.04	51	0.13	132	3.35	38	0.75	111
	Dual-bounds		2.91	4	9.58	1424	0	-	-	0	-	-	0.07	8	0.37	44	4.19	1	18.46	4
	Greedy		5.30	11	8.48	176	0	-	-	0	-	-	0.03	3	0.01	8	5.22	2	6.84	7
DIDP	CABS	h=greedy	0.00	292	2.79	-	20	0.00*	<1	20	0.00*	22	0.00	-	**0.00**	-	0.00*	15	1.48	-
		h=dual	0.00*	19	1.86	-	20	0.00*	<1	20	0.00*	<1	0.00	-	0.09	-	0.00*	2	0.81	-
		h=DQN	0.00	154	12.05	t.o.	20	0.00*	44	20	0.00*	765	0.00	t.o.	**0.05**	t.o.	0.00*	780	**0.77**	t.o.
		π=PPO	0.00	32	3.89	t.o.	20	0.00*	40	20	0.00*	860	0.00	t.o.	**0.00**	t.o.	0.00*	477	**0.19**	t.o.
	ACPS	h=greedy	0.00*	62	**2.45**	-	20	0.00*	<1	20	0.00*	2	0.00	-	**0.00**	-	0.00*	4	**1.17**	-
		h=dual	0.00*	8	6.35	-	20	0.00*	<1	20	0.00*	<1	0.00	-	0.21	-	0.00*	<1	2.14	-
		h=DQN	0.00	371	10.09	t.o.	20	0.00*	11	20	0.00*	99	0.00	t.o.	**0.03**	t.o.	0.00*	180	**0.50**	t.o.
		π=PPO	0.00	345	**2.45**	t.o.	20	0.00*	11	20	0.00*	123	0.00	t.o.	**0.00**	t.o.	0.00*	119	**0.14**	t.o.
	APPS	h=greedy	0.00*	66	**3.46**	-	20	0.00*	<1	20	0.00*	4	0.00	-	**0.00**	-	0.00*	4	**2.86**	-
		h=dual	0.00*	8	8.30	-	20	0.00*	<1	20	0.00*	<1	0.00	-	0.58	-	0.00*	<1	4.74	-
		h=DQN	1.08	t.o.	31.57	t.o.	20	0.00*	13	20	0.00*	141	0.00	t.o.	**0.03**	t.o.	0.00*	187	5.01	t.o.
		π=PPO	0.20	t.o.	**4.17**	t.o.	20	0.00*	12	20	0.00*	141	0.00	t.o.	**0.00**	t.o.	0.00*	119	**0.10**	t.o.

Solution Quality in Terms of Solve Time. While DQN and PPO guidance demonstrate significantly higher solution quality per node expansion compared to dual-bound guidance, their performance gains over time are relatively limited. The primary cause lies in the time required to expand a single node. As shown in Table 1, generating a solution using DQN or PPO takes much longer than using dual-bound or greedy heuristics (e.g., DQN takes 313 times longer than dual-bound to sample 1280 times for TSP $n = 50$). While our experiments highlight the potential of RL-based heuristics, they also emphasize the need to address the computational overhead associated with these methods.

7 Conclusion

The initial demonstration of DIDP solver performance was based on search guidance with dual bounds defined in the model. Through experiments on three anytime algorithms, we demonstrated that RL can provide heuristic guidance that improves solution quality with fewer node expansions. These findings show the effectiveness of RL-guided search within anytime algorithms and help to elucidate the conditions where RL guidance is most beneficial, such as in domains where heuristic quality plays a critical role in solution improvement. The inherent structural similarity between DP and RL models offers a natural synergy,

enabling RL to be easily integrated into the DIDP framework. With further work on automating RL model building and reducing the time to evaluate states, RL-guided DIDP has the potential to serve as a practical and powerful tool for combinatorial optimization.

References

1. Agostinelli, F., McAleer, S., Shmakov, A., Baldi, P.: Solving the Rubik's cube with deep reinforcement learning and search. Nat. Mach. Intell. **1**(8), 356–363 (2019). https://doi.org/10.1038/s42256-019-0070-z
2. Arulkumaran, K., Deisenroth, M.P., Brundage, M., Bharath, A.A.: Deep reinforcement learning: a brief survey. IEEE Signal Process. Mag. **34**(6), 26–38 (2017). https://doi.org/10.1109/MSP.2017.2743240
3. Atamtürk, A., Narayanan, V.: Polymatroids and mean-risk minimization in discrete optimization. Oper. Res. Lett. **36**(5), 618–622 (2008). https://doi.org/10.1016/j.orl.2008.04.006
4. Bellman, R.: Dynamic Programming. Princetion University Press, Princeton (1957)
5. Bello, I., Pham, H., Le, Q.V., Norouzi, M., Bengio, S.: Neural combinatorial optimization with reinforcement learning. In: The Fifth International Conference on Learning Representations (2017). https://openreview.net/forum?id=rJY3vK9eg
6. Boisvert, L., Verhaeghe, H., Cappart, Q.: Towards a generic representation of combinatorial problems for learning-based approaches. In: International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research, pp. 99–108. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-60597-0_7
7. Cappart, Q., Chételat, D., Khalil, E.B., Lodi, A., Morris, C., Veličković, P.: Combinatorial optimization and reasoning with graph neural networks. J. Mach. Learn. Res. **24**(130), 1–61 (2023)
8. Cappart, Q., Moisan, T., Rousseau, L.M., Prémont-Schwarz, I., Cire, A.A.: Combining reinforcement learning and constraint programming for combinatorial optimization. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, pp. 3677–3687. AAAI Press (2021). https://doi.org/10.1609/aaai.v35i5.16484
9. Dai, H., Khalil, E.B., Zhang, Y., Dilkina, B., Song, L.: Learning combinatorial optimization algorithms over graphs. In: Advances in Neural Information Processing Systems, pp. 6351–6361. Curran Associates Inc. (2017)
10. Flood, M.M.: The traveling-salesman problem. Oper. Res. **4**(1), 61–75 (1956). https://doi.org/10.1287/opre.4.1.61
11. Fu, Z.H., Qiu, K.B., Zha, H.: Generalize a small pre-trained model to arbitrarily large TSP instances. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, pp. 7474–7482 (2021). https://doi.org/10.1609/aaai.v35i8.16916
12. Gehring, C., et al.: Reinforcement learning for classical planning: viewing heuristics as dense reward generators. In: Proceedings of the International Conference on Automated Planning and Scheduling, vol. 32, pp. 588–596 (2022). https://doi.org/10.1609/icaps.v32i1.19846
13. Huang, S., Ontañón, S.: A closer look at invalid action masking in policy gradient algorithms. arXiv preprint arXiv:2006.14171 (2020)
14. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: a survey. J. Artif. Intell. Res. **4**, 237–285 (1996). https://doi.org/10.1613/jair.301

15. Kool, W., van Hoof, H., Gromicho, J., Welling, M.: Deep policy dynamic programming for vehicle routing problems. In: International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research, pp. 190–213. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-08011-1_14
16. Kool, W., van Hoof, H., Welling, M.: Attention, learn to solve routing problems! In: International Conference on Learning Representations (2019). https://openreview.net/forum?id=ByxBFsRqYm
17. Kuroiwa, R., Beck, J.C.: Domain-independent dynamic programming: generic state space search for combinatorial optimization. In: Proceedings of the 33rd International Conference on Automated Planning and Scheduling (ICAPS), vol. 33, pp. 236–244. AAAI Press (2023). https://doi.org/10.1609/icaps.v33i1.27200
18. Kuroiwa, R., Beck, J.C.: Solving domain-independent dynamic programming problems with anytime heuristic search. In: Proceedings of the 33rd International Conference on Automated Planning and Scheduling (ICAPS). AAAI Press (2023). https://doi.org/10.1609/icaps.v33i1.27201
19. Kuroiwa, R., Beck, J.C.: Domain-independent dynamic programming. arXiv preprint arXiv:2401.13883 (2024)
20. Lee, J., Lee, Y., Kim, J., Kosiorek, A., Choi, S., Teh, Y.W.: Set transformer: a framework for attention-based permutation-invariant neural networks. In: Proceedings of the 36th International Conference on Machine Learning, vol. 97, pp. 3744–3753. PMLR (2019). https://proceedings.mlr.press/v97/lee19d.html
21. Li, S., Wang, R., Tang, M., Zhang, C.: Hierarchical reinforcement learning with advantage-based auxiliary rewards. In: Advances in Neural Information Processing Systems. Curran Associates Inc., Red Hook, NY, USA (2019)
22. Martello, S., Toth, P.: Knapsack Problems: Algorithms and Computer Implementations. Wiley, Hoboken (1990)
23. Mnih, V., et al.: Asynchronous methods for deep reinforcement learning. In: Proceedings of the 33rd International Conference on Machine Learning, vol. 48, pp. 1928–1937. PMLR (2016). https://proceedings.mlr.press/v48/mniha16.html
24. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**, 529–533 (2015). https://doi.org/10.1038/nature14236
25. Nazari, M., Oroojlooy, A., Takáč, M., Snyder, L.V.: Reinforcement learning for solving the vehicle routing problem. In: Advances in Neural Information Processing Systems, vol. 31, pp. 9861–9871. Curran Associates, Inc. (2018)
26. Orseau, L., Lelis, L., Lattimore, T., Weber, T.: Single-agent policy tree search with guarantees. In: Advances in Neural Information Processing Systems, vol. 31. Curran Associates, Inc. (2018)
27. Orseau, L., Lelis, L.H.: Policy-guided heuristic search with guarantees. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, pp. 12382–12390. AAAI Press (2021). https://doi.org/10.1609/aaai.v35i14.17469
28. Puterman, M.L.: Markov decision processes. In: Handbooks in Operations Research and Management Science, vol. 2, pp. 331–434. Elsevier (1990)
29. Savelsbergh, M.W.: Local search in routing problems with time windows. Ann. Oper. Res. **4**, 285–305 (1985). https://doi.org/10.1007/BF02022044
30. Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P.: Trust region policy optimization. In: Proceedings of the 32nd International Conference on Machine Learning, vol. 37, pp. 1889–1897. PMLR (2015). https://proceedings.mlr.press/v37/schulman15.html
31. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)

32. Silver, D., et al.: Mastering the game of Go with deep neural networks and tree search. Nature **529**, 484–503 (2016). https://doi.org/10.1038/nature16961
33. Silver, D., et al.: Mastering chess and shogi by self-play with a general reinforcement learning algorithm. arXiv preprint arXiv:1712.01815 (2017)
34. Sun, Z., Yang, Y.: Difusco: graph-based diffusion solvers for combinatorial optimization. In: Advances in Neural Information Processing Systems, vol. 36, pp. 3706–3731. Curran Associates Inc. (2023)
35. Sutton, R.S., Barto, A.G.: The reinforcement learning problem. In: Reinforcement Learning: An Introduction, pp. 51–85. MIT Press (1998)
36. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. In: International Conference on Learning Representations (2018). https://openreview.net/forum?id=rJXMpikCZ
37. Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R.R., Smola, A.J.: Deep sets. In: Advances in Neural Information Processing Systems, vol. 30, pp. 3394–3404. Curran Associates Inc. (2017)
38. Zhang, C., Song, W., Cao, Z., Zhang, J., Tan, P.S., Xu, C.: Learning to dispatch for job shop scheduling via deep reinforcement learning. In: Advances in Neural Information Processing Systems, vol. 33, pp. 1621–1632. Curran Associates Inc. (2020)

Acquiring and Selecting Implied Constraints with an Application to the BinSeq and Partition Global Constraints

Jovial Cheukam Ngouonou[1,2,4], Ramiz Gindullin[1,2], Claude-Guy Quimper[4(✉)], Nicolas Beldiceanu[1,2], and Rémi Douence[1,2,3]

[1] IMT Atlantique, Nantes, France
jovial.cheukam-ngouonou.1@ulaval.ca, ramiz.gindullin@it.uu.se, {nicolas.beldiceanu,Remi.Douence}@imt-atlantique.fr
[2] LS2N, Nantes, France
[3] INRIA, Nantes, France
[4] Université Laval, Québec City, Canada
Claude-Guy.Quimper@ift.ulaval.ca

Abstract. We propose a machine-assisted approach to synthesise implied constraints for global constraints based on combinatorial objects. By reusing the Bound Seeker (BS) [8], we generate thousands of relationships between features. We present a scalable algorithm that automatically selects the relationships that filter the most, which we manually prove. We consider the Partition and the BinSeq constraints, which model the different ways of dividing a collection of objects into clusters, or the repartition of shifts in a 0–1 sequence. We use Partition and BinSeq in the Balanced Academic Curriculum Problem (BACP), and the Balanced Shift-Scheduling Problem (BSSP), where we optimise the distribution of the work to balance the workload. For 2 models of the BACP and 2 models of the BSSP, we show how the filtering inferred by the BS improves the cost of the solution found on different solvers. This filtering proved optimality for all CSPLib instances of the BACP.

1 Introduction

Constraint solvers rely on Filtering Algorithms (FAs) that are designed for each constraint. In recent decades, an effort has led to the custom development of FAs [5,7,29] to reduce the search space. Synthesising FAs can be traced back to research on the Rabbit solver [23], and studies on synthesising propagators for low-arity constraints [17]. However, the need to consider the interaction of an increasing number of constraint parameters raises the question of how to synthesise implied constraints that can be used in different solvers. We propose a novel machine-assisted approach to generate implied constraints corresponding to a set of bounds used for filtering a constraint for which a reformulation already exists. The BS [8] discovers each bound, and our method chooses the most important

G. Tack (Ed.): CPAIOR 2025, LNCS 15763, pp. 155–172, 2025.
https://doi.org/10.1007/978-3-031-95976-9_10

ones that we manually prove after the selection process. We consider satisfaction or optimisation problems whose solution is given by a combinatorial object, e.g. a graph, a 0–1 sequence, and a partition. The constraints are imposed on the features of this object. e.g. in the balanced academic curriculum problem, one wants to partition courses into terms. The solution is a partition that is constrained to spread the workload between terms. One can define a constraint optimisation problem whose objective is to minimise the sum of the squares of the partition sizes. Modelling this to obtain a strong filtering is not trivial.

The BS [8,20] identifies conjectures of sharp bounds between the features of a combinatorial object. These conjectures are inequalities between a single feature and a function over other features, e.g. consider a partition of n elements where $\overline{M}$ is the size of the largest partition and S is the sum of the squares of the partition sizes. The BS discovers the sharp bound $S \leq (n \bmod \overline{M})^2 - \overline{M}(n \bmod \overline{M}) + n\overline{M}$. This bound can be added, as a redundant constraint, to a model to enhance filtering. As our machine-assisted synthesis method is independent of bound generation, we treat the BS [8] as a black box that takes combinatorial object instances as input and outputs valid inequalities, i.e. arithmetic constraints, for all input data.

The BS can find thousands of relations between features, but not all of these bounds are equally useful for filtering. We designed an algorithm that selects a subset of these bounds that are relevant for filtering the variable domains of a constraint satisfaction problem whose solution is an instance of the combinatorial object. Finally, we add the inequalities associated with the bounds to the constraint model. The filtering performed by these inequalities acts as a filtering algorithm for a global constraint encoding the combinatorial object. To illustrate the efficiency of our method, we choose the partition and 0–1 sequence combinatorial objects introduced in [20]. Partitions are used in rostering problems to distribute workload between employees or in the balanced academic curriculum problem [22] to group courses of a same term. 0–1 sequences are used in nurse scheduling problems to assign shifts to nurses wrt to a set of rules. An objective is to obtain partitions or shifts of similar size. It is usual to approximate this objective by minimising the size of the largest partition or the longest shift, as filtering algorithms for this objective are efficient. More balanced solutions can be obtained by minimising the squared size of partitions or shift sizes, but this is more difficult, and few constraints exist. Spread [26,31] minimises the variance of a vector, equivalent to minimising the sum of squares when the mean is fixed. Deviation [27,30] computes the sum of absolute differences between a vector's values and their mean. In contrast, Partition aims to spread the number of times these values occur. Balance [11] minimises the difference between the most and least frequent values, as does Partition, while also minimising the squared number of occurrences. This is where exploiting the inequalities found by the BS becomes useful. Our contributions include: (1) *Machine Assisted Implied Constraints Synthesis*. Based on a set of discovered bounds, we present a framework for generating FAs that can be easily integrated into multiple constraint solvers. (2) *Bound Selection*. We propose a method for selecting the most signifi-

cant bounds automatically from a large set of candidates. (3) *Global Constraints for Balanced Solutions.* We improve the search for better balanced solutions to partitioning and shift scheduling problems by introducing the *Partition* and *BinSeq* constraints. (4) *Theoretical Insights.* We prove a sharp lower bound on the sum of the squared partition sizes that penalises both results above and below the load average. The penalty increases quadratically, which helps to prevent large deviations.

Section 2 gives the background. Section 3 presents the framework for selecting redundant constraints. Section 4 defines the Partition and BinSeq constraints. Section 5 uses the framework of Sect. 3 to generate redundant constraints for Partition and BinSeq. Section 6 evaluates their impact on the BACP and BSSP.

2 Background

Presolvers. Presolvers simplify models by fixing variables, reducing constraints, and strengthening bounds in mixed integer programs [2]. SAT solvers use techniques like unit propagation and subsumption [12]. Constraint satisfaction problem preprocessing is complex due to constraint diversity, but subexpression elimination enhances filtering [28]. Presolving can globally improve models, as seen in time series pattern analysis for bound inference [4]. Redundant constraints can strengthen filtering and reduce search space. Gent *et al.* [18] synthesise filtering algorithms for small constraints via exponential offline preprocessing. Charnley *et al.* [14] generate implied constraints for finite algebra. Our work differs in that their implied constraints have a simpler form, and their constraint selection process is costly, analysing all possible subsets of constraints up to a limit.

Conjecture Discovery. Few systems search for bounds on the features of a combinatorial object. Among the best known are the S. Fajtlowicz's Graffiti program [16] and P. Hansen's AutoGraphiX system [3,21]. We describe a third, more recent system called the BS [8] to exploit the sharp bounds it discovers, to synthesise implied constraints. Let $\mathcal{O}$ be a combinatorial object identified by m features $x_1, x_2, \ldots, x_m$ taking integer values, where the maximum value of the first feature x_1 restricts the range of $x_2, \ldots, x_m$. The BS takes as input a set of extreme instances for $\mathcal{O}$ and learns inequalities of the forms $x_i \leq u_i(P)$ and $x_i \geq l_i(P)$, where l_i and u_i are symbolic functions and $P \subseteq \{x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_m\}$. Thus, there is a possibility of $m2^m$ unique bounds. These inequalities bound the feature x_i, and the BS guarantees that these bounds are tight, i.e. there exists an instance of $\mathcal{O}$ for which the equality holds. These bounds, called *conjectures*, were derived from some instances of $\mathcal{O}$ and may not apply to all possible instances. Turning these conjectures into theorems requires a formal proof.

3 The Framework

The framework takes a combinatorial object and returns a set of constraints. They are computed only once, and later used to strengthen filtering. The Bounds

Seeker breaks down the process into three steps. First, it starts with input instances of a combinatorial object and generates conjectures in the form of redundant constraints that enhance filtering. However, not all constraints result in additional filtering. In Step 2, we remove some constraints to create a subset that filters as effectively as if all constraints were included. Two constraint sets are equivalent if replacing one set with another in a constraint model solves a problem with the same number of backtracks. In Step 3, we manually prove the conjectures underlying the selected constraints.

3.1 Generating Candidate Constraints

We start with a constraint satisfaction problem whose solution is expressed in terms of a combinatorial object, e.g. a graph, a 0–1 sequence, or a partition. We identify the features $x_1, x_2, \ldots, x_m$ of this combinatorial object that are important for the model. These features are numerical values that characterise the solution. For a graph, this could be the number of nodes, edges, components, etc. The problem's definition usually constrains these features.

The BS [20] takes as input instances of a combinatorial object and a list of features that can be computed on these objects. It computes a set of conjectures $\mathcal{C} = \{C_1, \ldots, C_{|\mathcal{C}|}\}$. Each conjecture C_i is of the form $x_i \leq u_i(x_{j_1}, x_{j_2}, \ldots)$ or $x_i \geq l_i(x_{j_1}, x_{j_2}, \ldots)$. These inequalities could be added directly to the model of the constraint satisfaction problem to enhance filtering. Even if, in practice, the BS is unable to find all $m2^m$ possible distinct conjectures in a reasonable time, it is nevertheless able to compute thousands of them. The time spent on filtering these constraints could outweigh the time saved by pruning the search tree. Many conjectures may filter the same values, and some conjectures might have their filtering dominated by others.

Algorithm 2 is an algorithm that removes some conjectures on bounds from the set $\mathcal{C}$ without reducing the filtering. It returns a subset of conjectures that is minimal in the sense that removing any conjecture from this subset reduces the amount of filtering. Algorithm 2 solves a Constraint Satisfaction Problem (CSP). Given the input $\mathcal{C}$ of bound conjectures, let $M(\mathcal{C})$ be a CSP outputting all possible combinations of values for the features $x_1, x_2, \ldots, x_m$ wrt an upper limit ℓ for x_1, where the feature x_1 bounds the size of the generated object, leading to a finite set of solutions. Let $I(\mathcal{O})$ be the instances of the combinatorial object. $M(\mathcal{C})$ is defined by:

$$M(\mathcal{C}) \iff \begin{cases} x_1 \leq \ell \quad (1) & (x_1, x_2, \ldots, x_m) \in I(\mathcal{O}) \quad (2) \\ C_1 \wedge C_2 \wedge \ldots \wedge C_{|\mathcal{C}|} \quad (3) & x_i \in \mathrm{dom}(x_i)\ \forall i \in \{1, \ldots, m\} \quad (4) \end{cases}$$

The constraint (1) bounds a feature (often chosen to be the size) to make the number of solutions finite. The constraint (2) forces the features to be consistent with the definition of the combinatorial object. That is, the features must have values that correspond to an instance of the combinatorial object, e.g. a graph could not have nine edges but only two nodes. This constraint (2) can be encoded

Algorithm 1: SelectOne$(F, C, nback)$

```
1 if |C| = 1 then
2 |   nback' ← Enumerate(M(F));
3 |   if nback = nback' then return (∅, C) else return (C, ∅);
4 Evenly partition C into sets C1, C2; nback' ← Enumerate(M(F ∪ C2));
5 if nback = nback' then // K stands for Keep, R for Reject
6 |   (K, R) ← SelectOne(F, C2, nback); return (K, C1 ∪ R);
7 else return SelectOne(F ∪ C2, C1, nback) ;
```

Algorithm 2: Selection(C)

```
1 nback ← Enumerate(M(C)); F ← ∅;
2 repeat
3 |   (K, R) ← SelectOne(F, C, nback); F ← F ∪ K; C ← C \ (K ∪ R);
4 until K = ∅ ∨ C = ∅;
5 return F;
```

according to the modeller's preferences. It can be seen as the modeller's prior knowledge of the constraint. This may be a decomposition into constraints that entirely define the possible values for the features. The constraint (3) is the conjunction of the conjectures in $\mathcal{C}$. Let Enumerate be an algorithm that takes as input a model that returns the number of backtracks to enumerate all the solutions by a solver.

Algorithm 1 takes as input a set of forced conjectures F, a set of candidate conjectures C, and the number of backtracks *nback* taken by the solver to enumerate all solutions using all conjectures found by the BS. It returns a tuple of two sets: A set K containing a single conjecture to keep, as it contributes to the filtering or an empty set if none exists, and a set R of conjectures to reject, as they do not contribute to any filtering. Algorithm 1 works as a binary search to find the first constraint that impacts the filtering with $O(\log |C|)$ calls to the solver. If enumerating the solutions of $M(F \cup C_2)$ triggers *nback* backtracks, then the constraints in C_1 do not filter values that are not already filtered by the constraints in $F \cup C_2$. We can reject the constraints C_1 and continue searching in C_2. Otherwise, the enumeration of $M(F \cup C_2)$ triggers more backtracks, and there exists at least one constraint in C_1 that captures a filtering missed by C_2. The binary search can continue in C_1. Algorithm 2 calls Algorithm 1 multiple times and extracts, one by one, the conjectures that have an impact on the filtering. It executes $O(|F| \log(|C|))$ calls to the solver where $|F|$ is the number of selected conjectures. The algorithms might return different sets of constraints depending on the order they eliminate the constraints. They might not find the smallest subset, but the subset is minimal, since removing any conjecture increases the number of backtracks. We also did an incremental version of Algorithm 2 in which calls to the Enumerate function in Algorithm 1 were optimised. These calls detect whether enumerating all solutions with a subset of

conjectures takes as many backtracks as enumerating using all conjectures. The enumeration can stop as soon as the solver finds a solution with more backtracks than it required while using all constraints. In such a case, $nback \neq nback'$ even if we let the enumeration complete. Selected conjectures can be proved using a theorem prover or by a human to become theorems. Adding the constraints to the model and letting the solver filter them leads to a correct and reinforced filtering.

4 Defining the Partition and the BinSeq Constraints

Humans are interested in balanced solutions for assignment problems. Examples of such problems are (*i*) problems where the total amount of work or time assigned to each worker or the amount of coursework to be validated by students for each session in an academic curriculum must be evenly distributed, and (*ii*) problems where the shift lengths assigned to each nurse should not vary too much, while meeting the demands of each period and the regulation. In this context, several constraints, such as *NValue* [24], *Balance* [7,11], *Among* [9] and *Stretch* [25] were introduced in constraint programming; these constraints are unified by the *Partition* and the *BinSeq* constraints.

4.1 Defining the Partition Constraint

A partition of a set U is a collection of subsets $S_1, S_2, \ldots, S_P$ that are pairwise disjoint, but whose union gives U. An array $\mathcal{X} = [X_1, X_2 \ldots, X_n]$ of integers encodes a partition as follows. The value $i \in U$ belongs to the subset S_j if and only if $X_i = j$. The partition has six features: (*i*) n the dimension of the array, i.e. the cardinality of U, (*ii*) P the number of distinct values in $\mathcal{X}$, i.e. the number of subsets, (*iii*) $\underline{M}$ (resp. (*iv*) $\overline{M}$) the number of occurrences of the least (resp. most) frequent integer in $\mathcal{X}$, (*v*) $\overline{\underline{M}}$ the difference $\overline{M} - \underline{M}$, (*vi*) S the sum of the squares of the number of occurrences for each distinct integer in $\mathcal{X}$.

Definition 1. *We define Partition*$([X_1, X_2, \ldots, X_n], P, \underline{M}, \overline{M}, \overline{\underline{M}}, S)$ *as a constraint satisfied if and only if*

$$P = |\{X_1, X_2, \ldots, X_n\}| \qquad S = \sum_{j \in \mathcal{X}} |\{i \mid X_i = j\}|^2 \tag{5}$$

$$\underline{M} = \min_{j \in \mathcal{X}} |\{i \mid X_i = j\}| \qquad \overline{M} = \max_{j \in \mathcal{X}} |\{i \mid X_i = j\}| \qquad \overline{\underline{M}} = \overline{M} - \underline{M} \tag{6}$$

Example 1 (Example for the Partition *Constraint).* Given the array $\mathcal{X} = [1, 5, 6, 1, 1, 1, 1, 1, \ 1, 1, 6]$, *Partition*$(\mathcal{X}, 3, 1, 8, 7, 69)$ holds, as $\mathcal{X}$ contains $P = 3$ distinct values, the least (resp. most) frequent one being value 5 (resp. 1) with $\underline{M} = 1$ (resp. $\overline{M} = 8$) occurrences, the difference between the occurrences of the most and the least frequent used values being $\overline{\underline{M}} = 7$, and the sum of the squares of the number of occurrences of distinct values being $S = 8^2 + 2^2 + 1^2 = 69$.

A Decomposition of the PARTITION *Constraint* PARTITION generalises NVALUE where $\mathcal{X}$ are the values, P is the number of values, and the remaining variables are ignored. As enforcing domain consistency on NVALUE [10] is NP-Hard, so is it for PARTITION. Therefore, we introduce the following decomposition named (`DECOMP_PART`). The PARTITION constraint can be encoded using the Global Cardinality (GCC) and the NVALUE constraints as follows. Constraint (9) is redundant, but improves filtering. Let v be an upper bound on the number of partitions. For instance, one can fix v to $\max_{i\in[1,n]} \max(\mathrm{dom}(X_i))$.

$$NValue(\mathcal{X}, P) \quad (7) \qquad GCC(\mathcal{X}, [O_1, O_2, \ldots, O_v]) \quad (8)$$

$$\sum_{j=1}^{v} O_j = n \ \wedge \ \sum_{i=1}^{n} X_i = \sum_{j=1}^{v} j \cdot O_j \quad (9) \qquad Min0([O_1, O_2, \ldots, O_v], \underline{M}) \quad (10)$$

$$\overline{M} = \max(O_1, \ldots, O_v) \ \wedge \ \overline{\underline{M}} = \overline{M} - \underline{M} \ \wedge \ S = \sum_{j=1}^{v} O_j^2 \quad (11)$$

The $Min0([O_1, O_2, \ldots, O_v], \underline{M})$ constraint holds for the variables $O_1, \ldots, O_v$ with domains in $[0, n]$ iff at least one variable O_i (with $i \in [1, v]$) has a value in $\mathbb{N}^+$, and $\underline{M}$ is the smallest such value. Introducing the auxiliary variables $A_0, A_1, \ldots, A_v$, $Min0([O_1, O_2, \ldots, O_v], \underline{M})$ can be reformulated as (i) $A_0 = n$, (ii) $\forall i \in [1, v] : O_i = 0 \Rightarrow A_i = A_{i-1}$, $O_i > 0 \wedge O_i \geq A_{i-1} \Rightarrow A_i = A_{i-1}$, $O_i > 0 \wedge O_i < A_{i-1} \Rightarrow A_i = O_i$, and (iii) $\underline{M} = A_v \ \wedge \ \underline{M} > 0$.

4.2 Defining the *BinSeq* Constraint

Consider an array $\mathcal{X} = [X_1, X_2, \ldots, X_n]$ of 0/1 integers, where a *stretch* is a subsequence of maximal length of successive 1, and an *inter-distance* is a subsequence of maximum length of successive 0 located between 2 consecutive stretches.

Definition 2. *We define* $BinSeq([X_1, X_2, \ldots, X_n], N_1, G, \underline{G}, \overline{G}, \underline{\overline{G}}, GS, \underline{D}, \overline{D}, \underline{\overline{D}}, DS)$ *as a constraint satisfied if and only if*

- N_1 *is the number of values* 1 *in the sequence,*
- G *is the number of stretches of 1 s,*
- $\underline{G}$ *(resp.* $\overline{G}$*) is the length of the smallest (resp. longest) stretch of 1 s,*
- $\underline{\overline{G}}$ *is the difference between the lengths of the longest and the smallest stretch,*
- GS *is the sum of the squared lengths of the stretches of 1 s,*
- $\underline{D}$ *(resp.* $\overline{D}$*) is the length of the smallest (resp. longest) inter-distance of 0 s,*
- $\underline{\overline{D}}$ *is the difference* $\overline{D} - \underline{D}$*,*
- DS *is the sum of the squared lengths of the inter-distances of 0 s.*

When there is no stretch, $\underline{G} = \overline{G} = 0$*; when there is no inter-distance,* $\underline{D} = \overline{D} = 0$.

Example 2 (Example for the BinSeq *Constraint).* Given the array $\mathcal{X} = [0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1]$, the $BinSeq(\mathcal{X}, 7, 3, 1, 4, 3, 21, 1, 2, 1, 5)$ constraint holds, as the array $\mathcal{X}$ contains $N_1 = 7$ occurrences of 1 s, $G = 3$ stretches of minimum (resp. maximum) length $\underline{G} = 1$ (resp. $\overline{G} = 4$) with $\overline{\underline{G}} = \overline{G} - \underline{G} = 3$, the sum of the squared lengths of the stretches $GS = 2^2 + 4^2 + 1^2 = 21$, the minimum (resp. maximum) inter-distance $\underline{D} = 1$ (resp. $\overline{D} = 2$) with $\overline{\underline{D}} = \overline{D} - \underline{D} = 1$, and the sum of the squared lengths of the inter-distances $DS = 2^2 + 1^2 = 5$.

Decomposing the BinSeq *Constraint.* For any $M \in \{N_1, G, \underline{G}, \overline{G}, \overline{\underline{G}}, GS, \underline{D}, \overline{D}, \overline{\underline{D}}\}$, an automaton with $O(n)$ states accepts the sequence $X_1, X_2, \ldots, X_n, M$. For $M = DS$, the automaton has $O(n^2)$ states. The conjunction of these automata has $O(n^{11})$ states, proving that domain consistency can be enforced on BinSeq in polynomial time using the Regular constraint. Due to such size, we created two decompositions of the BinSeq constraint for use in our experiments, which we describe briefly for space reasons. The first decomposition, named (`DECOMP_SEQ_GCC`), has the advantage of using standard constraints and can therefore be encoded in MiniZinc. The key idea is to associate a unique odd (or even) number with each stretch (or inter-distance) in the array $\mathcal{X}$. For instance, the array $\mathcal{X}$ = [0,0,0,1,1,0,0,1,1,1,1,0,1] from Example 2 is mapped to $\mathcal{X}' = [0, 0, 0, 1, 1, 2, 2, 3, 3, 3, 3, 4, 5]$. Then a Global Cardinality Constraint (GCC) on the array $\mathcal{X}'$ exposes the number of occurrences of each value in $\mathcal{X}'$. The occurrence variables of this GCC constraint can easily express the ten features of the BinSeq constraint: occurrences of even (resp. odd) values correspond to the lengths of stretches of 0 s (resp. 1 s). The 2nd decomposition, named (`DECOMP_SEQ_AUT`), is based on the SICStus register automaton constraint [6], which is not available in MiniZinc. For each feature f in the BinSeq constraint, we associate a register automaton with at most 5 states to link the array $\mathcal{X}$ to the feature f. When tested with SICStus, this 2nd decomposition proved to filter the BinSeq constraint variables much better than the first decomposition.

4.3 Challenge Behind the Partition and the BinSeq Constraints

The weakness of the decompositions (`DECOMP_PART`), (`DECOMP_SEQ_GCC`) and (`DECOMP_SEQ_AUT`) of the Partition and BinSeq constraints is that while most of the features of these two constraints are linked to the variables $X_1, X_2, \ldots, X_n$, there is virtually no direct link between the features, apart from the equalities $\overline{M} = \overline{M} - \underline{M}$, $\overline{G} = \overline{G} - \underline{G}$ and $\overline{D} = \overline{D} - \underline{D}$. As these features do not vary independently, this poses a challenge for getting an efficient filtering algorithm. To alleviate this problem, we show in Sect. 5 how to extract such missing links.

5 A Machine Assisted Generation of Implied Constraints with an Application to Partition and BinSeq

We use the framework presented in Sect. 3 to strengthen the decompositions provided for Partition and BinSeq. We apply Algorithm 2 to select, from

a set of conjectures generated by the BS for the partition and 0–1 sequence combinatorial objects, those that do not reduce the amount of filtering when all conjectures are used. We show how we generate the conjectures, how we use Algorithm 2 to select them, and we prove the most complicated selected conjecture.

Generating the Conjectures for Partition *and* BinSeq. The first step is to generate the conjectures using the BS. We provided instances of partitions and 0–1 sequences $I(\mathcal{O})$ of size $n \leq 30$ and reused CP models, which break symmetries where the different parts of a partition and the different stretch lengths are sorted by increasing size. For each object, we use all the features introduced in the Partition and BinSeq constraints as *primary features*. For the Partition constraint, we also manually introduced a set of *secondary features* at the partition object level, which the BS uses to systematically search for all the sharp bounds of the partition object's features, whereas for the BinSeq constraint *we did not introduce any secondary features to avoid any human assistance*. These instances of partitions (or 0–1 sequences) were sent to the BS, which returned 92 (or 3739) conjectures.

Generating Secondary Features for Partition. Most of these secondary features correspond to conditional expressions, where the value associated with the base case is either the constant 0, or is tied only to the size of the smallest partition. As the primary features focus on the largest and smallest partitions, the secondary features provide statistics on the other partitions. VV is the number of partitions, excluding the largest and smallest, and NN is the number of values that are not in the smallest or largest partition. A is the average size of the partitions that are neither the smallest nor the largest, rounded down. $A = 0$ if no such partition exists. SS is the sum of the squared sizes of the smallest and largest partitions. If there is only one partition, SS is its squared size.

$$VV = \begin{cases} P - 2 \text{ if } P > 1 \\ 0 \qquad \text{otherwise} \end{cases} \quad (12) \qquad NN = \begin{cases} n - \underline{M} - \overline{M} \text{ if } P > 1 \\ 0 \qquad\qquad\quad \text{otherwise} \end{cases} \quad (13)$$

$$A = \begin{cases} \lfloor \frac{NN}{VV} \rfloor \text{ if } P > 2 \\ 0 \qquad\quad \text{otherwise} \end{cases} \quad (14) \qquad SS = \begin{cases} \underline{M}^2 + \overline{M}^2 \text{ if } P > 1 \\ \underline{M}^2 \qquad\qquad\quad \text{otherwise} \end{cases} \quad (15)$$

R is the number of elements remaining after removing $\underline{M}$ elements from each partition. MID is, when the number of largest partitions is maximal, the size of an intermediate partition, either the one in between the smallest and largest, or the smallest, if no such partition exists. RR is the number of largest partitions or 0 if all partitions have the same size. SM is the gap between the squared sizes of the smallest and largest partitions. $SMIN$ is the sum of the squared sizes of partitions when they are all at the smallest size and with one partition excluded.

$$MID = \begin{cases} \underline{M} + (R \bmod \overline{M}) \text{ if } \overline{M} > 0 \\ \underline{M} \qquad\qquad\qquad \text{otherwise} \end{cases} \quad (16) \qquad R = n - P \cdot \underline{M} \quad (17)$$

$$RR = \begin{cases} \left\lfloor \frac{R}{\overline{M}} \right\rfloor \text{ if } \overline{M} > 0 \\ 0 \qquad \text{otherwise} \end{cases} \quad (18)$$

$$SM = \overline{M}^2 - \underline{M}^2 \quad (19)$$

$$SMIN = \underline{M}^2 \cdot (P-1) \quad (20)$$

Applying. Algorithm 2 *to* Partition *and* BinSeqWe implemented Algorithm 2 in SICStus using the clp(FD) solver on a Mac Studio M2 Ultra. We define $M(\mathcal{C})$ as the decomposition (`DECOMP_PART`) for Partition and (`DECOMP_SEQ_AUT`) for BinSeq. To limit the number of conjectures to be proved, we impose a limit of 20 on the number of conjectures returned by Algorithm 2, and run the selection algorithm on increasing value of n. We also stopped the search process when the same set of conjectures was selected for three consecutive values of n. Algorithm 2 selects 4 conjectures out of 92 with $n = 7$ for Partition in 0.5 sec and 17 conjectures out of 3728 with $n = 10$ for BinSeq in 150 min. Using an incremental version of the selection algorithm sketches at the end of Sect. 3.1, these two times were reduced to 0.1 sec and 17 min. We have proved all the selected conjectures listed in Table 1, and we now focus on the most interesting one for the BACP problem, which is a sharp lower bound on the sum of the squares of the partition sizes wrt the n, P, $\underline{M}$ and $\overline{M}$ features of the Partition constraint.

A Sharp Lower Bound for the Sum of the Squares of the Partition Sizes. Using the four secondary features introduced in (12)–(15), the BS returned the following conjecture for which we provide a proof and intuition.

$$S \geq -A^2 \cdot VV - A \cdot VV + 2 \cdot A \cdot NN + SS + NN \quad (21)$$

The intuition behind (21) is that, to achieve the smallest possible sum of squares, the largest partition has size $\overline{M}$, the smallest partition has size $\underline{M}$, and the other partitions have size NN/VV. But since the size must be an integer, NN mod VV partitions have their size rounded up to $A + 1$, while the rest have their size rounded down to A. To prove (21), we first need to introduce Theorem 1.

