Proceedings of the 14th International Conference on the Practice and Theory of Automated Timetabling, PATAT 2024

August 27-30, 2024
Technical University of Denmark
Copenhagen, Denmark

ISBN: 978-0-9929984-6-2

Preface

This volume contains full papers, extended abstracts and demonstration abstracts of talks presented at the 14th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2024), held from August 27 to August 30, 2024, in Copenhagen, Denmark.

PATAT is a biennial conference dedicated to all theoretical and practical aspects of computer-aided timetable generation. It serves as the main forum for the EURO Working Group on Automated Timetabling (EWG-PATAT), an international community of researchers, practitioners and vendors. PATAT has been supporting a range of competitions and challenges, investing back in the timetabling community for the benefit of the field. Previous PATAT conferences have been held in Edinburgh, UK (1995), Toronto, Canada (1997), Konstanz, Germany (2000), KaHo St.-Lieven, Gent, Belgium (2002), Pittsburgh, PA USA (2004), Brno, Czech Republic (2006), Montréal Canada (2008), Belfast, Northen Ireland (2010), Son, Norway (2012), York, United Kingdom (2014), Udine, Italy (2016), Vienna, Austria (2018), Leuven, Belgium (2022).

This volume includes the abstracts of four invited talks by Kaisa Miettinen, Stefan Røpke, Sigrid Knust and Nysret Musliu, the abstract of one invited tutorial by Kate Smith-Miles, and 45 contributed submissions consisting of 12 full papers, 30 extended abstracts and 3 demonstration abstracts. Each contribution went through a review process consisting of two or three single-blind reviews. No particular precaution was adopted for handling contributions co-authored by committee members. The accepted submissions (45 out of 50) were presented in two parallel tracks at the conference.

We thank the sponsors of PATAT 2024 for their support: Eventmap, EWG European Working Group in Automated Timetabling and Macom (Lectio). We thank the members of the steering and program committees, who agreed to review the submissions. Finally, we are thankful to Katrine Heide Sørensen for secretarial support at the Technical University of Denmark.

Copenhagen and Odense,
August 2024

Thomas Jacob Riis Stidsen
Marco Chiarandini
Program Chairs
PATAT 2024

Organization

Program Committee Chairs

Thomas Jacob Riis Stidsen	Technical University of Denmark, Copenhagen (DK)
Marco Chiarandini	University of Southern Denmark, Odense (DK)
Ender Özcan	University of Nottingham (UK)

Steering Committee

Edmund K.Burke (Chair)	University of Leicester (UK)
Barry McCollum (Treasurer)	Queen's University Belfast, Northern Ireland (UK)
Patrick De Causmaecker	Katholieke Universiteit Leuven (BE)
Jeffrey H. Kingston	University of Sydney, Australia
Nysret Musliu	Vienna University of Technology (AT)
Ender Özcan (Executive Officer)	University of Nottingham (UK)
Hana Rudova	Masaryk University (CZ)
Andrea Schaerf	University of Udine (IT)
Greet Vanden Berghe (EWG-PATAT liaison)	Katholieke Universiteit Leuven (BE)

Program Committee

Ahmed Kheiri	Lancaster University (UK)
Aldy Gunawan	Singapore Management University (SG)
Alexandre Lemos	Outsystems (PT)
Andrea Schaerf	University of Udine (IT)
Andrew Parkes	University of Nottingham (UK)
Daniel Karapetyan	University of Nottingham (UK)
David Van Bulck	Ghent University (BE)
Dries Goossens	Ghent University (BE)
Efstratios Rappos	HEIG-VD (CH)
Elina Rönnberg	Linköping University (SE)
Ender Ozcan	University of Nottingham (UK)
Erwin Pesch	University of Siegen (DE)

Federico Della Croce	DIGEP Politecnico di Torino (IT)
Gerald Lach	MathPLan GmbH (DE)
Gerhard Post	University of Twente (NL)
Günther Raidl	Vienna University of Technology (AT)
Hana Rudová	Masaryk University (CZ)
Hesham Alfares	King Fahd University of Petroleum & Minerals (SA)
Jeffrey H. Kingston	The University of Sydney (AU)
John H. Drake	University of Leicester (UK)
Jonathan Thompson	Cardiff University (UK)
Kadri Sylejmani	University of Prishtina
Luca Di Gaspero	University of Udine (IT)
Marco Chiarandini	University of Southern Denmark (DK)
Marco Lübbecke	RWTH Aachen University (DE)
Mats Carlsson	Research Institutes of Sweden (SE)
Michel Gendreau	Polytechnique Montréal (CA)
Nysret Musliu	TU Wien (AT)
Panayiotis Alefragis	University of Peloponnese (GR)
Paolo Toth	University of Bologna (IT)
Pieter Smet	KU Leuven (BE)
Rhydian Lewis	Cardiff University (UK)
Sanja Petrovic	University of Nottingham (UK)
Sara Ceschia	University of Udine (IT)
Thomas Stidsen	Technical University of Denmark (DK)
Thomas Stützle	Université Libre de Bruxelles (BE)
Tomas Muller	Purdue University (US)
Túlio Angelo Machado Toffolo	KU Leuven (BE)

Sponsoring Institutions

EVENTMAP®	Eventmap
EWG PATAT AUTOMATED TIMETABLING	EURO Working Group on Automated Timetabling
Lectio	Macom

Table of Contents

Part I

Invited Talks

Some Views to Multiobjective Optimization with a Focus on Interactive Methods

Kaisa Miettinen

Faculty of Information Technology
University of Jyväskylä (JYU), Finland

Abstract

In various real decision problems, we must optimize several conflicting objective functions simultaneously. This means that we must solve multiobjective optimization problems. These problems have so-called Pareto optimal solutions representing different trade-offs and they cannot be ordered mathematically without some additional information. Typically, we assume that a domain expert called a decision maker provides preference information to guide the solution process. By applying appropriate methods, we can find the best balance among the trade-offs. In this talk, I classify multiobjective optimization methods based on the role of the decision maker and devote most attention to interactive methods, where the decision maker augments the problem formulation with domain expertise. The decision maker directs the iterative solution process with one's preferences to find the most preferred solution. At the same time, the decision maker gains insight into the interdependencies and trade-offs among the conflicting objective functions and can get convinced of the quality of the most preferred solution. I demonstrate the advantages of applying interactive methods with some example problems. In addition, I give a brief overview of the modular, open-source software framework DESDEO containing different interactive methods

Decomposition methods for sports scheduling problems

Sigrid Knust

Institute of Computer Science
University of Osnabrück, Germany

Abstract

Generating a sports league schedule is a challenging task due to the variety of different requirements which have to be addressed. The basic problem is to find a schedule for a single/double round robin tournament in which every team plays against each other team exactly once/twice, and every team plays one game per round. Additionally, several side constraints have to be respected, e.g., the avoidance of breaks (consecutive home/away games of a team), fairness issues (like opponent strengths, carry-over effects), the consideration of regions or wishes of teams and media.

This variety of specific problem settings has led to a multitude of alternative approaches (cf., e.g. [1] [2], [3], [4]). Due to its complexity, the problem is often solved by decomposition techniques, i.e., it is divided into different subproblems which are solved consecutively. In this talk, the following three approaches are discussed:

In a "first-schedule, then-break" approach, in the first stage it is decided which teams play against each other in which round. Afterwards, in the second stage home-away patterns (with a minimum number of breaks) corresponding to the pairings from the first stage are determined. In a "first-break, then-schedule" approach, at first home-away patterns are generated for the teams. Then, the subproblem of the second stage consists in finding a corresponding feasible schedule. In a "first assign modes, then schedule" approach, at first for each game a home team is fixed. In the second stage, all games are scheduled in these fixed modes taking into account additional constraints.

References

1. Kendall, G., Knust, S., Ribeiro, C.C., Urrutia, S.: Scheduling in sports: An annotated bibliography. Computers & Operations Research 37, 1–19 (2010).
2. Rasmussen,R.V.,Trick,M.A.:Round robin scheduling —a survey. EuropeanJournal of Operational Research 188, 617-636 (2008).
3. Ribeiro, C.C.: Sports scheduling: problems and applications. International Transactions in Operational Research 19, 201-226 (2012).
4. Van Bulck, D., Goossens, D., Schönberger, J., Guajardo, M.: RobinX: A three-field classification and unified data format for round-robin sports timetabling. European Journal of Operational Research 280, 568–580 (2020).

AI Techniques for Timetabling and Scheduling Problems

Nysret Musliu

Faculty of Informatics
Technical University of Vienna, Austria

Abstract

In this talk, we will first provide an overview of various AI-based methods proposed by our lab for solving problems in application domains such as employee timetabling and project scheduling. The topics covered will include solver-independent modelling, constraint programming, and hybrid techniques. In the second part of the talk, we will discuss methods that utilize machine learning techniques for automatic algorithm selection and heuristic algorithm design. We will also briefly present innovative decision support systems that incorporate our solution methods and an approach for preference explanation to guide decision-makers toward solutions that align with their expectations. The talk will conclude with a discussion of future challenges in the domain of scheduling and timetabling.

Adaptive Large Neighborhood Search

Stefan Røpke

Department of Technology, Management and Economics
Technical University of Denmark

Abstract

Adaptive Large Neighborhood Search (ALNS) is a metaheuristic that extends the Large Neighborhood Search heuristic (LNS) proposed by Paul Shaw. While traditional LNS employs a single method for destroying and repairing solutions iteratively, ALNS introduces multiple such methods. The algorithm keeps track of the performance of each method and attempts to utilize the best methods for the instance at hand. ALNS allows the user to incorporate domain-specific knowledge by adding tailored destroy and repair methods that can exploit the problem's structure or even be targeted at a subset of the instances that need to be solved.

This talk briefly introduces the ALNS algorithm and explores applications to timetabling problems. We discuss the relationship between ALNS and hyperheuristics and review efforts to parallelize ALNS. Additionally, we explore the integration of machine learning into ALNS, particularly focusing on enhancing the selection of destroy and repair methods.