Theorem 1 (minimisation of $S = \sum_i^P y_i^2$). *Let $y_1, y_2, \cdots, y_P$ be positive integers whose sum is equal to n and which minimise $S = \sum_{i=1}^P y_i^2$. Then n mod P of these integers are equal to $\lfloor n/P \rfloor + 1$, and the others are all equal to $\lfloor n/P \rfloor$.*

Proof. $y_1, y_2, \cdots, y_P$ minimise $\sum_{i=1}^P y_i^2$ only if $y_i - y_j \leq 1$. Assume i and j with $y_i - y_j \geq 2$ then $(y_i - 1)^2 + (y_j + 1)^2 + \sum_{k \notin \{i,j\}}^P y_k^2 = \sum_k^P y_k^2 - 2(y_i - y_j) + 2 < y_i^2 + y_j^2 + \sum_{k \notin \{i,j\}}^P y_k^2 = \sum_k^P y_k^2$. As $(y_i - 1) + (y_j + 1) + \sum_{k \notin \{i,j\}}^P y_k = n$, the previous strict inequality shows that the decomposition $y_1 - 1, y_2 + 1, y_3 \cdots, y_P$

Table 1. Bounds selected by Algorithm 2 for (A) PARTITION and (B) BINSEQ

(A) $\overline{M} \leq \min(P \cdot \overline{M} - n, \overline{M} - 1)$ $\quad \underline{M} \leq n - P \cdot \underline{M}$

$S \geq -A^2 \cdot VV - A \cdot VV + 2 \cdot A \cdot NN + SS + NN$ (21) $\quad S \leq MID^2 + SM \cdot RR + SMIN$

(B) $N_I \leq \min(G \cdot \overline{G}, n - G + 1)$ $\quad \overline{G} \leq \begin{cases} n + \underline{G} & \text{if } \underline{G} = n \cdot \underline{D} \\ \left\lfloor \frac{n - \underline{G} - \underline{D} - \min(\underline{D}, 1) - 1}{\min(\underline{D}, 1) + 2} \right\rfloor + \underline{G} & \text{otherwise} \end{cases}$

$\overline{G} \geq \left\lfloor \frac{n}{n - N_I + 1} \right\rfloor$ $\quad \overline{D} \leq [\![G \geq 2]\!] \cdot (n - G \cdot \underline{G} - G + 2)$ $\quad GS \geq \underline{G}^2 \cdot G$

$\underline{D} \leq \begin{cases} 0 & \text{if } G \leq 1 \\ \left\lfloor \frac{n - \overline{G} + 1 - G}{G - 1} \right\rfloor & \text{if } G > 1 \end{cases}$ $\quad \overline{G} \leq \begin{cases} n & \text{if } G = 1 \wedge \overline{D} = 0 \\ \min(G, 1) & \text{if } G \neq 1 \wedge \overline{D} = 0 \\ n - \overline{D} - (G-2) \cdot \underline{D} - G + \min(G, 1) & \text{if } G \neq 1 \wedge \overline{D} \geq 1 \end{cases}$

$GS \geq \max(\overline{G}^2 + 1 - [\underline{D} = 0] - [\overline{G} = 0], 0)$ $\quad GS \leq \begin{cases} \max({N_I}^2 + G - 1, 0) & \text{if } G \leq 1 \\ \max((N_I - G + 1)^2 + G - 1, 0) & \text{otherwise} \end{cases}$

$GS \leq \begin{cases} \max({N_I}^2, 0) & \text{if } \underline{D} = 0 \wedge \min(N_I, 1) = 1 \\ 0 & \text{if } \underline{D} = 0 \wedge \min(N_I, 1) = 0 \\ \max((N_I - 2)^2 + 2, 0) & \text{if } \underline{D} \geq 1 \end{cases}$ $\quad DS \geq \underline{D}^2 \cdot (G - 1)$

$DS \geq \overline{D}^2$ $\quad DS \leq \begin{cases} 0 & \text{if } N_I \leq 1 \\ (n - N_I)^2 & \text{otherwise} \end{cases}$ $\quad DS \geq \begin{cases} 0 & \text{if } G \leq 1 \\ \max((\overline{D} + 1)^2 + G - 2, 0) & \text{otherwise} \end{cases}$

$DS \leq \begin{cases} \max((n - N_I - (G-2))^2 + G - 2, 0) & \text{if } G \geq 2 \\ \max(G - 2, 0) & \text{otherwise} \end{cases}$ $\quad GS \geq \overline{G} \cdot (\overline{G} + 1) \cdot \min(G, 1) + \overline{G} + G$

$GS \leq \begin{cases} \max(n^2, 0) & \text{if } G = 1 \wedge \overline{D} = 0 \\ \max((\min(G, 1))^2 + G - 1, 0) & \text{if } G \neq 1 \wedge \overline{D} = 0 \\ \max((n - \overline{D} - (G-2) \cdot \underline{D} - G + 1)^2 + G - 1, 0) & \text{if } G \neq 1 \wedge \overline{D} \geq 1 \end{cases}$

minimises better $\sum_{i=1}^{P} y_i^2$ than the decomposition $y_1, y_2, \cdots, y_P$, a contradiction. The only way to have $y_i - y_j \leq 1$ and $\sum_i^P y_i = n$ is to set the values of the integers, as stated by Theorem 1, otherwise, we always have $\sum_{k=1}^{P} y_k \neq n$. □

Proof (Conjecture (21)). If $P = 1$, substituting (12) to (15) into (21) simplifies to $S \geq n^2$ which is tight and consistent with the definition *Partition*. If $P = 2$,

we have $n = \overline{M} + \underline{M}$. By substituting (12), Conjecture (21) is simplified by $S \geq \overline{M}^2 + \underline{M}^2$, which is also tight and consistent. If $P > 2$, then $A = \lfloor \frac{NN}{VV} \rfloor$ leading to $NN = A \cdot VV + NN \bmod VV$. After substituting NN by $A \cdot VV + NN \bmod VV$ inside Conjecture (21) we have $S \geq -A^2 \cdot VV + 2 \cdot A^2 \cdot VV + 2 \cdot A \cdot (NN \bmod VV) + SS + NN \bmod VV$ which simplifies to $S \geq A^2 \cdot VV + (2 \cdot A + 1) \cdot (NN \bmod VV) + SS$. By substituting $(2 \cdot A + 1)$ by $(A + 1)^2 - A^2$, and SS by $\overline{M}^2 + \underline{M}^2$, we obtain this inequality that is proved by Theorem 1: $S \geq \overline{M}^2 + \underline{M}^2 + (A + 1)^2 \cdot (NN \bmod VV) + A^2 \cdot (VV - (NN \bmod VV))$ □

6 Experiments on the BACP and the BSSP Problems

We test our framework on the Balanced Academic Curriculum Problem (BACP) [22] and the Balanced Shift-Scheduling Problem [15]. These problems involve combinatorial objects, such as partitions or binary sequences, where multiple features are restricted, making sharp bounds essential. We describe the BACP and the two models we use for it. We outline the BSSP and the two models we developed for it. We evaluate these models with and without using the conjectures selected by Algorithm 2 on the Chuffed [19] and SICStus clp(FD) [13] solvers.

6.1 Describing the BACP Problem and Its Models

Problem Description. The BACP is a problem in which a set of n courses must be assigned to v periods. Let X_i denote the period of course i. A lower and upper limit $\underline{m}$ and limit $\overline{m}$ for the number of courses per period must be respected. Let $\mathcal{P}rec$ be the set of pairs of courses (i, k) such that i is the prerequisite for k, and let (i) be the number of credits for course i. The *load*, i.e. the sum of the credits of the assigned courses, for each period, must be in $[\underline{L}_0, \overline{L}_0]$. The course load must be balanced over the different periods. There are many ways to balance the load. One can minimise the maximum load or the load range. To distribute the load more evenly, we minimise the sum of the squares of the loads.

Description of the Models. The first model, which we call (`BACP_PART`), contains the constraints (22) to (30). The array of variables $[X_1, X_2, \ldots, X_n]$ partitions the courses $1 \ldots n$ into time periods $1 \ldots v$. The constraints (23), (25), (26), and (27) enforce a lower and upper limit on the number of courses taught per period. The constraint (28) enforces precedences between the corresponding courses i and k. The *Partition* constraint (31) imposes that the parameter S is the sum of the squares for the load of each time period. The array $[\underbrace{X_1, \ldots, X_1}_{c(1) \text{ times}}, \ldots, \underbrace{X_n, \ldots, X_n}_{c(n) \text{ times}}]$ contains $c(i)$ occurrences of variable X_i for each course i. One occurrence of a value of X_i represents one credit of the course i. As X_i takes a value of a time period, the number of occurrences of a time period j in the array is the sum of credits of all courses for that time period, i.e. the load

for the period j is $\sum_{i=1}^{n}[X_i = j] \cdot c(i)$. Constraint (31) also ensures that $\overline{\underline{L}}$ is the range between the minimal and maximal load $\underline{L}$ and $\overline{L}$. Constraint (29) ensures the respect of the lower and upper limits of the load per period. (30) define the initial domains of the variables X_i and ensure that the time periods that are used are contiguous. The model minimises S for a better balance of load among periods, rather than minimising $\overline{\underline{L}}$ or $\overline{L}$, as it is usually done.

$$\text{Minimize } S \tag{22}$$

$$GCC([X_1, X_2, \ldots, X_n], [M_1, \ldots, M_v]) \tag{23}$$

$$\sum_{j=1}^{v} M_j = n \ \wedge \ \sum_{i=1}^{n} X_i = \sum_{j=1}^{v} j \cdot M_j \tag{24}$$

$$\underline{M} = \min(M_1, \ldots, M_v) \tag{25}$$

$$\overline{M} = \max(M_1, \ldots, M_v) \tag{26}$$

$$\underline{m} \leq \underline{M} \ \wedge \ \overline{M} \leq \overline{m} \tag{27}$$

$$\forall (i,k) \in \mathcal{P}rec, X_i < X_k \tag{28}$$

$$\underline{L}_0 \leq \underline{L} \ \wedge \ \overline{L} \leq \overline{L}_0 \tag{29}$$

$$\forall i \in [1,n], 1 \leq X_i \leq v \wedge X_i \leq P \tag{30}$$

$$Partition([\underbrace{X_1, \ldots, X_1}_{c(1) \text{ times}}, \ldots, \underbrace{X_n, \ldots, X_n}_{c(n) \text{ times}}], P, \underline{L}, \overline{L}, \overline{\underline{L}}, S) \tag{31}$$

A second model, we call (`BACP_CUM`), by Mats Carlsson [22], uses constraints (22) to (28) and (30) and encodes the rest using a CUMULATIVE constraint.

6.2 Describing the BSSP Problem and Its Models

Problem Description: We present the Balanced Shift-Scheduling Problem (BSSP) to optimise employee schedules, balancing work and rest periods, with no requirement for equal number of rests per employee. There is only one activity on which all employees work on. An employee has lunch and has rests between periods of work. An employee rests for at least four hr at the beginning and end of a day, with work periods ranging from 1 to 4 hr between rests. We divide a day into 96 slots of 15 mins. When the rest is between periods of work, it is at least 15 mins. long and not more than an hr long. Lunch is one hr long. An employee should have at least three rest periods during the workday, which should be between 6 and 8 hr. At time i, there should be m_i employees working. Let M_i be the actual number of employees working at time i. There is a penalty cost $\overline{p}_i$ associated with overemployment and a penalty cost $\underline{p}_i$ associated with underemployment. The total penalty cost is $\sum_{i=1}^{96}(M_i > m_i) \cdot \overline{p}_i + (M_i < m_i) \cdot \underline{p}_i$. As we seek the

most balanced work-rest distribution, we add to that penalty cost the sum of squared work periods GS_e and rest periods DS_e for each employee e.

Models Description: We encode an employee schedule with an array $\mathcal{X} = [X_1, \ldots, X_n]$ of 0/1 integers, where 1 represents a 15-min. work period and 0 a 15-min. rest period; e.g., [000111111111111001111000011101111011110111000] means that the employee starts with a 45-min. rest, works for 3 hr, takes a 30-min. break, and then works 1 hr before lunch. After lunch, s/he finishes his day with 4 hr of work, separated by 15-min. breaks. We are interested by the following features of $\mathcal{X}$: its size n, G the number of stretches of 1, i.e. the number of periods of work, N_1 the number of 1 in the array, i.e. the total work time, $\underline{G}$ ($\overline{G}$) the smallest (largest) length of a stretch of 1, i.e. the shortest (longest) period of work, $\overline{G}$ the difference $\overline{G} - \underline{G}$, GS the sum of the squares of the stretch lengths, $\underline{D}$ ($\overline{D}$) the smallest (largest) inter-distance between consecutive stretches of 1, i.e. the smallest (largest) rest between work periods, $\overline{D}$ the difference $\overline{D} - \underline{D}$, DS the sum of the squares of the inter-distance lengths.

We encode the BSSP using the BinSeq constraint. Let m be the maximum number of employees required in a day. Constraint (32) ensures balanced schedules and the lowest penalty cost. (33) connects the decision variables $\mathcal{X}_e = [X_{1\,e}, \ldots, X_{n\,e}]$ to $n, N_1, G, \underline{G}, \overline{G}, \overline{\underline{G}}, GS, \underline{D}, \overline{D}, \overline{\underline{D}}, DS$. The variable $X_{i\,e}$ is 1 iff employee e works at time i. (34) enforces employees to take at least 4 breaks of at least 15 min. and no more than 1 hr, including 1 lunch break of exactly 1 hr. (35) catches the number of employees working in each time slot. (36) ensures that an employee works between 6 and 8 hr and between 1 hr and 4 hr before a rest. As employees are required to have at least 4 hr of rest at the beginning and end of each workday, we exclude the first and last 16 time slots.

$$\text{Minimize} \sum_{e=1}^{m} GS_e + DS_e + \sum_{i=1}^{96} (M_i > m_i) \cdot \overline{p}_i + (M_i < m_i) \cdot \underline{p}_i \quad (32)$$

$$\forall e \in [1:m], BinSeq(\mathcal{X}_e, N_{1\,e}, G_e, \underline{G}_e, \overline{G}_e, \overline{\underline{G}}_e, GS_e, \underline{D}_e, \overline{D}_e, \overline{\underline{D}}_e, DS_e) \quad (33)$$

$$1 \leq \underline{D}_e \wedge \overline{D}_e = 4 \wedge 4 \leq G_e \quad (34) \qquad \forall i \in [1:64], M_i = \sum_{e=1}^{m} X_{ie} \quad (35)$$

$$24 \leq N_{1\,e} \leq 32 \ \wedge 4 \leq \underline{G}_e \wedge \overline{G}_e \leq 16 \quad (36)$$

Using the (`DECOMP_SEQ_GCC`) and (`DECOMP_SEQ_AUT`) decompositions of the BinSeq constraint, we obtain the two models (`BSSP_GCC`) and (`BSSP_AUT`).

6.3 Evaluation

Settings for the BACP. We evaluate the models on 31 real-world instances with up to 10 periods and 42–66 courses, all from CSPLib [1] with SICStus and Chuffed. For SICStus, we use two strategies: (*i*) When the learnt bounds are unused, the 1st strategy selects the most constrained variable from $\mathcal{X} = X_1, \ldots, X_n$, and performs a dichotomic search. (*ii*) The 2nd strategy branches first on $\overline{L}, S, P, \underline{L}, \overline{L}$, then on $\mathcal{X}$ using the 1st strategy, prioritising feature variables that makes the objective smaller. As it is more aggressive, we use it only with models that include bounds. For Chuffed, using the free search, the search branches on $\overline{L}, S, P, \underline{L}, \overline{L}$, and chooses the values in ascending order. Then, it branches on $\mathcal{X}$ in that order and chooses the values in descending order.

Settings for the BSSP. We evaluate the models on 10 real instances with 2–7 employees. For SICStus, we use an aggressive strategy that, after restricting $\overline{G}$, fixes the variables by increasing time slot i, restricting first the covering cost $(M_i > m_i) \cdot \overline{p}_i + (M_i < m_i) \cdot \underline{p}_i$, and then the variables X_{ie} for time slot i. For Chuffed in free search, we enumerate on variables X_{ie} by increasing time slot i.

Results. We generate the implied constraints only once, not for each instance. This process is roughly equivalent to a human designing and programming a filtering algorithm. It took the BS about one week on the Digital Research of Alliance of Canada clusters to generate all conjectures. We used a single core of a Mac Studio M2 Ultra with a fifteen-min. timeout. For each model and solver, we report the number of instances where the solver (*i*) proves optimality (#optimality), (*ii*) the timeout (#timeout), (*iii*) finds no solution (#not found), and (*iv*) the number of instances for which a model is the best (#best) for the corresponding solver. The best models are those proving optimality faster. If none do, we select the one with the lowest objective value in the least time. We also provide the sum of objective values ($\sum$obj) and the sum of times spent ($\sum$time) to find the best solution for all instances where a solution was found. Table 2 gives the aggregated results. For the BACP, none of the models proves optimality without using the bounds. When using bounds, although (`BACP_PART`) proves optimality for 12 instances with SICStus, it proves optimality for all instances with Chuffed.[1] The fastest model is (`BACP_PART`) with 222359 as the smallest sum of objective values found in 5min for all the instances when using bounds with Chuffed.[2] For the BSSP, Table 2 shows that the models find better solutions and are faster when using bounds. The fastest model is (`BSSP_GCC`) with 71 min as the total time on the 10 instances when using bounds, meaning that when using bounds, it is 1.14 times faster than the same model without using bounds. The sum of objective values 7005592 is large due to one instance where using bounds degraded the results. But for the other instances, the (`BSSP_GCC`) is six

[1] The 19 instances where SICStus with (`BACP_PART`) did not find any solution when using bounds is due to the aggressive search strategy quoted earliest.

[2] The incremental version of Algorithm 2 and all conjecture proofs are in an arXiv report.

Table 2. Results of the experiments where the best results are shown in bold

Problem	BACP								BSSP			
Solver	SICStus				Chuffed				SICStus		Chuffed	
Model	(BACP_PART)		(BACP_CUM)		(BACP_PART)		(BACP_CUM)		(BSSP_AUT)		(BSSP_GCC)	
Used bounds	no	yes	no	yes	no	yes	no	yes	no	yes	no	yes
#optimality	0	12	0	30	0	**31**	0	30	0	0	**6**	**6**
#timeout	31	17	31	1	31	**0**	31	1	10	10	**4**	**4**
#not found	2	19	2	**0**	**0**	**0**	**0**	**0**	5	**0**	**0**	**0**
#best	0	0	0	**19**	0	10	0	2	0	3	1	**6**
∑obj	/	/	/	**222359**	225375	**222359**	232769	222529	/	5697	**5524**	7005592
∑time	/	/	/	17min	5h	**5min**	5h	29min	/	137min	81min	**71min**

times the best among all models when using bounds. Although (BSSP_AUT) never proved optimality, using bounds, it found the best solution three times, but failed to find any solution for five instances without exploiting bounds.

7 Conclusion

Bound Seeker-generated conjectures can lead to better filtering for problems with multiple features of a combinatorial object. We synthesise redundant constraints supported by various solvers. Our algorithm addresses two major issues: the impossibility of proving thousands of conjectures produced by the BS and the need to identify impactful conjectures to improve filtering.

References

1. CSPLib: A problem library for constraints. https://www.csplib.org/. Accessed 06 Dec 2024
2. Achterberg, T., Bixby, R.E., Gu, Z., Rothberg, E., Weninger, D.: Presolve reductions in mixed integer programming. INFORMS J. Comput. **32**(2), 473–506 (2020). https://doi.org/10.1287/IJOC.2018.0857
3. Aouchiche, M., Caporossi, G., Hansen, P., Laffay, M.: Autographix: a survey. Electron. Notes Discret. Math. **22**, 515–520 (2005). https://doi.org/10.1016/j.endm.2005.06.090
4. Arafailova, E., Beldiceanu, N., Simonis, H.: Deriving generic bounds for time-series constraints based on regular expressions characteristics. Constraints Int. J. **23**(1), 44–86 (2018). https://doi.org/10.1007/s10601-017-9276-z
5. Baptiste, P., Pape, C.L., Nuijten, W.: Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems. Kluwer (2012)

6. Beldiceanu, N., Carlsson, M., Petit, T.: Deriving filtering algorithms from constraint checkers. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 107–122. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30201-8_11
7. Beldiceanu, N., Carlsson, M., Rampon, J.X.: Global constraint catalog, (revision a) (2012)
8. Beldiceanu, N., Cheukam-Ngouonou, J., Douence, R., Gindullin, R., Quimper, C.: Acquiring maps of interrelated conjectures on sharp bounds. In: Solnon, C. (ed.) 28th International Conference on Principles and Practice of Constraint Programming, CP 2022, July 31 to August 8, 2022, Haifa, Israel. LIPIcs, vol. 235, pp. 6:1–6:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022). https://doi.org/10.4230/LIPIcs.CP.2022.6
9. Beldiceanu, N., Contejean, É.: Introducing global constraints in chip. Math. Comput. Model. **20**(12), 97–123 (1994). https://doi.org/10.1016/0895-7177(94)90127-9
10. Bessiere, C., Hebrard, E., Hnich, B., Kiziltan, Z., Walsh, T.: Filtering algorithms for the NVALUE constraint. In: Barták, R., Milano, M. (eds.) CPAIOR 2005. LNCS, vol. 3524, pp. 79–93. Springer, Heidelberg (2005). https://doi.org/10.1007/11493853_8
11. Bessiere, C., et al.: The balance constraint family. In: O'Sullivan, B. (ed.) CP 2014. LNCS, vol. 8656, pp. 174–189. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10428-7_15
12. Biere, A., Järvisalo, M., Kiesl, B.: Preprocessing in SAT solving. In: Handbook of Satisfiability, pp. 391–435. IOS Press (2021)
13. Carlsson, M., Mildner, P.: Sicstus prolog – the first 25 years. CoRR abs/1011.5640 (2010). http://arxiv.org/abs/1011.5640
14. Charnley, J.W., Colton, S., Miguel, I.: Automatic generation of implied constraints. In: Brewka, G., Coradeschi, S., Perini, A., Traverso, P. (eds.) ECAI 2006, 17th European Conference on Artificial Intelligence, August 29 - September 1, 2006, Riva del Garda, Italy, Including Prestigious Applications of Intelligent Systems (PAIS 2006), Proceedings. Frontiers in Artificial Intelligence and Applications, vol. 141, pp. 73–77. IOS Press (2006). http://www.booksonline.iospress.nl/Content/View.aspx?piid=1649
15. Demassey, S., Pesant, G., Rousseau, L.M.: A cost-regular based hybrid column generation approach. Constraints **11**(4), 315–333 (2006). https://doi.org/10.1007/s10601-006-9003-7. https://publications.polymtl.ca/23185/, aRRAY(0x55680943b278)
16. Fajtlowicz, S.: On conjectures of Graffiti. Discret. Math. **72**(1–3), 113–118 (1988). https://doi.org/10.1016/0012-365X(88)90199-9
17. Gent, I.P., Jefferson, C., Linton, S., Miguel, I., Nightingale, P.: Generating custom propagators for arbitrary constraints. Artif. Intell. **211**, 1–33 (2014). https://doi.org/10.1016/J.ARTINT.2014.03.001
18. Gent, I.P., Jefferson, C., Miguel, I., Nightingale, P.: Generating special-purpose stateless propagators for arbitrary constraints. In: Cohen, D. (ed.) CP 2010. LNCS, vol. 6308, pp. 206–220. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15396-9_19
19. Chu, G., Stuckey, P.J., Schutt, A., Ehlers, T., Gange, G., Francis, K.: Chuffed 0.12.1, a lazy clause generation solver (2023). https://github.com/chuffed/chuffed
20. Gindullin, R., Beldiceanu, N., Ngouonou, J.C., Douence, R., Quimper, C.G.: Boolean-arithmetic equations: acquisition and uses. In: Proceedings of the 20th International Conference on the Integration of Constraint Programming (CPAIOR) (2023)

21. Hansen, P., Caporossi, G.: Autographix: an automated system for finding conjectures in graph theory. Electron. Notes Discret. Math. **5**, 158–161 (2000). https://doi.org/10.1016/S1571-0653(05)80151-9
22. Hnich, B., Kiziltan, Z., Walsh, T.: CSPLib problem 030: balanced academic curriculum problem (BACP) (1999). http://www.csplib.org/Problems/prob030
23. Laurière, J.: Constraint propagation or automatic programming. Technical report. 19, IBP-Laforia (1996). https://www.lri.fr/~sebag/Slides/Lauriere/Rabbit.pdf
24. Pachet, F., Roy, P.: Automatic generation of music programs. In: Jaffar, J. (ed.) CP 1999. LNCS, vol. 1713, pp. 331–345. Springer, Heidelberg (1999). https://doi.org/10.1007/978-3-540-48085-3_24
25. Pesant, G.: A filtering algorithm for the stretch constraint. In: Walsh, T. (ed.) CP 2001. LNCS, vol. 2239, pp. 183–195. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45578-7_13
26. Pesant, G., Régin, J.C.: Spread: a balancing constraint based on statistics. In: Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP 2005), pp. 460–474 (2005)
27. Schaus, P., Deville, Y., Dupont, P.: Bound-consistent deviation constraint. In: Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP 2007), pp. 620–634 (2007)
28. Rendl, A., Miguel, I., Gent, I.P., Gregory, P.: Common subexpressions in constraint models of planning problems. In: Bulitko, V., Beck, J.C. (eds.) Eighth Symposium on Abstraction, Reformulation, and Approximation, SARA 2009, Lake Arrowhead, California, USA, 8–10 August 2009. AAAI (2009). http://www.aaai.org/ocs/index.php/SARA/SARA09/paper/view/823
29. Régin, J.C.: Modélisation et Contraintes Globales en Programmation par Contraintes. HDR dissertation, Université de Nice (2004)
30. Schaus, P., Deville, Y., Dupont, P., Régin, J.-C.: The deviation constraint. In: Van Hentenryck, P., Wolsey, L. (eds.) CPAIOR 2007. LNCS, vol. 4510, pp. 260–274. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72397-4_19
31. Schaus, P., Deville, Y., Dupout, P., Régin, J.C.: Simplification and extension of the spread constraint. In: Proceedings of the Third International Workshop on Constraint Propagation and Implementation, pp. 77–91 (2006)

Integer and Constraint Programming for the Offline Nanosatellite Partition Scheduling Problem

Julien Rouzot[1,2(✉)], Mickaël Pereira[1(✉)], Christian Artigues[1(✉)], Romain Boyer[1(✉)], Frédéric Camps[1(✉)], Philippe Garnier[1,2(✉)], Emmanuel Hebrard[1(✉)], and Pierre Lopez[1(✉)]

[1] LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France
{julien.rouzot,mickael.pereira,christian.artigues,romain.boyer,frederic.camps,philippe.garnier,emmanuel.hebrard,pierre.lopez}@laas.fr
[2] IRAP, Université de Toulouse, CNRS, UT3, CNES, Toulouse, France
{julien.rouzot,philippe.garnier}@irap.omp.eu

Abstract. Effective scheduling of tasks for nanosatellites is essential, given their limited onboard resources and capabilities. In a typical nanosatellite mission, the onboard computer has to run payload tasks linked to the mission objective, such as observations and measurements for science campaigns, but also communication and avionic tasks needed for navigation and safety. We consider a partitioned real-time system where a partition is a job that embeds a set of elementary tasks implementing one of the above-described functions. The offline nanosatellite partition scheduling problem consists in scheduling a set of partitions on a single-core processor within a fixed time frame that will be repeated cyclically, while maximizing the number and duration of scheduled payload partitions. The paper first establishes similarities and differences with related scheduling problems. We then prove that the problem is strongly NP-hard. A Mixed Integer Linear Programming (MILP) matheuristic and a Constraint Programming (CP) model are proposed. We compare the MILP and CP approaches in a multi-objective context and demonstrate the relevance of the latter to solve the problem efficiently both on randomly generated instances and on a real nanosatellite case study of the University Space Center of Toulouse: the NIMPH project. The CP model is embedded in NanoSatScheduler, an open source software with a user interface designed for the offline nanosatellite partition scheduling problem. For the NIMPH real-case study, the proposed solution outperforms the semi-manual approach used so far. As a result, the NIMPH team has adopted NanoSatScheduler as an operational tool for their mission.

Keywords: Constraint Programming · Mixed Integer Linear Programming · Nanosatellites · Offline Partition Scheduling · Periodic Scheduling

G. Tack (Ed.): CPAIOR 2025, LNCS 15763, pp. 173–190, 2025.
https://doi.org/10.1007/978-3-031-95976-9_11

1 Introduction

In scientific nanosatellite missions, the onboard computer has to orchestrate scientific and avionic tasks related to the mission objective and the proper functioning of the nanosatellite. We consider the case where isolated threads, also called partitions, are run on a single-core processor. Each partition encapsulates a segment of the application code. The principle of a partitioned system ensures space and time isolation of applications with different levels of criticality [27]. Typically, some partitions correspond to payload tasks with low criticality but high interest for the mission, while other partitions correspond to avionic tasks with high criticality for nanosatellite safety. In this paper, we introduce the offline nanosatellite partition scheduling problem (ONPSP) and we refer to the partitions as *jobs* for a better match with the scheduling literature. These jobs are decomposed into elementary tasks of variable duration. The schedule is built on a short time frame called the cycle time (e.g. 1 s) and is repeated indefinitely until further notice. There are various intra- and inter-job constraints that complicate the scheduling process. The time lag between two executions of tasks of the same job can be constrained to have a maximum value. The latter constraint is used to ensure the regularity of critical tasks within the time frame. There are also minimum time-lag constraints within the tasks of the same job, which represent a minimum amount of time required between two executions of a payload task, that can be needed to save data, resetting an instrument between two observations or just to spread the scientific observations over time. There are also precedence constraints between tasks of different jobs, to represent a data flow between tasks. Finally, some jobs may have a set of fixed times at which a task must start, allowing the user to build upon an existing incomplete plan.

Although the models proposed in this paper are valid for many onboard partitioned systems, our work takes place in a practical study, the Nanosatellite to Investigate Microwave Photonics Hardware (NIMPH) mission, to be launched in 2025 by the University Space Center of Toulouse (Centre Spatial Universitaire de Toulouse, CSUT)[1]. This mission is an academic endeavor that is part of the larger NanoLab Academy project launched by the French National Center for Space Studies (Centre National d'Etudes Spatiales, CNES) in an effort to bring students to develop and exploit their own spacecraft to aid in novel scientific experiments. The NIMPH main mission is designed to prove the viability of microwave photonics hardware, i.e. a new kind of radio frequency transmitter that merges fiber optic and photonic devices, in space. The NIMPH nanosatellite is based on the CubeSat architecture [10] and the tasks to be performed must be scheduled on a single-core processor through the hypervisor XTratuM [15], used for all NanoLab Academy missions.

Several objectives can be considered depending on the mission priorities. In [21], we proposed a method to solve a variant of the problem aimed at minimizing the context switches [6], i.e. the number of times the processor switches

[1] https://www.csut.cnrs.fr/en/project/nanosatellite-to-investigate-microwave-photonics-hardware/.

from one job to another. The number of tasks and their durations were considered as fixed parameters for each job, the fixed start times were not considered and the problem complexity was left open. With the evolution of the real project, handling fixed start times and maximizing the number and duration of science jobs have become strict necessities, while context switches became of secondary importance and are thus ignored in this study.

We propose a MILP-based matheuristic that adapts the MILP proposed in [18] for a related nanosatellite scheduling problem and a CP model. Both aim at generating an optimized schedule based on user-defined multiple objectives, considering the number of task occurrences and durations across different partitions. The CP model is integrated into an open-source software called NanoSatScheduler that includes a user interface where the user can create/modify instances and review/modify the solutions provided by the solver.

The remainder of the paper is organized as follows. In Sect. 2, we define the problem and its objectives. In Sect. 3, the related work in the literature is briefly presented. In particular, the similarities and differences with the previously considered offline nanosatellite task scheduling problem, and other (partition) scheduling problems are discussed. Section 4 establishes that the ONPSP is strongly NP-hard. The MILP model and the matheuristic are detailed in Sect. 5. The CP model and the user interface embedded in NanoSatScheduler are presented in Sect. 6. Section 7 provides computational experiments. The two approaches are compared on randomly generated instances and on the real-case study. Conclusions are drawn in Sect. 8.

2 Problem Definition

In the ONPSP under consideration, each job $i \in \mathcal{J} = \{i = 1, \ldots, n\}$ consists of elementary tasks. For each job, there is a minimum number of tasks $\underline{m}_i$ and a maximum number of tasks $\overline{m}_i$ such that the actual number of scheduled tasks $m_i \in [\underline{m}_i, \overline{m}_i]$ is a decision variable. Set $\mathcal{M}_i = \{1, \ldots, \overline{m}_i\}$, denotes the indices of all potential tasks of job $i \in \mathcal{J}$ and once m_i is chosen, the scheduled tasks are assumed to have indices $\{1, \ldots, m_i\}$. All scheduled tasks must not overlap, as they run on a single-core processor, and within the time frame $[0, h]$, where h is the cycle time. Each task in each job i has a minimum duration $\underline{p}_i$ and a maximum duration $\overline{p}_i$ and the actual duration $p_{i,j} \in [\underline{p}_i, \overline{p}_i]$ of a scheduled task j of job i is also a decision variable. The start time of each scheduled task and the start time of its immediate successor within the job, if it is scheduled, must be separated by at least lag_i^{min} time units (minimum time lag). Additionally, the end time of each scheduled task and the start time of its successor, if it is scheduled, must be separated by no more than lag_i^{max} time units (maximum time lag)[2]. Due to periodicity, the time-lag constraints have also to be enforced between the last scheduled task of the job and the first scheduled task of the job. More precisely, if the first task of job i starts at time t, its last task starts at time

[2] Modeling start-to-start and end-to-start time lags is a request from the NIMPH team, but this can be easily modified for other nanosatellite missions.

t'_{start} and ends at t'_{end}, then $h - t'_{start} + t \geq lag_i^{min}$ and $h - t'_{end} + t \leq lag_i^{max}$. For each job i, we have a set of Q_i fixed start times $T_i = \{t_{i,1}, \ldots, t_{i,Q_i}\}$. There must be exactly one task of job i scheduled to start at each of the fixed start times in T_i. Moreover, there is a set E of precedence constraints between different jobs. A precedence constraint $(i, i') \in E$ states that if task (i', j) is scheduled, then task (i, j) should be scheduled before it in the time frame.

Depending on the mission contexts, the effort of one or more jobs must be optimized, either by maximizing the number of tasks or the overall duration of the corresponding job. Those multiple – and conflicting – objectives are user defined. The weights u_i and v_i respectively define the importance of maximizing the number of tasks and total duration for each job, thus allowing users to finely tune their objective. The objective function of the ONPSP is given by (1). We provide a solution for an ONPSP instance as an illustrative example of the problem in Fig. 1.

$$\sum_{i=1}^{n} \sum_{j=1}^{m_i} u_i + v_i p_{i,j} \tag{1}$$

Fig. 1. In this toy example, we have 3 jobs (A,B,C), with $\underline{m}_A = 1$, $\overline{m}_A = 3$; $\underline{m}_B = \overline{m}_B = 1$; $\underline{m}_C = \overline{m}_C = 2$. Job A is the only one with variable duration, $\underline{p}_A = 2$, $\overline{p}_A = 4$. The only precedence constraint is (A, B), so at least k tasks of job A must occur before the k-th task of job B. We have $lag_A^{min} = 5$ and $lag_C^{max} = 5$. $T_B = \{2\}$, so the only task of job B must start at $t = 2$. In this example, the objective is to maximize the number of tasks for job A first ($u_A = M$, with M a large number), then we want to maximize the total duration for A ($v_A = 1$). For the other jobs $i \neq A$, we have $u_i = v_i = 0$.

3 Related Works

The literature on scheduling for nanosatellites is divided between online (onboard) and offline scheduling methods, such as in our study.

Although Rigo et al. [19] claim that online scheduling is not adapted to CubeSat nanosatellites due to their limited onboard capacity and the need to get close to optimal solutions, online scheduling approaches have recently been investigated, in particular to evaluate online task scheduling strategies aimed at minimizing power consumption [3,7,12–14,24,25,28].

The literature on nanosatellite offline scheduling [4,18–20,23] focuses on mathematical programming approaches for the offline nanosatellite task scheduling problem (ONTSP). The ONTSP considers in the same model all kinds of tasks performed by the nanosatellite: payload, communication, or operating system tasks. The goal is to schedule them on a much larger time horizon than

in our problem, corresponding to at least an orbit (about 100 min) with a time discretization of 1 min or 30 sec. A task must be scheduled within a time window and can be started and stopped as required, but with a minimum and maximum number of startups until the required duration reaches a value in a predefined interval and the tasks can be performed in parallel. A limitation comes from energy requirements, and the required power cannot exceed a given threshold to save energy. The ONTSP can be viewed as a coarse-grained scheduling problem for the satellite mission, while the ONPSP is a fine-grained scheduling problem that can be seen as a lower-level problem whose solution is needed to implement the high-level plan defined by the ONTSP solution.

The ONPSP is close to other offline (partition) scheduling problems in real-time systems encountered in the literature. However, many of the proposed methods, including integer and constraint programming, deal with strictly periodic scheduling on multi-core processors [1,2,5,11,16,22]. The ONPSP involves a single-core processor and minimum/maximum time lags offer more flexibility than strict periodicity.

In the general scheduling literature, there are related single-machine problems, especially the single-machine problem with minimum and maximum time lags that is NP-hard in the general case. In [26], several complexity results have been established, but our problem appears as a special case due to the specific time lag between the last and first scheduled tasks of each job, as well as the presence of fixed start times. Similarly, our problem is a special case of NP-hard cyclic scheduling problems [9], since there is no overlap of task instances between two time frames. A specific study is therefore needed to formally establish the complexity of the problem.

4 Problem Complexity

We establish the complexity of the ONPSP when the number of tasks for each job i is fixed ($m_i = \underline{m}_i = \overline{m}_i$) and the duration of each task of each job i is also fixed ($p_{i,j} = \underline{p}_i = \overline{p}_i$), the weight of the job in terms of number of scheduled tasks is $u_i = 1$ and its weight concerning its total scheduled duration is $v_i = 0$. There are no precedence constraints ($E = \emptyset$). The considered decision variant of the problem asks whether we can find a schedule with the objective $\sum_{i=1}^{n} m_i$, i.e. whether we can schedule all tasks in the cycle time h.

The NP-hardness proof of the ONPSP follows from straightforward reductions from 3-Partition (see Figs. 2 and 3). The 3-Partition problem asks whether a set of $3m$ values $\{e_k \mid k = 1, \ldots, 3m\}$ of total value $\sum_{k=1}^{3m} e_k = mQ$ with $m \geq 1$ integer and $\frac{Q}{4} < e_k < \frac{Q}{2}$ can be partitioned into a set of m triplets, each of total value Q. The following theorems show that the problem difficulty comes from the fixed start times and the minimum/maximum time lags, even considered separately.

Theorem 1. *The ONPSP is strongly NP-hard, even if* $m_i = 1$, $lag_i^{min} = 0$ *and* $lag_i^{max} \geq h$, $\forall i = 1, \ldots, n$.

Proof. Consider the 3-Partition problem from which we build an instance of the ONPSP as follows. Let $h = m(Q+1) - 1$. We have $3m+1$ jobs where for $i = 1, \ldots, 3m$, $m_i = 1$, $lag_i^{min} = 0$, $lag_i^{max} = h$, $p_i = e_i$ and for $i = 3m+1$, $m_i = m-1$, $lag_i^{min} = 0$, $lag_i^{max} = h$, $p_i = 1$. Job $3m+1$ has a set of $m-1$ fixed start times $t_{3m+1,k} = \{(Q+1)k - 1\}$ for $k = 1, \ldots, m-1$. The fixed tasks of job $3m+1$ leave only m idle intervals, each of length Q, on which the unfixed jobs must fit exactly since $\forall i = 1, \ldots, 3m$, $p_i > \frac{Q}{4}$, it is not possible to schedule 4 jobs in a single interval. If strictly less than three jobs are scheduled within an interval, then an idle time must be present since $\forall i = 1, \ldots, 3m$, $p_i < \frac{Q}{2}$, which does not allow to schedule all tasks in the m intervals. Hence, the 3-Partition problem built this way has a solution if and only if the ONPSP has a solution. □

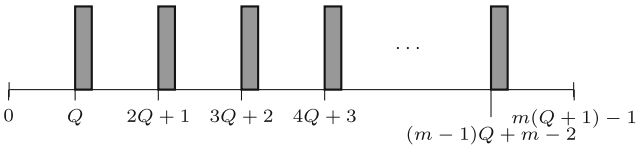


Fig. 2. Illustration of the reduction of 3-Partition to the ONPSP without lag. Grey rectangles represent the $m-1$ tasks with fixed start times.

Theorem 2. *The ONPSP is strongly NP-hard, even if* $T_i = \emptyset$, $\forall i = 1, \ldots, n$.

Proof. Consider the 3-Partition problem from which we build an instance of the ONPSP as follows. Let $h = m(Q+1)$. We have $3m+1$ jobs where, for $i = 1, \ldots, 3m$, $m_i = 1$, $lag_i^{min} = 0$, $lag_i^{max} = h$, $p_i = e_i$ and for $i = 3m+1$, $m_i = m$, $lag_i^{min} = lag_i^{max} = Q+1$, $p_i = 1$. It is easy to show that if the problem is feasible, there is a solution that schedules the first task of job $3m+1$ at time Q and the last job at time $h-1$. The m tasks of job $3m+1$ partition the cycle time in m idle time intervals of length Q where the other jobs must fit. The 3-Partition problem built this way has a solution if and only if the ONPSP has a solution. □

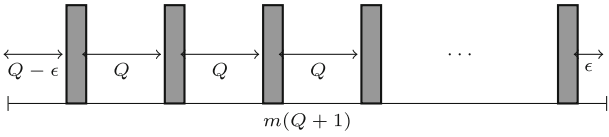


Fig. 3. Illustration of the reduction of 3-Partition to the ONPSP without fixed start times. Grey rectangles represent the m tasks with constant lag.

Without fix start times or time lags, the problem becomes polynomial if we only consider maximizing the number of tasks without precedence constraints. If we consider duration maximization, we can reduce the problem to Subset-Sum with or without precedence constraints (we need to select tasks with duration a_i such that $\sum a_i = C$, C being the time remaining when all mandatory tasks are inserted). The only case that remains is when we only want to maximize the number of tasks with precedence constraints, which corresponds in the 3-field scheduling notation to $1|d_i = d, prec|\sum U_i$. We are not aware of complexity results in this case.

5 MILP Model and Scaling Matheuristic

5.1 Time Indexed MILP

We propose a MILP of the problem directly transposed from the one proposed in [18] for the related offline nanosatellite task scheduling problem, incorporating a new modeling framework for the precedence constraints that are absent in [18].

Let $y_{i,t}$ be a binary decision variable that states whether job i is in process at time period t and let $z_{i,t}$ denote a binary decision variable indicating whether a task of job i starts at time t. For ease of notation, we denote as $\rho(t)$ the time period immediately following t in a cyclic schedule, i.e. $\rho(t) = t + 1$ if $t < h - 1$ and $\rho(h-1) = 0$ otherwise. Symmetrically, we denote as $\rho^{-1}(t)$ the time period immediately preceding t in a cyclic schedule. We have $\rho^{-1}(t) = t - 1$ if $t \geq 1$ and $\rho^{-1}(0) = h - 1$. We denote as $I(t, \delta)$ the cyclic interval of length δ starting at time period t that can be defined by $I(t, \delta) = \{\tau_1, \ldots, \tau_\delta\}$ with $\tau_1 = t$ and $\tau_q = \rho(\tau_{q-1})$ for $q = 2, \ldots, \delta$. Our MILP model is described below:

$$\max \sum_{i=1}^{n} \sum_{t=0}^{h-1} u_i z_{i,t} + v_i y_{i,t} \tag{2}$$

subject to:

$$z_{i,t} \geq y_{i,t} - y_{i,\rho^{-1}(t)} \qquad \forall i \in \mathcal{J}, \forall t = 0, \ldots, h-1 \tag{3}$$

$$z_{i,t} \leq y_{i,t} \qquad \forall i \in \mathcal{J}, \forall t = 0, \ldots, h-1 \tag{4}$$

$$\sum_{\tau=t}^{t+\underline{p}_i-1} y_{i,\tau} \geq \underline{p}_i z_{i,t} \qquad \forall i \in \mathcal{J}, \forall t = 0, \ldots, h-\underline{p}_i \tag{5}$$

$$\sum_{\tau=t}^{t+\underline{p}_i-1} z_{i,\tau} \leq 1 \qquad \forall i \in \mathcal{J}, \forall t = 0, \ldots, h-\underline{p}_i \tag{6}$$

$$\sum_{\tau=t}^{t+\overline{p}_i} y_{i,\tau} \leq \overline{p}_i + \sum_{\tau=t+1}^{t+\overline{p}_i} z_{i,\tau} \qquad \forall i \in \mathcal{J}, \forall t = 0, \ldots, h-\overline{p}_i-1 \tag{7}$$

$$\underline{m}_i \leq \sum_{t=0}^{h-1} z_{i,t} \leq \overline{m}_i \qquad \forall i \in \mathcal{J} \tag{8}$$

$$\sum_{\tau \in I(t, lag_i^{min})} z_{i,\tau} \leq 1 \qquad \forall i \in \mathcal{J}, \forall t = 0, \ldots, h-1 \tag{9}$$

$$\sum_{\tau \in I(t, lag_i^{max})} y_{i,\tau} \geq 1 \qquad \forall i \in \mathcal{J}, \forall t = 0, \ldots, h-1 \tag{10}$$

$$\sum_{i \in \mathcal{J}} y_{i,t} \leq 1 \qquad \forall t = 0, \ldots, h-1 \tag{11}$$

$$\pi_{i,i',0} = z_{i,0} - z_{i',0} \qquad \forall (i,i') \in E \tag{12}$$

$$\pi_{i,i',t} = \pi_{i,i',t-1} + z_{i,t} - z_{i',t} \qquad \forall (i,i') \in E, \forall t = 1, \ldots, h-1 \tag{13}$$

$$z_{i,t} = 1 \qquad \forall i \in \mathcal{J}, \forall t \in T_i \tag{14}$$

$$\pi_{i,i',t} \geq 0 \qquad \forall (i,i') \in E, \forall t = 0, \ldots, h-1 \tag{15}$$

$$z_{i,t} \in \{0,1\} \qquad \forall i \in \mathcal{J}, \forall t = 0, \ldots, h - \underline{p}_i \tag{16}$$

$$y_{i,i'} \in \{0,1\} \qquad \forall i \in \mathcal{J}, \forall t = 0, \ldots, h-1 \tag{17}$$

The objective (2) is to maximize the weighted sum of the number of tasks and the total duration of each job. Constraints (3) and (4) link variables $z_{i,t}$ and variables $y_{i,t}$, in the sense that $z_{i,t}$ marks the start of a task of job i and must be equal to 1 if the job is in process at time t but not at time $\rho^{-1}(t)$, the previous time period in the cyclic sense. Constraints (5) and (6) enforce the minimum duration for each task of each job. Constraint (5) states that if a job i is (re)started at time t, the task must be in process during at least $\underline{p}_i$ time periods and constraints (6) prevent a job from starting twice during an interval of length $\underline{p}_i$. Constraints (7) prevent a job from being scheduled more than $\overline{p}_i$ time periods within each interval of length $\overline{p}_i + 1$ unless it is restarted within this interval. Constraints (8) set the number of scheduled tasks of each job within the required bounds. Constraints (9) and (10) enforce the minimum and maximum time lags taking account of the cyclic context. Constraints (11) prevent tasks from being scheduled in parallel. Constraints (12–13) are discussed in more detail below. Constraints (14) are the fixed start time constraints. Constraints (15–17) give variable domains (the variable $\pi_{i,i',t}$ is also introduced below).

The MILP model proposed in [18] does not allow to tackle the precedence constraints. Modeling the precedence constraint between task (i,j) and task (i',j) in standard time-indexed MILP would require to index the "start" and "in-progress" variable by the task index j, such that constraint $\sum_{\tau=t}^{h-1} y_{i,j,t} + \sum_{\tau=0}^{t} z_{i',j,t}$, $\forall t = 0, \ldots, h-1$ models the precedence relation. However, this would create a much bigger MILP model with $2h \sum_{i=1}^{n} \overline{m}_i$ binary variables instead of $2hn$ binary variables.To overcome this difficulty we propose a new way of modeling the precedence constraints without increasing the number of binary variables, by introducing a continuous decision variable $\pi_{i,i',t}$ for each precedence constraint $(i,i') \in E$ and each time period t.

Theorem 3. *Constraints (12–13) enforce the precedence relation* $(i,i') \in E$.

Proof. Variable $\pi_{i,i',t}$ is incremented as soon as a task of job i starts and is decremented when a task of job i' starts. Thanks to the non-negativity of the

variables, a task of job i' can only start if a task of job i' without an already scheduled successor has started. □

5.2 Scaling Matheuristic

Unfortunately, for practical applications, even with this reduced model, the time-indexed MILP formulation may have a huge number of binary variables, particularly in our use case, where the cycle time lasts a second while task accuracy is of the order of a microsecond. To address this issue, we propose a scaling matheuristic that works as presented in Algorithm 1. Steps 1–3 downscales and rounds the data using a scale factor S to obtain an ONPSP aiming at being more restrictive than the original one. However, for fixed start times, no rounding choice is a priori more restrictive. We apply the following procedure, determined experimentally, to favor feasibility. We arbitrarily round up each fixed start time (step

Algorithm 1: Scaling Matheuristic

Data: an ONPSP instance P, a scaling factor $S > 0$
Result: an ONPSP solution $(m_i)_{i\in\mathcal{J}}, (s_{i,j}, p_{i,j})_{i\in\mathcal{J}, j=1,\ldots;m_i}$ or `fail`
`// Build a downscaled ONPSP instance` (P')
1 $h' \leftarrow \lfloor \frac{h}{S} \rfloor$;
2 $\underline{p}'_i \leftarrow \left\lceil \frac{\underline{p}_i}{S} \right\rceil$, $\overline{p}'_i \leftarrow \left\lfloor \frac{\overline{p}_i}{S} \right\rfloor$, $lag_i^{max\prime} \leftarrow \left\lceil \frac{lag_i^{min}}{S} \right\rceil$, $lag_i^{min\prime} \leftarrow \left\lfloor \frac{lag_i^{max}}{S} \right\rfloor$, $\forall i \in \mathcal{J}$;
3 $T'_i = \{t'_{i,k} = \left\lceil \frac{t_{i,k}}{S} \right\rceil \mid i \in \mathcal{J}, k = 1, \ldots, Q_i\}$;
4 add to (P') additional constraints: $y_{j,t} = 0, \forall j \in \mathcal{J}, \forall t \in \cup_{i\in\mathcal{J}}\{\rho^{-1}(t') | t' \in T'_i\} \setminus \cup_{i\in\mathcal{J}, t'\in T'_i} I(t', \underline{p}_i)$;
`// Solve` (P') `and build solution to` (P)
5 solve (P');
6 **if** `success` **then**
7 Get number of tasks m_i, start and duration $s'_{i,j}$, $p'_{i,j}$, $\forall i \in \mathcal{J}, j = 1, \ldots, m_i$;
8 Solve LP with variables $s_{i,j} \in [0, h]$, $p_{i,j} \in [\underline{p}_i, \overline{p}_i], \forall i \in \mathcal{J}, j = 1, \ldots, m_i$:

$$\max \sum_{i\in\mathcal{J}} \sum_{j=1}^{m_i} v_i p_{i,j} \tag{18}$$

$$s_{a,b} \geq s_{i,j} + p_{i,j} \qquad \forall a, b, i, j : s'_{a,b} > s'_{i,j} \tag{19}$$

$$s_{i,j} = t_{i,k} \qquad \forall i, j, i, k : s'_{i,j} = t'_{i,k} \tag{20}$$

$$s_{i,j+1} - s_{i,j} - p_{i,j} \leq lag_i^{max} \qquad i \in \mathcal{J}, j = 1, \ldots, m_i - 1 \tag{21}$$

$$h + s_{i,0} - s_{i,m_i} - p_{i,m_i} \leq lag_i^{max} \qquad i \in \mathcal{J} \tag{22}$$

$$s_{i,j+1} - s_{i,j} \geq lag_i^{min} \qquad i \in \mathcal{J}, j = 1, \ldots, m_i - 1 \tag{23}$$

$$h + s_{i,0} - s_{i,m_i} \geq lag_i^{min} \qquad i \in \mathcal{J} \tag{24}$$

if `success` **then**
9 return the solution found
10 return `fail;`

3), but we prevent any task to be scheduled in the previous time period, except if the earliest completion time of another fixed task is precisely this time period (Constraint added at step 4). The resulting MILP may be infeasible while the original problem is feasible, which is a clear drawback of this method, prone to further improvements. In the case of success in solving the downscaled problem, we build and solve an LP (step 8) to obtain the start times and durations in the original scale. The tasks are ordered in the order obtained by the downscaled MILP (constraint 18) and all constraints are linear. Note, however, that this LP still may have no solution if the decisions taken by the downscaled MILP are incompatible with the original scale.

6 NanoSatScheduler

In this section, we present NanoSatScheduler, an open source software tailored to solve ONPSP. This software embeds a CP model and a user interface allowing the user to create/modify their ONPSP instances and visualize/edit/save the solutions provided by the solver.[3]

The variables $s_{i,j} \in [0, h - \underline{p}_i]$ and $e_{i,j} \in [\underline{p}_i, h]$ indicate the start and end time of the j-th task of job i, respectively, and the variables $p_{i,j} \in [\underline{p}_i, \overline{p}_i]$ represent their duration. The variables $x_{i,j} \in [0, 1]$ indicate whether the j-th task of job i is actually present in the schedule. Finally, the variables $f_{i,k} \in [1, \overline{m}_i]$ will allow us to select which task will be associated with each date fixed by the user. Our objective is a combination of the occurrences and the total time used in the schedule for each partition (25).

Constraints (26) order the start variables in increasing chronological order. Constraints (27) link the presence variables so that the first variables are always used to indicate the presence of a task in the schedule. Those two constraints are both helpful to model the problem, but also to prevent the solver from exploring symmetrical solutions, thus improving its efficiency. We enforce the minimum number of tasks with constraints (28). The global constraints NoOverlap (29) prevent the *present* tasks – task (i, j) is present if and only if $x_{i,j} = 1$ – from overlapping. This global constraint is available in most CP solvers. The respect of *max-lag* is ensured with constraints (30), which enforce a maximum delay lag_i^{max} between the end and start of two consecutive present tasks. The consistency between the last present task and the first task is ensured with constraint (31). In the same fashion, the *min-lag* constraints are respected thanks to constraints (32–33). For each precedence between partitions in E, a *predecessor* task is required to be present and placed before a *successor* task if it is present. Finally, the global constraint Element$(x, vars, value)$ allows us to enforce that the variable in the list $vars$ at index x takes the value $value$. Here, we use this global constraint to ensure that each fixed time value $t_{i,k}$ is assigned to a start variable $s_{i,j}$. For this, we define variables $f_{i,k}$ representing the index of the variable in the list s_i that will be assigned the value $t_{i,k}$.

[3] Git repository: https://gitlab.laas.fr/roc/julien-rouzot/onpsp.

$$\max \sum_{i=1}^{n} \sum_{j=1}^{\overline{m}_i} u_i x_{i,j} + v_i p_{i,j} x_{i,j} \tag{25}$$

subject to:

$$s_{i,j} < s_{i,j+1} \qquad \forall i \in \mathcal{J},\ j \in \mathcal{M}_i \setminus \overline{m}_i \tag{26}$$

$$x_{i,j+1} \implies x_{i,j} \qquad \forall i \in \mathcal{J},\ j \in \mathcal{M}_i \setminus \overline{m}_i \tag{27}$$

$$x_{i,\underline{m}_i} = 1 \qquad \forall i \in \mathcal{J} \tag{28}$$

$$\textsc{NoOverlap}(s_{i,j}, p_{i,j}, x_{i,j})_{i \in \mathcal{J}, j \in \mathcal{M}_i} \tag{29}$$

$$x_{i,j+1} \implies s_{i,j+1} - e_{i,j} \leq lag_i^{max} \qquad \forall i \in \mathcal{J},\ j \in \mathcal{M}_i \setminus \overline{m}_i \tag{30}$$

$$!(x_{i,j+1}) \wedge x_{i,j} \implies h + s_{i,0} - e_{i,j} \leq lag_i^{max} \qquad \forall i \in \mathcal{J},\ j \in \mathcal{M}_i \setminus \overline{m}_i \tag{31}$$

$$x_{i,j+1} \implies s_{i,j+1} - s_{i,j} \geq lag_i^{min} \qquad \forall i \in \mathcal{J},\ j \in \mathcal{M}_i \setminus \overline{m}_i \tag{32}$$

$$!(x_{i,j+1}) \wedge x_{i,j} \implies h + s_{i,0} - s_{i,j} \geq lag_i^{min} \qquad \forall i \in \mathcal{J},\ j \in \mathcal{M}_i \setminus \overline{m}_i \tag{33}$$

$$x_{i',j} \implies x_{i,j} \wedge (s_{i,j} < s_{i',j}) \qquad \forall (i,i') \in E,\ j \in \mathcal{M}_i \tag{34}$$

$$\textsc{Element}(f_{i,k}, s_i, t_{i,k}) \qquad \forall i \in \mathcal{J},\ k \in 1, \ldots, Q_i \tag{35}$$

To make our scheduling tool accessible to the nanosatellite design teams and for wider distribution, we implemented a graphical user interface within NanoSatScheduler. This interface has the following features:

- *Instance Creation*: Users can enter the constraints of their problem by specifying parameters for each constraint for each partition, and indicate their objective function. These instances can be saved/loaded from an existing instance file.
- *Solution Visualization and Editing*: Once the instance is solved, the solution is displayed on the interface, along with solution details such as the objective function value, solution time, and the time used by each partition on the schedule. A solver-independent test tool verifies that all constraints are satisfied. Users can independently visualize each partition, and it is also possible to directly modify the solution through the interface, by setting fixed start times and/or changing tasks duration. This will trigger the solve process again and provide the user with a new solution. If the solution is satisfactory to the user, it can be saved in the appropriate format for direct use with the XTratuM hypervisor.

7 Experimental Results

XTratuM hypervisor [15] comes with a planning tool called XoncretE developed by the company FentISS. This software includes both a graphical interface and a console, and can be used by the NanoLab Academy mission teams. However, XoncretE has several drawbacks: Each occurrence (task) of each partition (job) must be manually declared in the interface; It is not possible to apply *min-lag*

and *max-lag* constraints, we can only provide an exact frequency for each partition; The tool does not perform optimization (it does not seek to maximize the number of tasks or their durations). The planning for the hypervisor is therefore usually done as follows: CNES provides the teams with a baseline plan with the partitions common to all nanosatellite missions, and each mission team attempts to manually insert their own partitions into the plan. This represents a laborious task, leading to suboptimal solutions for the teams. We will not compare ourselves here with XoncretE, which is not an optimization software, but we will use the NIMPH mission instance (which is the real-case study) to illustrate the contribution of our method compared to a solution constructed according to the method in place at NanoLab Academy. This instance has 13 partitions and up to 60 tasks to schedule. For the statistical analysis, we also generate random instances based on the NIMPH instance. Our instance generator can be configured to vary the size of the instances, the minimum/maximum number of each type of partition (i.e. with different type of constraints), and the average proportion of the schedule allocated to each. We ensure that the minimum occurrences are proportional to the given cycle time, in order to generate realistic and non-trivial instances. If, however, some instances generated this way are proven to be unsatisfiable in less than one second by our solver, these instances are discarded.

For our experiments, we use the CP-SAT solver from OR-Tools [17]. All experiments are run on a single-core Intel E5-2695 v3 2.3 Ghz CPU with 32 GB of RAM. For all experiments, the time limit is set to one hour for each instance. As maximizing the number of tasks is often the main goal, we run most of our experiments with this criterion only, but we also optimize both the number of tasks and durations for the NIMPH instance.

We begin by comparing the performance of our two models, MILP and CP, on a set of 100 synthetic instances derived from NIMPH, following the methodology described earlier. For the MILP model, we solve the instances using various time-step sizes (i.e. precision), while the CP model does not require such time scaling. In Table 1, the number of feasible solutions found over the 100 instances is shown. We observe that the matheuristic sometimes fails to build feasible solutions even if a feasible solution to the downscaled problem is found, due to the impact of time-step scaling. Increasing precision (i.e. reducing the step size) may lead to less failures in the reconstruction step but significantly slows the MILP resolution and, in many cases, even prevents it from converging to a solution. The 'gap' column reports the average gap to the best upper bound known when a valid solution is found. Our results show that the MILP model performs poorly on these instances, primarily because of the high number of time steps in NIMPH-like scenarios.

In contrast, the CP model does not require a variable for each job at every time step, allowing for faster solving without sacrificing precision. The CP model finds feasible solutions for all 100 instances and proves optimality for 82 of them. Among the optimally solved instances, 74 of them are solved in less than 1 min. As the MILP model proves impractical for our use case, we focus the remainder of this section on further experimentation with the CP model.

Table 1. Number of feasible solutions found by the downscaled MILP, number of feasible solutions (after the reconstruction step for the MILP method), mean optimality gap to the best bound known for the valid solutions, for our CP and MILP model with different time step size on 100 synthetic instances.

Method	Step size	MILP Feasible	Feasible	Gap
MILP	5 ms	78	72	5.1%
MILP	2 ms	66	60	7.6%
MILP	1 ms	35	34	12.9%
CP	1 μs	-	100	0.7%

We now demonstrate the interest of our CP model on the real instance, which corresponds to the NIMPH mission partitions. Here, we maximize two science partitions – namely EDMON and M2M – using lexicographical optimization, as NIMPH team has strict preferences on which one they want to favor. For each stage corresponding to the current partition to optimize, we first maximize the number of tasks, then the total duration of the partition[4]. For this real-world case study, the NIMPH team provided the initial schedule generated by CNES using XoncretE, where the payload partitions must be inserted. To ensure a fair comparison, we created a new instance in which the base schedule consists of tasks with fixed start times and duration. We then apply our lexicographical optimization to insert the payload tasks. The resulting solution represents the best schedule the NIMPH team could have achieved manually, assuming the solution is optimal[5]. Table 2 shows that the primary objective (i.e. maximizing the number of occurrences of the EDMON partition) is significantly worse when starting from a baseline schedule. We generate those solutions in less than 30 s, while minutes and generally hours are needed to manually achieve similar solutions. This highlights the limitations of the current scheduling workflow used at the NanoLab Academy, at least for the NIMPH mission, and demonstrates how our method can both improve schedules quality and significantly reduce the time required to generate them.

Table 2. Objective value for each objective ranked in a lexicographical order on NIMPH instance with and without using the baseline schedule provided by XoncretE.

Objective	XoncretE + handcrafted	NanoSatScheduler
Number of tasks EDMON	2	**5**
Duration EDMON	130.0	**307.2**
Number of tasks M2M	**4**	2
Duration M2M	**106.0**	40.0

[4] Maximizing the number of tasks before the total duration is a request from the NIMPH team.

[5] This is the case for the NIMPH instance.

Even if for NIMPH mission the objective is a strict lexicographical maximization, it might not be the case for other missions. As tuning the weights of the objective function can be a difficult process for the user, we propose to generate Pareto fronts, so the user can directly choose from a set of non-dominated solutions. To compute the Pareto solutions, we use the well-known epsilon-constraint method [8]. Figure 4 presents the Pareto front for NIMPH instance, if we want to maximize the number of tasks for EDMON partition against the number of tasks for M2M, and Fig. 5 presents the Pareto front when maximizing the durations. The user can then select the best solution for her/his use case. Note that any other objective combination can be generated (e.g. number of tasks against total duration for the same job). For NIMPH mission, we can see that we can generate a solution that dominates the one reached using lexicographical optimization when using the base schedule in terms of number of tasks.

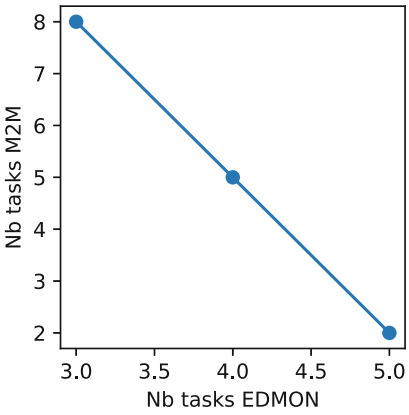


Fig. 4. Pareto front for the number of tasks of partition M2M against EDMON.

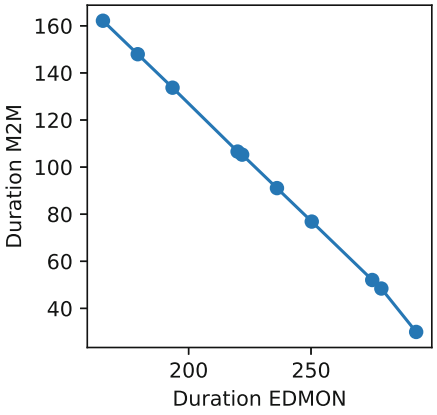


Fig. 5. Pareto front for the total duration of partition M2M against EDMON.

When we started working with NIMPH team, one of our first questions was about the cycle time length, as a single second cycle seemed quite small. According to the NIMPH team, semi-manually scheduling a second-long plan is already laborious, so scheduling longer plans is not considered, due to an overwhelming complexity. As our software handles one-second plans easily, we now analyze the impact of cycle time on the solutions quality. To that end, we generate 100 more synthetic instances and successively solve them and increase the cycle time and number of minimum/maximum tasks of each job accordingly. The time limit for each iteration is fixed to one hour and we add the previous objective function value as a lower bound for the next stage. Table 3 shows the mean number of tasks over a second, the mean solution time and the proportion of optimal solutions for 100 instances, for different cycle time lengths.

We observe that extending the cycle time allows for a slight increase (+0.6%) in the number of tasks over a second. Note that in this case, we would be able to perform more than 2000 additional tasks every hour. Additionally, the proportion of optimal solutions significantly declines as the size of the instances increases.

Table 3. Mean number of tasks (per second), mean solution time, and proportion of optimal solutions, for different cycle time lengths, on 100 synthetic instances.

Cycle time	Number of tasks (per s)	Solution time	Prop. optimal
1 s	110.6	6 min	91 %
2 s	111.1	22 min	66 %
4 s	111.2	45 min	32 %

Another problem of this arbitrary cycle time is the potential sub-optimality of the generated solutions. Indeed, the computed schedules can generally be compressed to fit a shorter cycle time, thus increasing the average number of tasks performed per second. To that end, we first compute a solution with the baseline cycle time and the given objective (maximize the number of tasks and/or durations of a given set of jobs). Then we solve the instance again, but we add the previous objective value as a constraint and minimize the cycle time. In Table 4, we display the mean cycle time, the mean number of tasks per second, the mean solution time. The second line correspond to solutions where the cycle time is minimized. Note that the proportion of optimal solutions is not compared in Table 4, as the objectives are different for the two stages. For this experiment, the number of tasks per second significantly increases (+3%), which shows that minimizing the cycle time is more efficient than extending it.

Table 4. Mean cycle time, mean number of tasks (per second), mean solution time, with and without minimizing the cycle time, on 100 synthetic instances.

Cycle time	Number of tasks (per s)	Solution time
1 s	110.6	6 min
0.969 s	114.0	34 min

8 Conclusion

In this paper, we presented both a MILP approach and a CP model to tackle the ONPSP, complemented by a user interface that enables visualization and interaction with the generated solutions. Additionally, we provide users with a set of non-dominated solutions, allowing them to explore and analyze the trade-offs

between different objectives. Our results demonstrate that the MILP matheuristic faces scalability issues. In contrast, the CP model proves to be highly efficient without compromising precision. For our real-case study, our method performs much better than the current workflow of semi-manually building the schedule, which convinced the NIMPH team to switch to NanoSatScheduler for the mission. Additionally, we show that extending the cycle time has a small impact on the solution quality, whereas minimizing it significantly increases the number of tasks that can be scheduled over time.

For future work, our primary objective is to distribute the software within the CNES NanoLab Academy to gather feedback from other nanosatellite mission teams and improve our approach. In [21], we proposed a CP model that aims at minimizing the context switches only, but this cannot be directly incorporated in our new model. We plan to enhance NanoSatScheduler by accounting for the context switches in the task durations, which will improve solution quality, as the context switch time can be ignored when tasks of the same partitions are continuous[6]. Lastly, it would be valuable to explore whether our method could be extended to broader use cases, particularly in the context of energy-aware scheduling for nanosatellite missions, as discussed in [4,18–20,23].

References

1. Al Sheikh, A., Brun, O., Hladik, P.E.: Partition scheduling on an IMA platform with strict periodicity and communication delays. In: 18th International Conference on Real-Time and Network Systems, pp. 179–188 (2010)
2. Balashov, V.V., Balakhanov, V.A., Kostenko, V.A.: Scheduling of computational tasks in switched network-based IMA systems. In: Proceedings of International Conference on Engineering and Applied Sciences Optimization, pp. 1001–1014 (2014)
3. Bernardo, V.P., Seman, L.O., Bezerra, E.A., Ribeiro, B.F.: Hardware-in-the-loop simulation of an on-board energy-driven scheduling algorithm for cubesats. IEEE Embed. Syst. Lett. **16**(1), 69–72 (2023)
4. Camponogara, E., Seman, L.O., Rigo, C.A., Morsch Filho, E., Ribeiro, B.F., Bezerra, E.A.: A continuous-time formulation for optimal task scheduling and quality-of-service assurance in nanosatellites. Comput. Oper. Res. **147**, 105945 (2022)
5. Chen, J., Du, C., Xie, F., Lin, B.: Scheduling non-preemptive tasks with strict periods in multi-core real-time systems. J. Syst. Architect. **90**, 72–84 (2018)
6. Comer, D., Fossum, T.V.: Operating System Design: Internetworking with Xinu. Prentice Hall (1987)
7. Dobiáš, P., Casseau, E., Sinnen, O.: Fault-tolerant online scheduling algorithms for cubesats. In: Proceedings of the 11th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures/9th Workshop on Design Tools and Architectures for Multicore Embedded Computing Platforms, pp. 1–6 (2020)
8. Haimes, Y.: On a bicriterion formulation of the problems of integrated system identification and system optimization. IEEE Trans. Syst. Man Cybern. **SMC-1**(3), 296–297 (1971)

[6] Currently, context switches are accounted for by adding a constant margin to the minimum task duration, so tasks always include the context switch time penalty.

9. Hanen, C., Munier, A.: Cyclic scheduling on parallel processors: an overview. In: Chrétienne, P., Coffman, E., Lenstra, J., Liu, Z. (eds.) Scheduling Theory and Its Applications. Wiley (1994)
10. Heidt, H., Puig-Suari, J., Moore, A., Nakasuka, S., Twiggs, R.: Cubesat: a new generation of picosatellite for education and industry low-cost space experimentation. In: 14th Annual/USU Conference on Small Satellites, Utah, USA (2000)
11. Karlsson, E., Rönnberg, E., Stenberg, A., Uppman, H.: A matheuristic approach to large-scale avionic scheduling. Ann. Oper. Res. **302**(2), 425–459 (2021)
12. Kessler Slongo, L., Vega Martinez, S., Vale Barbosa Eiterer, B., Augusto Bezerra, E.: Nanosatellite electrical power system architectures: models, simulations, and tests. Int. J. Circuit Theory Appl. **48**(12), 2153–2189 (2020)
13. Liubimov, O., Turkin, I.: Optimizing the cubesat on-board computer power consumption under hard real-time constraints. In: Conference on Integrated Computer Technologies in Mechanical Engineering–Synergetic Engineering, pp. 404–414. Springer, Cham (2023)
14. Martinez, S.V., et al.: An integrated thermal-electrical model for simulations of battery behavior in cubesats. Appl. Sci. **11**(4), 1554 (2021)
15. Masmano, M., Ripoll, I., Crespo, A., Metge, J.: Xtratum: a hypervisor for safety critical embedded systems. In: 11th Real-Time Linux Workshop, vol. 9 (2009)
16. Minaeva, A., Hanzálek, Z.: Survey on periodic scheduling for time-triggered hard real-time systems. ACM Comput. Surv. (CSUR) **54**(1), 1–32 (2021)
17. Perron, L., Didier, F.: CP-SAT (2024). https://developers.google.com/optimization/cp/cp_solver/
18. Rigo, C.A., Seman, L.O., Camponogara, E., Morsch Filho, E., Bezerra, E.A.: A nanosatellite task scheduling framework to improve mission value using fuzzy constraints. Expert Syst. Appl. **175**, 114784 (2021)
19. Rigo, C.A., Seman, L.O., Camponogara, E., Morsch Filho, E., Bezerra, E.A., Munari, P.: A branch-and-price algorithm for nanosatellite task scheduling to improve mission quality-of-service. Eur. J. Oper. Res. **303**(1), 168–183 (2022)
20. Rigo, C.A., Seman, L.O., Morsch Filho, E., Camponogara, E., Bezerra, E.A.: MPPT aware task scheduling for nanosatellites using MIP-based ReLU proxy models. Expert Syst. Appl. **234**, 121022 (2023)
21. Rouzot, J., et al.: Scheduling onboard tasks of the NIMPH nanosatellite. In: Liberatore, F., Wesolkowski, S., Parlier, G.H. (eds.) Proceedings of the 13th International Conference on Operations Research and Enterprise Systems, ICORES 2024, Rome, Italy, 24–26 February 2024, pp. 277–284. SCITEPRESS (2024)
22. Schild, K., Würtz, J.: Scheduling of time-triggered real-time systems. Constraints **5**, 335–357 (2000)
23. Seman, L.O., Rigo, C.A., Camponogara, E., Munari, P., Bezerra, E.A.: Improving energy aware nanosatellite task scheduling by a branch-cut-and-price algorithm. Comput. Oper. Res. **158**, 106292 (2023)
24. Turkin, I., Liubimov, O., Zakharenko, V.: Optimization of energy consumption of the cubesat on-board computer under real-time limitations. Aerosp. Tech. Technol. (3), 126–137 (2024)
25. Vega Martinez, S., Seman, L.O., Morsch Filho, E., Slongo, L.K., Bezerra, E.A.: On-board energy scheduling optimization algorithm for nanosatellites. Int. J. Circuit Theory Appl. **51**(8), 3915–3937 (2023)
26. Wikum, E.D., Llewellyn, D.C., Nemhauser, G.L.: One-machine generalized precedence constrained scheduling problems. Oper. Res. Lett. **16**(2), 87–99 (1994)

27. Windsor, J., Hjortnaes, K.: Time and space partitioning in spacecraft avionics. In: 2009 Third IEEE International Conference on Space Mission Challenges for Information Technology, pp. 13–20. IEEE (2009)
28. Zhang, W., Behbahani, A.S., Eltawil, A.M.: Joint frequency scheduling and power allocation for cubesat communication. In: 38th International Communications Satellite Systems Conference (ICSSC 2021), vol. 2021, pp. 184–188 (2021)

Accelerated Discovery of Set Cover Solutions via Graph Neural Networks

Zohair Shafi[1(✉)], Benjamin A. Miller[1,2], Tina Eliassi-Rad[1], and Rajmonda S. Caceres[2]

[1] Northeastern University, Boston, MA, USA
shafi.z@northeastern.edu
[2] MIT Lincoln Laboratory, Lexington, MA, USA

Abstract. Machine learning (ML) approaches are increasingly being used to accelerate combinatorial optimization (CO) problems. We investigate the Set Cover Problem (SCP) and propose Graph-SCP, a graph neural network method that augments existing optimization solvers by learning to identify a smaller sub-problem that contains the solution space. Graph-SCP uses both supervised learning from prior solved instances and unsupervised learning to minimize the SCP objective. We evaluate the performance of Graph-SCP on synthetically weighted and unweighted SCP instances with diverse problem characteristics and complexities, and on instances from the OR Library, a canonical benchmark for SCP. We show that Graph-SCP reduces the problem size by 60–80% and achieves runtime speedups of up to 10x on average when compared to Gurobi (a state-of-the-art commercial solver), while maintaining solution quality. This is in contrast to fast greedy solutions that significantly compromise solution quality to achieve guaranteed polynomial runtime. We showcase Graph-SCP's ability to generalize to larger problem sizes, training on SCP instances with up to 3,000 subsets and testing on SCP instances with up to 10,000 subsets.

Keywords: Graph Neural Networks · Combinatorial Optimization · Set Cover Problems

1 Introduction

A growing area of research explores machine learning (ML) solutions to combinatorial optimization (CO) problems. These ML solutions can be divided into two categories: (1) learning end-to-end models that generates feasible solutions (e.g., [11]) and (2) learning non-end-to-end models that predict branching heuristics and aids existing CO solvers (e.g., [14,31]). Bengio et al. [3] provide a comprehensive survey of methods used in each category. Here, we focus on the latter approach and use ML to aid conventional CO solvers.

We focus on the set cover problem (SCP), an NP-hard CO problem. In SCP, one is given the "universe," which is a set of elements $\{1, 2, \ldots m\}$. One is also

G. Tack (Ed.): CPAIOR 2025, LNCS 15763, pp. 191–208, 2025.
https://doi.org/10.1007/978-3-031-95976-9_12

given a collection of n sets whose union corresponds to the universe. To solve the problem, one must identify the *smallest* sub-collection of the n sets whose union is equal to the universe.

We use a graph neural network (GNN) model to accelerate the runtime of SCP solvers without sacrificing solution quality. Concretely, we propose *Graph-SCP*, where we cast an instance of SCP as a graph and learn a GNN to predict a subgraph that encapsulates the solution space. The nodes of this subgraph are passed into a conventional CO solver such as Gurobi [16] (a state-of-the-art commercial solver). This method of reducing the size of the input problem and passing it to traditional solvers enables them to run faster. Graph-SCP achieves between 60–80% reduction in input problem size, which leads to runtime improvements of up to 10x on average, while maintaining solution quality across a range of SCP instance characteristics.

Traditional solvers swiftly generate incumbent solutions, but require additional time to reach optimality. In our comparative analysis, we examine the incumbent solutions generated by both Graph-SCP and Gurobi, revealing that the reduction of the search space by Graph-SCP allows the solver to find incumbent and finally, optimal solutions faster. Our contributions are as follows:

- We propose Graph-SCP, a non-end-to-end ML-based framework for finding a subproblem to a given SCP instance. That subproblem is subsequently given to any traditional CO solver. Graph-SCP achieves the optimal objective value, while running up to 10x faster than conventional solvers.
- We provide insights into how features from alternative representations, such as a hypergraph representation of an SCP instance can be used to improve Graph-SCP's search. Our findings show that features derived from this hypergraph representation of the SCP instance lead to the largest speedup.
- We present a comprehensive evaluation of Graph-SCP, with SCP instances ranging across various densities, sizes, and costs (weighted vs. unweighted), as well as instances from the OR Library [2]. We demonstrate Graph-SCP's ability to generalize across problem characteristics. We compare against two benchmark state-of-the-art solvers, Gurobi [16] and SCIP [4], and discuss how various modeling choices affect the performance of Graph-SCP.