Stress-testing algorithms via Instance Space Analysis

Kate Smith-Miles

School of Mathematics and Statistics
University of Melbourne, Australia

Abstract

Instance Space Analysis (ISA) is a recently developed methodology to support objective testing of algorithms. Rather than reporting algorithm performance on average across a chosen set of test problems, as is standard practice, ISA offers a more nuanced understanding via visualisation of the unique strengths and weaknesses of algorithms across different regions of the instance space that may otherwise be hidden on average. It also facilitates objective assessment of any bias in the chosen test instances, and provides guidance about the adequacy of benchmark test suites and the generation of more diverse and comprehensive test instances to span the instance space. This tutorial provides an overview of the ISA methodology, and the online software tools (seematilda.unimelb.edu.au) that are enabling its worldwide adoption in many disciplines. Several case studies from classical operations research problems will be presented to illustrate the methodology and tools, including timetabling, travelling salesman problem, 0-1 knapsack; and applications to machine learning will also be highlighted.

Part II

Full Papers

An unconstrained binary model for the Uncapacitated Examination Timetabling Problem

Angelos Dimitsas[1][0000−0002−7892−046X], Panayiotis Alefragis[2][0000−0003−1113−8462], Christos Valouxis[3][0000−0001−6832−2311], and Christos Gogos[1][0000−0002−1313−1750]

[1] Dept. of Informatics and Telecommunications, University of Ioannina, Arta, Greece
{a.dimitsas,cgogos}@uoi.gr

[2] Dept. of Electrical and Computer Engineering, University of Peloponnese,Patras, Greece
alefrag@uop.gr

[3] Dept. of Electrical and Computer Engineering, University of Patras, Greece
cvalouxis@upatras.gr

Abstract. Quantum computing is offering a novel perspective for solving combinatorial optimization problems. To explore the possibilities offered by quantum computers, the problems can be formulated as Quadratic Unconstrained Binary Optmization (QUBO) models, taking under consideration the limitations of the current state of Quantum Annealers. QUBO represents a class of optimization problems that involve binary decision variables and quadratic objective functions. It has applications in a wide range of fields and can be solved using classical or quantum optimization techniques, depending on the problem size and complexity. In this work, we provide a QUBO formulation of the Uncapacitated Examination Timetabling Problem along with modifications for symmetry reduction in the context of solving it on a quantum computer. We also introduce a test-bed dataset of small instances suitable for modern annealers, along with optimal solutions to serve for comparison. To prove the efficiency of the formulation we test our model in D-Wave's hybrid annealer.

Keywords: QUBO, Quantum Annealing, Scheduling, Hybrid Quantum Computing, UETP.

1 Introduction

Educational timetabling problems involve the task of scheduling courses, classes, examinations, teachers, and resources within an educational institution to optimize various objectives while satisfying constraints. These problems are common in schools, colleges, and universities, and they can be quite complex.

Examination timetabling is a critical administrative task in educational institutions that involves scheduling examinations for students, ensuring that all examinations are conducted smoothly, and minimizing conflicts or constraints. This process can be complex due to various factors, including room availability, student preferences, and the need to optimize resource utilization.

The UETP is a specific variant of the examination timetabling problem that focuses on scheduling a set of examinations within a given time frame and without considering room capacities or constraints related to room allocation. In other words, it assumes that all examinations can be accommodated in any available room, making it a simplified version of the more complex capacitated examination timetabling problem. UETP has practical applications in educational institutions where examinations need to be scheduled within a specific time frame without considering room constraints. It is a foundational problem in examination timetabling, and solutions to UETP can be further extended to handle capacitated versions of the problem.

While UETP does not consider room capacities, it still has constraints to satisfy:

- No two examinations for the same student should be scheduled in the same time slot (to avoid conflicts).
- Each examination can only be scheduled once.

To encourage greater preparation and less stress for the students, their schedules should also contain sufficient gaps between examinations for all students. Carter et al. [3] introduced in 1996 the problem along with a dataset made of real-life instances, and numerous researchers have experimented with this dataset since then.

Quantum annealing is a specialized quantum computing approach used to solve optimization problems. It is considered one of the quantum computing paradigms, alongside with other methods like quantum gate-based computing. Quantum annealers (QAs) are designed to tackle optimization problems by leveraging quantum properties to potentially find more efficient solutions than classical computers for specific types of problems. Quantum annealing and QUBO are closely related concepts in the field of quantum computing and optimization. QUBO is a mathematical formulation used to express certain optimization problems, and quantum annealing is a quantum computing approach that can be applied to solve QUBO problems.

An outline of the paper follows. Section 2 contains a glimpse of the broad bibliography regarding the UETP and education timetabling in general along with QUBO formulations of other scheduling problems. Section 3 provides a brief description of the problem along with symmetries that have been identified in the past. Section 4 introduces the dataset we created to allow instances of the UETP to fit in modern QAs. Section 5 contains the QUBO model accompanied with a minimal example. Finally, in Section 6 we demonstrate the results obtained by testing our dataset using D-Wave's [1] cloud-based hybrid solvers.

2 Related Work

The related work about the examination timetabling problem in general and UETP in particular is very large. We refer the interested readers to the survey papers [9] and [4] while our recent paper [5] uncovers some of the symmetries that are found in UETP.

Regarding QUBO a nice introduction to the subject can be found at [6]. More specifically, QUBO models have been tried for several scheduling and timetabling problems [11]. For example the nurse scheduling problem has been addressed using QUBO in [7]. Other examples can be found in [8], [12]. Another resource that is worth mentioning is [10] which presents a list of QUBO formulations for several optimization problems.

Quantum computing is a fascinating relatively new computing paradigm that holds the promise of surpassing the limits of computation that currently exist. It is based on a new non Von Neumann architecture and several technology companies invest large amounts of money and resources in an effort to realize such systems. D-Wave is a leading company for quantum computing and in this paper we use the so-called hybrid solver of D-Wave for our experiments. An evaluation of quantum and hybrid solvers for combinatorial problems can be found at [2] published on arXiv.

3 Problem Description

UETP instances contain students, the examinations they participate and the total number of available periods P for the entire timetable. Uncapacitated, as indicated, denotes the absence of room restrictions. Additionally, none of the other restrictions that are typical found in actual examination scheduling exist. These limitations include the availability of the examiners, the order of the examinations, the grouping of the times on weekdays, and others. Consequently, UETP can be seen as an abstraction of the actual examination scheduling problem.

An instance can be thought as an undirected weighted graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$, where vertices $\mathbb{V}$ represent examinations and edges $\mathbb{E}$ represent common students between examinations. The number of students who take both of the examinations at the edge's ends makes up the edge's weight W_{v_1,v_2}. The only strict requirements are that a) each examination should only be scheduled once, and b) no student should be permitted to take more than one examination per period. The quality of a timetable is measured by an objective function. Each student applies a penalty of 16, 8, 4, 2, 1 for intervals of 1, 2, 3, 4 or 5 periods between each of his examinations respectively. Notation used in this paper is shown in Table 1.

Table 1: Notation used for describing UETP.

Sets	
$\mathbb{V}$	Set of examinations.
$\mathbb{E}$	Set of pairs of examinations with students in common.
$\mathbb{P}$	Set of periods.
Constants	
F_{p_i,p_j}	$\begin{cases} 2^{5-\lvert p_i-p_j\rvert}, & \text{if } 0 < \lvert p_i - p_j\rvert \leq 5 \\ 0, & \text{otherwise.} \end{cases}$
W_{v_1,v_2}	Total number of common students between examinations v_1 and v_2.

$$x_{v,p} = \begin{cases} 1, & \text{if examination } v \text{ is placed in period } p. \\ 0, & \text{otherwise.} \end{cases} \quad \forall v \in \mathbb{V} \quad \forall p \in \mathbb{P} \tag{1}$$

$$\min \sum_{(v_1,v_2)\in\mathbb{E}} \sum_{p_1\in\mathbb{P}} \sum_{p_2\in\mathbb{P}} F_{p_1,p_2} W_{v_1,v_2} x_{v_1,p_1} x_{v_2,p_2} \tag{2}$$

$$\text{s.t.} \quad x_{v_1,p} + p_{v_2,p} <= 1 \quad \forall (v_1, v_2) \in \mathbb{E} \quad \forall p \in 1..P \tag{3}$$

$$\sum_{p=1}^{P} x_{v,p} = 1 \quad \forall v \in \mathbb{V} \tag{4}$$

The penalty factor for the intervals of periods is calculated as in Table 1. Binary decision variables in Equation 1 denote the period that each examination is placed to. The objective function in equation 2 simply totals the penalties for all examinations. Finally, constraint 3 ensures than no two examinations sharing students are in the same period and constraint 4 obligates each examination to be placed once and only once.

3.1 Bidirectional Timetable Symmetry

It is easy to observe that if period $p \in 1..P$, changes to $P-p+1$ for all examinations, then we effectively get the original solution reversed. Since the cost is computed based on the distance among periods of scheduled examinations, the objective function is unaffected as demonstrated in equation 5.

$$\sum_{(v_1,v_2)\in\mathbb{E}} \sum_{p_1\in\mathbb{P}} \sum_{p_2\in\mathbb{P}} F_{p_1,p_2} W_{v_1,v_2} x_{v_1,p_1} x_{v_2,p_2} = \sum_{(v_1,v_2)\in\mathbb{E}} \sum_{p_1\in\mathbb{P}} \sum_{p_2\in\mathbb{P}} F_{p_1,p_2} W_{v_1,v_2} x_{v_1,(P-p_1+1)} x_{v_2,(P-p_2+1)} \tag{5}$$

4 Dataset

Different datasets regarding the UETP problem were made public over the years, but the sheer size of the included instances make them to big to fit in current state of the art annealers. While the number of qubits required for some small instances is acceptable, the nature of the problem i.e., the relation of two exams with students in common, results in an increase of the Non Zero Couplings in the matrix that is sent to the solver (usually called a QMatrix) provided to the solver, thus making most of these instances unfit for the annealer.

In order to demonstrate the proof of concept we opted to generate a dataset consisting of 50 small instances able to run on current annealers. To create an instance we randomly choose between 3 and 7 exams and generate a complete graph with them (all of them have students in common) the number of the periods available equals the number of nodes in the complete graph to make the instance compact e.g., there exists no solution with an empty period, then we proceed to add more exams and more conflicts while keeping the number of conflicts under 60. The students in common between the conflicting exams (the weight of their edge) is chosen arbitrarily between 1 and 100. However, this number could be higher as this will not result in more variables.