2 Background

Figure 1(A) shows an example of an SCP instance. We adopt an alternate formulation for SCP. Given a binary matrix $A \in \mathbb{R}^{m \times n}$, SCP is defined as covering all m rows (denoting the elements of the universe) by the minimum cost subset of the n columns (denoting the collection of sets whose union corresponds to the universe). Costs of each column are represented in the column vector $c \in \mathbb{R}^n$. The goal is to find the assignment vector $x \in \mathbb{R}^n$, where

$$x_j = \begin{cases} 1 & \text{if column } j \text{ is in solution} \\ 0 & \text{otherwise} \end{cases} \qquad \forall j \in n, \tag{1}$$

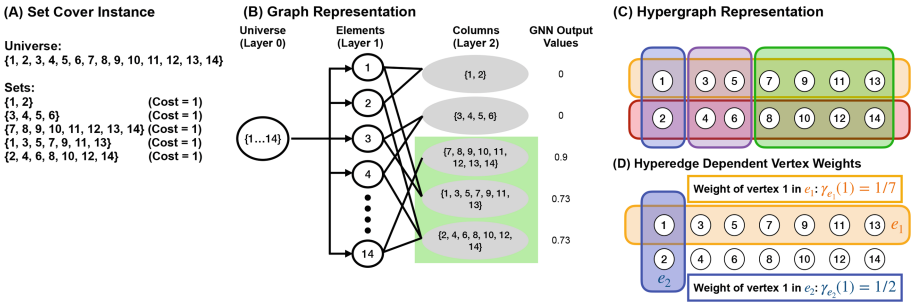


Fig. 1. (A) A simple example of a set cover problem with equal cost subsets. (B) The set cover instance represented using a bipartite graph abstraction. This graph is passed into a GNN which then predicts whether nodes (i.e. subsets) belong to the SCP solution (green highlighted nodes). The prediction values for Layer 0 and Layer 1 nodes (white-colored nodes) are discarded, since only Layer 2 nodes contribute to the solution. (C) The hypergraph representation for the same set cover instance. Each subset corresponds to a hyperedge shown as bounding boxes around the element nodes. (D) For each hyperedge e, we define hyperedge dependent weights $\gamma_e(v)$ for each element v of the universe as the inverse of the size of its associated hyperedge as shown.

The optimization problem is:

$$\min_x \sum_{j \in n} c_j x_j \text{ s.t. } \sum_{j \in n} A_{ij} x_j \geq 1, \ \forall 1 \leq i \leq m, x_j \in \{0, 1\}. \tag{2}$$

Given such a matrix, we can define its density as $d = \frac{q}{m \times n}$, where q is the number of non-zero entries in the matrix A [24].

Graph Abstractions for SCP. By treating the covering matrix as an adjacency matrix, with elements in the universe as rows and sets as columns, we can represent the SCP instance as a directed bipartite graph (see Fig. 1(B)). A node representing the universe is connected to each element node, and the element nodes are connected to the sets in which they appear.

We also represent the SCP instance as a hypergraph. Here, each element of the universe is a vertex and each set is a hyperedge (Fig. 1(C)). The hyperedges are demarcated by bounding boxes around the associated element nodes. This hypergraph representation offers an additional lens through which to explore the connections between elements and sets. For example, it enables an investigation into the structural characteristics of an SCP instance by examining features inherent to the hypergraph structure (such as hyperedge dependent vertex weights in Fig. 1(D)).

Graph Neural Networks. GNNs combine node-wise updates with message updates from neighboring nodes. Throughout this work, we utilize GraphSAGE [17], a message passing neural network. Formally, given a graph $G = (V, E, X)$ with nodes V, edges E, and node features X, GraphSAGE computes node representations as:

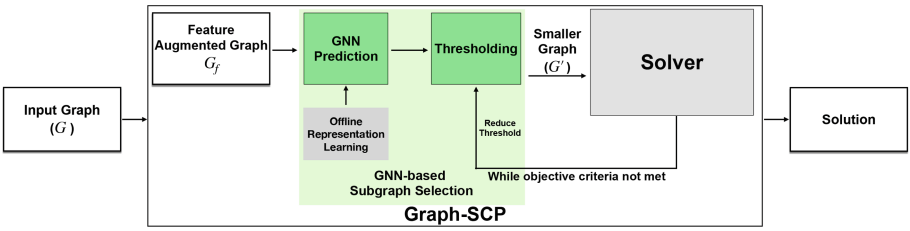


Fig. 2. Graph-SCP takes an SCP instance represented as a graph as input. The graph is augmented with a set of features on its nodes (G_f). A GNN is trained on this graph to predict a subgraph of nodes that likely contains the solution to the SCP instance. At runtime, the GNN is used (only once) to generate predictions. Graph-SCP picks nodes at a predefined percentile threshold as the subgraph. If the objective criteria are not met, the threshold is reduced, thus selecting a larger subgraph.

$$h_v^0 = \boldsymbol{x_v}, \tag{3}$$

$$h_{N(v)}^k \leftarrow AGGREGATE(\{h_u^{k-1}, \forall u \in N(v)\}), \tag{4}$$

$$h_v^k \leftarrow \sigma(W^k \cdot CONCAT(h_k^{k-1}, h_{N(v)}^k), \tag{5}$$

where $\boldsymbol{x_v}$ is the node feature vector for node v, k is the number of message passing layers, h_v^k is the hidden representation of node v at layer k, W^k is the weight matrix at layer k, $N(v)$ is the set of node v's neighbors, and $AGGREGATE$ can be any permutation invariant aggregation method like sum or average.

3 Proposed Method: Graph-SCP

Figure 2 provides an overview of Graph-SCP, which takes as input an SCP instance represented as a graph G (a.k.a. an *SCP graph*). Graph-SCP augments G with a set of features on its nodes (described in the next section). Then, a GNN is trained on the augmented graph, G_f, to predict whether a subgraph (i.e., a set of nodes) is likely to appear in the solution. The output of the GNN is subsequently given to a thresholding procedure that selects which subgraphs should be given to the solver. If the solver cannot find a solution, Graph-SCP reduces the threshold and selects a larger subgraph. This process continues until a solution is found.

3.1 Augmenting a SCP Graph with Features

Before Graph-SCP trains a GNN, it adds the following features to each node of the SCP graph:

Cost: We assign costs of 0 to the "Universe" node in layer 0 and "Element" nodes in layer 1 of the graph (Fig. 1B). The "Columns" nodes in layer 2 (representing subsets) have their costs set according to the input problem.

Cover: This feature is the cardinality of the set of elements represented by the node. For each of the "Columns" nodes in layer 2, it is the size of the set. For the "Elements" nodes in layer 1 of the graph, it is set to 1. For the "Universe" node, it is set to 0. This feature is similar to the one used by the standard greedy approximation algorithm for SCP [10].

Random Walks with Restarts (RWR) [34]**:** This feature represents the relative importance of nodes based on the structure of the graph. Our RWR algorithm starts a random walk from the "Universe" node (layer 0) and restarts the walk from the same node (see Fig. 1(B)). When the walker reaches stationary distribution, the score between each node in layers 1 and 2 and the universe node is used as a node feature. This feature value for the "Universe" node is set as 0. The restart probability is a hyperparameter and is set to 0.45 since the SCP graph is shallow.

Degree-Based Features: For each node in the SCP graph (Fig. 1(B)), we capture its degree and the average degree of its neighbors.

Hypergraph Features: Figure 1(C) shows an SCP instance cast as a hypergraph. For a hypergraph $H(V, E, \omega, \gamma)$, each hyperedge e has weights $\omega(e)$ (denoting the cost of each set). The elements within each hyperedge e, denoted by $v \in V$ have hyperedge-dependent weights $\gamma_e(v)$. These hyperedge-dependent weights correspond to the inverse of the hyperedge degree as shown in Fig. 1(D). Given this hypergraph, we compute its Laplacian matrix [9]. The algebraic connectivity of H, i.e., the second smallest eigenvalue of the Laplacian matrix of H, is denoted by $\mu(H)$. The contribution of an individual hyperedge e to the algebraic connectivity of H can be quantified as $c(e) = \mu(H) - \mu(H - e) \geq 0, \forall e \in E$. According to the definition of algebraic connectivity, H is connected when $\mu(H) > 0$. In the context of SCP, connectedness implies the coverage of every element in the universe. For unweighted SCP instances modeled as hypergraphs, the optimal solution corresponds to a connected hypergraph with the fewest hyperedges. The optimization objective can be expressed as maximizing algebraic connectivity and minimizing the number of hyperedges. A greedy approach that iteratively removes hyperedges with the smallest $c(e)$ will return an approximate solution to the SCP instance.

Compared to a standard greedy algorithm for SCP which relies only on the hyperedge degree as a heuristic (i.e., number of uncovered elements in a set), features derived from the hypergraph Laplacian capture richer information than the hyperedge degree alone (see Fig. 3).

Motivated by the above discussion, we compute two features derived from the hypergraph representation of SCP. Observe that the hypergraph representation does not include the universe node. Thus the hypergraph feature values for the "Universe" node are set to 0.

To generate hypergraph features for the "Elements" and "Columns" nodes, we use the following procedure. First, each "Elements" node is assigned a weight equal to the inverse of its hyperedge degree – i.e., the weight of an element within a subset corresponds to the inverse of the set size (see Fig. 1(D)). Second,

we define two probability transition matrices in order to generate the Laplacian matrix of the SCP hypergraph with edge-dependent vertex weights [9]: (*i*) a matrix for transitions from one "Elements" node to another via a hyperedge (a "Columns" node), and (*ii*) a matrix for transitions from one hyperedge to another via an "Elements" node. Third, we compute the eigenvalues of the two Laplacian matrices. These eigenvalues become the hypergraph-based features of the "Elements" and "Columns" nodes, respectively.

3.2 Using a GNN to Predict Membership in the Optimal Solution

Chen et al. [8] proved that given a dataset of SCP instances, a GNN can approximate the optimal solution of the linear program relaxation of the SCP instance. Building on this observation, we formalize reducing the SCP problem size as a subgraph selection learning task. In particular, our Graph-SCP uses a message passing GNN to learn the following binary classification task: predict which "Columns" nodes (see Fig. 1(B)) are likely to be part of the optimal solution. Formally, given a graph $G = (V, E)$ with node features X, we learn a GNN function $f(G, X)$ to predict a binary classification vector $y \in \mathbb{R}^{|V|}$, where 1 indicates the nodes in the graph that are likely to contain the SCP solution and 0 otherwise.

Objective Function. Graph-SCP's objective function consists of supervised and unsupervised components. The supervised component is a standard binary cross entropy loss that uses labels generated by Gurobi, whereas the unsupervised component directly minimizes the objective value of the linear program relaxation of the SCP instance. Formally, let $A \in \mathbb{R}^{m \times n}$ be the binary covering matrix and $c \in \mathbb{R}^{n \times 1}$ be the cost vector. Then the loss function is given by

$$L(y, \hat{y}) = \alpha(y \cdot \log \hat{y} + (1 - y) \cdot \log(1 - \hat{y})) + \tag{6}$$

$$\beta[\sum(\hat{y} \cdot c)^2 - \gamma \sum(A\hat{y} - 1) - \omega \sum(1 - A\hat{y})] \tag{7}$$

Observe that $\hat{y}$ only considers nodes in the graph that represent sets (i.e., "Columns" nodes from layer 2, as shown in Fig. 1(B)) and the log function is taken element wise. The term $\hat{y} \cdot c$ penalizes the cost and the number of sets picked. A feasible solution to the SCP instance requires that $A\hat{y} \geq 1$. To that end, the term $\sum(A\hat{y} - 1)$ penalizes constraints in A that are not satisfied and the term $\sum(1 - A\hat{y})$ encourages a sparse selections of sets. The hyperparameters α and β control the relative importance of the supervised and unsupervised components, whereas hyperparameters γ and ω control the constraint satisfaction and subset sparsity, respectively. In practice, we set $\alpha = 1, \beta = 10^{-4}, \gamma = 1$ and $\omega = 0.4$. The model is trained with an Adam optimizer with a learning rate of 10^{-4}.

Inference. The trained model outputs continuous values between 0 and 1 for each "Column" node from layer 2. To select a set of nodes, Graph-SCP selects nodes at a predefined percentile threshold and passes the selected nodes into a CO solver. Our experiments use Gurobi and SCIP; however, other solvers of choice can be used. Subsequently, Graph-SCP checks if the objective value *obj*

returned by the solver with the reduced problem size is at least a user-defined value u_{obj}, i.e., $obj \leq u_{obj}$. If not, the percentile threshold is reduced (by a fixed decrement size, set to 10 in our experiments), thus selecting a larger set of nodes to pass into the solver. Graph-SCP uses solutions from each iteration to warm-start the solver for the next iteration. This iterative refinement process mirrors Column Generation [15] methods used in tackling large-scale CO problems.

In our experiments, we set u_{obj} to be the optimal objective value, thereby forcing Graph-SCP to run until the optimal value is reached. It is important to note that u_{obj} serves as a bound rather than a specific solution, indicating the proximity to optimality required for the solution; an aspect often specified in practical applications. In real-world scenarios where a predefined limit is not available, an additional iteration can be run to determine if further improvements are possible. Due to the warm-starting mechanism, where each iteration builds on the results of the previous one, these additional runs incur relatively low computational cost. Furthermore, we demonstrate that in situations constrained by runtime rather than a specific objective limit, Graph-SCP consistently identifies superior incumbent solutions compared to Gurobi (see Fig. 6).

Node Feature Importance. Figure 3 demonstrates the value of the node features described in the previous section for Graph-SCP's prediction task. For each "Elements" node and each "Columns" node, we extract the node embeddings from the penultimate layer of their GNN. To visualize these embeddings, we use PaCMAP [38] and map the embeddings in 2 dimensions. To quantify how well various sets of features can separate solution nodes from non-solution nodes, we measure the average distance of a solution node to other solution nodes (intra distance) and the average distance of a solution node to non-solution nodes (inter distance) in the GNN embedding space. Ideally, the intra distance is small, the inter distance is large and difference between inter and intra distance is large. With this definition in place, we see in Fig. 3 that separability is the highest when using all of the aforementioned node features, i.e., subplots (A) and (E) where the difference between inter and intra distances is the largest.

4 Experimental Setup

GNN Architecture. Graph-SCP uses GraphSAGE [17] with 2 message-passing layers with 1024 neurons and ReLU activation, followed by a fully connected layer with 1024 neurons and sigmoid activation. Each layer is followed by a dropout layer (dropout rate of 0.4) and batch normalization. The model has approximately 2.1 million parameters.

Training Data. To generate training data, we create SCP instances (see Table 1) and solve them using Gurobi [16], a state-of-the-art CO solver. We label the sets that are part of the solution with 1 and all other sets with 0. In the case of multiple solutions, all sets that are part of *any* solution are labeled as 1. The SCP instances are generated with various densities and characteristics.

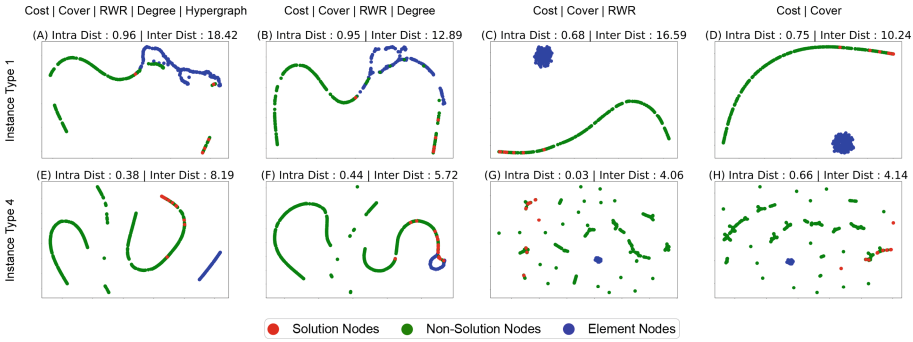


Fig. 3. Visualization of node embeddings in two dimensions using PaCMAP [38]. The embeddings are extracted from the penultimate layer of Graph-SCP's message passing GNN with different sets of node features. The features used are displayed above each column. Each row shows embeddings for a different type of SCP instance (see Table 1). To quantify how well various sets of features can separate solution nodes from non-solution nodes, we measure the average distance of a solution node to other solution nodes (intra distance) and the average distance of a solution node to non-solution nodes (inter distance) in the GNN embedding space. Observe how (A) and (E) have the largest inter distance with small intra distances, which indicates that Cost, Cover, RWR, degree-based, and hypergraph-based features are useful in separating solution nodes from non-solution nodes.

We also use the canonical OR Library at the testing stage. The instances generated for training reflect the OR library in range of densities, number of columns, and number of rows, but incorporate additional variation in the distribution of costs. We categorize the instances into 5 instance types and show results for each. Detailed characteristics of each instance type are shown in Table 1. Graph-SCP was trained using 75 instances from each type for a total of 300 instances, *excluding* Instance Type 5, which is used only during testing. The test set consists of 30 instances from each type. Observe that we train up to a maximum of 3000 columns (Type 4) but test on instances with up to 10,000 columns.

We train and test on an Intel Xeon Gold 6148 CPU with 24 cores and one NVIDIA Tesla V100 GPU (16 GB). Our code repository is available at https://github.com/zohairshafi/Graph-SCP.

Baseline Methods. We compare Graph-SCP against several approaches w.r.t. runtime and objective achieved.

- **Gurobi**: A state-of-the-art, commercial CO solver [16].
- **Greedy Baseline**: Sets with the largest number of uncovered elements are picked at each step. The greedy baseline has faster runtimes but an approximation ratio of $\ln(n)$ where n is the size of the universe [10].
- **Lagrangian Relaxation Heuristic**: Recent works highlight how well designed heuristics can outperform a learned ML model for CO problems [1] or how ML methods only marginally improve simpler alternatives [29].

Table 1. Characteristics of SCP instances used during training and testing of Graph-SCP. Here, m is the number of rows, n the number of columns, and d is the density of the instance. Details of the sets in the OR Library and their results are in Table 2.

Instance	m	n	d	Cost
Type 1	100–400	100–1000	0.22–0.29	Uniform [100–200]
Type 2	100–300	100–500	0.16–0.28	Equal
Type 3	200–350	300–350	0.13–0.18	Poisson ($\lambda = 20$)
Type 4	200–250	1000–3000	0.04–0.05	Poisson ($\lambda = 20$)
Type 5 (OR Library)	100–500	1000–10000	0.02–0.2	Uniform [1–100]

To that end, we also compare against a heuristic algorithm for SCP that combines Lagrangian relaxation and the greedy method as proposed by [41].

- **Predict and Search** (PS-Gurobi): [18] use a GNN to predict the marginal probability of each set (i.e., a "Columns" node) followed by searching for a solution in a well-defined ball around the predicted solution.
- **Random Subset Selection**: As a sanity check for the subgraph selection and to understand the contribution of learning from solved SCP instances, we test against a random node selection strategy. Given an initial relative subgraph size parameter k, we randomly select $k\%$ nodes to pass to the solver. For our experiments, we repeat this process for $k = \{20, 50, 80\}$.
- **GCNN Branching**: We compare against the GCNN branching framework by Gasse et al. [14], where a GNN is used to approximate the strong branching heuristic of a branch-and bound-solver. To replicate their setup exactly, we allow cutting plane generation at the root node only and deactivate solver restarts and test on SCIP [4] only.

5 Results

Figure 4 shows runtime speed-up performance and solution quality of Graph-SCP, greedy, heuristic, PS-Gurobi and random subgraph selection baselines relative to Gurobi's performance. Note that speedups are shown in the log scale. The runtime speedup factor is equal to $\frac{\text{Gurobi runtime}}{\text{Graph-SCP runtime}}$. The runtime for Gurobi and Graph-SCP are calculated as the wall clock time until the optimal objective is found. In the case of Graph-SCP, this includes the time taken to propagate through the GNN as well as every run of the solver if the threshold is reduced and the solver is rerun, until the optimal solution is found. For the heuristic, greedy and random baselines, the runtime is the wall clock time until the algorithm returns an output. We see that Graph-SCP achieves the optimal objective across all instances. In terms of speedup, the largest speedup is about 70x for Instance Type 2 (Fig. 4(B)) and the smallest speedup is about 3x for Instance Type 4 (Fig. 4(D)).

Note that the greedy algorithm runs significantly faster across all 5 instances, however it has poor objective values. The heuristic algorithm achieves better

Table 2. Results for the sets of instances defined in the OR Library [2]. Each set (A, B, NRE etc.) has its own definitive characteristics. Here, m is the number of rows, n the number of columns, d is the density of the instance and '(s)' is seconds. Graph-SCP has an average speedup of 3.2x across all sets while achieving the optimal objective and an average of approximately 84% reduction in problem size.

Set	# Instances	m	n	Cost [Uniform]	d	Speedup Factor	Mean Runtime Gurobi (s)	Size Reduction
6	5	200	1000	1–100	0.05	1.26	0.10	75%
A	5	300	3000	1–100	0.02	1.28	0.22	84%
B	5	300	3000	1–100	0.05	2.34	0.97	89%
C	5	400	4000	1–100	0.02	2.19	0.69	83%
D	5	400	4000	1–100	0.05	3.48	2.92	86%
NRE	5	500	5000	1–100	0.1	5.96	36.90	94%
NRF	5	500	5000	1–100	0.2	8.15	32.6	96%
NRG	5	1000	10000	1–1000	0.02	1.17	2003.27	71%

objective values than greedy, albeit at the cost of runtime speedup. In each subplot in Fig. 4, the area of the plot above the red line (random subgraph selection baseline) corresponds to the benefits of learning offline strategies for picking a smaller problem size that contains the solution. Observe how the slope of the line indicates that random selection performs poorly if a better objective is required. We analyze the reduction in problem size as the ratio of number of subsets passed to Gurobi by Graph-SCP to the original number of subsets. For each of the four instances shown, Graph-SCP achieves a size reduction of $72.29\% \pm 11.21\%$, $87.76\% \pm 8.19\%$, $65.53\% \pm 15.31\%$ and $73.21\% \pm 7.46\%$, respectively. We present a detailed breakdown of results and characteristics for Instance Type 5 (instances from the OR Library [2]) in Table 2. These results highlight the ability of Graph-SCP to generalize to other instances given that Instance Type 5 was not part of the training set. We see an average speedup across Type 5 instances of 3.2x and an average problem size reduction of 84.75%.

Results for the sets of instances defined in the OR Library citech12beasley1990or. Each set (A, B, NRE etc.) has its own definitive characteristics. Here, m is the number of rows, n the number of columns, d is the density of the instance and '(s)' is seconds. Graph-SCP has an average speedup of 3.2x across all sets while achieving the optimal objective and an average of approximately 84% reduction in problem size.

Generalization. We highlight generalization properties of Graph-SCP by testing against problem instances with varying complexities as defined in [14], where instances are classified as 'easy', 'medium' or 'hard', with 500 rows and 500, 1000 and 2000 columns, respectively, and a density of 0.05. To replicate the setting exactly, we use SCIP [4] as our back-end solver instead of Gurobi, with cutting plane generation at the root node only and solver restarts disabled. Figure 5 shows results from this experiment, where we see that Graph-SCP outperforms the GCNN branching method from [14]. Note that we use the same model trained on Instance Types 1–4 (with a maximum of 400 rows).

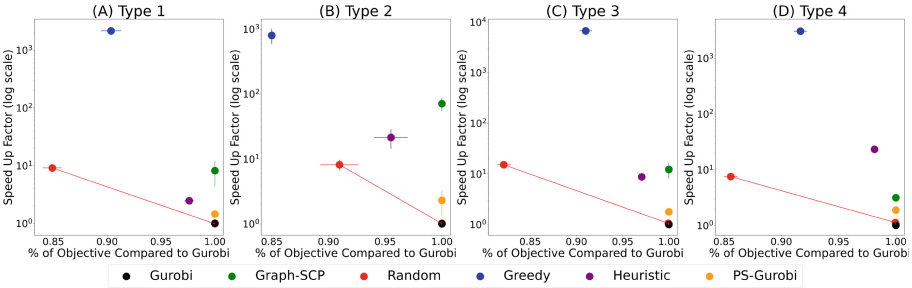


Fig. 4. Graph-SCP achieves the optimal objective with faster runtimes. We see the largest speedup of about 70x for Instance Type 2 (B) and the smallest speedup of about 3x for Instance Type 4 (D). Note that the greedy algorithm runs significantly faster across all 4 instances, but obtains poor objective values. In each subplot, the area above the red line (random sample) correspond to the benefits of learning. Thirty instances were used across each type with error bars showing standard error. (Color figure online)

We observe that traditional CO solvers find high quality incumbent solutions relatively quickly, with the majority of the time spent trying to find the optimal solution. To investigate the impact of Graph-SCP on these incumbent solutions, we set timeout limits to the solver and examine the primal gap achieved. Here, if obj_{bound} is the best known objective bound and obj_{val} is the current best objective solution, then the primal gap is defined as: $primal\ gap = \frac{obj_{bound} - obj_{val}}{obj_{val}}$ Fig. 6 shows results averaged across 30 instances in each of the instance types defined in Table 1. Graph-SCP achieves better incumbent solutions before both Gurobi and Graph-SCP reach the optimal value.

Gurobi vs. SCIP. We compare Graph-SCP with a Gurobi backend vs. Graph-SCP with a SCIP backend in Table 3. Graph-SCP hyperparameters like the initial threshold are held constant across both solvers. We see that Graph-SCP yields higher acceleration with SCIP than with Gurobi. We posit this could be due to SCIP being more sensitive to the size of the input problem than Gurobi.

Table 3. Speedup factors and mean runtimes in seconds (s) of Graph-SCP on two different back-end solvers: Gurobi and SCIP. All Graph-SCP hyperparameters are held constant between the runs. The speedup factors for Graph-SCP are higher with SCIP than with Gurobi. However, Gurobi uses parallelism better than SCIP, and thus has lower mean runtimes.

Instance Type	Speedup Factor Gurobi	Mean Runtime Gurobi (s)	Speedup Factor SCIP	Mean Runtime SCIP (s)
Type 1	8.19 ± 3.82	5.61 ± 0.53	11.61 ± 3.02	21.87 ± 3.07
Type 2	70.94 ± 16.11	17.2 ± 4.87	491.53 ± 134.74	168.15 ± 48.07
Type 3	12.26 ± 4.2	35.57 ± 4.11	18.48 ± 5.05	313.5 ± 46.26
Type 4	3.65 ± 0.24	40.74 ± 7.02	11.96 ± 5.28	751.16 ± 176.33

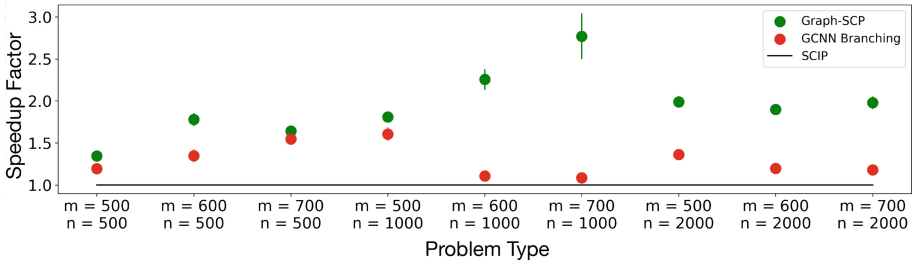


Fig. 5. Generalization properties of Graph-SCP against problem instances with complexities as defined in [14]. Here, m is the size of the universe and n is the number of subsets, with densities $d = 0.05$. We use the same model trained on instances of Types 1 through 4, that are smaller with a maximum of 400 rows. Graph-SCP achieves faster speedups in runtimes when compared to GCNN Branching. Ten instances were sampled for each problem type with error bars showing standard error.

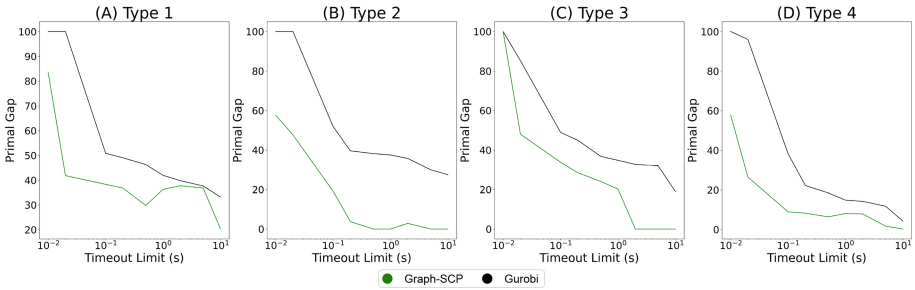


Fig. 6. Incumbent solutions generated by Graph-SCP and Gurobi at specific timeout values. The y-axis shows the primal gap (lower is better) and the x-axis shows the timesteps at which the solver was stopped. Graph-SCP achieves better incumbent solutions at each time step. Results are averaged over 30 instances from each type.

5.1 Discussions and Limitations

Graph-SCP has optimal performance on instances with densities within the range of 0.1 and 0.2. However, the acceleration in runtime diminishes for instances below a density of 0.02. We attribute this to the highly imbalanced nature of the classification task, where the number of solution sets relative to all possible sets becomes skewed. To demonstrate, we test on instances with a fixed number of rows (600) and columns (1000) while varying the density. Costs are picked uniformly between 0 and 100. Results are shown in Fig. 7, where we see that the speedup achieved by Graph-SCP is stable across a wide range of densities and gradually decreases as the density of the problem decreases.

Figure 8 shows the distribution of the number of times the threshold is reduced for each instance type as well as the initial starting threshold values. All instance types can share the same initial threshold since Graph-SCP automatically reduces it as needed. In our experiments, we determine the initial threshold

value empirically by scanning threshold values until most validation instances yield a feasible (but not necessarily optimal) solution within the selected subgraph. Figure 8 demonstrates that different instance types exhibit varying patterns in how thresholds are reduced and highlights the flexibility of our setup. This approach enables Graph-SCP to dynamically adapt to the unique characteristics of each instance type, optimizing performance based on the specific demands of the problem.

For larger SCP instances, Graph-SCP is faster than directly solving the problem because the reduction in the problem size is substantial. For smaller SCP instances, Graph-SCP can be marginally slower than directly solving the problem, because the overhead of a forward pass through the GNN is larger than the benefit of reducing the problem size. We also show the impacts of various components of Graph-SCP through an ablation study in Table 4.

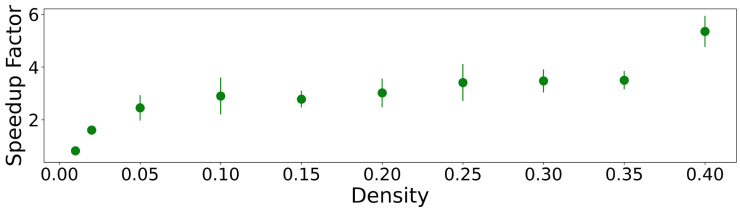


Fig. 7. Speedup factors of Graph-SCP for problems with varying densities. We generate instances with 600 rows (size of universe), 1000 columns (number of sets), and costs picked uniformly in [0, 100]. Speed up remains stable across a wide range of densities and as density gradually reduces, the speed up achieved by Graph-SCP reduces.

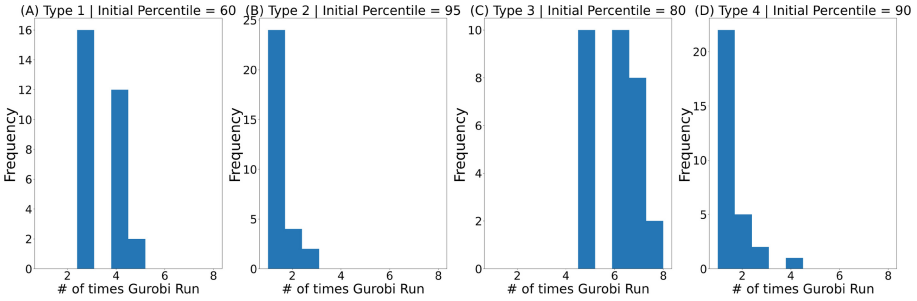


Fig. 8. For each instance type, we show the distribution of the number of times Graph-SCP reduces the threshold. The initial staring threshold is shown above each subplot. Observe that different instance types exhibit varying patterns in how thresholds are reduced, highlighting the flexibility of this setup. This enables Graph-SCP to dynamically adapt to the unique characteristics of each instance type, optimizing performance based on the specific demands of the problem.

Table 4. Speedup factors when training Graph-SCP without the unsupervised component result in lower acceleration. Further removal of hypergraph node features leads to an additional decline in acceleration.

Method	Type 1	Type 2	Type 3	Type 4
Graph-SCP	9.97 ± 2.13	70.148 ± 18.27	11.76 ± 2.43	3.20 ± 0.84
w/o Unsupervised	6.75 ± 3.43	64.59 ± 22.00	5.83 ± 0.56	2.23 ± 0.32
w/o Unsupervised & Hypergraph	4.45 ± 1.73	60.869 ± 18.85	3.54 ± 0.45	2.014 ± 0.427

Finally, we also compare Graph-SCP's performance by replacing GraphSAGE as the GNN used with Graph Attention Networks (GAT) [36], Graph Convolutional Network (GCN) [22], Graph Isomorphism Networks (GIN) [39], and Chebyshev GCN [12] with average runtime speed-ups of 3.52 ± 0.51, 3.74 ± 0.92, 4.16 ± 0.68 and 7.98 ± 0.74, respectively, on Instance Type 1. We posit that differences in performance are due to limiting the sizes of each model to approximately 2 million parameters to be comparable to each other.

6 Related Work and Conclusions

Existing literature explores the use of ML to replace or augment traditional solvers for various CO problems with the objective of accelerating runtimes and, where appropriate, improving the quality of solutions. The proposed methods can be classified into two approaches: learning a model in an end-to-end manner to *replace* a CO solver, or learning a model to *aid* an existing CO solver.

Under the category that uses ML to replace a solver, Numeroso et al. [28] apply the Neural Algorithmic Reasoning framework proposed by Veličković et al. [35] to jointly optimize for the primal and dual of a given problem. They train models to find joint solutions for both the min-cut and max-flow problems and show that their solution leads to better overall performance. For NP-hard problems, Khalil et al. [11] propose a reinforcement learning approach that uses graph embeddings to learn a greedy policy that incrementally builds a solution. Heydaribeni et al. [19] model constrained CO problems as a hypergraph followed by mapping continuous values generated by the Hypergraph Neural Network to integer node assignments using simulated annealing. Schuetz et al. [30] and Hu et al. [20] use GNNs to minimize the Quadratic Unconstrained Binary Optimization (QUBO) formulations of CO problems. Boisvert et al. [5] provide a generic method for encoding arbitrary CO problems into a graph structure for use with GNNs. Yau et al. [40] propose GNNs that can capture provably optimal message passing algorithms for a large class of combinatorial optimization problems. *The aim of our work is to focus on accelerating an existing solver*, allowing us to carry over its performance guarantees. In a similar context, Ireland et al. [21] and Tian et al. [33] look at problems like minimum vertex cover and max-cut and use a GNN to identify promising nodes for large scale graphs. Han et al. [18] train a GNN to predict a marginal probability of each variable

and then construct a trust region to search for high quality feasible solutions. Verhaeghe et al. [37] cast the resource constrained project scheduling problem as a directed graph and use GNNs to extract information to help speed up a traditional solver. Li et al. [25] use a GNN to predict if a vertex is part of the optimal solution followed by a tree search to arrive at the final solution. However, Bother et al. [6] later showed that the results in [25] are not reproducible. Liu et al. [26] investigate the performance of branching rules with respect to the size of the search neighborhood and devise a framework for guiding the search of the branching heuristic. Ding et al. [13] also use a learned GCN model to predict branching cuts. They modify the solver by adding a local branching rule in their approximate case and performing an actual branch at the root node in the exact case. Nair et al. [27] propose two methods, Neural Branching and Neural Diving. While we operate exclusively on the input space irrespective of a solver, neural diving aims to explore the branch and bound tree in a depth-first manner. They differ from traditional diving in that they do not start from the root and explore all the way to the leaf node, but start from root, stop mid way and allow a solver to solve the remaining subproblem. Our method similarly uses a traditional solver to find a solution. Gasse et al. [14] use a GNN to learn computationally expensive branch-and-bound heuristics for MIP solvers. The learning is done offline and the learned models are then used at runtime in place of the heuristic to achieve faster runtimes while maintaining the quality of the solution. This work is similar to our approach in that it uses ML to aid a CO solver; however, it does so by speeding up the internal workings (branching heuristic) of the solver. These methods lie on the lines of modifying or improving the branch and bound heuristic. Kruber et al. [23] use supervised learning to determine when a Dantzig-Wolfe decomposition should be applied to a Mixed Integer Program (MIP). Graph-SCP differs in that it reduces the input problem size complexity, and therefore can be used in conjunction with any existing solver, while achieving the optimal solution. Shen et al. [32] use support vector machines to speed up column generation methods by predicting the solution to the pricing problem at each step. We point the interested reader to surveys by Bengio et al. [3] and Cappart et al. [7] that provide a detailed summary of research in this field.

Our contribution is Graph-SCP, a GNN framework to accelerate runtimes for both conventional and ML augmented CO solvers. By identifying a smaller subproblem that encapsulates the solution space, Graph-SCP achieves runtimes up to approximately 10x faster, while maintaining the reliability associated with traditional solvers. A distinguishing feature of Graph-SCP is its GNN prediction module, which is executed only once. This efficiency is in contrast to many ML solutions for CO problems. We illustrate Graph-SCP's robust generalization across SCP instances with diverse characteristics and complexities, showing its effectiveness on instances from the canonical OR Library. Through comparative studies, we offer insights into the impact of different modeling choices on Graph-SCP's performance. Crucially, since Graph-SCP operates only in the input space, it seamlessly integrates with any traditional or ML-augmented CO solver.

Acknowledgments. Distribution Statement A. Approved for public release. Distribution is unlimited. This material is based upon work supported by the Under Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Under Secretary of Defense for Research and Engineering.