To test the annealer against the optimal solutions we employ GoogleOR-Tools CP-SAT Solver to solve the problem instances to optimality. The characteristics and optimal solutions values are presented in Table 2.

Table 2: Instances and characteristics

Instance	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Examinations	20	19	8	13	22	19	20	17	18	20
Periods	6	6	5	5	4	6	7	5	7	6
Conflict density	0.23	0.28	0.37	0.55	0.19	0.25	0.24	0.3	0.27	0.27
Optimal	10512	10736	13540	17376	16152	9916	7458	13242	8365	12720
Instance	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Examinations	24	24	20	18	19	15	23	17	27	8
Periods	4	4	6	5	4	5	5	5	7	6
Conflict density	0.15	0.15	0.23	0.29	0.25	0.38	0.17	0.33	0.12	0.38
Optimal	16312	13460	10018	13088	16700	12724	9640	11704	8281	12255
Instance	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**
Examinations	19	19	26	17	20	19	27	23	19	17
Periods	4	5	7	7	6	7	7	7	6	4
Conflict density	0.25	0.27	0.14	0.29	0.24	0.24	0.13	0.17	0.24	0.32
Optimal	15592	15628	5316	7464	10445	8301	5718	9265	7700	19680
Instance	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**
Examinations	21	25	17	15	22	26	20	17	18	19
Periods	6	6	7	4	4	7	6	5	5	7
Conflict density	0.21	0.16	0.33	0.39	0.19	0.14	0.24	0.31	0.29	0.25
Optimal	8749	8834	8780	20364	17412	6735	11691	12356	14434	7221
Instance	**41**	**42**	**43**	**44**	**45**	**46**	**47**	**48**	**49**	**50**
Examinations	24	15	17	21	23	8	18	24	19	18
Periods	7	5	5	4	6	6	5	6	6	6
Conflict density	0.16	0.44	0.33	0.23	0.17	0.38	0.27	0.18	0.25	0.3
Optimal	5885	15108	18630	24604	8456	9731	7964	8723	9407	9654

5 Unconstrained Binary Model

The general form of a QUBO objective function can be expressed as follows:

$$\min \quad \sum_{i=1}^{n} q_{ii}x_i + \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} q_{ij}x_ix_j$$

where:

- x_i are binary variables.
- n is the number of binary variables.
- q_{ii} represents the linear coefficient associated with variable x_i.
- q_{ij} represents the quadratic coefficient associated with the interaction between variables x_i and x_j.

The model for a QUBO problem will always be the same. What makes the difference is the choice of the values in the QMatrix. For this problem our binary decision variables assume the value 1 when a specific exam is scheduled in a period. For exams in conflict the corresponding quadratic coefficient is calculated as $F_{p_1,p_2}W_{v_1,v_2}/2$. We divide by two because the QMatrix is symmetric. As the nature of QUBO formulation is inherently unconstrained we choose a large enough number M to impose penalties and incentives in the objective function that can act as constraints. We chose M to equal the sum of all edges multiplied by 16 to ensure that no worse solution exists when you violate the conflicting exams constraint 3. To provide the incentive to schedule all exams, as dictated by constraint 4, we set the value of an exam being placed to $-M$ and to M if the exam is placed twice.

We use a minimal problem presented in Figure 1 to demonstrate the resulting QMatrix in Table 3. Note that this toy example involves 5 examinations and 3 periods.

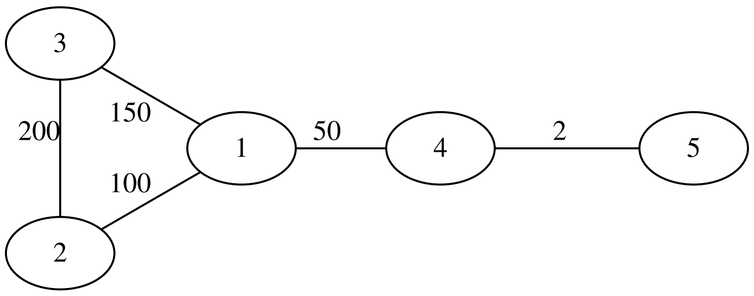


Fig. 1: Minimal problem graph (5 examinations, 3 periods).

We can also choose to eliminate the bidirectional timetable symmetry discussed in Section 3.1 by restricting any two conflicting exams to be placed in a certain order. If

Table 3: Q Matrix

	E_1P_1	E_1P_2	E_1P_3	E_2P_1	E_2P_2	E_2P_3	E_3P_1	E_3P_2	E_3P_3	E_4P_1	E_4P_2	E_4P_3	E_5P_1	E_5P_2	E_5P_3
E_1P_1	-M	M	M	M	800	400	M	1200	1200	M	400	200	0	0	0
E_1P_2	M	-M	M	800	M	800	1200	M	1200	400	M	400	0	0	0
E_1P_3	M	M	-M	400	800	M	1200	1200	M	200	400	M	0	0	0
E_2P_1	M	800	400	-M	M	M	M	1600	800	0	0	0	0	0	0
E_2P_2	800	M	800	M	-M	M	1600	M	1600	0	0	0	0	0	0
E_2P_3	400	800	M	M	M	-M	800	1600	M	0	0	0	0	0	0
E_3P_1	M	1200	1200	M	1600	800	-M	M	M	0	0	0	0	0	0
E_3P_2	1200	M	1200	1600	M	1600	M	-M	M	0	0	0	0	0	0
E_3P_3	1200	1200	M	800	1600	M	M	M	-M	0	0	0	0	0	0
E_4P_1	M	400	200	0	0	0	0	0	0	-M	M	M	M	16	8
E_4P_2	400	M	400	0	0	0	0	0	0	M	-M	M	16	M	16
E_4P_3	200	400	M	0	0	0	0	0	0	M	M	-M	8	16	M
E_5P_1	0	0	0	0	0	0	0	0	0	M	16	8	-M	M	M
E_5P_2	0	0	0	0	0	0	0	0	0	16	M	16	M	-M	M
E_5P_3	0	0	0	0	0	0	0	0	0	8	16	M	M	M	-M

we choose that exam 1 must be scheduled before exam 3 the binary value E_1P_1 is not needed anymore as exam 1 cannot be placed in the first period and for each combination where exam 1 is scheduled before exam 3 we again provide the value of M to place a heavy penalty if such a combination is selected. The QMatrix with these modifications is presented in Table 4.

Table 4: Q Matrix without bidirectional timetable symmetry

	E_1P_2	E_1P_3	E_2P_1	E_2P_2	E_2P_3	E_3P_1	E_3P_2	E_3P_3	E_4P_1	E_4P_2	E_4P_3	E_5P_1	E_5P_2	E_5P_3
E_1P_2	-M	M	800	M	800	1200	M	**M**	400	M	400	0	0	0
E_1P_3	M	-M	400	800	M	1200	1200	M	200	400	M	0	0	0
E_2P_1	800	400	-M	M	M	M	1600	800	0	0	0	0	0	0
E_2P_2	M	800	M	-M	M	1600	M	1600	0	0	0	0	0	0
E_2P_3	800	M	M	M	-M	800	1600	M	0	0	0	0	0	0
E_3P_1	1200	1200	M	1600	800	-M	M	M	0	0	0	0	0	0
E_3P_2	M	1200	1600	M	1600	M	-M	M	0	0	0	0	0	0
E_3P_3	**M**	M	800	1600	M	M	M	-M	0	0	0	0	0	0
E_4P_1	400	200	0	0	0	0	0	0	-M	M	M	M	16	8
E_4P_2	M	400	0	0	0	0	0	0	M	-M	M	16	M	16
E_4P_3	400	M	0	0	0	0	0	0	M	M	-M	8	16	M
E_5P_1	0	0	0	0	0	0	0	0	M	16	8	-M	M	M
E_5P_2	0	0	0	0	0	0	0	0	16	M	16	M	-M	M
E_5P_3	0	0	0	0	0	0	0	0	8	16	M	M	M	-M

6 Experiments and results

Our experiments were performed using the hybrid Quantum Annealer provided by D-Wave. A time limit of 20 seconds was given for each problem instance and the results are presented in Table 5. The justification for using only 20 seconds of running time per instance is due to the small sizes of the problems and the limited time that the hybrid solver of D-Wave can use the Quantum infrastructure for the non-pay version of D-Wave Leap.

Results show that there is potential for using Quantum Annealers for solving UETP problems. Some results are optimal, while others are near optimal, as can be seen in Figure 2 which shows how far from the optimal solution the results for the 50 problem instances are.

Table 5: Results

Instance	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Decision Variables	119	113	79	64	87	113	139	84	125	119
Non Zero Couplings	1950	2061	1290	1215	908	1851	2596	1230	2370	2226
Objective	12082	12687	13540	17376	16784	11265	9515	13302	9176	14899
Difference	3.47%	3.26%	11.38%	0.00%	6.62%	0.60%	6.58%	4.12%	3.56%	4.16%
Instance	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Decision Variables	95	95	119	89	75	74	114	84	188	95
Non Zero Couplings	864	864	1950	1310	842	1165	1405	1330	2669	1926
Objective	17504	15336	11244	13996	16700	13312	11346	12630	11224	12255
Difference	0.00%	0.00%	0.96%	3.18%	6.06%	0.11%	2.31%	3.94%	1.76%	2.88%
Instance	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**
Decision Variables	75	94	181	118	119	132	188	160	113	67
Non Zero Couplings	826	1375	2729	2277	2004	2392	2904	2645	1821	814
Objective	15592	16662	8448	8584	12264	9922	9499	11914	9091	19680
Difference	1.68%	0.00%	1.13%	4.06%	1.90%	7.54%	0.00%	0.00%	1.60%	3.49%
Instance	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**
Decision Variables	125	149	118	59	87	181	119	84	89	132
Non Zero Couplings	1971	2229	2524	770	880	2788	2016	1250	1325	2439
Objective	10426	11357	10342	20364	17968	10187	12908	12564	14794	8798
Difference	4.01%	4.45%	12.42%	6.25%	4.14%	0.00%	4.37%	6.25%	4.08%	0.79%
Instance	**41**	**42**	**43**	**44**	**45**	**46**	**47**	**48**	**49**	**50**
Decision Variables	167	74	84	83	137	95	89	143	113	107
Non Zero Couplings	2638	1320	1335	946	1977	1890	1250	2214	1857	1974
Objective	8789	15108	18790	24604	11036	10404	8156	11369	11097	11133
Difference	10.20%	2.47%	0.42%	0.62%	4.92%	9.90%	0.00%	0.21%	0.00%	1.67%

7 Conclusions

In this paper we tried to present a proof of concept idea about using a QUBO formulation and a Quantum Annealer solver for solving UETP. We created a custom dataset of small problem instances keeping in mind the current limitations of the Quantum Annealers and modeled the problem according to QUBO. We then run experiments on the non-pay version of D-Wave's Leap architecture. Our results are promising, although they do not manage to find optimal solutions for all problem instances given only 20 seconds for each problem, they achieve near optimal results for most of the cases.

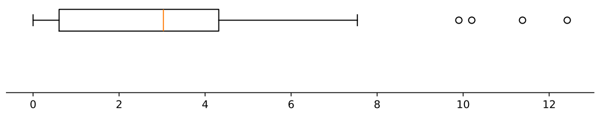


Fig. 2: Difference in percentage from the optimal solution for 50 problem instances.

References

1. D-wave systems web page, https://www.dwavesys.com/, accessed on April, 2024
2. Bertuzzi, A., Ferrari, D., Manzalini, A., Amoretti, M.: Evaluation of quantum and hybrid solvers for combinatorial optimization. arXiv (2024)
3. Carter, M.W., Laporte, G., Lee, S.Y.: Examination timetabling: Algorithmic strategies and applications. Journal of the operational research society **47**, 373–383 (1996)
4. Ceschia, S., Di Gaspero, L., Schaerf, A.: Educational timetabling: Problems, benchmarks, and state-of-the-art results. European Journal of Operational Research **308**(1), 1–18 (2023)
5. Dimitsas, A., Gogos, C., Valouxis, C., Nastos, V., Alefragis, P.: A proven optimal result for a benchmark instance of the uncapacitated examination timetabling problem. Journal of Scheduling (Mar 2024). https://doi.org/10.1007/s10951-024-00805-0
6. Glover, F., Kochenberger, G., Du, Y.: Quantum bridge analytics i: a tutorial on formulating and using qubo models. 4or **17**(4), 335–371 (2019)
7. Ikeda, K., Nakamura, Y., Humble, T.S.: Application of Quantum Annealing to Nurse Scheduling Problem. Scientific Reports **9**(1), 12837 (Sep 2019). https://doi.org/10.1038/s41598-019-49172-3, number: 1 Publisher: Nature Publishing Group
8. Ossorio-Castillo, J., Pena-Brage, F.: Optimization of a refinery scheduling process with column generation and a quantum annealer. Optimization and Engineering **23**(3), 1471–1488 (Sep 2022). https://doi.org/10.1007/s11081-021-09662-8
9. Qu, R., Burke, E.K., McCollum, B., Merlot, L.T., Lee, S.Y.: A survey of search methodologies and automated system development for examination timetabling. Journal of scheduling **12**, 55–89 (2009)
10. Ratke, D.: List of QUBO formulations, https://blog.xa0.de/post/List-of-QUBO-formulations/, accessed on April, 2024
11. Stollenwerk, T., Basermann, A.: Experiences with Scheduling Problems on Adiabatic Quantum Computers (Jan 2016)
12. Venturelli, D., Marchand, D., Rojo, G.: Job shop scheduling solver based on quantum annealing. In: Proc. of ICAPS-16 Workshop on Constraint Satisfaction Techniques for Planning and Scheduling (COPLAS). pp. 25–34 (2016)

Instance Space Analysis for the Bus Driver Scheduling Problem

Tommaso Mannelli Mazzoli[1] [0000-0002-5861-3155], Lucas Kletzander[2] [0000-0002-2100-7733], Nysret Musliu[2] [0000-0002-3992-8637], and Kate Smith-Miles[3] [0000-0003-2718-7680]

[1] TU Wien, Vienna, Austria
[2] Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling, DBAI, TU Wien, Vienna, Austria
{tommaso.mazzoli,lucas.kletzander,nysret.musliu}@tuwien.ac.at
[3] ARC Training Centre in Optimisation Technologies, Integrated Methodologies and Applications (OPTIMA), School of Mathematics and Statistics, The University of Melbourne, Parkville, Victoria, Australia smith-miles@unimelb.edu.au

Abstract. The Bus Driver Scheduling Problem is a combinatorial optimisation problem with important real-world applications. The goal is to assign bus drivers to predetermined bus tours in order to minimise an objective function that considers the total time of each employee's one-day work shift and their dissatisfaction.

Due to the amount of complex rules specified by a collective agreement and European laws, this problem is highly constrained. Thus, exact methods are computationally intractable. In recent work, two metaheuristics have been proposed to solve this problem: Large Neighbourhood Search (LNS) and Construct, Merge, Solve and Adapt (CMSA). In the literature, 65 real-world-like instances have been used to test the algorithms. Among those instances, LNS seems to outperform CMSA; nevertheless, the reason was still obscure.

In order to investigate the reason, we use Instance Space Analysis to show the weaknesses and strengths of the two solution methods. First, we greatly extend an instance generator to be able to generate varied real-world-like and synthetic instances. This allows us to expand the previous set of instances from the literature.

We then present a set of features that describe the hardness of the instances. The features consider the structure of the instance, such as the average break length for each vehicle or the distribution of bus tours in the city. We observe that even if LNS outperforms CMSA in real-world-like instances, it does not for some synthetic ones.

Using Instance Space Analysis, each instance is projected into a 2D plane based on selected features. We see clusters of instances in the instance space, and the real-world-like are in the centre. The bus tour structure appears to have an impact on the performance of the algorithms. Using this information, we can gain insights into the weaknesses and strengths of the two algorithms.

Keywords: Instance Space Analysis, Bus Driver Scheduling Problem, Scheduling, Combinatorial Optimisation

1 Introduction

This work deals with the Bus Driver Scheduling Problem and how to (1) generate new *diverse* instances and (2) how to objectively assess how the state-of-the-art algorithms perform on those diverse instances.

The Bus Driver Scheduling Problem (BDSP) is a sub-problem of the general Transportation Planning System, that includes Vehicle Scheduling, Crew Rostering, and Timetabling [11]. The goal is to assign bus drivers to predetermined routes while minimising a specified objective function that considers operating costs as well as employee dissatisfaction with their work shifts.

The problem has a clear practical relevance. Recently, a variant of the BDSP with complex break constraints has been studied [2,3,1,7,5]. In this setting, there a set of 65 real-world-like instances (named *Realistic*) and performance results for two state-of-the-art algorithms: CMSA [7] and LNS [5]. Based on the 65 Realistic instances, LNS appears to outperform CMSA. However, we must scrutinise the diversity of these test instances and seek to ensure they span a range of instance characteristics before conclusions can be drawn about the merits of each algorithm.

Instance Space Analysis [10] is a methodology that allows the diversity of a set of test instances to be visually examined, and insights into the strengths and weaknesses of algorithms, across the mathematically defined boundaries of the instance space, to be observed.

Our research contributions of this paper are:

- We developed an instance generator with whom we have generated diverse instances.
- We propose a set of features of the test instances to characterise their similarities and differences.
- We compare the two main state-of-the-art algorithms for the BDSP on real-world-like and synthetic instances.
- We scrutinise the instance space and point out the regions of instance space that are not yet sufficiently covered.

This paper is organised as follows: Section 2 describes the Bus Driver Scheduling Problem in detail. In Section 3 we present the Instance Space Analysis framework, and the meta-data we built for the Instance Space Analysis. In Section 4, we show and visualise the Instance Space Analysis. Finally, in Section 5 we present the conclusions of the work and its future possible developments.

2 The Bus Driver Scheduling Problem

The investigated Bus Driver Scheduling Problem deals with the assignment of bus drivers to vehicles that already have a predetermined route for one day of operation. The problem specification is taken from the literature [2].

2.1 Problem Input

The input of the BDSP consists in three pieces of data:

– For a set P of *positions* (for instance, a set of bus stops), a time distance matrix $D = (d_{ij}) \in \mathbb{R}^{|P|\times|P|}$ is given, where d_{ij} represents the time needed for an employee to go from position i to j when not actively driving a bus. If no transfer is possible, then $d_{ij} = M$, where M is a very large number. If $i \neq j$, then d_{ij} is called *passive ride time*, whereas d_{ii} represents the time it takes to switch vehicle at the same position, but is not considered passive ride time.
– For each position $p \in P$ two values $\mathit{startWork}_p$ and $\mathit{endWork}_p$: they represent respectively the amount of working time required to start or end a work shift at that position.
– A set L of *bus legs*: each leg $\ell \in L$ is a 5-tuple:

$$\ell = (\mathit{tour}_\ell, \mathit{startPos}_\ell, \mathit{endPos}_\ell, \mathit{start}_\ell, \mathit{end}_\ell),$$

representing the trip of a vehicle between two stops at a certain time:
- $\mathit{tour}_\ell \in \mathbb{N}$ is the ID of the vehicle
- $\mathit{startPos}_\ell, \mathit{endPos}_\ell \in P$ are respectively the starting and the ending positions of the leg
- $\mathit{start}_\ell \in \mathbb{R}$ is the time when the vehicle departs from position $\mathit{startPos}_\ell \in \mathbb{R}$
- $\mathit{end}_\ell \in \mathbb{R}$ is the time at which the vehicle arrives to position endPos_ℓ

Legs with the same tour t do not overlap: the intervals $(\mathit{start}_\ell, \mathit{end}_\ell)$ for ℓ such that $\mathit{tour}_\ell = t$ are disjoint. The set L is totally ordered by *start*, using *tour* as tie-breaker.

ℓ	tour_ℓ	start_ℓ	end_ℓ	$\mathit{startPos}_\ell$	endPos_ℓ
1	1	420	495	0	1
2	1	520	530	1	2
3	1	540	550	2	1
4	1	558	570	1	0

Table 1: A Bus Tour Example

Table 1 shows a short example of one particular bus tour. The vehicle starts at time 420 (6:40 AM) at position 0, does multiple legs between positions 1 and 2 with waiting times in between and finally returns to position 0.