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

References

1. Angelini, M.C., Ricci-Tersenghi, F.: Modern graph neural networks do worse than classical greedy algorithms in solving combinatorial optimization problems like maximum independent set. Nat. Mach. Intell. **5**(1), 29–31 (2023)
2. Beasley, J.E.: Or-library: distributing test problems by electronic mail. J. Oper. Res. Soc. **41**(11), 1069–1072 (1990)
3. Bengio, Y., Lodi, A., Prouvost, A.: Machine learning for combinatorial optimization: a methodological tour d'horizon. Eur. J. Oper. Res. **290**(2), 405–421 (2021)
4. Bestuzheva, K., et al.: Enabling research through the SCIP optimization suite 8.0. ACM Trans. Math. Softw. **49**(2) (2023)
5. Boisvert, L., Verhaeghe, H., Cappart, Q.: Towards a generic representation of combinatorial problems for learning-based approaches. In: CPAIOR (2024)
6. Böther, M., Kißig, O., Taraz, M., Cohen, S., Seidel, K., Friedrich, T.: What's wrong with deep learning in tree search for combinatorial optimization. In: ICLR (2021)
7. Cappart, Q., Chételat, D., Khalil, E.B., Lodi, A., Morris, C., Velickovic, P.: Combinatorial optimization and reasoning with graph neural networks. JMLR **24**, 130–1 (2023)
8. Chen, Z., Liu, J., Wang, X., Yin, W.: On representing linear programs by graph neural networks. In: ICLR (2022)
9. Chitra, U., Raphael, B.: Random walks on hypergraphs with edge-dependent vertex weights. In: ICML, pp. 1172–1181 (2019)
10. Chvatal, V.: A greedy heuristic for the set-covering problem. Math. Oper. Res. **4**(3), 233–235 (1979)
11. Dai, H., Khalil, E.B., Zhang, Y., Dilkina, B., Song, L.: Learning combinatorial optimization algorithms over graphs. In: NeurIPS (2017)
12. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: NeurIPS (2016)
13. Ding, J.Y., et al.: Accelerating primal solution findings for mixed integer programs based on solution prediction. In: AAAI (2020)
14. Gasse, M., Chételat, D., Ferroni, N., Charlin, L., Lodi, A.: Exact combinatorial optimization with graph convolutional neural networks. In: NeurIPS (2019)
15. Gilmore, P.C., Gomory, R.E.: A linear programming approach to the cutting-stock problem. Oper. Res. 849–859 (1961)

16. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2023). https://www.gurobi.com
17. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: NeurIPS (2017)
18. Han, Q., et al.: A GNN-guided predict-and-search framework for mixed-integer linear programming. In: ICLR (2022)
19. Heydaribeni, N., Zhan, X., Zhang, R., Eliassi-Rad, T., Koushanfar, F.: Distributed constrained combinatorial optimization leveraging hypergraph neural networks. Nat. Mach. Intell. 1–9 (2024)
20. Hu, C.: Assessing and enhancing graph neural networks for combinatorial optimization: novel approaches and application in maximum independent set problems. arXiv preprint arXiv:2411.05834 (2024)
21. Ireland, D., Montana, G.: Lense: learning to navigate subgraph embeddings for large-scale combinatorial optimisation. In: ICML (2022)
22. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: ICLR (2017)
23. Kruber, M., Lübbecke, M.E., Parmentier, A.: Learning when to use a decomposition. In: International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (2017)
24. Lan, G., DePuy, G.W., Whitehouse, G.E.: An effective and simple heuristic for the set covering problem. Eur. J. Oper. Res. **176**(3), 1387–1403 (2007)
25. Li, Z., Chen, Q., Koltun, V.: Combinatorial optimization with graph convolutional networks and guided tree search. In: NeurIPS (2018)
26. Liu, D., Fischetti, M., Lodi, A.: Learning to search in local branching. In: AAAI (2022)
27. Nair, V., et al.: Solving mixed integer programs using neural networks. arXiv preprint arXiv:2012.13349 (2020)
28. Numeroso, D., Bacciu, D., Veličković, P.: Dual algorithmic reasoning. In: ICLR (2023)
29. Santana, Í., Lodi, A., Vidal, T.: Neural networks for local search and crossover in vehicle routing: a possible overkill? In: CPAIOR (2023)
30. Schuetz, M.J., Brubaker, J.K., Katzgraber, H.G.: Combinatorial optimization with physics-inspired graph neural networks. Nat. Mach. Intell. (2022)
31. Shafi, Z., Miller, B.A., Chatterjee, A., Eliassi-Rad, T., Caceres, R.S.: Grasp: accelerating shortest path attacks via graph attention. arXiv preprint arXiv:2310.07980 (2023)
32. Shen, Y., Sun, Y., Li, X., Eberhard, A., Ernst, A.: Enhancing column generation by a machine-learning-based pricing heuristic for graph coloring. In: AAAI (2022)
33. Tian, H., Medya, S., Ye, W.: Combhelper: a neural approach to reduce search space for graph combinatorial problems. In: AAAI (2024)
34. Tong, H., Faloutsos, C., Pan, J.Y.: Fast random walk with restart and its applications. In: ICDM (2006)
35. Veličković, P., Blundell, C.: Neural algorithmic reasoning. Patterns **2**(7), 100273 (2021)
36. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. In: ICLR (2018)
37. Verhaeghe, H., Cappart, Q., Pesant, G., Quimper, C.G.: Learning precedences for scheduling problems with graph neural networks. In: Constraint Programming (2024)

38. Wang, Y., Huang, H., Rudin, C., Shaposhnik, Y.: Understanding how dimension reduction tools work: an empirical approach to deciphering t-SNE, UMAP, TriMAP, and PaCMAP for data visualization. JMLR **22**(201), 1–73 (2021)
39. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? In: ICLR (2018)
40. Yau, M., Lu, E., Karalias, N., Xu, J., Jegelka, S.: Are graph neural networks optimal approximation algorithms? In: NeurIPS (2024)
41. Zhu, G.: A new view of classification in astronomy with the archetype technique: an astronomical case of the np-complete set cover problem. arXiv preprint arXiv:1606.07156 (2016)

Constrained Machine Learning Through Hyperspherical Representation

Gaetano Signorelli(✉) and Michele Lombardi

University of Bologna, Bologna, Italy
{gaetano.signorelli2,michele.lombardi2}@unibo.it

Abstract. The problem of ensuring constraints satisfaction on the output of machine learning models is critical for many applications, especially in safety-critical domains. Modern approaches rely on penalty-based methods at training time, which do not guarantee to avoid constraints violations; or constraint-specific model architectures (e.g., for monotonocity); or on output projection, which requires to solve an optimization problem that might be computationally demanding. We present the Hypersherical Constrained Representation, a novel method to enforce constraints in the output space for convex and bounded feasibility regions (generalizable to star domains). Our method operates on a different representation system, where Euclidean coordinates are converted into hyperspherical coordinates relative to the constrained region, which can only inherently represent feasible points. Experiments on a synthetic and a real-world dataset show that our method has predictive performance comparable to the other approaches, can guarantee 100% constraint satisfaction, and has a minimal computational cost at inference time.

1 Introduction

Modern machine learning (ML) techniques are widespread in a variety of fields, including some in which the predicted output must satisfy a set of constraints, for consistency (e.g., physical laws) or safety (e.g., autonomous driving) reasons.

State-of-the-art methods are generally based on architectural choices, output projection or penalty-based approaches. Architectural choices can be applied only for specific constraints, satisfied by design by selecting appropriate ML models; for instance, the architectures from [6,19] and [17] can specifically handle monotonicity constraints. Output projection avoids constraint violation by identifying, given a possibly infeasible prediction, the feasible point that has either minimum distance or maximum likelihood. However, this operation requires solving a possibly costly optimization problem. Penalty-based approaches introduce extra terms in the loss function (see e.g. [18]) that penalize constraint violation. These methods incur no inference-time overhead and, under certain assumptions, can guarantee constraint satisfaction. However, guarantees are limited to the training data and do not apply to unseen examples.

In this paper, we propose the *Hyperspherical Constrained Representation*: a novel solution, which can provide feasibility guarantees both in and out of the

G. Tack (Ed.): CPAIOR 2025, LNCS 15763, pp. 209–217, 2025.
https://doi.org/10.1007/978-3-031-95976-9_13

training distribution, for convex and bounded constrained spaces (and generalizable to star domains with some careful design choices). The method relies on a conversion from the canonical Euclidean space to an alternative system inspired by hyperspherical coordinates, designed to be incapable of representing infeasible points. The conversion introduces minimal overhead at inference time and enables training via classical supervised learning.

Our main contribution is the introduction of an original method to enforce constraints in ML for convex and bounded regions that is: (i) capable of providing satisfaction guarantees; (ii) significantly faster than projection-based strategies; and (iii) competitive in terms of accuracy with the alternative approaches.

2 Problem Statement

We focus on a specific class of hard constraints in the output space of a ML model, i.e. those whose conjunction defines a convex and bounded region. The method can also be generalized to star domains (i.e. sets where every point can be connected by a line to a single origin) with some careful design choices.

Formally, let $\mathcal{X} \subseteq \mathcal{R}^k$ and $\mathcal{Y} \subseteq \mathcal{R}^n$ be respectively the input and output space for a ML task; let $\{x_j, y_j\}_{j=1}^N$ be the training set and let $f_\theta : \mathcal{X} \to \mathcal{Y}$ be a function (predictive model) parameterized on θ. Let $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \to \mathcal{R}$ be a loss function. Given a set of m constraints $c_1(y), c_2(y), \ldots, c_m(y)$ defined as convex functions on the output space $\mathcal{Y}$, let:

$$C(y) = \bigwedge_{i=1}^{m} (c_i(y) \leq 0) \tag{1}$$

be the predicate defining the feasible region, which is convex due to the convexity of the $c_i(y)$ functions. Then the problem of learning a model with feasibility guarantees can be formalized as:

$$\arg\min_{\theta} \left\{ \sum_{j=1}^{N} \mathcal{L}(y_j, f_\theta(x_j)) \text{ s.t. } C(f_\theta(x)) \quad \forall x \in \mathcal{X} \right\} \tag{2}$$

where the loss is computed on the training set, but the constraints should hold for every point in the input space.

3 Related Work

A typical approach to enforce constraints in ML models consists in adding penalty terms to the loss function at training time. The extra terms are weighted by λ parameters, which under certain circumstances correspond to Langrangian multipliers [15]. The approach is usually presented as a form of *regularization*. Examples include the regularization technique from [18], which is based on the weighted model count presented by [1]. In [2,16] the penalty terms are derived

via fuzzy logic from first-order logic constraints. There are many other regularization approaches in the literature, e.g. [4] and [9]. Regularization approaches can handle many types of constraints, but they cannot generally guarantee feasibility out of the training distribution.

A second common strategy to enforce constraints consists in projecting the model output y to a feasible point $\hat{y}$ such that the property $p(\hat{y})$ holds. This approach provides feasibility guarantees even on unseen examples, but it involves solving an optimization problem that may be computationally expensive.

A third viable option is to enforce constraints by designing specific ML models and training algorithms. These approaches tend to be restricted to specific constraints classes. Examples include monotonic lattices [6], deep lattice networks [19], and COMET [17], which enforce monotonicity in neural networks; specific classes of safety related constraints are considered in [11,12]. The Multiplexnet from [8] is a much more general example, where a layer is responsible for satisfying constraints in disjunctive normal form (DNF), provided that the individual terms of the disjunction are sufficiently simple. ProbLog [14] exploits its own framework to train neural networks that satisfy constraints, but only discrete variables can be modeled. The DeepSaDe method from [5] relies on a custom network architecture and a training algorithm that solves a Max SMT model at training time, this can guarantee feasibility for a broad class of constraints, but at the cost of a very long training time and a loss in accuracy.

Compared to the previous methods, our approach guarantees constraint satisfaction in and out of distribution with no overhead at inference time, has accuracy on par with the best alternatives, and can be trained via classical supervised learning after a pre-processing step (with small or limited computational cost).

4 Method

We now present the Hyperspherical Constrained Representation (HCR), which exploits a change in representation to enforce constraints. In detail, we apply a transformation in the output space from the canonical Euclidean representation to a coordinate system that spans exactly the feasible region. We propose a representation system inspired by the hyperspherical coordinates, where each point is represented by two elements: an angle and a distance with respect to an origin. Similarly, our hyperspherical system relies on fixing an origin, which has to be feasible, and then representing points in the output space by means of a direction, expressed as a normalized vector, and a distance, expressed as a value in the range $(0, 1)$. This pair determines the relative position of the point along the segment connecting the origin with the frontier of the feasible region. Convexity ensures that every distance value identifies a feasible point, boundedness ensures that a unique point on the frontier is associated to a given direction. No point outside of the frontier can be represented by construction and, since the direction vector is unrestricted, any feasible point can be represented in this fashion. HCR works by predicting directly into the hyperspherical space and then converting back to Euclidean coordinates.

Formalization: Let $O \in \mathcal{R}^n \mid C(O)$ be the origin of the system. Each point $y \in \mathcal{R}^n \mid C(y)$ can be converted to hyperspherical coordinates using a function:

$$\phi(y) : \mathcal{R}^n \rightarrow \mathcal{R}^n \times [0, 1] \tag{3}$$

Let $\phi(y) = (d, r)$, such that $d \in \mathcal{R}^n$ is the point direction and $r \in [0, 1]$ is the point distance; these are defined as:

$$d(y) = \frac{y - O}{\|y - O\|_2}, \quad r(y) = \frac{\|y - O\|_2}{\mathrm{s}(d(y))} \tag{4}$$

where s determines the distance from O and the intersection between the ray identified by $d(y)$ and the frontier of the feasible region. This is defined as:

$$s(d) = \min\{t \mid \exists i \in \{1, 2, \dots m\} \mid c_i(t\, d) = 0\} \tag{5}$$

Conversely, hyperspherical coordinates can be converted into Euclidean ones using the inverse of ϕ:

$$\phi(y)^{-1} : \mathcal{R}^n \times [0, 1] \rightarrow \mathcal{R}^n \tag{6}$$

which is defined as:

$$y = O + d \cdot r \cdot s(d) \tag{7}$$

The bottleneck in both conversion processes is computing $s(d)$, which requires in the worst case to solve m equations in the form $c_i(t\, d) = 0$ in the scalar variable t. For many constraint types, this can be done efficiently via the Brent's, Newton-Raphson's or Halley's method. Notably, since computing intersection is typically less expensive than solving a constrained optimization problem, our method can be expected to be faster than projection at inference time.

A Numerical Example: To better explain our method, we include a simple example of the conversion procedure. Let us assume that $n = 2$ and $m = 1$, with the constrained region being a circular area of radius equal to 10 and centered around the origin $O = (0, 0)$, i.e. $C \equiv \|y\|_2 - 10 \leq 0$. Given a point $y_0 = (5, 0)$, the hyperspherical coordinates are computed by:

1. Compute $d_0 = d(y_0) = y_0/5 = (1, 0)$.
2. Compute $s_0 = s(d_0)$, corresponding to the minimum value t such that $d_0 \cdot t_0$ intersects one constraint in C. In this example, there is a single constraint and the intersection along d_0 is trivially at $S = (10, 0)$, so that $s(d_0) = 10$. Formally, this is obtained by solving the equation $\|O + t \cdot d_0\|_2 - 10 = 0$.
3. Compute $r_0 = r(y_0) = 5/10 = 0.5$.

Hence, the hyperspherical coordinates of y_0 are $((1, 0), 0.5)$. The objects involved in the example are all depicted in Fig. 1. Reconstructing y_0 involves computing the intersection point S again and then applying Eq. (7). Note that, in practice, constraints with a fixed known radius R with respect to the origin, as in this example, can be easily handled as $s = R$ by definition.

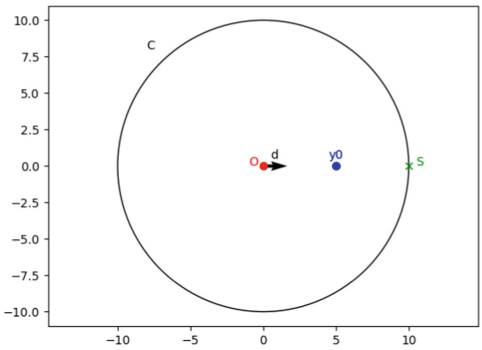


Fig. 1. Example of an instance of conversion. The constrained region is C: a circle centered at $O = (0, 0)$ and radius $R = 10$. The point $y_0 = (5, 0)$ is converted to hyperspherical coordinates $(d = (1, 0), r = 0.5)$ by finding the intersection point S.

Acceleration Technique: In an effort to further speed up the solution of Eq. (5), we employ an acceleration technique. Rather than computing the intersections with all m constraints, we exploit the fact that any point sufficiently far from the original one along the direction d will: (i) violate at least one constraint; (ii) typically violate only a subset of the constraints. Based on this observation, we pick up a point along d and gradually move it far away, via multiplication by a scaling hyperparameter (`BASE-MULTIPLIER`). For a proper choice of the multiplier, a violation will be found in a few iterations; every iteration can be very fast, since checking constraint violation is much faster than computing an intersection. Our experiments show that setting the `BASE-MULTIPLIER` to approximately the radial length of the constrained region reduces the search to $1 - 2$ constraints on average. This technique is described in Algorithm 1, which computes and returns the restricted set of candidate constraints to be checked, in order to identify the intersection point.

Training Process: The training process requires to convert all points in the training set to their corresponding hyperspherical coordinates by means of ϕ. Then, the model can be trained via supervised learning on the hyperspherical space. In case some training points are outside the feasible region, they need to be made feasible (e.g. via projection) before the hyperspherical conversion. While the computational cost of this operation can be non negligible, it is only incurred at training time, where computational resources are typically plentiful and response times are not an issue.

Generalization to Star Domains: The main assumption for HCR requires C to be a convex set, so that every line segment between two points lies inside the feasible region. This assumption allows to arbitrarily choose an origin, guaranteeing one and only one intersection point with the boundaries of C along all the possible directions. However, this property must hold only for the origin. Thus, the convexity assumption can be relaxed in favor of a star domain, assuming

Algorithm 1. Restrict constraints trick

```
function RESTRICT-CONSTRAINTS(O, d)
    i ← 0
    while i < MAX-ITERATIONS do
        mult ← BASE-MULTIPLIER · (1 + 0.5i)
        point ← O + d · mult
        violated ← ENVIRONMENT.violated-constraints(point)
        if length(violated) > 0 then
            return violated
        i ← i + 1
    return ENVIRONMENT.constraints
```

C to be a radially convex set. This relaxation expands the applicability of our method to all the cases where at least one point $s_0 \in C$ is known, such that $\forall s \in C$ the line segment between s_0 and s lies in C.

5 Experiments

We executed experiments on two distinct datasets, comparing our method with the following models: a simple neural network, with no constraint enforcement; a network trained with penalty (Lagrangian) terms, and multiplier calibrated via dual ascent [3]; a network paired with inference-time projection.

Synthetic Benchmark: We first built a synthetic dataset with a single constraint, i.e. a hypersphere in n dimensions, centered at the origin and having a radius R. We generate (feasible) data using different distributions for the training and test set, to reproduce common scenarios where violations may occur when performing out-of-distribution inference. In detail, we sample k features from uniform distributions, with $x_{train} \sim \mathcal{U}(-0.8, 0.8)$ and $x_{test} \sim \mathcal{U}(-1.0, 1.0)$. We subsequently generate the weights matrix $W \in \mathcal{R}^n \times \mathcal{R}^k$ sampling from $\sim \mathcal{U}(-10.0, 10.0)$ and then normalizing so that $\sum_{i=1}^{k} W_{ij} = 1 \quad \forall j \in \{1, 2, \ldots, n\}$. We generate $y_{train} = R \cdot W x_{train}$ and $y_{test} = R \cdot W x_{test}$. This procedure creates samples inside the hypercube circumscribed to the hypersphere. All the points outside of the feasible region are then projected inside it by solving: $\arg\min_{\hat{y}} \|\hat{y} - y\|_2 \mid \|\hat{y}\|_2 < R$. In all our experiments we have $R = 10$, $k = 128$ and $n = 768$, to stress both the computation cost and accuracy of the methods on complex scenarios. We use 500 samples for training and 1000 for testing.

M4 Forecasting Competition Dataset: As a second dataset, we use real time-series from the M4 forecasting competition [13]. This dataset is made up of $100k$ time series, split in temporal categories (hourly, daily, weekly, quarterly and yearly). The tasks consists in predicting values in an output window (with size n), based on observed values from an input window (with size k). We impose a max-deviation constraint between consecutive points in the prediction window, i.e. such that $|v_i - v_{i+1}| \leq d_{max} \forall i \in \{1, 2, \ldots, n-1\}$. The value d_{max} corresponds to the largest deviation between two consecutive values in the training

set. Similarly, we impose an upper and lower bound for each value, so to define a convex constrained area in the form of a polytope in $\mathcal{R}^n$. Our focus for these experiments is on violations that might occur from out-of-distribution inference, rather than from a lack of alignment between the constraints and the data distribution. For this reason, we preprocess all time series via projection to ensure that the constraints are satisfied also in test data. We use 20% of the data as training set to encourage overfitting and constraint violations. We set $k = n$, while n is the same used in [13] and it depends on the temporal split.

Table 1. Results on synthetic dataset. Mean-squared-error, percentage of feasible outputs, average and maximum post-processing time (projection or hyperspherical conversion) are reported for the test set.

Method	MSE	Inside ratio	Avg. time	Max time
Simple	0.071 ± 0.001	0.341 ± 0.009	*NA*	*NA*
Lagrangian	0.068 ± 0.018	0.912 ± 0.029	*NA*	*NA*
Projection	0.050 ± 0.001	1.000 ± 0.000	0.0700 ± 0.001	0.1250 ± 0.004
HCR	$\mathbf{0.010 \pm 0.002}$	1.000 ± 0.000	$\mathbf{0.0001 \pm 0.000}$	$\mathbf{0.0002 \pm 0.000}$

Results: For both datasets, we implement a simple neural network consisting of an encoding layer and a linear regression head for all the models, except for ours, having two regression heads: one for d, followed by a normalization step; the other for r, followed by a sigmoid function. We train all the models using the Adam algorithm [10] and the MSE loss function. We use a single feed-forward layer for the synthetic case and a Long Short-term Memory model [7] for the M4 dataset as encoding layers. We apply data standardization to target values for all the models except ours; x is also standardized for the M4 dataset. We repeated the experiments using 10 different seeds for the synthetic dataset, and 30 different time series for the M4 dataset. For this second dataset, we only report results based on the higher dimensional (most critical) temporal category (hourly), having $k = n = 48$ and $m = 190$. Results are shown in Tables 1 and 2.

Experiments show similar performances in terms of MSE accuracy compared to the other methods; particularly for the synthetic dataset, where our method outperforms the baselines. Moreover, they highlight the gap in post-processing time between our method and projection: the HCR takes a negligible amount of time, with low variations depending on n, m and the structure of the feasible region; projection takes a highly variable time, mostly depending on the nature of the given constraint, up to 700 times higher than our approach for a single quadratic constraint in a high-dimensional space.

Table 2. Results on M4 dataset. Relative mean-squared-error, percentage of feasible outputs, average and maximum post-processing time (projection or hyperspherical conversion) are reported for the test set.

Method	R-MSE	Inside ratio	Avg. time	Max time
Simple	0.125 ± 0.051	0.894 ± 0.111	*NA*	*NA*
Lagrangian	0.191 ± 0.050	0.980 ± 0.041	*NA*	*NA*
Projection	$\mathbf{0.124 \pm 0.049}$	1.000 ± 0.000	0.001 ± 0.001	0.004 ± 0.011
HCR	0.130 ± 0.047	1.000 ± 0.000	$\mathbf{0.0001 \pm 0.000}$	$\mathbf{0.001 \pm 0.000}$

6 Conclusions

We have introduced a novel method to tackle constrained machine learning for convex and bounded feasible regions in output space. Our method guarantees constraints satisfaction using a conversion at inference time that requires, both theoretically and empirically, a negligible amount of time, outperforming projection methods, while still providing a comparable predictive accuracy. We believe that the HCR method could be useful in settings such as control applications, where safe predictions are required and the limited resources make the use of projection difficult. Furthermore, our method combined with a common root-finding method (e.g., Newton-Raphson) would be fully differentiable and it could be used in pipelines where constraints are required in intermediate steps (i.e., embeddings). As a future work, we plan to further explore use cases for the discussed scenarios and directions to expand the method for non-convex regions.

References

1. Chavira, M., Darwiche, A.: On probabilistic inference by weighted model counting. Artif. Intell. **172**(6–7), 772–799 (2008). https://doi.org/10.1016/J.ARTINT.2007.11.002
2. Diligenti, M., Gori, M., Sacca, C.: Semantic-based regularization for learning and inference. Artif. Intell. **244**, 143–165 (2017). https://doi.org/10.1016/J.ARTINT.2015.08.011
3. Fioretto, F., Van Hentenryck, P., Mak, T., Tran, C., Baldo, F., Lombardi, M.: Lagrangian duality for constrained deep learning. In: Dong, Y., Ifrim, G., Mladenić, D., Saunders, C., Van Hoecke, S. (eds.) ECML PKDD 2020. LNCS (LNAI), vol. 12461, pp. 118–135. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-67670-4_8
4. Fischer, M., Balunovic, M., Drachsler-Cohen, D., Gehr, T., Zhang, C., Vechev, M.: Dl2: training and querying neural networks with logic. In: International Conference on Machine Learning, pp. 1931–1941. PMLR (2019)
5. Goyal, K., Dumancic, S., Blockeel, H.: Deepsade: learning neural networks that guarantee domain constraint satisfaction. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 38, pp. 12199–12207 (2024). https://doi.org/10.1609/AAAI.V38I11.29109

6. Gupta, M., et al.: Monotonic calibrated interpolated look-up tables. J. Mach. Learn. Res. **17**(109), 1–47 (2016)
7. Hochreiter, S.: Long short-term memory. Neural Computation. MIT-Press (1997)
8. Hoernle, N., Karampatsis, R.M., Belle, V., Gal, K.: Multiplexnet: towards fully satisfied logical constraints in neural networks. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 36, pp. 5700–5709 (2022). https://doi.org/10.1609/AAAI.V36I5.20512
9. Hu, Z., Ma, X., Liu, Z., Hovy, E., Xing, E.: Harnessing deep neural networks with logic rules. arXiv preprint arXiv:1603.06318 (2016). https://doi.org/10.18653/V1/P16-1228
10. Kingma, D.P.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
11. Leino, K., Fromherz, A., Mangal, R., Fredrikson, M., Parno, B., Păsăreanu, C.: Self-correcting neural networks for safe classification. In: International Workshop on Numerical Software Verification, pp. 96–130. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-21222-2_7
12. Lin, X., Zhu, H., Samanta, R., Jagannathan, S.: Art: abstraction refinement-guided training for provably correct neural networks. In: Formal Methods in Computer Aided Design. vol. 1, pp. 148–157. TU Wien Academic Press (2020). https://doi.org/10.34727/2020/ISBN.978-3-85448-042-6_22
13. Makridakis, S., Spiliotis, E., Assimakopoulos, V.: The m4 competition: 100,000 time series and 61 forecasting methods. Int. J. Forecast. **36**(1), 54–74 (2020)
14. Manhaeve, R., Dumančić, S., Kimmig, A., Demeester, T., De Raedt, L.: Neural probabilistic logic programming in deepproblog. Artif. Intell. **298**, 103504 (2021). https://doi.org/10.1016/J.ARTINT.2021.103504
15. Rockafellar, R.T.: Lagrange multipliers and optimality. SIAM Rev. **35**(2), 183–238 (1993). https://doi.org/10.1137/1035044
16. Serafini, L., Garcez, A.: Logic tensor networks: deep learning and logical reasoning from data and knowledge. arXiv preprint arXiv:1606.04422 (2016)
17. Sivaraman, A., Farnadi, G., Millstein, T., Van den Broeck, G.: Counterexample-guided learning of monotonic neural networks. Adv. Neural. Inf. Process. Syst. **33**, 11936–11948 (2020)
18. Xu, J., Zhang, Z., Friedman, T., Liang, Y., Broeck, G.: A semantic loss function for deep learning with symbolic knowledge. In: International Conference on Machine Learning, pp. 5502–5511. PMLR (2018)
19. You, S., Ding, D., Canini, K., Pfeifer, J., Gupta, M.: Deep lattice networks and partial monotonic functions. In: Advances in Neural Information Processing Systems, vol. 30 (2017)

PySCIPOpt-ML: Embedding Trained Machine Learning Models into Mixed-Integer Programs

Mark Turner[1(✉)], Antonia Chmiela[1], Thorsten Koch[1,2], and Michael Winkler[3]

[1] Zuse Institute Berlin, Berlin, Germany
{turner,chmiela,koch}@zib.de
[2] Institute of Mathematics, Technische Universität Berlin, Berlin, Germany
[3] Gurobi GmbH, Ulmenstr. 37-39, 60325 Frankfurt am Main, Germany
winkler@gurobi.com

Abstract. A standard tool for modelling real-world optimisation problems is mixed-integer programming (MIP). However, for many of these problems, information about the relationships between variables is either incomplete or highly complex, making it difficult or even impossible to model the problem directly. To overcome these hurdles, machine learning (ML) predictors are often used to represent these relationships and are then embedded in the MIP as surrogate models. Due to the large amount of available ML frameworks and the complexity of many ML predictors, formulating such predictors into MIPs is a highly non-trivial task. In this paper, we introduce PySCIPOpt-ML, an open-source tool for the automatic formulation and embedding of trained ML predictors into MIPs. By directly interfacing with a broad range of commonly used ML frameworks and an open-source MIP solver, PySCIPOpt-ML provides a way to easily integrate ML constraints into optimisation problems. Alongside PySCIPOpt-ML, we introduce SurrogateLIB, a library of MIP instances with embedded ML constraints, and present computational results over SurrogateLIB, providing intuition on the scale of ML predictors that can be practically embedded. The project is available at https://github.com/Opt-Mucca/PySCIPOpt-ML.

1 Introduction and Related Work

Many problems coming from real-world applications can be modelled using *Mixed-Integer Programming (MIP)*, which can be expressed as follows:

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & f_0(x) \\ \text{s.t.} \quad & f_i(x) \leq 0, \ \forall i \in [m], \\ & x_i \in \mathbb{Z}, \ \forall i \in \mathcal{I}. \end{aligned} \tag{P}$$

Each function $f_i : \mathbb{R}^n \to \mathbb{R}$ for $i \in [m]$ is continuous, and $\mathcal{I} \subseteq [n]$ denotes the index set of the integer-constrained variables. Due to a MIP's ability to represent

G. Tack (Ed.): CPAIOR 2025, LNCS 15763, pp. 218–234, 2025.
https://doi.org/10.1007/978-3-031-95976-9_14

complex systems and decision-making processes across various industries, MIP has become a standard tool to solve optimisation problems arising in real-world applications [2,50].

A common challenge of finding a suitable MIP formulation (P) of a system is the presence of unknown or highly complex relationships. This poses a problem since traditional MIP approaches rely on precise definitions of constraints and the objective function. To address this challenge, *Machine Learning (ML)* predictors are often used to approximate these relationships and are then embedded in (P) as surrogate models. To embed a trained ML predictor $g : \mathbb{R}^n \to \mathbb{R}$, it must first be formulated as a series of MIP constraints before it can be added to (P). However, with the growing popularity and accessibility of different ML frameworks [1,6,14,31,47,48], automatically formulating trained ML predictors coming from these different architectures into MIPs has become a highly non-trivial, yet necessary task.

Contribution. In this paper, we introduce PySCIPOpt-ML, an open-source tool for the automatic formulation and embedding of trained ML predictors into MIPs. By directly interfacing with a broad range of commonly used ML frameworks, PySCIPOpt-ML allows for the easy integration of ML constraints into MIPs, which can then be solved by the state-of-the-art open-source solver SCIP [10]. The package supports various models from Scikit-Learn [48], XGBoost [14], LightGBM [31], Keras [15], PyTorch [47], and ONNX [6], making it a general tool to easily optimise MIPs with embedded ML constraints (see Fig. 1). To represent predictors as MIPs, we use formulations based on SOS1 constraints [19], which admit valid formulations in the absence of user specified bounds. Alongside PySCIPOpt-ML, we introduce SurrogateLIB, an expandable library of MIP instances with embedded ML constraints based on real-world data. Computational results over this library of instances are presented, aiming to provide intuition on the scale of ML predictors that can be practically embedded.

To summarise, our contributions are the following:

1. We introduce the **Python package PySCIPOpt-ML**[1], which **directly interfaces various ML frameworks with the open-source solver SCIP** (see Sect. 2).
2. We utilise **MIP formulations prioritising practicality**, since they do not rely on user defined bounds or expensive bound propagation techniques (see Sect. 3)
3. We introduce a **new library of MIP instances with embedded ML constraints called SurrogateLIB** [55], as well as the corresponding instance generators (see Sect. 4).
4. We present **computational experiments showcasing the scale of predictors** that can be practically embedded (see Sect. 5).

Related Work. In recent years, more and more industries have leveraged ML frameworks to approximate unknown or complex relationships in optimisation

[1] https://github.com/Opt-Mucca/PySCIPOpt-ML.

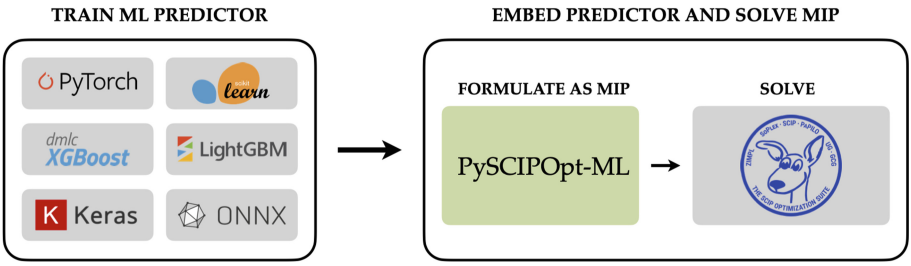


Fig. 1. How trained ML predictors can be automatically formulated as MIPs and embedded in an optimisation problem using our Python package PySCIPOpt-ML.

problems. For instance, ML predictors have been used to optimise cancer treatment plans [8], minimise energy consumption [44], and in the design of energy systems [30]. Embedded ML predictors have also been used in the optimisation of gas production and water management systems [41], as well as in the optimisation of retail prices [18] and locations [28], to name only a few examples. For surveys of embedded ML predictors through surrogate optimisation, see [9,35,43].

The increasing demand for embedding ML predictors into MIPs has motivated the development of various tools for their automatic integration [37]. The commercial MIP solver Gurobi [25] released the Python package Gurobi Machine Learning [24], which integrates trained regression models into optimisation problems that can then be solved with Gurobi. Other packages that rely on Gurobi are OptiCL [42], which supports many Scikit-Learn predictors [48], and Janos [7], which supports linear and logistic regression, as well as neural networks with rectified linear unit (ReLU) activation functions. EML [36] in contrast uses CPLEX [17], and enables the embedding of decision trees and neural networks through the Keras API [15]. ENTMOOT [53] focuses on representing gradient-boosted tree models from LightGBM [31]. ReluMIP [40] allows the user to integrate ReLU neural networks trained with TensorFlow [1] within optimisation problems modelled with either Pyomo [11] or Gurobi. MeLon [51] primarily focuses on embedding neural networks, support vector machines, and Gaussian processes in C++, but via read and write functionality has limited Python support to Keras and Scikit-Learn. The newest alternative, MathOptAI [38], is based in the JuMP [39] ecosystem for Julia, and supports binary decision trees, Gaussian processes, and a wide range of activation function for neural networks with an interface to PyTorch. Finally, OMLT [3,13,57] focuses on embedding a wide variety of neural networks and gradient-boosted trees via the general MIP modelling framework Pyomo and the general purpose ML interface ONNX [6].

While many of these packages focus on supporting specific types of predictors, PySCIPOpt-ML allows the user to embed a wide variety of predictors. Specifically, for both regression and classification tasks: neural networks with various activation functions, linear and logistic regression models, decision trees, gradient boosted trees, random forests, support vector machines, and centroid clustering.

In addition, unlike the other packages, PySCIPOpt-ML interfaces directly with an open-source MIP solver. This allows users to dynamically and transparently improve the solving process, e.g., by dynamic bound tightening [27]. However, we note that the functionality of the JuMP ecosystem enables MathOptAI to approach a similar level of functionality. Using such a tool also implicitly comes with structure detection of the embedded predictor, which removes the hurdle of identifying exactly what constraints belong to the embedded predictor. As a consequence, users can focus entirely on the development of special methods to handle these constraint types.

Even though this direct interface is one of the biggest advantages of PySCIPOpt-ML, we would like to note here that any MIP formulation can be easily extracted by writing it to a file[2]. This file can then be optimized with any solver with PySCIPOpt-ML acting as a solver-independent modelling tool.

2 About the Package

PySCIPOpt-ML directly interfaces with the popular ML frameworks Scikit-Learn [48], XGBoost [14], LightGBM [31], Keras [15], PyTorch [47], and the open format ONNX [6]. Each framework allows the user to train ML predictors of different types, which can then be embedded into a MIP by our package. A detailed overview of what PySCIPOpt-ML supports is given in Table 1. We note that embedding more advanced ML predictors, such as transformers and graph neural networks, is a highly non-trivial task as they are typically programmed with custom forward functions. PySCIPOpt-ML currently only supports ML predictors that have clearly defined evaluation functions provided directly by the ML framework itself.

An advantage of PySCIPOpt-ML is that it seamlessly joins all parts of the modelling, training, and optimisation processes. At no point is there a need to export the ML predictor or MIP model into some intermediary language or framework. A central function (`add_predictor_constr`) acts as the main interface between the ML frameworks training the predictor and the solver SCIP solving the problem. To create and solve a MIP problem with an embedded ML predictor, the user must only do the following:

1. Train the ML predictor, `predictor`, using one of the supported ML frameworks.
2. Define the optimisation problem, `scip_model`. This involves adding the constraints and objective not directly related to the ML predictor.
3. Create variables that represent `(input_data,output_data)`. The output variables are optional as they can be inferred by the ML predictor. Embed the predictor into the MIP by simply calling:

```
add_predictor_constr(scip_model, predictor, input_data, output_data).
```

4. Solve the updated `scip_model` with SCIP.

[2] Simply call the function `scip_model.writeProblem()` where `scip_model` refers to the MIP model.

Figure 2 visualises this workflow.

Table 1. Overview of ML predictors supported by each automatic embedding framework. The following abbreviations are used: SCIP is PySCIPOpt-ML, EML is Emperical Machine Learning, ENT is ENTMOOT, GRB is Gurobi Machine Learning, JAN is Janos, MOA is MathOptAI, MEL is MeLon, OCL is OptiCL, OMLT is OMLT, and RMIP is reluMIP. Predictor-framework pairs that do not exist or are not supported by any tool are indicated by "-".

ML Predictor	PyTorch	Tensorflow/Keras	Scikit-Learn	LightGBM	XGBoost
Decision Trees	-	-	**SCIP**/EML/GRB/ OCL/OMLT	**SCIP**/EML/ GRB/OMLT	**SCIP**/GRB/OMLT
Gradient Boosted Trees	-	-	**SCIP**/GRB/OCL/ OMLT	**SCIP**/ENT/ GRB/OMLT	**SCIP**/GRB/OMLT
Random Forests	-	-	**SCIP**/GRB/OCL/ OMLT	**SCIP**/ENT/ GRB/OMLT	**SCIP**/GRB/OMLT
Neural Networks (ReLU)	**SCIP**/GRB/ MOA/OMLT/	**SCIP**/EML/GRB/ MEL/OMLT/RMIP	**SCIP**/GRB/JAN/ OCL/OMLT	-	-
Neural Networks (other)	**SCIP**/MOA/OMLT	**SCIP**/EML/ MEL/OMLT	**SCIP**/OMLT	-	-
Linear Regression	-	-	**SCIP**/GRB/JAN/ OCL	-	-
Logistic Regression	-	-	**SCIP**/GRB/JAN/ OCL	-	-
Support Vector Machines	-	-	**SCIP**/MEL/OCL	-	-
Centroid Clustering	-	-	**SCIP**	-	-

*For all supported frameworks and models PySCIPOpt-ML offers multi-regression and multi-classification.

In summary, PySCIPOpt-ML helps to easily create, embed, and solve optimisation problems with ML surrogate models and spares the user from navigating multiple platforms at the same time.

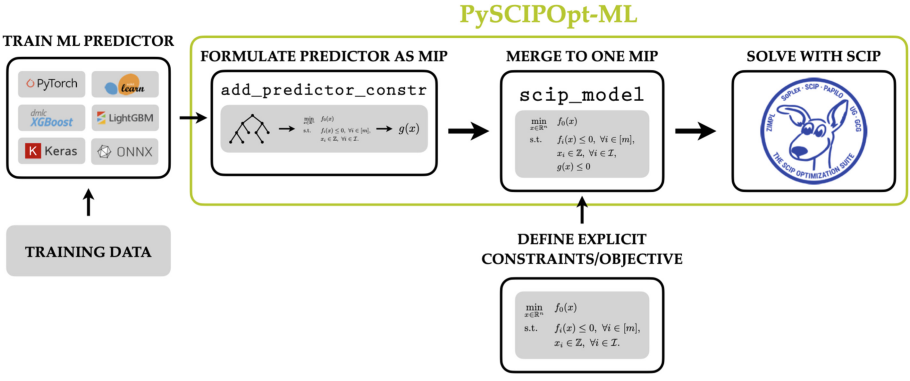


Fig. 2. The workflow when embedding ML predictors into a MIP with PySCIPOpt-ML.

3 MIP Formulations of ML Predictors

To embed an ML predictor in a MIP, we first need to represent the predictor itself as a MIP. The formulation of the predictor has a significant effect on the ability to efficiently optimise the larger optimisation problem. There has been a substantial amount of work on determining the best performing formulations for popular ML predictors. In particular, neural networks with ReLU activation functions [4,23,33,51] and ensemble based methods that use decision trees [3,24,45,53] have received a lot of attention in the past. Often, the most natural formulation of these predictors involves quadratic constraints. Due to performance issues, however, formulations that utilise big-M and indicator constraints are in general preferred. In the remainder of this section, we will give an overview of how PySCIPOpt-ML models different ML predictors as MIPs.