For Realistic instances, the number of legs is proportional to the number of bus tours with approximately $n_{\text{legs}} \approx 10 \cdot n_{\text{tours}}$.

2.2 Solution

A solution S to the problem is an assignment $S\colon L \to E$, where $E \subseteq \mathbb{N}$ is the set of employees. The number of drivers is not given, but one can imagine setting it as large as needed to have a feasible solution.

Equivalently, it is useful to represent a solution by a set of *shifts*, that is the work scheduled to be performed by a driver in one day [11]. More precisely, the shift of driver

$e \in E$ is the preimage $L_e = S^{-1}(\{e\})$ with the total order induced by L. Hence, the notion of *consecutive* bus legs in a shift is well-defined.

Each shift of a driver $e \in E$ must be feasible according to the following criteria:

- No overlapping bus legs are assigned to e.
- Changing tour or position between consecutive legs $i, j \in L_e$ requires

$$start_j \geq end_i + d_{endPos_i, startPos_j}.$$

- The shift L_e respects all hard constraints regarding work regulations as specified below. These refer to different measures of time related to an employee e containing the set of bus legs L_e, as visualised in Figure 1.

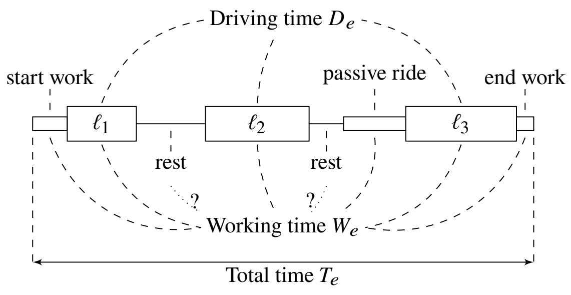


Fig. 1: Example shift [2]

Driving Time Regulations. The *driving time* of a shift L_e is

$$D_e = \sum_{i \in L_e} (end_i - start_i)$$

The driving time D_e cannot exceed $D_{\max} = 9$ h. The driving time is subject to additional rules regarding driving breaks. A *driving break* between two consecutive bus legs i and j is

$$diff_{ij} = start_j - end_i$$

After at most 4 h of driving time, one of the following has to occur:

- One driving break of at least 30 min.
- Two driving breaks of at least 20 min each.
- Three driving breaks of at least 15 min each.

Once we reach all required breaks, the next block of at most 4 h starts.

Total Time Regulations. The *total time* of a shift L_e is the span from the start of work until the end of work:

$$T_e = end_\ell + endWork_{endPos_\ell} - (start_f - startWork_{startPos_f})$$

where f is the first leg and ℓ is the last leg in the shift L_e. A hard limit of $T_e \leq T_{\max} = 14\,\mathrm{h}$ is enforced.

Shift Splits. We say that the employee e has a *shift split* if L_e contains two consecutive legs i and j such that:

$$start_j - end_i - r_{endPos_i, startPos_j} \geq 3\,\mathrm{h}$$

where $r_{p,q} = d_{p,q}$ if $p \neq q$, else $r_{p,p} = 0$. Denote by $split_e$ the number of shift splits and by $splitTime_e$ the total amount of time the driver e spends on a shift split. A shift split resets the driving time (i.e., it counts as a *driving* break). A shift contains up to two shift splits.

Shift splits are unpaid, so they are badly regarded by bus drivers. This will play a role in designing the objective function.

Working Time Regulations. The working time W_e cannot exceed 10 h and has a soft minimum of 6.5 h. If the employee is working for a shorter period of time, the difference has to be paid anyway.

A minimum rest break is required according to the following options:

- $W_e < 6\,\mathrm{h}$: no rest break;
- $6\,\mathrm{h} \leq W_e \leq 9\,\mathrm{h}$: at least 30 min;
- $W_e > 9\,\mathrm{h}$: at least 45 min.

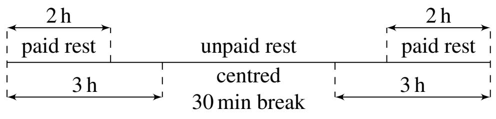


Fig. 2: Rest break positioning [2]

The rest break may be split into one part of at least 30 min and one or more parts of at least 15 min. The first part has to occur after at most 6 h of working time. Whether rest breaks are paid or unpaid depends on break positions according to Figure 2. Every period of at least 15 min of consecutive rest break is unpaid as long as it does not intersect the first 2 or the last 2 hours of the shift (a longer rest break might be partially paid and partially unpaid). The maximum amount of unpaid rest is limited:

- If 30 consecutive minutes of rest break are located such that they do not intersect the first 3 h of the shift or the last 3 h of the shift, at most 1.5 h of unpaid rest are allowed;
- Otherwise, at most one hour of unpaid rest is allowed.

2.3 Objective function

We minimise the objective function combining cost and employee satisfaction defined in previous work [2]:

$$z = \sum_{e \in E} \left(2\, W'_e + T_e + ride_e + 30\, change_e + 180\, split_e \right) \tag{1}$$

The objective function z represents a linear combination of six criteria for each employee e. The actual paid working time $W'_e = \max\{W_e, 390\}$ is the main objective (containing actual working time and additional payments for short shifts), and it is combined with the total time T_e to reduce long unpaid periods for employees. The next sub-objectives reduce the passive ride time $ride_e$ and the number of tours changes $change_e$, which is beneficial for both employees and efficient schedules. The last objective aims to reduce the number of split shifts $split_e$ as they are very unpopular. The weights were determined by previous work [2] based on preferences agreed by different stakeholders at Austrian bus companies and employee scheduling experts. Details of the objective function can be found in previous work [2,3,1].

3 Instance Space Analysis

Instance Space Analysis (ISA) is a methodology proposed by Smith-Miles et al. in 2014 [8] that extends the algorithm selection problem framework of Rice [6]. In ISA, instances are represented as vectors of features. The instances are then projected onto the 2D plane to separate hard and easy instances. Figure 3 illustrates the Instance Space Analysis framework.

The *problem space* $\mathcal{P}$ contains all the theoretically possible instances of the BDSP. Nevertheless, we only have results for a (smaller) subset of instances $\mathbf{I} \subset \mathcal{P}$, for which we have experiment results.

The first component of the meta-data are some chosen features, used to characterize the mathematical and statistical properties of the instances that (1) describe the similarities and differences between instances in (2) have correlation with the performance of one or more algorithms.

For a given instance $x \in \mathbf{I}$, we calculate the feature vector $\boldsymbol{f}(x)$, which represents an instance in the *feature space*, $\mathcal{F}$.

The second component are the performance measures. We imagine to have the algorithm space $\mathcal{A}$ representing the set of algorithms available to solve all instances in $\mathbf{I}$. For each algorithm $\alpha \in \mathcal{A}$ and each instance $x \in \mathbf{I}$, we have a performance measure, $y(\alpha, x)$: an element of a vector that describes the performance space, $\mathcal{Y}$, and requires a user-defined measure of "goodness," such as the objective function value obtained for a fixed computational budget. Both the features and performance measures for all the instances in $\mathbf{I}$, and all algorithms in $\mathcal{A}$ constitute the meta-data, which we represent as two matrices $\boldsymbol{F} = [f_1, \ldots, f_n] \in \mathbb{R}^{m \times n}$ and $\boldsymbol{Y} = [y_1, \ldots y_n] \in \mathbb{R}^{a \times n}$, where m is the number of features, n is the number of problem instances, and a is the number of algorithms. Hence, the meta-data is the set $\{\boldsymbol{F}, \boldsymbol{Y}\}$.

In the original framework proposed by Rice in 1976 [6], a selection mapping S was learned directly from features and performance. Later, in the expanded framework

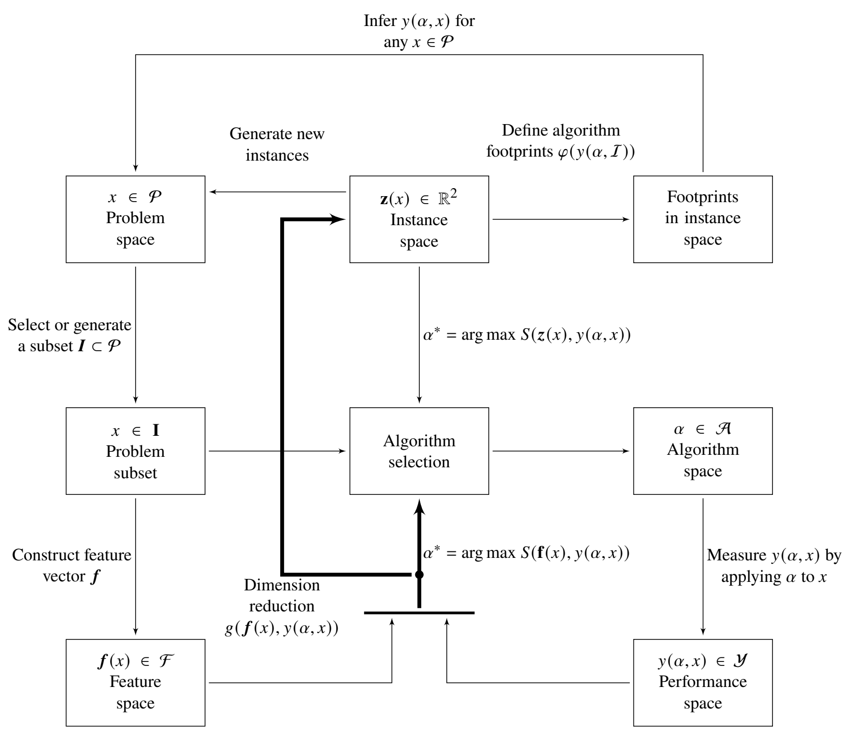

Infer $y(\alpha, x)$ for any $x \in \mathcal{P}$
Generate new instances
Define algorithm footprints $\varphi(y(\alpha, \mathcal{I}))$
$x \in \mathcal{P}$ Problem space
$\mathbf{z}(x) \in \mathbb{R}^2$ Instance space
Footprints in instance space
Select or generate a subset $\boldsymbol{I} \subset \mathcal{P}$
$\alpha^* = \arg\max S(\mathbf{z}(x), y(\alpha, x))$
$x \in \mathbf{I}$ Problem subset
Algorithm selection
$\alpha \in \mathcal{A}$ Algorithm space
Construct feature vector $\boldsymbol{f}$
$\alpha^* = \arg\max S(\mathbf{f}(x), y(\alpha, x))$
Measure $y(\alpha, x)$ by applying α to x
Dimension reduction $g(\boldsymbol{f}(x), y(\alpha, x))$
$\boldsymbol{f}(x) \in \mathcal{F}$ Feature space
$y(\alpha, x) \in \mathcal{Y}$ Performance space

Fig. 3: ISA framework [10] extending the original by Rice [6]

introduced by Smith-Miles et al. in 2014 [8], and extended by Smith-Miles and Muñoz in 2023 [10], instances are projected from the feature space into a lower-dimensional 2D space using the dimension reduction $g\,(\boldsymbol{f}(x), y(\alpha, x))$. It aims to yield an optimal projection that looks for linear trends in both features and algorithmic performance across the resultant instance space, in order to gain interpretable insights. This allows us to get a visualization and enables a more detailed evaluation of algorithmic performance, as well as algorithm selection based on the position of an instance in the instance space.

In the conceptual framework delineated in the preceding section, the ISA methodology involves six fundamental steps:

1. Acquiring experimental meta-data for a designated set of instances $\mathbf{I}$ ran across a set of algorithms $\mathcal{A}$. This includes capturing feature values $\boldsymbol{F}$ for all instances and recording performance metrics $\boldsymbol{Y}$ for all algorithms across all instances.
2. Creating an instance space through a feature selection process applied to the meta-data $\{\boldsymbol{F}, \boldsymbol{Y}\}$, with consideration for a user-defined benchmark for optimal performance, encompassing its theoretical boundaries.
3. Employing machine learning techniques to generate predictions for automated algorithm selection.

4. Establishing algorithm footprints and evaluating associated metrics.
5. Analysing the instance space to extract insights and assess the adequacy of the available meta-data.
6. Generating supplementary meta-data as needed, and iterating from Step 2 onwards, or concluding the process if no further iterations are warranted.

3.1 Problem Subset *I*

In 2020, Kletzander and Musliu [2] proposed a set of 50 real-world-like instances for the BDSP. This set was later [1] extended with 15 new (again real-world-like) instances. The instances are publicly available[4]. For these instances, the number of bus tours ranges from 10 bus tours (about 70 legs) up to 250 (about 2300 bus legs). These instances are build to reproduce particular properties seen in a specific industrial use-case, however, in other settings involving different locations or rule sets, real-world instances might exhibit very different properties.

In order to cover a larger portion of the instance space, we greatly extended the original instance generator. It can generate instances with a larger number of bus stops and more diverse distributions of bus legs and tours during the day. The generator uses 44 parameters.

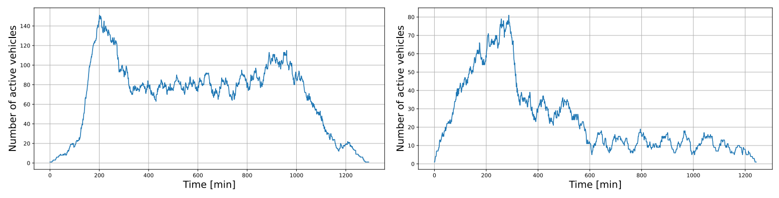
We changed some of this parameters (one at time) and we then generated 219 new diverse instances. The new instances are divided into 12 distinct class families (named sources). A brief description of the types is given in Table 2.

Table 2: The 12 types of instance sources. The third column gives the value for the existing benchmark instances.

Name	Characteristic	Standard
breakMax	No breaks between two consecutive bus legs	$[3, 35]$ min
distanceAvailability	The probability that 2 stops are connected is 0.1	0.9
distanceVariation	Add uniform $\mathcal{U}_{[1,100]}$ distance perturbation	$\mathcal{U}_{[1,1.2]}$
legRegularity	Probability of reusing the last leg is 0.1	0.9
numStations	There are 1000 bus stops	10
morningPeak	Morning peak is 5 times the regular demand	1.8
legPeriodMax	Max number of break lengths in use per tour is 5	3
shortLeg	Every leg length is in the interval $[5, 15]$ min	$[20, 60]$ min
gridSpread	Bus stops drawn using the distribution $\mathcal{N}^2(0, 1000)$	$\mathcal{N}^2(0, 50)$
legMax	The maximum leg length is 240 min	60 min
legMin	The minimum leg length is 5 min	20 min

Figure 4 shows the demand distribution of two instances. In both cases there is a significant morning peak when both employees and students need numerous buses within a brief period, followed by a decrease in activity. With the instance generator, we can create instances like in Figure 4b, where the peak in the morning is extremely high.

[4]https://cdlab-artis.dbai.tuwien.ac.at/papers/sa-bds/



(a) Real-life-like (b) Synthetic

Fig. 4: Demand Distribution of active vehicles.

3.2 Algorithm Space $\mathcal{A}$

We ran Instance Space Analysis with two algorithms from previous work.

The first algorithm is Construct, Merge, Solve and Adapt (CMSA) [7], a matheuristic algorithm.

The second algorithm is Large Neighbourhood Search (LNS) [5], where a destroy operator creates a sub-instance of the BDSP by removing all the legs of a number of drivers based on tours that are shared by these drivers. Then, LNS uses Column Generation as repair operator. Even if the solution of the sub-problem is not optimal, it is in fact close enough to the optimal with a very small optimality gap.

We set 5 min as timeout for both algorithm. We considered the average of the objective function values over 5 runs. All executions were performed on a cluster with 11 nodes using Ubuntu 22.04.2 LTS. Each node has two Intel Xeon E5-2650 v4 (max 2.20 GHz, f12 physical cores, no hyperthreading). For each run, we set a memory limit of 4.267 GB and use one thread. The implementation is in Python, executed with PyPy 7.3.11. Column Generation is implemented in Java, using OpenJDK 20, and CPLEX 22.11 for the master problem.

3.3 Feature Space $\mathcal{F}$

In order to describe the difficulty of each instance, we collect a set of features. A feature is a number that characterizes a proprieties of an instances. We collect a set of 84 features, described in Table 3.

A special feature is the number of relative relief opportunities of an instance, defined as follow.

Definition 1 (Relative Relief Opportunity (RRO)). *Let $p \in P$ be a position and $t \in \mathbb{R}$. We define a relief opportunity in p at time t (in minutes) as the proportion of bus legs that are passing through position p in the time window $[t, t + 60\,\mathrm{min})$:*

$$RRO(p,t) = \frac{|\{\ell \in L \mid startPos_\ell = p \;\wedge\; start \in [t, t+60)\}|}{|\{\ell \in L \mid startPos_\ell = p\}|}$$

Note that for each position $p \in P$, we can evaluate, e.g., $\max_t RRO(p,t)$ and then $\max_{p \in P} \max_t RRO(p,t)$ that tells us what is the maximum relief opportunity across the positions throughout the day.

Table 3: Set of 84 features used in Meta-Data. With *glorious seven* we mean the seven descriptive statistics: Max, Min, Average, Median, Std, First quartile and Third quartile.

Feature Name	Description
Size-related: Dimension of the problem (4 features)	
Number of Tours	Number of distinct bus tours of the problem
Number of Legs	Number of distinct bus legs of the problem
Number of Positions	Number of bus stops (positions) used
Number of Active vehicles	Max number of active vehicles during the day
Geometry: (1 feature)	
Average distance	Average distance between bus stops
Bus Tours: Glorious seven across all tours for each feature (35 features)	
Total Time per tour	Total span time for each tour
Number of breaks per tour	Number of breaks between consecutive legs
Number of proper breaks per tour	Number of breaks of $\geq$ 15 min for each tour
Number of legs per tour	Number of bus legs for each tour
Number of large legs per tour	Number of legs with length $\geq$ 2 h for each tour
Distributions: Glorious seven across all legs for each feature (14 features)	
Drive	Bus legs lengths
Breaks statistics	Length of breaks between consecutive legs
RRO: Max, Min, Avg, Median, Std across all positions (25 features)	
Max RRO	Max Number of RROs over the time horizon
Min RRO	Min Number of RROs over the time horizon
Mean RRO	Mean Number of RROs over the time horizon
Median RRO	Median Number of RROs over the time horizon
Std RRO	Std Number of RROs over the time horizon
Bin Packing Problem [4]**:** k is the longest leg length (5 features)	
Huge	Proportion of legs that have length $\lvert\ell\rvert > k/2$
Large	Proportion of legs with $k/3 < \lvert\ell\rvert \leq k/2$
Medium	Proportion of legs with $k/4 < \lvert\ell\rvert \leq k/3$
Small	Proportion of legs with $k/10 < \lvert\ell\rvert \leq k/4$
Tiny	Proportion of legs with $\lvert\ell\rvert \leq k/10$

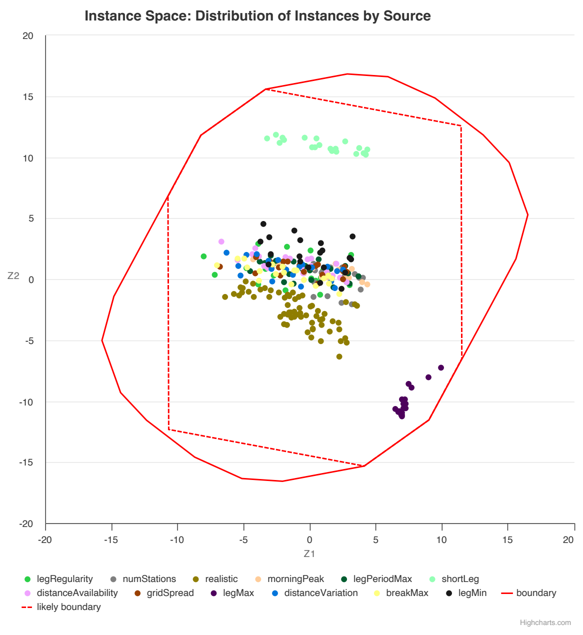


Fig. 5: Bus Driver Scheduling Problem instance space defined by Equation (2). We recognise three main clusters: legMax, shortLeg and all the rest. We also observe that the Realistic instances (in brown) are in the middle of the Instance Space.