3.1 Neural Networks

The current standard MIP formulation for feedforward neural networks with ReLU activation functions uses big-M constraints [4,5,13,23,29]. These constraints are used by all alternative packages listed in Table 1 to embed neural networks, although OMLT does offer a complementary constraint based formulation in addition to big-M formulations. Consider a single neuron in a fully connected hidden-layer k, where $k \in \mathbb{N}$. Let $x \in \mathbb{R}^n$ be the output of the previous layer $k-1$, $y \in \mathbb{R}_{\geq 0}$ be the output of the given neuron, and $w \in \mathbb{R}^n$ and $b \in \mathbb{R}$ be the weights and bias connecting the previous layer to the neuron. The standard big-M formulation requires bounds $l, u \in \mathbb{R}$, where $l \leq \sum_{i=i}^{n} w_i x_i + b \leq u$, and a binary variable for the neuron, namely $z \in \{0, 1\}$. The formulation is defined as:

$$
\begin{aligned}
& y \geq 0, \\
& y \geq \sum_{i=1}^{n} w_i x_i + b, \\
& y \leq \sum_{i=1}^{n} w_i x_i + b - l(1 - z), \\
& y \leq uz.
\end{aligned}
$$

Big-M formulations are prone to numerical issues and increased running time when the bounds on the input are large, and thus benefit greatly from tighter bounds [12,32]. For this reason, when optimising MIPs with embedded neural networks, either feasibility based or optimality based bound tightening is typically performed prior to optimisation [13,23]. It is therefore important that the user specifies bounds on the variables of the wider optimisation problem. Since this is not always possible, PySCIPOpt-ML gives the user the option to decide between the classical big-M formulation or a novel SOS1-based formulation to represent ReLU activation functions. By doing the latter, we aim to maintain a valid formulation even in the absence of variable bounds, and minimise the pegging of solver performance to bound tightening procedures in the case of provided

bounds. The downside of the SOS1-based formulation is that the SOS-constraint does not feature in the LP relaxation and must be enforced by branching [19]. However, they are still widely used in practice, e.g., in piece-wise approximating non-linear functions for gas transport [46] and survivable network design for telecommunication networks [54].

Our SOS1 formulation for an intermediate neuron with ReLU activation function introduces $s \in \mathbb{R}_{\geq 0}$ as a slack variable for the neuron. The formulation is defined as:

$$\begin{aligned} & y = \sum_{i=1}^{n} w_i x_i + b + s, \\ & \text{SOS1}(s, y), \\ & y \geq 0. \end{aligned}$$

An SOS1 constraint enforces that at most one variable in the constraint can be non-zero. The above formulation ensures that the slack is only non-zero when $\sum_{i=1}^{n} w_i x_i + b$ is negative. In this case, the slack must also take exactly the value of $-b - \sum_{i=1}^{n} w_i x_i$, as the SOS1 constraint then forces y to 0.

To illustrate the advantages of the SOS1 formulation, we present an example comparing it to the standard big-M formulation. Assume we have $l = -10^9$, $u = 10^9$, and that the expression $\sum_{i=1}^{n} w_i x_i + b$ evaluates to 0. We can consider the case that $z = 10^{-6}$, which is within its integer feasibility tolerance of 0, and would be valid from the solver's perspective. Consider the constraints however:

$$\begin{aligned} & y \geq 0 \\ & y \geq 0 \\ & y \leq 0 + 10^9(1 - 10^{-6}) \\ & y \leq 10^9 * 10^{-6} \end{aligned}$$

In the big-M formulation for this poorly scaled example, the output y is in the range $[-\epsilon, 10^3 + \epsilon]$, where $\epsilon \in [0, 10^{-6}]$ is the constraint feasibility tolerance of the MIP solver. Thus, the output of the neural network for the MIP can vary greatly compared to its actual prediction from the same input. Meanwhile, due to our SOS1 formulation containing no big-M constraints, the error of the individual node's prediction is bounded by the feasibility tolerance value. We note however, that in practice such an occurrence is uncommon, since it occurs only when the LP solution happens to be integer feasible and the binary variable is not yet fixed. We observed that when solving MIPs with embedded ML predictors using SCIP [10], most binary variables are resolved through branching, with the MIP solver fixing the binary variable and not allowing any epsilon changes to its value.

While we have focused on the ReLU activation function formulation, we note that PySCIPOpt-ML also supports other activation functions such as sigmoid, and tanh. As PySCIPOpt-ML uses SCIP as a solver, these formulations can be added directly as non-linear expressions, requiring no reformulation or linearisation.

3.2 Tree-Based Predictors

To model tree-based predictors like decision trees, gradient boosted trees, or random forests, PySCIPOpt-ML uses the formulation first introduced in Gurobi Machine Learning [24]. As opposed to quadratic formulations [45] and linear big-M formulations [3], the Gurobi Machine Learning formulation uses a series of indicator constraints for each leaf node ensuring that any input maps to the appropriate leaf node of the tree. Currently, MathOptAI also adopts the Gurobi Machine Learning formulation, while other packages listed in Table 1 use big-M and quadratic formulations. In what follows, we will give an overview of how decision trees are modelled in PySCIPOpt-ML. Note that the MIP formulations of gradient boosted trees and random forests are simply a linear combination of the individual decision tree formulations.

For a decision tree, we denote the set of leaves as $\mathcal{L} \subseteq [m]$, the input to the tree as $x \in \mathbb{R}^n$, where $m \in \mathbb{N}$ is the number of nodes of the tree and $n \in \mathbb{N}$ is the number of features, and the output of the tree as $y \in \mathbb{R}$. Each leaf $l \in \mathcal{L}$ is defined by a number of constraints on the input features of the tree that correspond to the branches taken in the path leading to l. Additionally, each leaf $l \in \mathcal{L}$ has an associated output value $\gamma_l \in \mathbb{R}$. We formulate decision trees by introducing one binary decision variable $\delta_l \in \{0, 1\}$ for each leaf of the tree, which indicates whether the given input maps to l (i.e., $\delta_l = 1$) or not (i.e., $\delta_l = 0$). Since in a decision tree exactly one leaf is chosen, we have to add the following constraint:

$$\sum_{l \in \mathcal{L}} \delta_l = 1.$$

To ensure that the input vector maps to the correct leaf, however, we need to introduce additional notation and constraints. For a node $v \in [m] \setminus \mathcal{L}$ in the tree, we denote by $i_v \in [n]$ the feature used for splitting, i.e., making the decision, and by $\theta_v \in \mathbb{R}$ the value at which the split is made. At a leaf $l \in \mathcal{L}$, we have a set $\mathcal{L}_l$ of inequalities of the form $x_{i_v} \leq \theta_v$ corresponding to the left branches leading to l and a set $\mathcal{R}_l$ of inequalities of the form $x_{i_v} > \theta_v$ corresponding to the right branches. For each leaf, the inequalities describing $\mathcal{L}_l$ and $\mathcal{R}_l$ are imposed with indicator constraints modelling the following relationships:

$$\begin{aligned} \delta_l = 1 &\implies x_{i_v} \leq \theta_v - \frac{\epsilon}{2}, \quad \forall x_{i_v} \leq \theta_v \in \mathcal{L}_l, \\ \delta_l = 1 &\implies x_{i_v} \geq \theta_v + \frac{\epsilon}{2}, \quad \forall x_{i_v} > \theta_v \in \mathcal{R}_l. \end{aligned}$$

In our implementation, $\epsilon \geq 0$ can be specified by a keyword parameter `epsilon` in the central function `add_predictor_constr` for embedding a trained predictor as described in Sect. 2. By default, PySCIPOpt-ML sets $\epsilon = 0$. Note that when ϵ is smaller than the default feasibility tolerance in SCIP[3] (as it is by default), and you have a solution where $|x_{i_v} - \theta_v| \approx \epsilon$, then SCIP can select an arbitrary child node of that decision in the tree. Note also that larger values for ϵ risk removing

[3] In SCIP 9.1.1 the feasibility tolerance is set to 10^{-6}.

solutions containing values $x_{i_v} \approx \theta_v$. These removed solutions can potentially change the optimal solution of the instance or make the instance infeasible.

The final output of the tree, y, is then modelled via an indicator constraint for each leaf $l \in \mathcal{L}$:

$$\delta_l = 1 \implies y = \gamma_l.$$

3.3 Argmax Formulation

For many classification tasks, the ML predictor's evaluation is determined by an argmax function call. That is, if each class j from the index set $\mathcal{J}$ has some predicted score $y_j \in \mathbb{R}$, the predictor outputs the class with the highest score. To model this functionality with MIP, we define a variable $m \in \mathbb{R}$, binary variables $z_j \in \{0, 1\}, j \in \mathcal{J}$, and slack variables $s_j \geq 0, j \in \mathcal{J}$. The formulation we use is then given by:

$$\begin{aligned} & y_j + s_j - m = 0, \ \forall j \in \mathcal{J} \\ & \text{SOS1}(z_j, s_j), \ \forall j \in \mathcal{J} \\ & \sum_{j=1}^{|\mathcal{J}|} z_j = 1. \end{aligned}$$

In the above formulation, the variable m implicitly represents $\max\{y_j : j \in \mathcal{J}\}$: If m is less than the maximum, there exists a $j' \in \mathcal{J}$ such that $y_{j'} \geq m$, and for which $y_{j'} + s_{j'} - m = 0$ cannot be valid. On the other hand, if m is greater than the maximum, then all slack variables are non-zero, and $\sum_{i=1}^{|\mathcal{J}|} z_j = 1$ cannot be satisfied due to the SOS1 constraints. Because m is the maximum, exactly one $z_{j'}$ is set to 1, where $y_{j'} = \max\{y_j : j \in \mathcal{J}\}$.

To truly return the argmax instead of a binary variable corresponding to the location of the maximum value, one can introduce a variable $a \in \mathbb{Z}$, and the corresponding constraint

$$a = \sum_{j=1}^{|\mathcal{J}|} j * z_j.$$

3.4 Centroid Clustering

To model centroid-clustering-based predictors like those trained via k-means clustering [26], let us denote by $d_j \in \mathbb{R}$ the distance variables between the input vector $x \in \mathbb{R}^n$ and the centroid $c^j \in \mathbb{R}^n$ of cluster $j \in \mathcal{J}$. Using the squared L2-norm, we therefore have:

$$d_j = \sum_{i=1}^{n} (x_i - c_i^j)^2, \ \forall j \in \mathcal{J}.$$

The output variable $y \in \{0,1\}^{|\mathcal{J}|}$ is then a binary vector modelling argmin(d):

$$y_j = \begin{cases} 1, & \text{if } j = \text{argmin}(d), \\ 0, & \text{otherwise.} \end{cases}, \ \forall j \in \mathcal{J}.$$

This relation is modelled using the argmax-function on the negative values (see Sect. 3.3).

Since this approach is quite flexible, we can accommodate any distance measure. To provide a linearized version, PySCIPOpt-ML offers the option to use the L1-norm instead, although it should be noted that points can be misclassified when using this formulation as it is an approximation. To model this, let $\bar{d}^j, \underline{d}^j \in \mathbb{R}^n_{\geq 0}$ be variables such that $\bar{d}^j - \underline{d}^j$ denotes the difference per dimension between $x \in \mathbb{R}^n$ and a given centroid $c^j \in \mathbb{R}^n$, $j \in \mathcal{J}$. The formulation is then

$$\begin{aligned} &\bar{d}^j - \underline{d}^j = x - c^j, \ \forall j \in \mathcal{J}, \\ &SOS1(\bar{d}^j_i, \underline{d}^j_i), \ \forall i \in [n], j \in \mathcal{J}, \\ &d_j = \sum_{i=1}^{n} (\bar{d}^j_i + \underline{d}^j_i), \ \forall j \in \mathcal{J}. \end{aligned}$$

4 Instance Library

Accompanying PySCIPOpt-ML, we present a new instance library, SurrogateLIB [55], consisting of MIPs that contain embedded ML predictors as either constraints or components of the objective function.

SurrogateLIB is motivated by the large availability of data in the ML community and the need for more homogeneous non-trivial instance sets in the MIP community. Its goal is to be a library of instances with controllable complexity that is representative of MIPs with embedded ML predictors. To achieve this, semi-realistic MIP scenario generators are constructed in an array of application areas, including auto manufacturing, water treatment, and adversarial attacks, where real-world data is readily available for training the embedded ML predictors. By avoiding synthetically generated data to train the ML predictors, we aim to showcase the practical benefits of embedding predictive models in optimisation problems. The current iteration of SurrogateLIB contains 1016 instances.

Alongside the library, we also include instance generators of the introduced problems that allow users to create homogeneous instances of varying difficulty. All generators feature two random seeds as common arguments. One is for randomising the ML predictor's training, and the other is for randomising some constraints of the MIP formulation. Thus, PySCIPOpt-ML and SurrogateLIB present an opportunity to generate semi-realistic MIPs of controllable complexity by adjusting the size of the embedded ML predictors as well as the parameters of the instances. This way, we provide a set of homogeneous instances that are sufficiently diverse and relatively easy to solve without being trivial. Table 2 presents a summary for the current generators in SurrogateLIB.

Table 2. Overview of instance generators available in SurrogateLIB. *Output type* is labelled as (M)ulti-(R)regression or (C)lassification. *#(Fixed) Input* refers to the number of MIP variables that are input to each embedded ML predictor, both for those that can take free values and for those that are fixed. For example, in the tree planting instance, of the seven input features to each of the tree survival rate ML predictors, six are fixed by properties of the instance and one is free to be changed by the solver. *#ML Pred.* is the number of embedded ML predictors by default in the generator.

Example Origin	Output Type	#(Fixed) Input	#ML Pred.	Brief Description
AdversarialAttack [22,34]	MR	(0)/256	1	Maximise chance of incorrect image classification
Auto Manufacturer[a]	R	(0)/10	3	Maximise sales of sufficiently different car
City Planning[b]	R	(7)/14	25	Maximise living quality of new residents
FunctionApproximation [23]	R	(0)/5	2	Maximise function while satisfying equality constraints
Palatable DietProblem [42,49]	R	(2)/25	1	Minimise transport cost while meeting nutritional/palatability requirements
Tree Planting [56]	R	(6)/7	100	Maximise tree survival rates
Water Potability[c]	C	(0)/9	50	Maximise amount of drinkable water after treatment
Wine Quality [16]	R	(0)/11	5	Maximise quality of mixed wine from suppliers
WorkloadDispatching [36]	R	(0)/3	48	Maximise minimum efficiency of compute cores

[a] https://www.kaggle.com/datasets/gagandeep16/car-sales.
[b] https://www.kaggle.com/datasets/krzysztofjamroz/apartment-prices-in-poland.
[c] https://github.com/MainakRepositor/Datasets/tree/master.

5 Computational Experiments

One question that often arises when practitioners deal with surrogate models is the following: How large of a ML predictor can actually be embedded without blowing up solution time? To answer this question we will present a set of experiments aiming to provide some intuition for the scale of embedded ML predictors that practitioners can expect to optimise over.

To determine the impact of embedding a ML predictor on the solution time, we use SurrogateLIB and progressively embed larger ML predictors until a 30 min time limit is reached. We embed fully connected feedforward neural networks with ReLU activation functions and gradient-boosted decision trees (GBDT). For the neural network, we test both the SOS1 and big-M formulation, where

the hidden layer size is fixed to either 16 or 32. For GBDTs we fix the maximum depth of each estimator to either 2 or 5, and use an $\epsilon = 0.0001$ in the formulation. Increasing the size of each embedded ML predictor then corresponds to either introducing an additional hidden layer of fixed size or an additional estimator of fixed depth. The function approximation problem has been removed so as to leave only semi-real-world examples. GBDTs are not used for the image based adversarial attack problem and GBDTs of depth 5 are not used for the workload dispatching problem due to the low input dimension.

We fixed the hidden layer sizes of the neural networks and depth of individual estimators in the GBDTs as we wanted only a single axis on which to increase the size of the embedded predictors. Size 16 and 32 for hidden layers were chosen as they are common choices for smaller neural network architectures, and depths 2 and 5 were chosen for GBDTS as they are appropriate for a wide range of SurrogateLIB instances. We reduced the experiment to only two types of ML predictors as they admit valid integer linear formulations, can be scaled, and encompass simpler ML predictors, e.g., decision trees and linear regression. We note that for all instances tight bounds on the input variables are used, which are either derived from the natural interpretation of the setting or from some small factor of the maximum and minium over the training data.

For all experiments, SCIP 9.1.1 [10] is used for MIP solving with SOPLEX 7.1.1 [10] as the LP solver, PyTorch 2.3.1 [47] is used for training the neural networks, and Scikit-learn 1.5.2 [48] is used for training the GBDTs. All experiments are run on a cluster equipped with Intel Xeon Gold 5122 CPUs with 3.60GHz and 96GB main memory. Each instance is run 27 times using all combinations of the random seeds $\{0, 1, 2\}$ for the SCIP random seed, the data randomisation seed, and the training random seed. For the purposes of the experiment, a problem is classified as having status time limit reached if more than half of the runs reached the 30 min time limit.

During preliminary experiments, we observed a common problem where deeper neural networks would collapse into predicting a constant output. We believe this is caused by vanishing gradients, and is the result of using too deep of a neural network to represent simpler relations. The resulting neural network then contains a layer of dead neurons and collapses to outputting constant values, see the folding and collapsing operation in [52]. The MIP containing the embedding is then either often quickly identified as infeasible or trivial as the embedding's constant output cannot be changed. Such a result is ultimately misleading for determining the scale of ML predictors that can be embedded. To overcome this issue, we initialised the neural networks with a Xavier uniform distribution [20], introduced gradient clipping [21], and added an L2 penalty. Additionally, we stopped the experiment at a maximum of 15 layers or estimators.

The summarised result for the experiment is presented in Table 3. The Table shows that the scale of ML predictors that can be practically embedded is highly predictor-dependent. For example, in the auto manufacturer instance, a predictor of scale 1, 2, 5, or 15^+ can be solved to optimality in under 30 min depending on

Table 3. Maximum embedding size for ML predictors in SurrogateLIB such that the resulting MIP can on average be solved within 30 min. Entries with "0" indicate that the starting embedding size could not be solved for the corresponding problem-formulation pair. Entries with "-" indicate that the corresponding experiment was skipped.

	Adversarial Attack	Auto Manufacturer	City Planning	Palatable Diet
ReLU-SOS-32	2	1	0	9^*
ReLU-BigM-32	3	2	0	6^*
ReLU-SOS-16	6	2	0	7^*
ReLU-BigM-16	6	5	0	7^*
GBDT-5	-	15^+	2	15^+
GBDT-2	-	15^+	15^+	15^+
	Tree Planting	Water Potability	Wine Manufacturer	Workload Dispatching
ReLU-SOS-32	0	8	1	0
ReLU-BigM-32	1^*	8	1	0
ReLU-SOS-16	0	8	1	0
ReLU-BigM-16	2^*	8	3	0
GBDT-5	15^+	6	4	-
GBDT-2	15^+	15^+	15^+	15^+

*Results after this depth are unreliable due to collapsing neural networks.
+Maximum embedding size for experiment was reached.

the embedded predictor type and formulation. Furthermore, we can also observe a dependence on the instance type. Whereas PySCIPOpt-ML is able to solve instances of the Water Potability problem with relatively large embedded neural network predictors, it fails to solve the Workload Dispatching problem with the smallest embedded neural network predictor size. This is despite the Water Potability problem having a larger input space for each embedded ML predictor and more embedded ML predictors. This shows that SurrogateLIB contains a broad range of generators of varying difficulty, making it a versatile tool to test new approaches for solving MIPs with surrogate models.

In Table 3 we can also observe the relative performance differences of the SOS1 and big-M formulations. Aside from the palatable diet problem, which suffers from collapsing neural networks, the big-M formulation always performs at least as well as the SOS1 formulation. We note however, that in our experiments we used tight bounds, which are likely advantageous for the big-M formulation. For more insight into the scale of these instances we refer to Table 4, which shows the average size of each problem from Table 3.

Table 4. The average number of variables and constraints of SurrogateLIB for embedding sizes of Table 3

	Adversarial Attack	Auto Manufacturer	City Planning	Palatable Diet
ReLU-SOS-32	714/715	841/853	5912/6317*	824/814
ReLU-BigM-32	810/907	1129/1333	5912/7117*	602/784
ReLU-SOS-16	810/811	841/853	4712/5117*	362/352
ReLU-BigM-16	810/907	1273/1525	4712/5517*	362/464
GBDT-5	-	2904/14180	5215/4192	1001/5884
GBDT-2	-	954/1931	5812/6192	159/463
	Tree Planting	Water Potability	Wine Manufacturer	Workload Dispatching
ReLU-SOS-32	41100/40506*	40000/39168	715/746	18625/4848*
ReLU-BigM-32	79500/104506	40000/51968	715/586	18625/6384*
ReLU-SOS-16	21900/21306*	20800/19968	475/346	16321/2544*
ReLU-BigM-16	41100/53306	20800/26368	955/1066	16321/3312*
GBDT-5	165209/66314	11226/8800	1495/8109	-
GBDT-2	41930/57946	6448/7816	910/2581	20485/22842

*Minimum problem size, which still did not solve within the time limit

6 Conclusion

In this paper we have introduced the Python package PySCIPOpt-ML, which automatically embeds trained ML predictors in MIPs. We have briefly introduced the API, expanded on some of the core MIP formulations including novel formulations for ReLU neural networks and the argmax function, and presented a set of computational results that provide an intuition for the scale of ML predictors that can be embedded in practice. For the final point, we introduced a library of homogeneous semi-real-world MIP instance generators, which form the extendable library SurrogateLIB.

Acknowledgements. We thank Mohammed Ghannam, Robert Luce, Julian Manns, and Franziska Schlösser for their help with deploying SCIP via PyPI. The work for this article has been conducted in the Research Campus MODAL funded by the German Federal Ministry of Education and Research (BMBF) (fund numbers 05M14ZAM, 05M20ZBM).

References

1. Abadi, M., et al.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems (2015). https://www.tensorflow.org/, software available from tensorflow.org
2. Achterberg, T.: Constraint integer programming. Ph.D. thesis, TU Berlin (2007)
3. Ammari, B.L., et al.: Linear model decision trees as surrogates in optimization of engineering applications. Comput. Chem. Eng. **178** (2023). https://doi.org/10.1016/j.compchemeng.2023.108347

4. Anderson, R., Huchette, J., Ma, W., Tjandraatmadja, C., Vielma, J.P.: Strong mixed-integer programming formulations for trained neural networks. Math. Program. **183**(1-2), 3–39 (2020)
5. Badilla, F., Goycoolea, M., Muñoz, G., Serra, T.: Computational trade-offs of optimization-based bound tightening in relu networks. arXiv preprint arXiv:2312.16699 (2023)
6. Bai, J., Lu, F., Zhang, K., et al.: ONNX: Open Neural Network Exchange (2019). https://github.com/onnx/onnx
7. Bergman, D., Huang, T., Brooks, P., Lodi, A., Raghunathan, A.U.: JANOS: an integrated predictive and prescriptive modeling framework. INFORMS J. Comput. **34**(2), 807–816 (2022)
8. Bertsimas, D., O'Hair, A., Relyea, S., Silberholz, J.: An analytics approach to designing combination chemotherapy regimens for cancer. Manag. Sci. **62**(5), 1511–1531 (2016)
9. Bhosekar, A., Ierapetritou, M.: Advances in surrogate based modeling, feasibility analysis, and optimization: a review. Comput. Chem. Eng. **108**, 250–267 (2018)
10. Bolusani, S., et al.: The scip optimization suite 9.0. arXiv preprint arXiv:2402.17702 (2024)
11. Bynum, M.L., et al.: Pyomo-Optimization Modeling in Python, vol. 67, 3rd edn. Springer, Cham (2021)
12. Camm, J.D., Raturi, A.S., Tsubakitani, S.: Cutting big m down to size. Interfaces **20**(5), 61–66 (1990)
13. Ceccon, F., et al.: OMLT: optimization & machine learning toolkit. J. Mach. Learn. Res. **23**(1), 15829–15836 (2022)
14. Chen, T., Guestrin, C.: XGBoost: a scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2016, pp. 785–794. ACM (2016)
15. Chollet, F., et al.: Keras (2015). https://keras.io
16. Cortez, P., Cerdeira, A., Almeida, F., Matos, T., Reis, J.: Modeling wine preferences by data mining from physicochemical properties. Decis. Support Syst. **47**(4), 547–553 (2009)
17. Cplex, I.I.: V12. 1: User's manual for CPLEX. Int. Bus. Mach. Corporation **46**(53), 157 (2009)
18. Ferreira, K., Lee, B., Simchi-Levi, D.: Analytics for an online retailer: demand forecasting and price optimization. Manuf. Serv. Oper. Manag. **18** (2015)
19. Fischer, T., Pfetsch, M.E.: Branch-and-cut for linear programs with overlapping SOS1 constraints. Math. Program. Comput. **10**, 33–68 (2018)
20. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, pp. 249–256. JMLR Workshop and Conference Proceedings (2010)
21. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016)
22. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572 (2014)
23. Grimstad, B., Andersson, H.: ReLU networks as surrogate models in mixed-integer linear programs. Comput. Chem. Eng. **131**, 106580 (2019)
24. Gurobi Optimization: Gurobi MachineLearning (2023). https://github.com/Gurobi/gurobi-machinelearning/
25. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2024). https://www.gurobi.com

26. Hartigan, J.A., Wong, M.A.: Algorithm as 136: a k-means clustering algorithm. J. R. Stat. Soc. Ser. C (Appl. Stat.) **28**(1), 100–108 (1979)
27. Hojny, C., Zhang, S., Campos, J.S., Misener, R.: Verifying message-passing neural networks via topology-based bounds tightening. arXiv preprint arXiv:2402.13937 (2024)
28. Huang, T., Bergman, D., Gopal, R.: Predictive and prescriptive analytics for location selection of add-on retail products. Produ. Oper. Manag. **28** (2018)
29. Huchette, J., Muñoz, G., Serra, T., Tsay, C.: When deep learning meets polyhedral theory: a survey. arXiv preprint arXiv:2305.00241 (2023)
30. Jalving, J., et al.: Beyond price taker: conceptual design and optimization of integrated energy systems using machine learning market surrogates. Appl. Energy **351**, 121767 (2023)
31. Ke, G., et al.: LightGBM: a highly efficient gradient boosting decision tree. Adv. Neural. Inf. Process. Syst. **30**, 3146–3154 (2017)
32. Klotz, E., Newman, A.M.: Practical guidelines for solving difficult mixed integer linear programs. Surv. Oper. Res. Manag. Sci. **18**(1–2), 18–32 (2013)
33. Kronqvist, J., Misener, R., Tsay, C.: Between steps: intermediate relaxations between big-m and convex hull formulations. In: Stuckey, P.J. (ed.) CPAIOR 2021. LNCS, vol. 12735, pp. 299–314. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-78230-6_19
34. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
35. Lombardi, M., Milano, M.: Boosting combinatorial problem modeling with machine learning. arXiv preprint arXiv:1807.05517 (2018)
36. Lombardi, M., Milano, M., Bartolini, A.: Empirical decision model learning. Artif. Intell. **244**, 343–367 (2017)
37. López-Flores, F.J., Ramírez-Márquez, C., Ponce-Ortega, J.M.: Process systems engineering tools for optimization of trained machine learning models: comparative and perspective. Ind. Eng. Chem. Res. **63**(32), 13966–13979 (2024)
38. Los Alamos Advanced Network Science Initiative: MathOptAI.jl (2024). https://github.com/lanl-ansi/MathOptAI.jl
39. Lubin, M., Dowson, O., Garcia, J.D., Huchette, J., Legat, B., Vielma, J.P.: JuMP 1.0: recent improvements to a modeling language for mathematical optimization. Math. Program. Comput. **15**(3), 581–589 (2023)
40. Lueg, L., Grimstad, B., Mitsos, A., Schweidtmann, A.M.: reluMIP: Open Source Tool for MILP Optimization of ReLU Neural Networks (2021). https://doi.org/10.5281/zenodo.5601907. https://github.com/ChemEngAI/ReLU_ANN_MILP
41. López-Flores, F., Lira-Barragn, L., Rubio-Castro, E., El-Halwagi, M., Ponce-Ortega, J.: Hybrid machine learning-mathematical programming approach for optimizing gas production and water management in shale gas fields. ACS Sustain. Chem. Eng. **11** (2023)
42. Maragno, D., Wiberg, H., Bertsimas, D., Birbil, Ş.İ., den Hertog, D., Fajemisin, A.O.: Mixed-integer optimization with constraint learning. Oper. Res. (2023)
43. McBride, K., Sundmacher, K.: Overview of surrogate modeling in chemical process engineering. Chem. Ing. Tec. **91**(3), 228–239 (2019)
44. Misaghian, M.S., Tardioli, G., Cabrera, A.G., Salerno, I., Flynn, D., Kerrigan, R.: Assessment of carbon-aware flexibility measures from data centres using machine learning. IEEE Trans. Ind. Appl. **59**(1), 70–80 (2023)
45. Mistry, M., Letsios, D., Krennrich, G., Lee, R.M., Misener, R.: Mixed-integer convex nonlinear optimization with gradient-boosted trees embedded. INFORMS J. Comput. **33**(3), 1103–1119 (2021)

46. Moritz, S.: A mixed integer approach for the transient case of gas network optimization. Ph.D. thesis, Technische Universität (2007)
47. Paszke, A., et al.: PyTorch: an imperative style, high-performance deep learning library. Adv. Neural. Inf. Process. Syst. **32**, 8024–8035 (2019)
48. Pedregosa, F., et al.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
49. Peters, K., et al.: The nutritious supply chain: optimizing humanitarian food assistance. INFORMS J. Optim. **3**(2), 200–226 (2021)
50. Petropoulos, F., et al.: Operational research: Methods and applications. arXiv preprint arXiv:2303.14217 (2023)
51. Schweidtmann, A.M., Mitsos, A.: Deterministic global optimization with artificial neural networks embedded. J. Optim. Theory Appl. **180**(3), 925–948 (2019)
52. Serra, T., Kumar, A., Ramalingam, S.: Lossless compression of deep neural networks. In: Hebrard, E., Musliu, N. (eds.) CPAIOR 2020. LNCS, vol. 12296, pp. 417–430. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58942-4_27
53. Thebelt, A., Kronqvist, J., Mistry, M., Lee, R.M., Sudermann-Merx, N., Misener, R.: ENTMOOT: a framework for optimization over ensemble tree models. Comput. Chem. Eng. **151**, 107343 (2021)
54. Turner, M., Berthold, T., Besançon, M., Koch, T.: SNDlib-MIPs: a new set of homogeneous MILP instances (2023). https://doi.org/10.5281/zenodo.8021237
55. Turner, M., Chmiela, A., Koch, T., Winkler, M.: SurrogateLIB: an extendable library of mixed- integer programs with embedded machine learning predictors (2024). https://doi.org/10.5281/zenodo.11231147
56. Wood, K.E., Kobe, R.K., Ibáñez, I., McCarthy-Neumann, S.: Tree seedling functional traits mediate plant-soil feedback survival responses across a gradient of light availability. PLoS ONE **18**(11), e0293906 (2023)
57. Zhang, S., Campos, J.S., Feldmann, C., Sandfort, F., Mathea, M., Misener, R.: Augmenting optimization-based molecular design with graph neural networks. Comput. Chem. Eng. 108684 (2024)

Satellite Communication Resources Management in a Earth Observation Federation of Constellations

Henoïk Willot[1(✉)], Jean-Loup Farges[1], Gauthier Picard[1], and Philippe Pavero[2]

[1] DTIS, ONERA, Université de Toulouse, Toulouse, France
{henoik.willot,jean-loup.farges,gauthier.picard}@onera.fr
[2] Airbus Defence and Space, Toulouse, France
philippe.pavero@airbus.com

Abstract. This paper addresses a new problem arising from the novel concept of multi-mission federation in Earth observation. This problem arises because of the development of several competing ground station networks that could be used by a component of federated missions called Satellite Communication and Resource Management System (SCRMS). Among the functions of SCRMS selection of contacts is a challenging decision problem. It is an optimization problem with Boolean variables corresponding to the selection of some contacts among potential contacts. Constraints enforce the fulfillment of the communication needs of the satellites. The considered criteria are the communication cost and the conflicts and jamming between satellites when communicating with sites, optimized lexicographically. We propose a linear program, several incomplete search schemes and greedy allocation schemes to address this problem, and evaluate them on realistic systems. Results indicate that the linear program complies with computation time requirements and produces better contact selection plans than other optimization schemes.

Keywords: Earth observation · communication resources · GSaaS · Integer Linear Programming · Incomplete Search

1 Introduction

Traditionally space missions devoted to Earth Observation (EO) have their own resources. Each mission has its constellation of satellites, its ground stations and its control center. Ground stations are devoted to communication with satellites and allow to upload observation plans and download observations. More recently, third-party communication stations using the Ground Station as a Service (GSaaS) paradigm were developed [3,11]. In this paradigm customers book communication resources from a ground segment provider instead of building their own stations. This provider enables them to communicate, download, and process data from their satellites, on a pay-as-you-go basis. Customers of GSaaS

G. Tack (Ed.): CPAIOR 2025, LNCS 15763, pp. 235–251, 2025.
https://doi.org/10.1007/978-3-031-95976-9_15

are mainly operators of Low Earth Orbit (LEO) satellites owning or not owning ground stations. The main objective of operators owning their own stations and using GSaaS is to communicate more frequently with their satellites.

A multi-agent federation layer to coordinate systems composed of independent Earth observation missions has been proposed [7]. The goal of this federation is to allow clients requesting acquisitions of large areas to easily access several constellations of satellites and communication sites to compose and download their acquisitions, in a reduced time compared to conventional uncoordinated requests. A Satellite Communication and Resource Management System (SCRMS) has been identified as a key agent of the federation layer. SCRMS shall manage communication using not only ground stations owned by missions of the federation but also GSaaS. SCRMS was functionally decomposed and a first mathematical model of the problem to be solved by one of its functions has been proposed.

The work presented here is devoted to the development and the experimental assessment of algorithms for solving this allocation problem. Its main contribution is the provision of well grounded guidelines for algorithm selection for the communication site booking function.

Sections 2 overviews the concept of SCRMS developed in the DOMINO-E project [6]. Section 3 briefly discusses the related works. Section 4 expounds the definitions and the decision model related to communication booking. Then, the algorithms proposed for solving the decision problem are detailed in Sect. 5. Section 6 is devoted to the presentation and analysis of experimental results and finally this article finishes with concluding remarks.

2 Satellite Communication Resource Management

2.1 Interactions with Other Agents

SCRMS is in charge of providing opportunities for communicating with ground to all satellites of all federated missions. It manages communication between N satellites, seeking for communication windows for uplink and downlink data transfer, and sites including one or several ground stations. Each ground station controls a specific antenna. Missions provide orbits of their satellites and ground station networks publish coordinates and visibility masks of their stations. When the use of GSaaS is necessary, SCRMS interacts with external station networks in order to book communication resources.

2.2 Functional Decomposition

Because SCRMS needs to be aware of all communication opportunities, to choose the best communication opportunities and to actually book resources necessary for the chosen communication opportunities, it is decomposed in three functions:

- computation of potential contacts,
- selection of contacts and
- booking of selected contacts.

Computation of Potential Contacts. Propagating orbits and computing the events with respect to sites coordinates and visibility masks leads for each satellite i to L_i potential contacts with the ground. Each contact $l \in \{1, \dots L_i\}$ is characterized by:

- $s_{i,l}$ its site on ground and
- $[\underline{u}_{i,l}, \overline{u}_{i,l}]$ its time window.

Selection of Contacts. The satellite i may not need all its L_i available contacts to ensure uploading of observation plans and downloading of observations. This is a decision problem that is described in Sect. 4.

Booking of Selected Contacts. For owned stations the selected contacts can be used directly, but for GSaaS the protocol of the station provider shall be respected. It includes opening delay for booking and the possibility of a rejection of the booking of a contact. Rejected booking shall be recorded for future calls to the selection of contacts function.

3 Related Works

The problem consisting in scheduling downloads from satellites of a constellation to a dedicated ground station network is well covered by the literature. For instance, [5] optimize by evolutionary computation the scheduling model for an observation data topic whose value of observation data decay shortly if the topic is not complete at specified time. Another example is the use of a greedy scheduling algorithm for maximizing the total amount of data downloaded to Earth [4]. On the ground station side, research is also active. Indeed, Monte Carlo simulation allows the assessment of a single ground station saturation in terms of data downlink requirements and increasing number of conflicting passes with the growth of the number of satellites [15]. An automated tool, based on Integer Linear Programming, solves the problem of deconflicting requests for multi-site ground networks serving numerous satellite operators [14]. Scheduling for ground station with equal distribution of redundant contact between requests, associated with data synchronization of satellite downlinks and transmission error recovery, improves ground station network performance, cf. [13].

However, the problem of optimizing communication needs of several constellations using several ground station networks with different access conditions is a new problem resulting from the development of several competing ground station networks such as AWS, see [1], and KSAT, see [9]. Thus the scope of this paper is the development of mathematical problems expressing constraints and criteria related to communication booking. The formulation we propose, based on Integer Linear Programming, and more generally the underlying combinatorial optimization problem can be linked, up to a certain extend, to the Weighted Partition Set Cover problem [8].

4 Decision Problem Modeling

This section provides the definitions and the decision problem model related to communication booking in SCRMS.

4.1 Definitions

Definition 1 (Need). *A need is a needed contact duration associated to a satellite, a site list and a time window. The kth need of satellite i is characterized by:*

- *a communication duration* $D_{i,k}$,
- *a time window* $[\underline{t}_{i,k}, \overline{t}_{i,k}]$,
- *a site list* $\mathcal{S}_{i,k}$, *and*
- *a radio communication band.*

The satellite i has K_i needs.

Definition 2 (Need associated to a contact). *A need is associated to a contact if this contact is able to contribute to fulfilling the needed contact duration. The lth contact of satellite i is associated to the kth need of satellite i if and only if:*

- $s_{i,l} \in \mathcal{S}_{i,k}$, *and*
- $\underline{u}_{i,l} \geq \underline{t}_{i,k}$,
- $\overline{u}_{i,l} \leq \overline{t}_{i,k}$, *and*
- *the need radio communication band is included in the set of radio communication bands provided by the site* $s_{i,l}$.

Using those definitions, it is possible to determine, for each contact l of satellite i, the set of needs it is able to fulfill. This set is denoted $\mathcal{Z}_{i,l}$.

4.2 Variables

The proposed model doesn't consider partial use of contacts by satellites. In consequence, decision variables are $x_{i,l} \in \{0, 1\}$ for $i \in \{1, \dots N\}$ and $l \in \{1, \dots L_i\}$. $x_{i,l}$ corresponds to the selection of contact l by satellite i. The set of all $x_{i,l}$ is noted as $\mathbf{x}$.

4.3 Constraints

Each need of each satellite shall be satisfied, leading to the constraints for $i \in \{1, \dots N\}$ and $k \in \{1, \dots K_i\}$:

$$\sum_{l \in \{1, \dots L_i\} / k \in \mathcal{Z}_{i,l}} d_{i,l} x_{i,l} \geq D_{i,k} \tag{1}$$

where $d_{i,l}$ is the duration of a contact l of satellite i. It is given by $\overline{u}_{i,l} - \underline{u}_{i,l}$.

4.4 Criteria

If the problem is feasible, setting all $x_{i,l}$ variables to one would respect Eq. 1. However, there is a financial cost associated to contact selection and selecting some contacts together induces a risk of not being able to actually use them because of conflict or mutual jamming. For is reason two criteria are relevant for selecting contacts:

1. the total booking cost for the federation and
2. the total level of conflict and jamming suffered by the federation.