4 Results and Evaluation

We perform the Instance Space Analysis using the Matlab toolkit MATILDA available from https://matilda.unimelb.edu.au. The settings for MATILDA are the default settings except the performance threshold set as 0.0.

Summing up, we have 284 instances from two distinct sources, 84 features described in Table 3, and 2 algorithms: CMSA [7] and LNS [5].

4.1 Instance Distribution

Figure 5 shows the distribution of the instance sources across the instance space. We notice that Realistic instances are located around the centre of the instance space, meaning that the feature values are average. Moreover, shortLeg and legMax instances appear to be close to the theoretical boundary of the Instance Space. Those are instances where the leg length has drastically changed from the value of the Realistic instances.

The red solid outer line represents the theoretical boundaries made by considering all the feasible combinations of features and their upper and lower bounds. The red dotted line instead is the likely boundary.

In total, we have 84 features. Equation (2) shows the projection matrix applied to the ten features after the preprocessing (that includes normalisation and Box-Cox transformation). We observe that four out of ten features describe the distribution of *drive*, that is, the leg length. Two features are related to the Total Time per Tour, i.e., the maximum and minimum total span of each tour, from the very beginning to the very end. The other four features represent the distribution of the Relative Relief Opportunities.

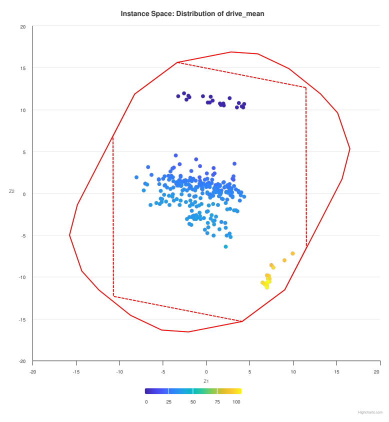
$$\begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} = \begin{bmatrix} 0.2636 & -0.7982 \\ -0.1663 & -0.6664 \\ 0.6380 & -0.5291 \\ -0.7371 & -1.4743 \\ -0.6819 & -0.5257 \\ -0.4333 & -0.8723 \\ -0.7684 & 0.3775 \\ -1.3987 & -0.2818 \\ -0.4252 & 0.4900 \\ -0.6950 & 0.0500 \end{bmatrix}^{\mathrm{T}} \cdot \begin{bmatrix} \mathit{driveMax} \\ \mathit{driveMean} \\ \mathit{driveMedian} \\ \mathit{drive3rdQuartile} \\ \mathit{maxTotalTimePerTour} \\ \mathit{minTotalTimePerTour} \\ \mathit{stdMaxRro} \\ \mathit{stdMeanRro} \\ \mathit{stdMedianRro} \\ \mathit{stdStdRro} \end{bmatrix} \qquad (2)$$

Figure 6 shows the distribution of four features. In Figure 6a we see that, as expected, the average length of bus legs is very high for legMax instances (where the leg length belongs to the interval $[20, 240]$), whereas for shortLeg instances is extremely low (here the leg length belongs to the interval $[5, 15]$). Figure 6b shows the distribution of the standard deviation (over the bus stops) of the minimum number of Relative Relief Opportunities during the day. This essentially is related to the number of possible changes/moves that we can do during the day for each bus stop. This value appears to be very low among the legMax instances, where the legs are usually large and, therefore the number of possible vehicle changes during the day is reduced. The other two images are about the total time per tour. In Figure 6c we see a clear distinction between Realistic instances and the new generated one. This is because the new generated instances have a lower maximum of length of bus tours. Figure 6d shows the Minimum total time per tour.

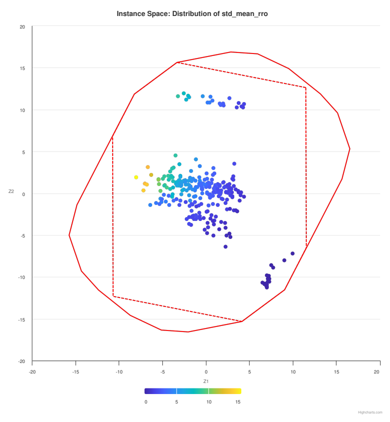
4.2 Algorithm Evaluation

Fig. 7 shows the binary performance distribution of the two algorithms. We observe that in the middle cluster, LNS performs better than CMSA. However, CMSA appears to have better results closer to the theoretical boundaries. Fig. 8 shows the prediction of the Support Vector Machine, supporting this idea.

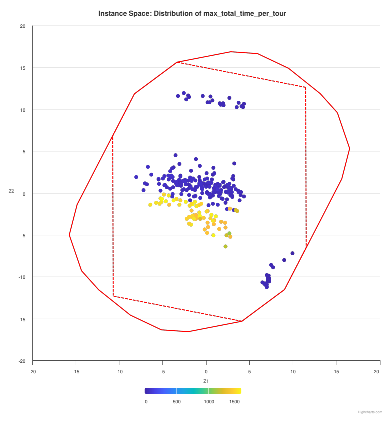
We believe that the "structure" of the bus tour impacts the performance of the two algorithms. In particular, LNS removes all the bus legs with some selected bus tours. In contrast, CMSA randomly generates a number of greedy solutions at every iteration and, therefore, does not directly exploit the bus tours. LNS seems to perform better than CMSA for most of the instances (including Realistic), but not for all. We observe that CMSA gets better solutions for legMax and shortLeg instances. These instances have very short tours. Thus, LNS does not benefit much from removing all the legs associated with the same tour. Hence, CMSA (which does not explicitly depend on the structure of bus tours) provides good results with no significant difference from the others.



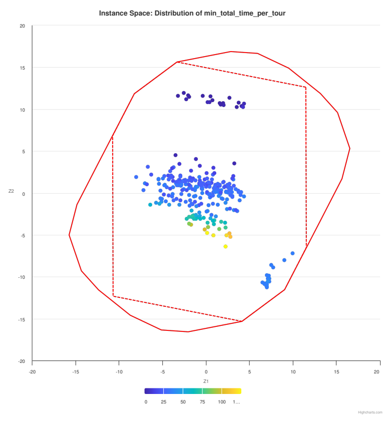
(a) Average leg length



(b) stdMeanRRO



(c) Max Total Time x Tour



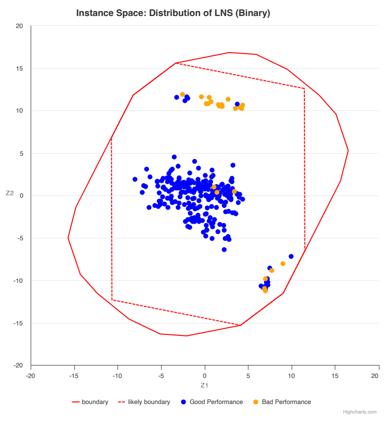
(d) Min Total Time x tour

Fig. 6: Distribution of features. The colors represent the feature values. Axes as defined by the equation (2).

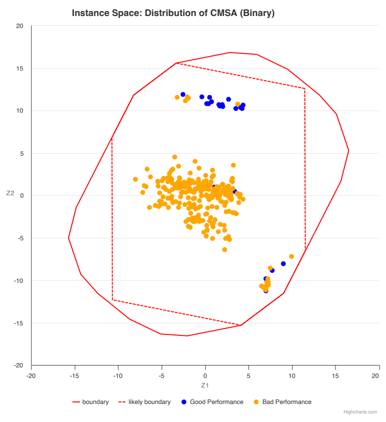
4.3 Filling the Gaps

Thanks to Instance Space Analysis, we observe that there is still a considerably extended region between the four clusters in the instance space. This reveals opportunities to generate new instances in order to fill this gap. A first possible way to do that is by changing or adding parameters of the instance generator and trying to explore the instance space.

A more elaborated one is to fix a target (e.g., a portion of the Instance Space to fill up) and generate instances through a Genetic Algorithm that evolves new instances in



(a) LNS



(b) CMSA

Fig. 7: Binary performance distribution. We see that CMSA performs better in some of the new generated instances close to the boundary.

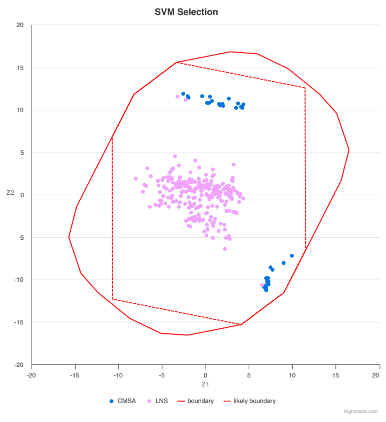


Fig. 8: Algorithm selection using SVM

the desirable region, as done by Smith-Miles [9]. However, this procedure is problem-dependent and requires more investigation.

5 Conclusions and Future Investigations

In this paper, we have applied Instance Space Analysis to the Bus Driver Scheduling Problem for the first time. We evaluated the performance of two metaheuristic techniques for the BDSP, providing insights into the strength of LNS and the boundaries of its good performance. We greatly increased the capabilities of the instance generator and extended the previous set of instances with new, diverse ones. We defined and evaluated a novel set of features, seeing which features help the most to explain algorithm performance.

In the future, we want to fill the instance space by creating more instances that are even more diverse than the ones present now. At first, we will consider other public transportation systems, possibly located in different countries. Then, we will create instances with new combinations of parameters like long leg lengths and short bus tour lengths. Ideally, we want to use a Genetic Algorithm to automatically evolve the instances to fill up certain regions in the Instance Space. The goal is to perform automatic algorithm selection and outline the region of the instance space where one algorithm performs better than another. Thanks to this problem's structure, this will also be helpful for related problems such as vehicle routing.

Furthermore, we will test other solution methods using other quality metrics, such as the GAP from the best-known solution or the area under the curve of the trajectory of solutions found during the search.

Acknowledgements. 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. This work is also partially funded by the Doctoral Program Vienna Graduate School on Computational Optimization, Austrian Science Foundation (FWF), under grant No.: W1260-N35 (https://vgsco.univie.ac.at/).