Total Cost. With respect to cost, the federation may use the sites owned by its missions at no cost or the sites of ground station networks with a cost depending on their billing method. The cost C is given by:

$$C = \sum_{i=1}^{N} \sum_{l=1}^{L_i} c_{i,l} x_{i,l} \tag{2}$$

where

$$c_{i,l} = c^0_{i,l} + e_{i,l} d_{i,l} \tag{3}$$

The cost value associated to contact l of satellite i depends on the billing method of the provider of this contact. If this provider is payed per contact $c^0_{i,l}$ is the cost per contact for the provider of the site involved in the contact l of satellite i. If it is payed per minute of contact, in the term $e_{i,l} d_{i,l}$, $e_{i,l}$ is the cost per unit of time for the provider of the site involved in the contact l of satellite i. The cost value associated to contact l of satellite i depends on the billing method of the provider of this contact. If this provider is payed per contact $c^0_{i,l}$ is the cost per contact for the provider of the site involved in the contact l of satellite i. If it is payed per minute of contact, in the term $e_{i,l} d_{i,l}$, $e_{i,l}$ is the cost per unit of time for the provider of the site involved in the contact l of satellite i.

Conflict and Jamming. Conflict and jamming may occur when two contacts of two different satellites have overlapping time windows and the same site. It can be written as:

$$J = \sum_{i=1}^{N-1} \sum_{l=1}^{L_i} (f_{i,l} x_{i,l} + \sum_{j=i+1}^{N} \sum_{m=1}^{L_j} b_{i,l,j,m} x_{i,l} x_{j,m}) \tag{4}$$

where $f_{i,l}$ characterizes the conflict and jamming between contact l of satellite i and satellites not belonging to the federation. It is only relevant for contacts related to not owned sites and its value shall be estimated by learning on the long term a model of the site. $b_{i,l,j,m}$ characterizes the conflict and jamming between contact l of satellite i and contact m of satellite j. If contact l of satellite i does not overlap with contact m of satellite j or if those contacts are not related to

the same site, i.e. $s_{i,l} \neq s_{j,m}$, or if their respective communication bands differ, $b_{i,l,j,m} = 0$. Otherwise, the value of $b_{i,l,j,m}$ depends on site characteristics.

If the site has a single ground station, it is a conflict and the strength of the conflict is characterized by the ratio between intersection and union of time intervals:

$$b_{i,l,j,m} = \frac{\min\{\overline{u}_{i,l}, \overline{u}_{j,m}\} - \max\{\underline{u}_{i,l}, \underline{u}_{j,m}\}}{\max\{\overline{u}_{i,l}, \overline{u}_{j,m}\} - \min\{\underline{u}_{i,l}, \underline{u}_{j,m}\}} \tag{5}$$

If the site has multiple antennas, jamming may occur at time σ in the overlapping period $[\max\{\underline{u}_{i,l}, \underline{u}_{j,m}\},\ \min\{\overline{u}_{i,l}, \overline{u}_{j,m}\}]$, such that $\alpha_{i,l,j,m}(\sigma) \leq \underline{\alpha}$ where $\alpha_{i,l,j,m}(\sigma)$ is the angle between satellites i and j, seen from the site, during their contacts l and m respectively, and $\underline{\alpha}$ is a critical value for angles between satellites seen from a site. We have:

$$\cos \alpha_{i,l,j,m}(\sigma) = Q_{i,l}(\sigma).Q_{j,m}(\sigma) \tag{6}$$

where $Q_{i,l}(\sigma)$ is the unit vector pointing from the antenna site $s_{i,l}$ to the satellite i. It is given by:

$$Q_{i,l}(\sigma) = \frac{P_{i,l}(\sigma) - P'_{s_{i,l}}(\sigma)}{\|P_{i,l}(\sigma) - P'_{s_{i,l}}(\sigma)\|} \tag{7}$$

were $P_{i,l}(\sigma)$ and $P'_{s_{i,l}}(\sigma)$ are respectively, in a common frame, the position of satellite i at time σ and the position of station $s_{i,l}$. In that case, $b_{i,l,j,m}$ can be computed as the ratio of, on the one hand, the duration of $\alpha_{i,l,j,m}(\sigma) \leq \underline{\alpha}$ holding in the intersection of the two contacts to, on the other hand, the duration of the union of the two contacts. The positions during the intersection of the two contacts shall be obtained from orbit description of the two satellites. If the common frame is not rotating with the Earth, the position of the station shall be obtained considering Earth rotation.

Lexicographic Order. Because C and J haven't a common measure, it is difficult to weight them in an aggregated criterion. Risky federation managers would prefer to minimize C first while cautious federation managers would try to minimize J first. For this reason criteria shall be considered in a lexicographic order and two strategies shall be considered: either C is optimized first and J is optimized restricting the solutions to those providing the optimal C value or J is optimized first and C is optimized restricting the solutions to those providing the optimal J value. The strategies optimizing C first and J first are named MINI and SECURE respectively.

5 Algorithms

For solving the communication booking decision problem, i.e. finding an admissible assignment of true or false to each variable in $\mathbf{x}$, we propose four types of algorithms: (1) Round-robin algorithms, (2) greedy algorithms, (3) search algorithms and (4) Integer Linear Programming (ILP) algorithms. We present each algorithm in the following sub-sections.

5.1 Round-Robin

The round-robin procedure is a system to consider fair distribution of items among agents [2]. The round-robin allocation scheme proposed here associates needs to agents and contacts to items. Thus, each need is considered in turn and selects an additional contact as to optimize its valuation. The order to consider the needs either depends on the needs' valuation (the need with the best valuation chosen first), or depends on the hardness of the need (the hardest need chosen first). This results in two round-robin allocation schemes: **valuation-guided round-robin** (coined v_RR) and **hardness-guided round-robin** (coined h_RR). Needs' valuation and hardness are defined as follows. For the valuation of need k of satellite i all contacts that can contribute to the need k and are not selected, i.e. $l \in \{1, \dots L_i\}/k \in \mathcal{Z}_{i,l} \wedge x_{i,l} = 0$, are examined for a possible selection, i.e. $1 \rightarrow x_{i,l}$ and resulting values of criteria J and C are computed. The valuation of the need is the best couple of criteria considering the lexicographic order. The need's hardness is computed by:

$$\frac{D_{i,k}}{1 + \sum_{l \in \{1, \dots L_i\}/k \in \mathcal{Z}_{i,l}} d_{i,l}(1 - x_{i,l})} \tag{8}$$

Note that when no contact is selected for a need its hardness is the smallest value possible, i.e. almost the ratio between the needed duration and the total potential contact duration associated to the need, and that selecting contacts increases its hardness.

5.2 Greedy Procedures

A greedy allocation scheme consists in allocating some contacts by making the choice that seems best at the moment, without revising this decision afterwards. It is a very fast (polynomial) method, but it might result in sub-optimal allocations. Here, two ways of making allocation choices are proposed: either the main focus is the need or the main focus is the contact. In the first way needs are recursively selected for full fulfillment while in the second way contacts are recursively selected.

For the first way, we defined two greedy allocation algorithms: a **need valuation-guided greedy** (coined v_G), that first fulfills the best need wrt. its valuation (cost then jamming, or jamming then cost, depending on the strategy), and a **need hardness-guided greedy** (coined h_G), that first fulfills the need that is the hardest to fulfill: the one with the smallest margin of freedom, cf. Eq. (8).

The second way leads to **contact valuation-guided greedy** (c_G). Very similar to v_G, instead of looping over needs and selecting the best need at each step, it loops over all the non booked contacts, and select the best one. It repeats until all the contacts have been considered or all the needs are fulfilled.

5.3 Local Searches

Round-robin and greedy procedures are constructive heuristics: they produce partial solution at each step until reaching admissibility. At the opposite local searches work on complete solutions trying to improve criteria while maintaining admissibility. Those methods need an initial solution and the definition of the neighborhood of a solution, i.e. the set of solutions that are defined as reachable from a given solution.

Concerning the initial solution, several possibilities can be considered, such as:

1. setting all $x_{i,l}$ to 1,
2. for each i setting $x_{i,l}$ to 1 by increasing l index order until all Eq. (1) related to i and $k \in \{1, \dots K_i\}$ are satisfied,
3. for each i setting $x_{i,l}$ to 1 by increasing $c_{i,l}$ order until all Eq. (1) related to i and $k \in \{1, \dots K_i\}$ are satisfied or
4. using any constructive heuristic.

Results of initial tests conduct to choose the third option when the strategy is MINI and the fourth option with v_G when the strategy is SECURE. Those choices are driven by a trade-off between computation time to find the initial solution and quality of the initial solution.

Concerning the definition of the neighborhood two options have been investigated here:

- a small neighborhood consisting of flipping any one of the $x_{i,l}$ variables forbidding $1 \rightarrow 0$ flips that violate Eq. (1) and
- a large neighborhood consisting in making a given number of $1 \rightarrow 0$ flips and reconstructing admissibility.

For the small neighborhood we defined two search schemes: **best first search** (coined BFS) and **depth first search** (coined DFS). Both use the development of the neighborhood of a solution: from a solution all new solutions in the neighborhood are evaluated. In BFS, the process selects first the solution with the best valuation among all the solutions that are already evaluated but not already developed. In DFS, the process selects the best neighboring solution of the current solution. Note that BFS is local w.r.t. all already evaluated solutions while DFS is local w.r.t. the current solution.

Large Neighborhood Search (LNS) is a metaheuristic method employed in optimization problems, particularly those involving combinatorial optimization [12]. It is an advanced version of the simpler neighborhood search technique. For the large neighborhood the algorithm mainly alternates between *destroy*, *repair* and *accept* operations to find better solutions. The definition of those operations leads to **large neighborhood search** (coined LNS):

- *destroy* unset randomly a proportion of positive values in $\mathbf{x}$. In other word, it cancels some contacts.
- *repair* set variables for each unfulfilled need using h_G principle.

– *accept* only filter improving solutions.

This metaheuristic requires several meta-parameters to be set:

– The maximum number of iterations,
– The ratio of contact assignments destroyed at each iteration and
– The maximum number of iterations without improvement before stopping.

5.4 Integer Linear Programming

ILP solvers are general purpose software that are able to solve optimization problems with a linear criterion and linear constraints. Examples of such software are IBM CPLEX[1] [10] and Google OR-Tools[2]. The problem described by Eqs. (1), (2) and (4) is almost linear. Indeed, only Eq. (4) is not linear. The general idea for using a ILP approach is to linearize this equation.

Linearization of the Conflict and Jamming Criterion. In order to linearize the conflict and jamming criterion, a new variable is introduced for each couple of contacts, contact l of satellite i and contact m of satellite j, that presents a non null $b_{i,l,j,m}$ parameter:

– $\forall i \in \{1, \dots N-1\}, \forall l \in \{1, \dots L_i\}, \forall j \in \{i+1, \dots N\}, \forall m \in \{1, \dots L_j\}$ such that $b_{i,l,j,m} > 0$, the variable is $y_{i,l,j,m} \in \{0, 1\}$.

This new variable stands for the product $x_{i,l}x_{j,m}$. The set of all those new decision variables is noted $\mathbf{y}$. Those new variables allow a rewriting of the conflict and jamming criterion in a linear form:

$$J = \sum_{i=1}^{N-1} \sum_{l=1}^{L_i} (f_{i,l}x_{i,l} + \sum_{j=i+1}^{N} \sum_{m \in \{1 \dots L_j\} : b_{i,l,j,m} > 0} b_{i,l,j,m} y_{i,l,j,m}) \tag{9}$$

However $y_{i,l,j,m}$ shall stick to $x_{i,l}x_{j,m}$ on $\{0, 1\} \times \{0, 1\}$. In order to have this property the following constraints are introduced:

$$\forall i \in \{1, \dots N-1\}, \forall l \in \{1, \dots L_i\}, \forall j \in \{i+1, \dots N\}, \forall m \in \{1, \dots L_j\} : b_{i,l,j,m} > 0$$

$$y_{i,l,j,m} \leq x_{i,l} \tag{10}$$

$$y_{i,l,j,m} \leq x_{j,m} \tag{11}$$

$$y_{i,l,j,m} \geq x_{i,l} + x_{j,m} - 1 \tag{12}$$

Equation (10) and Eq. (11) ensure that if $x_{i,l}$ or $x_{j,m}$ is null $y_{i,l,j,m}$ is also null. In those three cases Eq. (12) is verified. Equation (12) ensures that in the fourth case, i.e. if $x_{i,l} = x_{j,m} = 1$, $y_{i,l,j,m} = 1$. In this case Eq. (10) and Eq. (11) are verified. Finally, note that because the MILP solver is minimizing and $b_{i,l,j,m}$ are positive, reciprocal constraints corresponding to Eqs. (10) and (11) are not absolutely necessary.

[1] https://www.ibm.com/products/ilog-cplex-optimization-studio.
[2] https://developers.google.com/optimization.

Problem Provided to the Solver. In consequence, the problem provided to the ILP solver has the following characteristics:

- Variables: $\mathbf{x}$ and $\mathbf{y}$
- Constraints: Eqs. (1), (10), (11) and (12).
- Criteria: Eqs. (2) and (9)

Note that it is a bi-criteria ILP problem. CPLEX solves bi-criteria problems while Google OR-Tools is not able to directly handle lexicographic bi-objective optimization. For Google OR-Tools we implemented the lexicographic bi-objective optimization by procedurally chaining two mono-objective optimization processes, whose order depends on the booking strategy. The value of the first criterion found at the end of the first optimization is used as a constraint for the optimization of the second criterion. We made preliminary experiments with both solvers, but **only retain Google OR-Tools**, because solution quality was identical to IBM CPLEX's one, and execution time was in the same order of magnitude, while being an open source software. In the following this Google OR-Tools based solver is coined ILP.

6 Experimental Evaluation

6.1 Scenarios

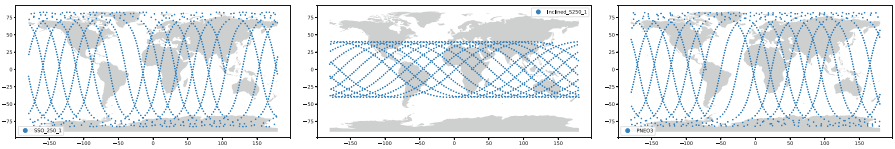
Sixty different scenarios, corresponding to sixty consecutive days, are defined for the assessment of algorithms. Those scenarios differ with respect to the communication needs of the satellites and constellations and with respect to positions of satellites on their orbits. Each day, a contact selection plan is computed for the next ten days. This computation is performed by applying ten times the considered algorithm changing the initial position of satellites. The sums of criteria over the days are provided as indicators for the considered scenario. The timeout is applied to each day of a scenario. In our case, we set 30 s per day-related subproblem, which means 5 min per scenario.

We developed all the proposed algorithms in Java (SDK 1.8). Experiments have been run on a 20-core Intel(R) Xeon(R) CPU E5-2660 v3 @ 2.60GHz, 62GB RAM, Ubuntu 18.04.5 LTS.

Constellations. There are four constellations in the configuration. The first three constellations are handled by the system, while the last one contains the external satellites that can produce unforeseen jammings. The three internal constellations are configured as followed:

- PNEO: Two S950 satellites on the same SSO orbit, in phase opposition,
- SSO: Four S250 satellites on the same SSO orbit, in quadrant phases,
- Inclined: Two pairs of S250 satellites in phase opposition, each pair on an 40° inclined orbit. The orbits have a 180° RAAN difference.

Figure 1 shows sample ephemerides of each constellation.



(a) SSO_250_1 (b) Inclined_S250_1 (c) PNEO3

Fig. 1. Sample ephemerides for some satellites used in the scenarios

Station Providers. There are four providers considered in the configuration:

- **Owned**: contains the stations that are owned by the system. Its cost is considered free.
- **Preferred**: this simulates a provider with optimized cost and location.
- **Normal**: this simulates a bonus provider with basic contract.
- **Expensive**: this simulates a premium provider that should be used only in emergency.

The stations are created with a 5° mask, and are positioned as illustrated in Fig. 2.

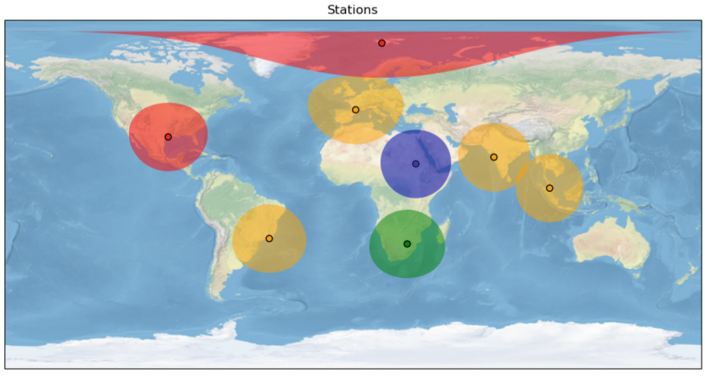


Fig. 2. Stations and their respective visibility circles at 500km altitude: violet is Owned, green is Preferred, yellow is normal, red is Expensive. (Color figure online)

Needs. Needs consists of communication requirements to be fulfilled at each period of time (every 24h for Band X and every 6h for Band S). Needs are expressed in terms of minimum communication duration for each band, for each satellite and/or constellation. Initial needs are given in Table 1. Then, all along the 60-days period, the needs evolve as illustrated in Fig. 3, as to simulate growing or decreasing needs depending on the period of the two months.

Table 1. Initial routine needs

constellation	satellite	band	duration (per period in min)
PNEO_S950		S	5
PNEO_S950		X	50
SSO_S250		S	5
SSO_S250		X	70
SSO_S250	SSO_S250_1	X	60
Inclined_S250		S	5
Inclined_S250		X	50
Inclined_S250	Inclined_S250_1	X	50

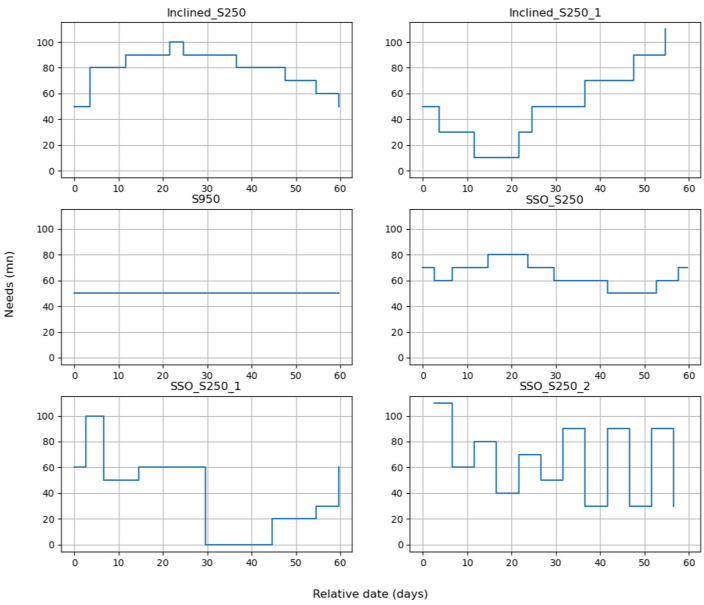


Fig. 3. Band X needs evolution over time

6.2 Results

Figure 4 presents the average values of cost, jamming and conflict, and computation time for the two strategies and the nine algorithms.

Cost. All methods applying a MINI strategy which aims at minimizing costs present a lower cost than any method applying a SECURE strategy, indicating that the priority given to the cost by the MINI stragegy is always efficient. Moreover, all the solution methods converge to solutions with a cost around 23000, when following a MINI strategy. When aiming to minimize the jamming and conflicts (SECURE strategy) the search methods reach solutions around 35000, but there is a large difference between the best method ILP, with an average cost

around 28000, and the worst methods c_G, with an average cost around 48000, and h_G, with an average cost around 40000. However, in SECURE strategy costs shall be compared only for equivalent value of jamming and conflict.

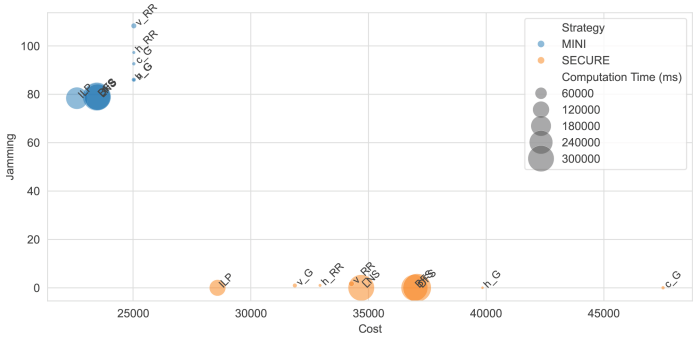


Fig. 4. Pareto analysis of the main performance metrics, cost and jamming.

Conflicts and Jamming. All methods applying a SECURE strategy present a lower conflict and jamming than any method applying a MINI strategy, indicating that the priority given to conflict and jamming by the SECURE strategy is always efficient. For conflicts and jamming the same discrepancies arise when following a MINI strategy: ILP and search-based solvers find quasi optimal solutions around 80, while other solvers struggle with values between 82 and 105. The good news is that, when following a SECURE strategy, all the algorithms manage to find almost jamming free allocations. This is due to the fact that there are numerous contacts, and it is easy to find configuration without jamming. But let's note that we consider here only jamming with owned constellations. In real contexts, it is highly probable that external constellations generate extra jamming. To assess this we need a model of the other constellations and of the service acceptance behavior.

Algorithm Performance w.r.t. the Strategy. Figure 5 illustrates on the cost indicator that the workload generates a wide variance, that impacts all the algorithms in the same way. For this reason, significance of differences between algorithms is computed by first computing the difference each day and then checking the resulting variable for significant mean value. Table 2 presents such computations while using the SECURE strategy. With this kind of analysis, it is possible to compare algorithm performance w.r.t. the requirement derived from the strategy. For comparisons of algorithms while using the MINI strategy the most important criterion is the cost and the less important criterion is the jamming. It is possible to note that ILP is always better than other algorithms. Both BFS and DFS are slightly equivalent but significantly better than LNS.

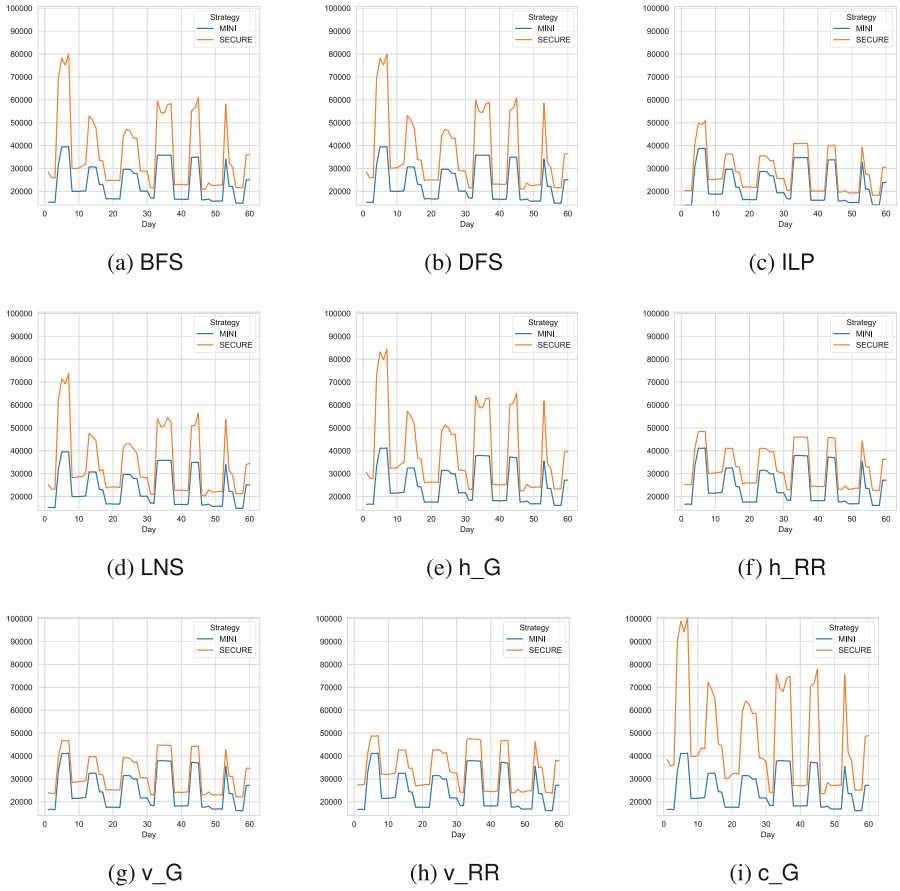


Fig. 5. Solution cost all along the successive scenarios for all the solvers

LNS is better than greedy algorithms. There is no significant difference between hardness guided greedy and valuation guided greedy and both are better than contact greedy which is better than hardness guided Round-Robin, which in turn is better than value guided Round-Robin, but those differences are due to the less important criterion. When comparing the algorithms for the SECURE strategy the most important criterion is the jamming and the less important criterion is the cost. As indicated by Table 2, at a 95% level the ranking is partly driven by the most important criterion, i.e. the jamming. In that case it is possible to note that ILP is always better than other algorithms. For LNS, BFS, DFS, h_G and c_G, there is no difference on the most important criterion, i.e. the jamming. Then the order LNS > BFS > DFS > h_G > c_G is driven by significant differences in the less important criterion, i.e. the cost. LNS, BFS, DFS, h_G and c_G are better than v_G and Round-Robin algorithms (h_RR and v_RR) on the basis of a significant difference on the most important criterion. v_G is

better than h_RR on the basis of a significant difference on the less important criterion and better than v_RR on the basis of a significant difference on the most important criterion. h_RR is better than v_RR because of a significant difference on the most important criterion. In conclusion, it is worth noting that of course the ILP is the best option since it performs better on both MINI and SECURE strategies. However, the local search and LNS are very close.

Table 2. Average degradation of criteria w.r.t. reference algorithms (left column) when using algorithms in SECURE strategy; in a cell on first line jamming degradation and on the second line cost degradation; * and ** indicate respectively 95% and 99.7% statistical significance on a test with data twined per day

Other Reference	LNS	BFS	DFS	h_G	c_G	v_G	h_RR	v_RR
ILP	0.02*	0.02*	0.02*	0.02*	0.02*	0.96*	1.01*	1.68*
	6104**	8363**	8477**	11254**	18924**	3284**	4354**	5691**
LNS		0	0	0	0	0.94*	0.99*	1.67*
		2259**	2373**	5150**	12820**	−2821**	−1751*	−413
BFS			0	0	0	0.94*	0.99*	1.67*
			114**	2891**	10561**	−5080**	−4009**	−2672*
DFS				0	0	0.94*	0.99*	1.67*
				2777**	10447**	−5194**	−4123**	−2786*
h_G					0	0.94*	0.99*	1.67*
					7670**	−7970**	−6900**	−5563**
c_G						0.94*	0.99*	1.67*
						−15641**	−14570**	−13233**
v_G							0.05	0.72*
							1070**	2407**
h_RR								0.68*
								1337**

Computation Time. We observed that search algorithms requires 5min (the fixed timeout) while greedy and round-robin may only require few seconds each scenario day. Interestingly, the ILP does not need to reach to time out to prove the optimality of the solutions. That makes us think this is a very promising option here. Compromises between cost (resp. jamming) and computation time confirm this tendency: for the SECURE strategy, all the greedy algorithms being able to reach jamming-free allocations, h_G and c_G, are good candidates, but they struggle to find low cost allocations. However, time might not be a strong constraint since these algorithms should be executed only a few times a day depending on the availability of the services. Looking at the variability due to the workload, it is interesting to note that ILP sometimes solves the problems

very fast (as fast as greedy and round-robin solvers). This is due to some configurations that are very easy to solve, even optimally.

7 Conclusions

The result of this research is the design, development and assessment of nine algorithms for the communication resource management in EO federation of constellations. The algorithms we developed mostly provide good solutions, when following a strategy aiming at minimizing costs. However, when it comes to minimize the jamming first, while minimizing costs, the best algorithms (search-based, and ILP) requires more computation time (still less than 5 min). Yet, this computation time is very reasonable, since such procedures should only be called few times each day.

One limitation of the assessment presented here is that it is done in open loop: acceptance ot rejection of selected contacts by GSaaS is not simulated and the optimization of the scenario for the next day starts from scratch. Closed loop assessment is a challenge that will be addressed soon. For this, GSaaS behavior and resolution of conflicts and jamming with competing constellation shall be simulated. In closed loop settings, the uniform time constraint sharing among days is questionable. Indeed, the first day of the optimization horizon is the most critical but also the one that should have the largest number of contacts already booked. Learning time constraint sharing among day would be an interesting challenge to address.

Other limitations are related to the model and its link with real life elements. First the cost as defined by Eq. (2) does not cover all the billing possibilities. For instance, it is possible to imagine communication packages where a given amount is pre-paid at low cost, thus free of charge when considering real time operation, and excesses with respect to this amount are paid more expensively. Then, in the test only global communication needs are associated to satellites. However, when communication is urgent, needs can be associated to areas of the Earth surface, for instance where a disaster is occurring. In that case, the need for S and X bands are respectively before and after the satellite is flying over the area. The $\mathcal{Z}_{i,l}$ set is theoretically able to model such a situation, however the difficulty is to determine which contacts have such needs in their $\mathcal{Z}_{i,l}$ set. Indeed, it requires fine visibility computations depending on the areas of interest. Finally, the impact of external constellations on the jamming is modeled by $f_{i,l}$ coefficients, but the value of those coefficients is difficult to set. One track would be to simulate external constellations and their respective workload, to assess an average jamming value for each potential contact. A second track would be to build a model of the acceptance behavior of service providers, as to provide allocations that are robust to the providers' response. Finally, a third track could try to learn those coefficients using past contact rejections from the GSaaS and past jamming from external constellations.

References

1. AWS, A.W.S.: AWS ground station (2023). https://aws.amazon.com/ground-station/
2. Caragiannis, I., Kurokawa, D., Moulin, H., Procaccia, A.D., Shah, N., Wang, J.: The unreasonable fairness of maximum Nash welfare. ACM Trans. Econ. Comput. **7**(3) (2019). https://doi.org/10.1145/3355902
3. Carcaillon, E., Bancquart, B.: Market perspectives of ground segment as a service, p. 60462 (2020)
4. Castaing, J.: Scheduling downloads for multi-satellite, multi-ground station missions (2014)
5. Chen, H., Zhai, B., Wu, J., Du, C., Li, J.: A satellite observation data transmission scheduling algorithm oriented to data topics. Int. J. Aerosp. Eng. **2020**, 1–16 (2020)
6. DOMINO-E Consortium (2024). https://domino-e.eu/
7. Farges, J.L., et al.: Going beyond mono-mission earth observation: using the multi-agent paradigm to federate multiple missions. In: 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), pp. 2674–2678. International Foundation for Autonomous Agents and Multiagent Systems (2024)
8. Inamdar, T., Varadarajan, K.R.: On partial covering for geometric set systems. In: Speckmann, B., Tóth, C.D. (eds.) 34th International Symposium on Computational Geometry, SoCG 2018, Budapest, Hungary, 11–14 June 2018. LIPIcs, vol. 99, pp. 47:1–47:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2018). https://doi.org/10.4230/LIPICS.SOCG.2018.47
9. KSAT: Ground network services (2023). https://www.ksat.no/ground-network-services/
10. Manual, C.U.: IBM ilog cplex optimization studio. Version **12**(1987–2018), 1 (1987)
11. Nguyen, L.: Ground stations as a service (GSAAS) for near real-time direct broadcast earth science satellite data. Technical report, NASA (2012). https://esto.nasa.gov/forums/estf2021/Presentations/June10/Nguyen_GSON_ESTF2021.pdf
12. Pisinger, D., Ropke, S.: Large neighborhood search. In: Gendreau, M., Potvin, J.-Y. (eds.) Handbook of Metaheuristics. ISORMS, vol. 272, pp. 99–127. Springer, Cham (2019). https://doi.org/10.1007/978-3-319-91086-4_4
13. Schmidt, M.: Ground station networks for efficient operation of distributed small satellite systems. Universität Würzburg (2011)
14. Vazquez, R., Perea, F., Vioque, J.G.: Resolution of an antenna–satellite assignment problem by means of integer linear programming. Aerosp. Sci. Technol. **39**, 567–574 (2014)
15. Writt, A.J.: Optimization of cubesat ground stations for increased satellite numbers. Ph.D. thesis, Monterey, CA; Naval Postgraduate School (2018)

Shaping Reward Signals in Reinforcement Learning Using Constraint Programming

Chao Yin(✉), Quentin Cappart, and Gilles Pesant

Department of Computer and Software Engineering, Polytechnique Montréal, Montreal, Canada
{chao.yin,quentin.cappart,gilles.pesant}@polymtl.ca

Abstract. Reinforcement learning is a machine learning paradigm in which an agent learns through interaction a policy, that is what sequence of actions to take in some environment in order to maximize the reward it receives. Reward shaping modifies the reward function in an attempt to accelerate agent training. Under some conditions, potential-based reward shaping (PBRS) preserves the optimal policies of the original problem. We propose a PBRS method that automatically derives shaped rewards from a constraint programming (CP) model of the environment and from the probability mass functions over the domains of its variables, as provided by the CPBP framework. In the context of an AI planning task, we investigate the effect of CP modeling choices on the effectiveness of our reward shaping method. Our experiments show that our method significantly shortens training while being rather insensitive to modeling choices, and that the resulting agent's performance scales well beyond instance sizes seen during training.

Keywords: Reinforcement Learning · Constraint Programming · Belief Propagation · Potential-Based Reward Shaping · AI Planning

1 Introduction

Reinforcement learning (RL) is a machine learning paradigm in which an agent learns what sequence of actions to take in some environment (i.e. a *policy*) by interacting with it in order to maximize the *reward* it receives. It has achieved superhuman game-playing performance [24,35] albeit at the expense of a huge amount of data. For example, AlphaStar used the equivalent of 200 years of StarCraft II gameplay, and even so with additional guidance from expert players to improve *sample efficiency* during training by identifying valuable game states and rewarding the agent for entering them [39]. Because RL may require large amounts of training data, its use is limited in domains where the collection of data is more costly [5,37].

Designing a good reward function is important to reduce the amount of training data necessary. A simple reward such as returning 1 when the agent has successfully completed the task and 0 otherwise (including for each intermediate

G. Tack (Ed.): CPAIOR 2025, LNCS 15763, pp. 252–269, 2025.
https://doi.org/10.1007/978-3-031-95976-9_16

step), is easy to specify but typically makes training much longer (i.e. it is not sample efficient). This has been coined as the problem of *sparse rewards*, where rewards are rarely non-zero. *Reward shaping* [30] refers to modifying the reward function in order to shape the behaviour of the learning agent. It is a way to inject heuristic knowledge about the environment which potentially makes the RL task much more sample efficient. Such rewards can either be handcrafted by human experts [4,29] or created in an automated fashion through learning [16, 38]. However, care must be taken when modifying the reward function because it can have unintended consequences [9,29] such as the optimal policy using the modified reward function being definitely suboptimal in the original setting. For example, *CoastRunners* is a video game in which the player has to finish a boat race as quickly as possible and *boost* items are laid out in the map to provide speed ups. While training an agent to play this game at OpenAI [9], they found that since the boost items can respawn and agents are rewarded for acquiring them (through reward shaping), their agent learned to get a high return by repeatedly taking boost items as they respawn instead of finishing the race.

Recent investigations of constraint programming (CP) to guide RL have had some success with regard to sample efficiency. An RL framework had been proposed to finetune a sequential generative neural network (NN) given a set of rules by shaping the reward in an *ad hoc* manual fashion based on these rules [18]. Improving on this, Lafleur et al. [20] used a CP model of the rules to rate actions according to the likelihood of no rule violation being introduced or, failing that, to the least additional violation it could bring. Yin et al. [42] further improved the latter approach by shaping the reward function using the minimum or expected value of a global violation variable in the CP model. Both works take advantage of the CPBP framework [28], which provides a probability mass function over the domain of each variable.

Building on this prior work, our paper defines an automated reward shaping method relying on a CP model of the RL environment. We show that our method preserves the optimality of policies when adding reward shaping. We investigate the effect of CP modeling choices, and of the use of the CPBP framework, on the effectiveness of our reward shaping method. In the context of AI planning, we further automate the process by deriving the CP model directly from a standard representation for planning problems.

In the rest of the paper, Sect. 2 presents the necessary background, Sect. 3 discusses the remaining relevant literature, Sect. 4 describes our methodological contribution, Sect. 5 demonstrates and evaluates our methodology for classical AI planning, before concluding in Sect. 6.

2 Technical Background

In the following section, we provide the necessary technical background to follow our methodology. This includes a review of Markov decision processes, potential-based rewards in RL, classical AI planning, and belief propagation in CP.

2.1 Markov Decision Process

In the typical RL setting, the environment is modeled as a (finite) *Markov decision process* (MDP). Briefly, an MDP is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P})$ where $\mathcal{S}$ is a (finite) set of states, $\mathcal{A}$ is a (finite) set of actions and $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the (probabilistic) transition function of the MDP such that $\sum_{s' \in \mathcal{S}} \mathcal{P}(s, a, s') = 1$ for all states s and actions a. Function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathbb{R} \to [0, 1]$ defines the probability of the reward $r \in \mathbb{R}$ received after taking action a at state s, with $\int_{\mathbb{R}} \mathcal{R}(s, a, r)\ dr = 1$ for any s and a. We also write them as $\mathcal{P}(s'|s, a)$ and $\mathcal{R}(r|s, a)$. Let an agent choose a sequence of actions in an MDP, indexed by time (action) step $t \in \mathbb{N}_0$. In state s_t, the agent takes an action a_t, receives a reward r_{t+1} from distribution $\mathcal{R}(r_{t+1}|s_t, a_t)$, and moves to the next state s_{t+1} according to distribution $\mathcal{P}(s_{t+1}|s_t, a_t)$. We say the MDP is *episodic* if there is some terminal state s_* that cannot be left once entered and $\mathcal{R}(0|s_*, a) = 1$ for all actions a. For episodic MDP, we use G_t to denote the discounted sum of rewards obtained after time step t: $G_t = \sum_{k=t+1}^{h} \gamma^{k-(t+1)} r_k$ with h being the *horizon* (length) of the episode and $\gamma \in (0, 1]$ being the discount factor. We say the return is undiscounted if $\gamma = 1$. If the MDP is not finite-horizon, we take the limit of $h \to \infty$ and a discount factor < 1 so the limit exists. In our work, we assume a finite-horizon episodic MDP.

We define a *policy* $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ as a function that gives the probability of taking an action a at a state s. We impose that $\sum_{a \in \mathcal{A}} \pi(s, a) = 1$ for each $s \in \mathcal{S}$ and also write $\pi(a|s)$ to make it clear this is a distribution over $\mathcal{A}$ for a given $s \in \mathcal{S}$. Given policy π, we define the *value function* $V_\pi : \mathcal{S} \to \mathbb{R}$ that gives the expected return of a state when following the policy π from that state:

$$V_\pi(s) = \mathbb{E}_\pi[G_t|s_t = s].$$

We also define the *Q-value function* which gives the expected return when taking an action and following π subsequently:

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t|s_t = s, a_t = a].$$

Consider a policy π^* such that $V_{\pi^*(s)} \geq V_{\pi(s)}$ for all states s and any policy π: we call π^* an *optimal policy*, which is not necessarily unique. The goal of a RL algorithm is to find an optimal policy as quickly as possible or with the least number of samples (i.e. episodes).

One family of RL methods to learn an optimal policy are known as *policy gradient methods*, where a policy π_θ parameterized by θ is updated to maximize the expected return of the initial state s_0, $V_\pi(s_0)$. The REINFORCE [40] algorithm is a policy gradient method which approximates the gradient $\nabla_\theta V_\pi(s_0)$ using trajectories sampled from π_θ. There have since been innovations to reduce the variance in the estimated gradient with the use of a state-dependent baseline $b(s)$. In particular, a learned value function $V_\pi(s)$ can be used by employing a neural network (the *critic network*) [36], such methods being typically known as *actor-critic*. *Proximal Policy Optimization* [33] is an actor-critic method that also constrains how much the policy is allowed to change at each update, which provides stability.

2.2 Potential-Based Reward Shaping

Potential-based reward shaping (PBRS) [26] preserves optimal policies of the original MDP. For a given MDP, we call $\Phi : \mathcal{S} \to \mathbb{R}$ a potential function which assigns to each state a real value. Consider the following modification to the reward function,

$$\mathcal{R}'(s, a, r) = \mathcal{R}(s, a, r + \gamma\Phi(s') - \Phi(s))$$

where $\mathcal{R}$ and $\mathcal{R}'$ are the original and resulting reward functions, respectively, γ is the discount factor and s' is the next state after taking action a at state s. We call

$$r_t^{\Phi} = \big(\Phi(s_t) - \Phi(s_{t-1})\big) + r_t$$

the *shaped reward* and

$$G_t^{\Phi} = \sum_{k=t+1}^{h} \gamma^{k-(t+1)} r_k^{\Phi}$$

the *shaped return*. Ng et al. [26] show that the resulting MDP retains its optimal policies, assuming the MDP has an infinite horizon. Devlin and Kudenko [11] show that the previous result also holds for dynamic potential functions $\Phi_t(s)$ dependent on the time step t. Despite the fact that Eck et al. [14] show that PBRS adds bias in the finite horizon MDP setting, Grzes et al. [17] give an additional condition on Φ, namely that $\Phi(s) = 0$ for any terminal state s, so that the optimal policy invariance holds. Their result also applies to dynamic PBRS.

Beyond guaranteeing policy invariance, theoretical analysis has also shown that PBRS makes the resulting MDP easier to learn. To understand what makes reward shaping successful, Laud et al. [21] introduced the concept of *reward horizon* which can be characterized as the minimum number of steps taken by an agent before receiving meaningful feedback. One easily sees that sparse rewards would have high reward horizons. They showed, using Q-learning, that MDPs with lower reward horizons are easier to learn. While practitioners [4,16,23,41] of PBRS appreciate the policy invariance guarantee, they rely on empirical evidence to evaluate sample efficiency.

2.3 AI Planning

The goal of AI planning consists of choosing a sequence of actions (called a *plan*) to arrive at a goal state starting from an initial state. The classic example is *Blocks World* (BW), a block stacking problem where initial and goal configurations are given together with a set of actions to modify a configuration. Figure 1 shows a simple BW instance with $move(x, y)$ actions which place block x on top of y, the latter being either the ground or another block.[1] Plan $move(B, ground)$, $move(A, B)$ reaches the goal configuration. Classical AI planning problems are typically expressed in one of a few formalisms (e.g., SAS+ [8], PDDL [22]) so that

[1] We use here the same set of actions as Dong et al. [13] for later comparison but the more standard definition would include pick-up/put-down actions for a hand.

they may be processed by a planning solver. In our work we use SAS+ for reasons that will soon become clear. Note that this is not a limitation since a PDDL description can be automatically converted to SAS+.

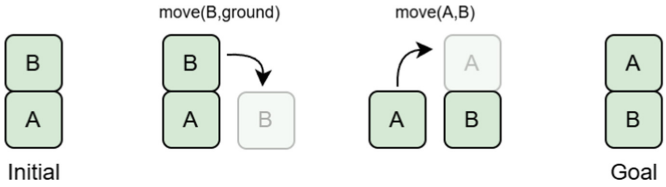


Fig. 1. An instance of Blocks World.

A SAS+ *model* of an AI planning problem is a tuple $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ with $\mathcal{V}$ a finite set of variables, $\mathcal{O}$ a finite set of operators (actions) each with a set of preconditions, effects, and optionally a cost, s_0 the initial state, and s_* the goal state. For the instance presented in Fig. 1, a possible SAS+ model would be $\mathcal{V} = \{v_A, v_B\}$ with $v_A \in \{air, B\}$, $v_B \in \{air, A\}$ indicating what is on top of a given block, $\mathcal{O} = \{move(A, ground), move(A, B), move(B, ground), move(B, A)\}$ with appropriate preconditions (e.g., $v_A = air$ for the first operator) and effects (e.g. $v_B = A$ for the second operator), $s_0 = \{v_A = B, v_B = air\}$, and $s_* = \{v_A = air, v_B = A\}$.

We now introduce the concept of transition system which serves as a bridge between AI planning and CP. It allows us to view a SAS+ model as an automaton that can be readily expressed as a constraint in CP. A *transition system* is a tuple $\Theta = (S, L, c, T, s_0, S_*)$ with S a finite set of states, L a finite set of labels, $c : L \to \mathbb{R}$ a cost function, $T \subseteq S \times L \times S$ a set of transitions (denoted $s \xrightarrow{\ell} s'$), $s_0 \in S$ the initial state, and $S_* \subseteq S$ the set of accepting states. The *induced transition system* of planning problem Π, denoted $\Theta(\Pi)$, is obtained by taking assignments of $\mathcal{V}$ as states, interpreting operators as labels, and defining transitions on labels so that they are consistent with the preconditions and effects of the corresponding operators. Figure 2 (left) depicts the induced transition system of the previous SAS+ model.

Because the size of the state space of the induced transition system is exponential in the number of variables and hence quickly becomes intractable, one can instead decompose it into *atomic factors*, one for each variable in $\mathcal{V} = \{v_1, v_2, \ldots, v_m\}$. The *factored transition system* (FTS) [34] of Θ is the tuple $\langle \Theta^1, \ldots, \Theta^m \rangle$ of transition systems, each defined by $\Theta^j = \langle S^j, L, c, T^j, s_0^j, S_*^j \rangle$ sharing the same labels, costs, and the rest basically being projected onto v_j. Figure 2 (right) depicts the corresponding FTS.

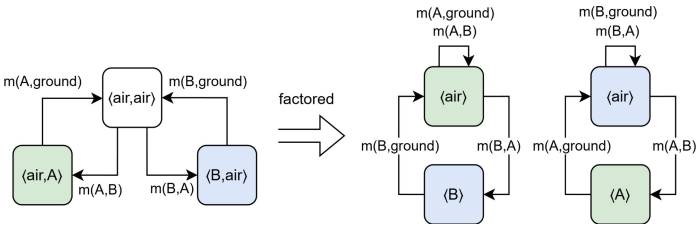


Fig. 2. Induced transition system of the Blocks World instance at Fig. 1 (left) and corresponding FTS $\langle \Theta^A, \Theta^B \rangle$ (right). Blue and green states respectively represent s_0 and s_*. (Color figure online)

2.4 CP-Based Belief Propagation

The MiniCPBP solver[2] complements standard constraint propagation in CP through an additional message-passing phase akin to belief propagation that outputs from each constraint probability mass functions over the domain of the individual variables in its scope, representing how frequently a domain value appears in a solution to that constraint [28]. This propagation of probabilities approximates the marginal distributions over individual variables for the whole CP model. These marginals have been used to design branching heuristics to solve combinatorial problems [2,7] and as a bridge in neuro-symbolic AI [20,31,42].

3 Related Work

Sample efficiency in RL refers to the ability of a learning algorithm to perform certain tasks without requiring large amounts of data. Although the question of how to fully characterize the limitations of sample efficiency remains an open question, methods have been established in the literature to evaluate and compare the sample efficiency of algorithms using a combination of theoretical (when the number of states is relatively small) [1,12,19] and empirical [15,24] results.

Reward shaping has been used to improve sample efficiency by injecting domain expertise in the form of a reward signal. There has been work on the automated creation of shaped rewards by modeling the environment. For example, *plan-based reward shaping* by Grzes et al. [16] for planning problems. A model for the planning problem is explicitly created and a plan is obtained using an AI planner. The potential function is the progress of the agent with respect to the plan. *Reward Machines* by Icarte et al. [38] is a type of finite state machine which is used to model the reward structure of the problem. Since the state is finite, the value functions, which are then used as potential functions, can be readily obtained using tabular methods. *Belief Reward Shaping* by Marom et al. [23] uses a potential function created from prior belief about the environment which decays as a value function is learned alongside it. The

[2] https://github.com/PesantGilles/MiniCPBP.

use of a state-dependent function to improve sample efficiency is also used in policy gradient methods in the form of a critic network. Schulman et al. [32] made the connection between reward shaping and their *Generalized Advantage Estimation* scheme which learns a value function as their critic network. *Neural Logic Machines* (NLM) by Dong et al. [13] is a neural-symbolic model which can solve problems like BW—we use their work to conduct our experiments. The NLM takes as input a set of objects which are used to predict predicates of increasing complexity at every depth of its structure and predicts an action to take using the final layer of predicates. Zimmer et al. [43] propose to implement a novel critic network which uses the NLM to perform value prediction as they noticed that the NLM does not use a variance reduction method.

In almost all of these methods, data is required to tune the prior beliefs or parameters of the critic network before an informative shaped reward can be generated. Our proposed reward shaping method requires no data: it uses a combination of constraint propagation and BP to create a potential function which we detail in Sect. 4. We show that it is possible to use a CP-based technique to compute a baseline (the potential) and improve learning performance without the need to learn a critic network.

4 Semi-automated Potential-Based Reward Shaping

Consider a discrete-action, fully-observable MDP to represent an episodic, finite-horizon RL setting. We propose a two-step method to create a (dynamic) potential function Φ which maps each state (and time step) to a real value called its potential in semi-automated fashion, to be used for reward shaping. We call this a semi-automated method since it requires the creation of a CP model, which in general involves human expertise. Note however that we will automate that step as well in Sect. 5 when we consider AI planning. Once a CP model for the environment has been created, a potential function is automatically derived.

4.1 A CP Model of the Environment

The first step requires designing a CP model $\mathcal{M}$ to represent a given RL environment. In $\mathcal{M}$, the successive actions taken by the agent during an episode are represented by a corresponding sequence of finite-domain *action variables* $\langle x_0, \ldots, x_{h-1} \rangle$ each having as their domain the set of actions $\mathcal{A}$. An auxiliary *return variable* g is also defined to represent the return with its domain the set of obtainable returns by the agent depending on which actions will be taken. Constraints on these variables should express restrictions on actions based on the current state of the environment and mimic the transition dynamic as well as the reward function of the environment. Given the state-based definition of MDPs, some candidate constraints naturally come to mind. In the full probabilistic context, one has the ElementaryMarkov [27], MarkovTransition [25], and NgramMarkov [6] constraints but so far none of them have been equipped to be used in the CPBP framework (and actually they are not a good fit; see

Sect. 6). In the case where both the transition function and the reward function are deterministic, a CostRegular constraint [10] can be used and is already equipped for BP. Consequently this is the context that we implement in Sect. 5.

We offer a few insights on the important design decisions that need to be carefully considered in the first step of our method. Providing a general outline for how to create a CP model from any given environment, however, is beyond the scope of our work. Since our goal is to compute a potential for PBRS with the help of a CP model, there are two main considerations when designing the CP model, the first being which solutions to exclude from the solution space and the second being the ability to filter out values from the domain of the return variable g. We mentioned previously that the constraints are used to mimic the transition dynamics and reward function of the environment. Variable g should contain all possible returns obtainable by the agent by following any sequence of h actions. However it might be of interest to only allow certain sequences of actions that a user considers *desirable*. Overly long sequences can be excluded through the choice of horizon h. Sequences of actions that do not allow the agent to reach a user-specified goal state could be excluded with a stronger CostRegular constraint using a more informed automaton. The motivation behind the latter is that we would expect the estimated value using a model with a stronger filtering ability to be closer to the true value. In this sense, the filtering ability of our CP model may be related to the accuracy of our chosen potential value.

4.2 Extracting Potentials from the Model

In our second step, we now wish to obtain $\Phi(s)$ from our CP model that has a variable g which represents the obtainable return from state s. We use constraint and belief propagation to compute a potential from the domain of the return variable g. Constraint propagation filters out inconsistent values from the domain of g and belief propagation builds a discrete probability distribution $\hat{\theta}_g$ over it, which approximates the true marginal distribution on g. From this distribution, we compute the expected value

$$\mathbb{E}[g] = \sum_{d \in dom(g)} d \cdot \hat{\theta}_g(d)$$

interpreted as the average return of all solutions (consistent sequences of actions) to $\mathcal{M}$ and used as our potential $\Phi(s)$. Without the use of BP, we can still compute a potential function by simply taking the arithmetic average of the values in g's domain, and we evaluate the impact of doing this in Sect. 5.

In the dynamic PBRS approach [11], which uses a dynamic potential function $\Phi_t(s)$, the CP model is created once at the beginning of the RL episode. After each action a_t taken by the agent, the corresponding action variable x_t is assigned a_t. We can then extract the potential with the method described previously. In the traditional PBRS approach [26], the potential must be independent of the time step t. We can do this by creating a new model $\mathcal{M}$ at each time step such that its internal state is initialized as the current state of the environment with a new

sequence of unassigned action variables $\langle x_0, \ldots, x_{h-1} \rangle$. We favour the dynamic approach, which is simpler and faster on the CP side.

Optimal Policy Invariance. The condition given by Grzes et al. [17] applies to the setting where either the MDP has terminal states or it has a terminal time h. We make the observation that their result can also apply to the setting of an MDP having both terminal states and a terminal time h. In the case of MDPs with terminal states, we have the following shaped return [17],

$$G_t^{\Phi} = G_t + \gamma^{N-t}\Phi(s_*) - \Phi(s_t),$$

where N is the time step at which terminal state s_* was entered. They argued that since only the $\gamma^N \Phi(s_N)$ term depends on actions beyond time step t, setting it to 0 would preserve the optimal policy of the original reward function. Accordingly our PBRS method, dynamic or otherwise, preserves optimal policies by setting $\Phi(s_t)$ to 0 when either $s_t = s_*$ or $t = h$.

5 An Application to AI Planning

We now demonstrate our two-step method on RL applied to classical deterministic AI planning and show how to automatically derive a CP model from an AI planning representation.

Given a SAS+ model $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$, its induced transition system is essentially a deterministic automaton with costs associated to each transition and an initial and a goal state (recall Section 2). A single CostRegular constraint on the sequence of action variables $\langle x_0, \ldots, x_{h-1} \rangle$ would be able to directly model this, its solutions being in one-to-one correspondence with valid plans of length at most h, and with identical costs. Recall, however, that nontrivial planning problems have a state space too large to be represented explicitly as such. Instead we use the induced FTS $\langle \Theta^1, \ldots, \Theta^m \rangle$ built from the atomic factors $\Theta^j = \langle S^j, L, c, T^j, s_0^j, s_*^j \rangle$, each modeled by a CostRegular constraint, yielding CP model

$$\begin{aligned} \textsc{CostRegular}(\langle x_0, \ldots, x_{h-1} \rangle, \bar{T}^j, s_0^j, s_*^j, c^j, g^j) \qquad & 1 \leq j \leq m \\ \textsc{Sum}(\langle g^1, \ldots, g^m \rangle, g) \qquad & \\ x_i \in L \cup \{\textsc{eos}\} \qquad & 0 \leq i \leq h-1 \\ g \in \{-h \cdot max(c), \ldots, -1, 0\}. \qquad & \end{aligned}$$

Transition function $\bar{T}^j$ extends T^j by adding an *accept* state and an EOS label in the domain of action variables: together with the automaton, this allows shorter plans and ensures that they are then right-padded with EOS labels, thus adding some static symmetry breaking. Costs c are partitioned into $c^1, \ldots, c^m$ such that each action label has its cost appear in a single automaton (and zero cost elsewhere). A natural way to partition is to relate it to the atomic factor(s) on which the action label has an effect. Note that whereas each CostRegular constraint has its own automaton inherited from the corresponding factor, the sequence of action variables is shared.

Table 1. Solutions to the CP model of the BW instance of Fig. 1.

x_0	x_1	x_2	x_3	g
$m(B, ground)$	$m(A, B)$	EOS	EOS	-2
$m(B, ground)$	$m(B, A)$	$m(B, ground)$	$m(A, B)$	-4
$m(B, ground)$	$m(A, B)$	$m(A, ground)$	$m(A, B)$	-4

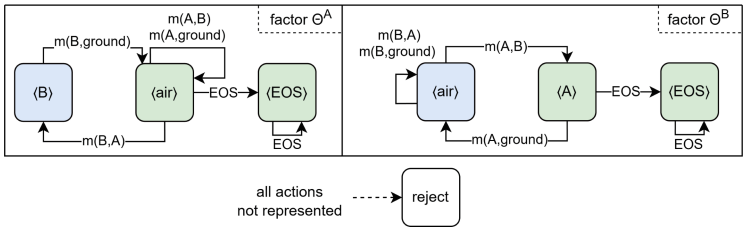


Fig. 3. Automata for the CostRegular constraints related to factors Θ^A and Θ^B. Blue and green states respectively represent initial and accepting states. (Color figure online)

Example 1. *Recall the* BW *instance illustrated in Fig. 1, its induced* FTS *depicted in Fig. 2 and let* $h = 4$. *For* BW, *each action has the same cost and thus yields a reward of* -1. *Our* CP *model will consist of a sequence of four action variables, the return variable and a* CostRegular *constraint for each of the two factors, with its automaton depicted in Fig. 3. To perform dynamic* PBRS *we create a state-dependent potential function* $\Phi_t(s)$, *initialize each* CostRegular *constraint to the initial state of the environment, and assign actions taken by the agent to the corresponding action variables in the* CP *model.*

Table 1 gives the three solutions to this CP *model, of returns* -2 *or* -4. *Constraint propagation only filters the domain of* g *to* $\{-2, -3, -4\}$. *Using a solver that implements the* CPBP *framework allows us to approximate the marginal distribution over that domain, which is respectively* $(1/3, 0, 2/3)$, *yielding* $\mathbb{E}[g] = -3.\overline{3}$ *(MiniCPBP actually approximates it to* -3.4*) which we interpret as our potential* Φ_t *for that state, an estimation of the return, provided the agent reaches the goal state. If the agent takes action* $x_0 = move(B, ground)$ *(see Fig. 4), the state of the automaton for A changes from B to air. The new potential* Φ_1 *is* $-2.\overline{3}$ *and shaped reward* $r_1^\Phi = \big(\Phi_1(s_1) - \Phi_0(s_0)\big) + r_1 = (-2.\overline{3} + 3.\overline{3}) - 1 = 0$ *is returned to the agent. Say the agent now takes action* $x_1 = move(A, B)$ *which changes the state for B from air to A. The problem is now solved since both automata are in an accepting state. The agent receives the shaped reward* $(0 + 2.\overline{3}) - 1 = 1.\overline{3}$.

If the agent had taken action $x_1 = move(B, A)$ *instead, the potential would have been* -2 *(from the remaining second solution in Table 1). Note that* $s_2 = s_0$, *but* $\Phi_2(s_2) \neq \Phi_0(s_0)$ *since there are now fewer time steps before the episode terminates, which is what makes this* PBRS *approach dynamic. The shaped reward would have been* $\big(\Phi_2(s_2) - \Phi_1(s_1)\big) - 1 = -0.\overline{6}$, *a much worse reward than* $1.\overline{3}$.

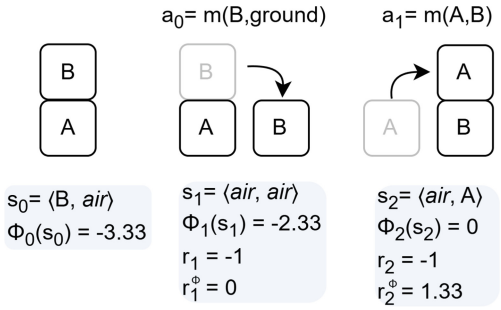


Fig. 4. BW planning problem in our RL setting with potential-based reward shaping.

To summarize, starting from a standard SAS+ representation of a classical planning problem, we automatically derive an equivalent CP model through the induced FTS. This model acts as the environment, providing both its state and the reward expressed from g.

5.1 CP Models

In order to evaluate the impact of modeling choices in our experiments, we consider three SAS+ representations of increasing complexity Π_1, Π_2, Π_3 and their corresponding CP models $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3$, of increasing filtering ability. This increase in filtering ability comes at the cost of identifying insights about the problem which makes them dependent on human expertise. We therefore investigate empirically the relevance of this increase in filtering power.

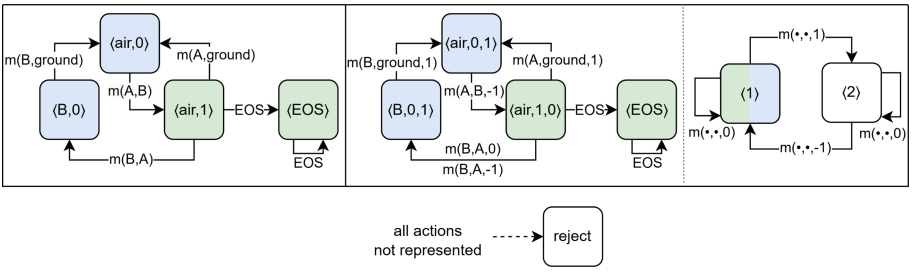


Fig. 5. Factors of block A using Π_2 (left) and Π_3 (middle). Factor for an automata counting the number of stacks using Π_3 (right)

In Π_1 each variable v_j in $\mathcal{V}$ represents the entity that is on top of block j, which can be another block or *air* (see Fig. 3). Π_2 adds an indicator of whether the block is on top of the correct entity in the goal state (see Fig. 5 for the corresponding automaton). It therefore has up to twice as many states as Π_1.

Representation Π_3 (see Fig. 5) further adds an indicator of whether the entity below is the ground, increasing again the number of states up to twice as many. This information is required to keep track of the number of stacks of blocks. This indicator allows for the actions to be marked as either removing a block from the ground, adding it or making no change, increasing the number of actions by up to three times. An additional automaton exploits this information to constraint the number of stacks which allows the model to filter more values.

5.2 Experimental Protocol

We evaluate empirically our reward shaping method on the planning problem *blocks world* solved using NLM [13]. We measure sample efficiency during training, training time, and out-of-distribution performance of the trained agents. Briefly, NLM is a neural architecture which, among other tasks, learns to solve BW with REINFORCE [40]. It uses a sparse reward function that returns 1 if the agent successfully solves the problem and negative reward -0.01 for each action taken. We use it as baseline and replace their sparse reward function with our shaped reward. We study the effect of using the three CP models previously defined and of exploiting the approximate marginal distribution over the domain of g.

Table 2. Hyper-parameters for training. Entries with x/y/z indicate different parameters used for the baseline model, models with BP, and without BP, respectively.

Parameter	Value	Parameter	Value
B^{init}	2	B^{grad}	12
discount rate	0.99/1.0/1.0	h	$4 \times B$
scaling factor (succ)	1/1/2	scaling factor (fail)	$1/\frac{1}{20}/\frac{2}{20}$

We use the same experimental setup as Dong et al. [13]. In particular, they use curriculum learning [3] to train on instances featuring an increasing number B of blocks, starting at B^{init}. For a given problem, the agent is given $h = 4 \times B$ actions to solve it. The agent is evaluated every 10 epochs on 3000 random problem instances and if its success rate reaches a certain threshold, B increases by 1. The threshold is initially 0.95 and increased to 1.0 in a linear fashion as B is incremented. When B reaches B^{grad} and the agent passes the final evaluation, we say that the agent has graduated. The agent has 500 epochs to reach B^{grad} and graduate, otherwise we say that it has failed to graduate. The models are trained on an Nvidia RTX 4060 Ti 16Go GPU. Each operation of the CP solver during training is carried out on a CPU AMD Ryzen 9 5950X 16-Core Processor at 3.40 GHz. Both the NLM implementation and our code are available.[3]

Table 2 gives the hyper-parameters. We introduced a new hyper-parameter, a constant scaling factor for the reward received by the agent, determined empirically in order to keep it in the range [-1,1] as a way to normalize it. The scaling

[3] https://github.com/ChYinn/CPBP_reward_shaping.

factor is different depending on whether or not the episode ends in failure. In the latter case, the return is highly negative and needs a much smaller scaling factor to normalize it to a negative number between -1 and 0. This should not introduce any bias since the agent would obtain higher rewards by ending the episode in success.

5.3 Sample Efficiency

We trained our agents using shaped rewards from our three CP models, both with and without BP. For each agent we use 10 random seeds. Without BP we were unable to train the agents beyond curriculum level 10 without the performance of the agent collapsing to a 0% success rate, which is not the case when using BP. This already tells us that shaped rewards using BP improve training by allowing the curriculum level to reach higher. Both graphs at Fig. 6 feature a top and bottom part sharing the same x-axis, which is the total number of epochs used for training. The y-axis of the top part is the median curriculum level achieved across all 10 seeds, with shading between its 25^{th} and 75^{th} quantiles. The y-axis of the bottom part is the proportion of seeds for which the agent graduated.

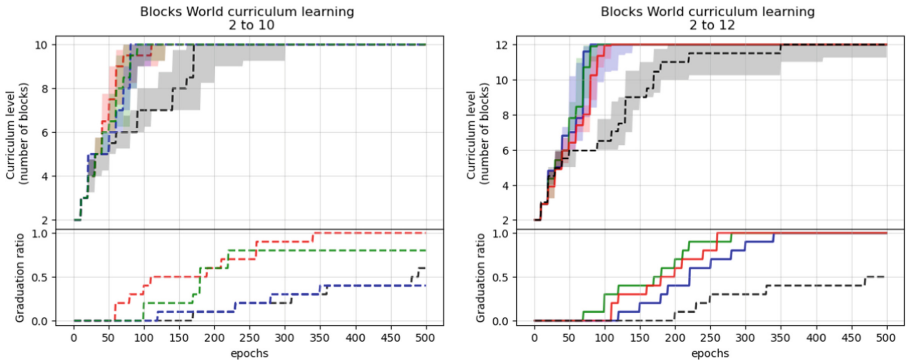


Fig. 6. Training of Blocks World agents using NLM and different rewards. Solid curves with BP: $\mathcal{M}_1$ (blue), $\mathcal{M}_2$ (green), $\mathcal{M}_3$ (red). Dashed coloured curves follow the same color code but are without BP. Black, dashed curve is the baseline with sparse reward. (Color figure online)

The left graph reports curriculum training from 2 to 10 blocks without BP. Agents using shaped rewards outperform the agent using the sparse reward. We observe an impact of the choice of CP model on performance, with stronger models advancing more quickly and consistently through curriculum levels.

The right graph reports curriculum training from 2 to 12 blocks with BP. Agents using shaped rewards advance through the curriculum much faster than the agent using the sparse reward. The bottom part also shows that the agents using shaped rewards graduate earlier and more consistently, graduating all 10

seeds whereas the agent with the sparse reward only graduated in 5 out of 10 seeds, which is consistent with the original experiments by Dong et al. Another important observation is that, surprisingly, the choice of CP model had little impact on training performance. This suggests that the use of BP is able to bridge the gap between models.

5.4 Out-of-Distribution Performance

Table 3 reports the ratio of solved instances on 1000 much larger BW instances averaged across all seeds. Our agents generalize much better than the baseline (which includes seeds that have not graduated after 500 epochs of training), all three CP models achieving very high success rates and with model $\mathcal{M}_3$ exhibiting a near perfect score. So the improvement in performance seen during training from the use of our shaped rewards is also observed on out-of-distribution problems, which suggests that any bias introduced due to the use of our reward scaling factor, if any at all, is negligible.

Table 3. Success rate for BW problems of varying number of blocks.

Model	No. blocks				
	10	20	30	40	50
Baseline (NLM)	0.87	0.80	0.75	0.72	0.69
$\mathcal{M}_1$ (with BP)	1.00	1.00	0.99	0.97	0.96
$\mathcal{M}_2$ (with BP)	1.00	1.00	1.00	0.99	0.98
$\mathcal{M}_3$ (with BP)	1.00	1.00	1.00	1.00	0.99

5.5 Training Time

Figure 7 evaluates the impact of using CP and BP on training time. The top plot shows how the time to compute the potential function, and hence the shaped reward, grows with instance size during curriculum learning: as one would expect, the complexity of the CP model (and the corresponding size of the automata) as well as the use of BP have an impact, but that growth is considerably slower with the smaller $\mathcal{M}_1$ (blue curves). The bottom plot provides an indication in absolute terms of the time spent per training epoch for each model.

Given that all three models with BP behave similarly with regard to sample efficiency and out-of-distribution performance, the lighter model $\mathcal{M}_1$ would be the best choice here to reduce training time.

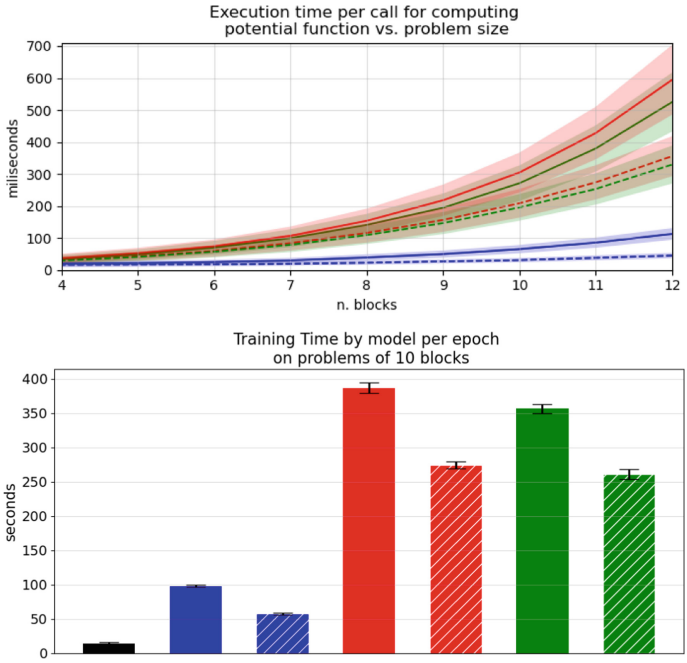


Fig. 7. The top plot shows the execution time in milliseconds per call to compute the potential value of a Blocks World state of n. blocks, averaged over 100 episodes of training. Shading represents the standard deviation. The bottom bar plot shows the execution time per epoch of training on Blocks World instances of 10 blocks. Colors are as usual, hatched bars represent models without the use of BP.

6 Conclusion

Designing a good reward function is an important step in RL to reduce the amount of data necessary to train an efficient model. In this paper, we showed that CP can be used as a semi-automated reward shaping method while preserving the optimality of policies. Our potential-based reward shaping method, when used with BP, is more sample efficient than a sparse reward. This is consistent with our previous findings [42] in another application domain. We also observed that with BP, the effectiveness of our shaped rewards was little affected by the choice of CP model, which helps automate the approach. While there is a computational cost to using CP to train RL agents, they trained using less data, more consistently, and the resulting agents generalize much better.

So far we experimented in a deterministic setting and we would like to consider a probabilistic setting. The existing Markov constraints [6,25,27] fall short of what we need, that is, both the sequence of states and of actions exposed as variables in the scope of the constraint and a fixed probabilistic transition function. We plan to design such a constraint, equipped with its domain filtering and weighted counting algorithms. Our work also only applies to MDPs with

discrete state and action spaces which are required in order to use CP for modeling. Future work can explore methods to model MDPs with continuous state and action spaces by discretizing them or by partially modeling the MDP using abstractions in the fashion of Reward Machines [38].

Acknowledgments. This research was made possible through funding from NSERC Discovery Grant RGPIN-2017-05783.

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

References

1. Agrawal, S., Jia, R.: Optimistic posterior sampling for reinforcement learning: worst-case regret bounds. In: Advances in Neural Information Processing Systems, vol. 30. Curran Associates, Inc. (2017)
2. Babaki, B., Omrani, B., Pesant, G.: Combinatorial search in CP-based iterated belief propagation. In: Simonis, H. (ed.) CP 2020. LNCS, vol. 12333, pp. 21–36. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58475-7_2
3. Bengio, Y., Louradour, J., Collobert, R., Weston, J.: Curriculum learning. In: Proceedings of the 26th Annual International Conference on Machine Learning, Montreal, Quebec Canada, pp. 41–48. ACM (2009). https://doi.org/10.1145/1553374.1553380
4. Berner, C., et al.: Dota 2 with Large Scale Deep Reinforcement Learning. CoRR (2019). http://arxiv.org/abs/1912.06680
5. Bojarski, M., et al.: End to End Learning for Self-Driving Cars (2016). https://doi.org/10.48550/arXiv.1604.07316. http://arxiv.org/abs/1604.07316
6. Bonlarron, A., Régin, J.: Markov constraint as large language model surrogate. In: Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI 2024, Jeju, South Korea, 3–9 August 2024, pp. 1844–1852. ijcai.org (2024). https://www.ijcai.org/proceedings/2024/204
7. Burlats, A., Pesant, G.: Exploiting entropy in constraint programming. In: Ciré, A.A. (ed.) Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 20th International Conference, CPAIOR 2023, Nice, France, 29 May–1 June 2023, Proceedings. Lecture Notes in Computer Science, vol. 13884, pp. 320–335. Springer (2023). https://doi.org/10.1007/978-3-031-33271-5_21
8. Bäckström, C.: Expressive equivalence of planning formalisms. Artif. Intell. **76**(1–2), 17–34 (1995). https://doi.org/10.1016/0004-3702(94)00081-B
9. Clark, J., Amodei, D.: Faulty reward functions in the wild. https://openai.com/index/faulty-reward-functions/
10. Demassey, S., Pesant, G., Rousseau, L.M.: A cost-regular based hybrid column generation approach. Constraints **11**(4), 315–333 (2006). https://doi.org/10.1007/s10601-006-9003-7
11. Devlin, S., Kudenko, D.: Dynamic potential-based reward shaping. In: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 1, AAMAS 2012, pp. 433–440. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2012)

12. Domingues, O.D., Ménard, P., Kaufmann, E., Valko, M.: Episodic reinforcement learning in finite MDPs: minimax lower bounds revisited. In: Proceedings of the 32nd International Conference on Algorithmic Learning Theory, pp. 578–598. PMLR (2021)
13. Dong, H., Mao, J., Lin, T., Wang, C., Li, L., Zhou, D.: Neural logic machines. In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, 6–9 May 2019 (2019)
14. Eck, A., Soh, L.-K., Devlin, S., Kudenko, D.: Potential-based reward shaping for finite horizon online POMDP planning. Auton. Agent. Multi-Agent Syst. **30**(3), 403–445 (2015). https://doi.org/10.1007/s10458-015-9292-6
15. Fedus, W., et al.: Revisiting fundamentals of experience replay. In: International Conference on Machine Learning (ICML). arXiv (2020)
16. Grzes, M., Kudenko, D.: Plan-based reward shaping for reinforcement learning. In: 2008 4th International IEEE Conference Intelligent Systems, vol. 2, pp. 10-22–10-29 (2008). https://doi.org/10.1109/IS.2008.4670492
17. Grześ, M.: Reward shaping in episodic reinforcement learning. In: Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, pp. 565–573. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2017)
18. Jaques, N., Gu, S., Bahdanau, D., Hernández-Lobato, J.M., Turner, R.E., Eck, D.: Sequence tutor: conservative fine-tuning of sequence generation models with KL-control. In: Precup, D., Teh, Y.W. (eds.) Proceedings of the 34th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 70, pp. 1645–1654. PMLR (2017). https://proceedings.mlr.press/v70/jaques17a.html
19. Jin, C., Allen-Zhu, Z., Bubeck, S., Jordan, M.I.: Is Q-learning provably efficient? In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 31. Curran Associates, Inc. (2018)
20. Lafleur, D., Chandar, S., Pesant, G.: Combining reinforcement learning and constraint programming for sequence-generation tasks with hard constraints. In: Solnon, C. (ed.) 28th International Conference on Principles and Practice of Constraint Programming (CP 2022). Leibniz International Proceedings in Informatics (LIPIcs), vol. 235, pp. 30:1–30:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2022). https://doi.org/10.4230/LIPIcs.CP.2022.30. ISSN: 1868-8969
21. Laud, A., DeJong, G.: The influence of reward on the speed of reinforcement learning: an analysis of shaping. In: Proceedings of the Twentieth International Conference on International Conference on Machine Learning, ICML 2003, pp. 440–447. AAAI Press, Washington, DC, USA (2003)
22. Malik, G.: PDDL The Planning Domain Definition Language (1998). https://www.cs.cmu.edu/~mmv/planning/readings/98aips-PDDL.pdf
23. Marom, O., Rosman, B.: Belief reward shaping in reinforcement learning. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, no. 1 (2018). https://doi.org/10.1609/aaai.v32i1.11741
24. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015). https://doi.org/10.1038/nature14236
25. Morin, M., Quimper, C.-G.: The Markov transition constraint. In: Simonis, H. (ed.) CPAIOR 2014. LNCS, vol. 8451, pp. 405–421. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07046-9_29

26. Ng, A.Y., Harada, D., Russell, S.J.: Policy invariance under reward transformations: theory and application to reward shaping. In: Proceedings of the Sixteenth International Conference on Machine Learning, ICML 1999, pp. 278–287. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1999)
27. Pachet, F., Roy, P.: Markov constraints: steerable generation of Markov sequences. Constraints Int. J. **16**(2), 148–172 (2011). https://doi.org/10.1007/S10601-010-9101-4
28. Pesant, G.: From support propagation to belief propagation in constraint programming. J. Artif. Intell. Res. **66**, 123–150 (2019). https://doi.org/10.1613/jair.1.11487
29. Popov, I., et al.: Data-efficient Deep Reinforcement Learning for Dexterous Manipulation (2017). https://doi.org/10.48550/ARXIV.1704.03073
30. Randlov, J., Alstrøm, P.: Learning to drive a bicycle using reinforcement learning and shaping. In: Proceedings of the 15th International Conference on Machine Learning, pp. 463–471 (1998)
31. Saikali, D., Pesant, G.: Constrained molecule generation modelled using the grammar constraint. In: 23rd workshop on Constraint Modelling and Reformulation (ModRef), CP 2024 (2024)
32. Schulman, J.: High-dimensional continuous control using generalized advantage estimation. In: International Conference on Learning Representations (ICLR) (2016). http://arxiv.org/abs/1506.02438
33. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal Policy Optimization Algorithms. CoRR (2017). https://doi.org/10.48550/ARXIV.1707.06347. Version Number: 2
34. Sievers, S., Helmert, M.: Merge-and-shrink: a compositional theory of transformations of factored transition systems. J. Artif. Intell. Res. **71**, 781–883 (2021). https://doi.org/10.1613/jair.1.12557
35. Silver, D., et al.: Mastering the game of Go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016). https://doi.org/10.1038/nature16961
36. Sutton, R.S., Barto, A.G.: Reinforcement learning: an introduction. Adaptive computation and machine learning. MIT Press, Cambridge (1998)
37. Sünderhauf, N., et al.: The limits and potentials of deep learning for robotics. Int. J. Robot. Res. **37**(4–5), 405–420 (2018). https://doi.org/10.1177/0278364918770733
38. Toro Icarte, R., Klassen, T.Q., Valenzano, R., McIlraith, S.A.: Reward machines: exploiting reward function structure in reinforcement learning. J. Artif. Intell. Res. **73**, 173–208 (2022)
39. Vinyals, O., Babuschkin, I., Czarnecki, W., et al.: Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature **575**(7782), 350–354 (2019)
40. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach. Learn. **8**(3), 229–256 (1992). https://doi.org/10.1007/BF00992696
41. Ye, D., et al.: Towards playing full MOBA games with deep reinforcement learning. In: Proceedings of the 34th International Conference on Neural Information Processing Systems. NIPS 2020. Curran Associates Inc., Red Hook (2020)
42. Yin, C., Cappart, Q., Pesant, G.: An improved neuro-symbolic architecture to fine-tune generative AI systems. In: Dilkina, B. (ed.) Integration of Constraint Programming, Artificial Intelligence, and Operations Research, vol. 14743, pp. 279–288. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-60599-4_19
43. Zimmer, M., et al.: Differentiable logic machines. Trans. Mach. Learn. Res. (TMLR) (2021).https://doi.org/10.48550/ARXIV.2102.11529. Version Number: 5

Author Index

G. Tack (Ed.): CPAIOR 2025, LNCS 15763, pp. 271–272, 2025.
https://doi.org/10.1007/978-3-031-95976-9