References

1. Kletzander, L., Mazzoli, T.M., Musliu, N.: Metaheuristic algorithms for the bus driver scheduling problem with complex break constraints. In: Proceedings of the Genetic and Evolutionary Computation Conference. p. 232–240. GECCO '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3512290.3528876
2. Kletzander, L., Musliu, N.: Solving large real-life bus driver scheduling problems with complex break constraints. Proceedings of the International Conference on Automated Planning and Scheduling **30**(1), 421–429 (Jun 2020), https://ojs.aaai.org/index.php/ICAPS/article/view/6688
3. Kletzander, L., Musliu, N., Van Hentenryck, P.: Branch and price for bus driver scheduling with complex break constraints. Proceedings of the AAAI Conference on Artificial Intelligence **35**(13), 11853–11861 (May 2021), https://ojs.aaai.org/index.php/AAAI/article/view/17408
4. Liu, K., Smith-Miles, K., Costa, A.: Using Instance Space Analysis to Study the Bin Packing Problem. Ph.D. thesis, PhD thesis (2020)

5. Mazzoli, T.M., Kletzander, L., Hentenryck, P.V., Musliu, N.: Investigating large neighbourhood search for bus driver scheduling. In: 34th International Conference on Automated Planning and Scheduling (2024), https://openreview.net/forum?id=d4TzG4ivNu
6. Rice, J.R.: The algorithm selection problem. In: Rubinoff, M., Yovits, M.C. (eds.) Advances in Computers, Advances in Computers, vol. 15, pp. 65–118. Elsevier (1976). https://doi.org/https://doi.org/10.1016/S0065-2458(08)60520-3
7. Rosati, R.M., Kletzander, L., Blum, C., Musliu, N., Schaerf, A.: Construct, merge, solve and adapt applied to a bus driver scheduling problem with complex break constraints. In: AIxIA 2022 – Advances in Artificial Intelligence, pp. 254–267. Springer International Publishing (2023). https://doi.org/10.1007/978-3-031-27181-6_18
8. Smith-Miles, K., Baatar, D., Wreford, B., Lewis, R.: Towards objective measures of algorithm performance across instance space. Computers& Operations Research **45**, 12–24 (2014). https://doi.org/https://doi.org/10.1016/j.cor.2013.11.015, https://www.sciencedirect.com/science/article/pii/S0305054813003389
9. Smith-Miles, K., Bowly, S.: Generating new test instances by evolving in instance space. Computers & Operations Research **63**, 102–113 (2015)
10. Smith-Miles, K., Muñoz, M.A.: Instance space analysis for algorithm&nbsp;testing: Methodology and software tools. ACM Comput. Surv. **55**(12) (mar 2023). https://doi.org/10.1145/3572895
11. Wren, A.: Scheduling vehicles and their drivers-forty years' experience. Tech. rep., University of Leed (2004)

Solving the Employee Task Distribution Problem with Multiple Objectives

Matthias Horn[1], Marie-Louise Lackner[1], Christoph Mrkvicka[2], Nysret Musliu[1], Jakob Preininger[2], and Felix Winter[1]

[1] Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling, DBAI, TU Wien, Favoritenstraße 9, 1040 Vienna, Austria
{matthias.horn,marie-louise.lackner,nysret.musliu,felix.winter}
@tuwien.ac.at
[2] MCP GmbH, Canovagasse 7, 1010 Vienna, Austria
{christoph.mrkvicka,jakob.preininger}@mcp-alfa.com

Abstract. Assigning employees to operations effectively is a frequent task across diverse industry areas. The challenge of this industry application is to provide a solution method that is flexible enough to be easily adjusted to a specific use case's constraints and optimization objectives. In this paper, we introduce and formally define the Employee Task Distribution Problem (ETDP), describe the highly configurable objective function and propose a solver-independent model that is solved using different constraint programming and mixed integer programming solvers. Furthermore, we prove that the ETDP is NP-hard. We evaluate the performance of our approach using a large benchmark set based on real-life instances and a range of exact state-of-the-art solvers. The best methods can find optimal solutions for nearly all benchmark instances within a realistic time bound for practical usage.

Keywords: Employee Task Distribution Problem, Assignment Problem, Constraint Programming, Integer Programming

1 Introduction

Across diverse industry areas, employees' time capacities must be effectively distributed among operations or tasks on a daily basis. Such an assignment can concern a single or several shifts and must respect various constraints such as capacity limits and employee qualifications. Moreover, a good assignment will make use of the available resources as effectively as possible. What it means to use resources effectively depends on the specific use case; a trade-off between a selection of the following goals needs to be found: maximize the overall output, prioritize bottleneck operations, preferably choose fixed employees instead of temporary or leased staff, maximize the quality of the output, or schedule employees to work on as few different operations as possible within one shift. In addition, when planning several shifts simultaneously, it can be desirable to give employees continuity in their tasks, i.e., to assign employees to the same operations in subsequent shifts.

In this paper, we introduce and define this novel real-life planning problem; we refer to it as the *Employee Task Distribution Problem* (ETDP). This problem originates from collaborations with several industry partners. The challenge of solving this practical problem is twofold: First, the solution method needs to be highly configurable to handle different practical use cases (with varying constraints and objectives). Second, high-quality solutions need to be found quickly to allow for a responsive user experience in industrial settings (the assignments need to be updated often and time efficiently).

The main contributions of this paper are as follows. We propose a solver-independent mathematical model with flexible constraints and an objective function that can handle priorities and relative weights of various objective components. This model is implemented using the high-level modeling language MiniZinc [22]. We evaluate our solution approach in a series of experiments conducted using state-of-the-art *Mixed Integer Programming* (MIP) and *Constraint Programming* (CP) solvers. Our benchmark set consists of 216 instances based on real-life scenarios and is publicly available online [15]. The most successful solvers can find optimal solutions within less than a minute. In addition, we provide computational hardness results for the ETDP, showing that several variants of the ETDP are $\mathcal{NP}$-hard. Thereby, we identify which constraints and components of the objective functions cause computational challenges. These theoretical results are also reflected in our experimental evaluation.

The remainder of the paper is organized as follows: We formally introduce the ETDP and define the objective components in Section 2. Our solver-independent mathematical model is presented in Section 3. We analyze the computational complexity of ETDP in Section 4 and present our experimental evaluation and results in Section 5.

1.1 Related Work

At its core, the ETDP can be seen as a transportation problem for which the employees correspond to suppliers, the operations to customers, and the amount shipped from some supplier to a customer corresponds to the amount of time units the corresponding employee is assigned to the corresponding operation. For a transportation problem, every supplier can ship to any of the customers at a given shipping cost per unit. The goal is to decide on the amounts shipped between every supplier-customer pair to minimize the total cost of meeting customer demands. In the ETDP context, these shipping costs can be interpreted as costs related to employee qualifications, employee types, and operations types. The transportation problem is one of the oldest problems in operations research [20]. See [25,8] for more details on transportation problems. It can be solved efficiently in polynomial time by formulating it as a linear program or minimum-cost flow problem with linear cost function (see the early works [17,14,6,9] and e.g. [26,3,16] for more recent publications). However, due to special constraints and objectives, the ETDP is more complex than the classical transportation problem and these polynomial-time algorithms are not applicable.

The Fixed-Charge Transportation Problem (FCTP) [13,18], is a generalization of the transportation problem for which fixed costs are incurred whenever a route between a supplier and a customer is opened. These fixed costs are added to the unit costs incurred per unit shipped from the given supplier to the given customer. The objective of the FCTP is to decide which routes are opened and which amounts are shipped on these routes to minimize the total cost. This objective is related to the objective of the ETDP to schedule employees to work on as few different operations as possible within one shift. See Theorem 1 in Section 4 for details on the relation of the ETDP to the FCTP.

The ETDP is also related to the assignment problem [24], for which tasks must be assigned to agents to minimize overall costs. In contrast to the ETDP, a one-to-one assignment needs to be found for the assignment problem. Also, the generalized assignment problem (GAP) [4,23] is not general enough to capture the nature of the ETDP. Here, multiple tasks may be assigned to one agent, but tasks may not be split among multiple agents.

Another loosely related problem is the resource-constrained project scheduling problem (RCPSP), which considers scheduling subject to resource and preceding constraints like minimum and maximum time lags. The problem is widely studied in the literature. For an overview see [11,12]. However, the RCPSP focuses much more on the scheduling aspect, whereas our problem abstracts away timing information of operations and employees and focuses more on the assignment part.

2 The Employee Task Distribution Problem

An instance of the *Employee Task Distribution Problem* (ETDP) consists of a set of employees, a set of operations, a set of time buckets, and a set of qualifications. The goal is to assign a number of time units to every bucket-employee-operation triple so that all constraints are respected and the objective function is minimized. Assigning 0 time units to a triple is always admissible.

An overview of all the instance parameters of an ETDP instance is given in Table 1. Let $\mathcal{B} = \{1, 2, \ldots, u\}$ be the set of time buckets. These time buckets are used to model subsequent non-overlapping blocks of time within which the assignments are made. Such time buckets can be interpreted as shifts, work days, calendar weeks, etc. The set of employees $\mathcal{E} = \{1, 2, \ldots, m\}$ is specified by the employees' supplied time capacities $\mathrm{sc}(b, e) \in \mathbb{N} \cup \{0\}\ \forall b \in \mathcal{B}, e \in \mathcal{E}$ and the employees' priority levels $\mathrm{ep}(b, e) \in \mathbb{N}^+$. The set of operations is denoted by $\mathcal{O} = \{1, 2, \ldots, n\}$. For every operation $o \in \mathcal{O}$ and time bucket $b \in \mathcal{B}$, we are given the demanded time capacity $\mathrm{dc}(b, o) \in \mathbb{N} \cup \{0\}$, the priority level $\mathrm{op}(b, o) \in \mathbb{N}^+$, the minimum (positive) assigned capacity $\mathrm{mc}(b, o) \in \mathbb{N} \cup \{0\}$ and $\mathrm{mp}(b, o) \in \mathbb{N} \cup \{0\}$, the maximum number of operations any employee assigned to o can be assigned to in total within the same time bucket. The qualification matrix $Q = (Q(o, e))_{o \in \mathcal{O}, e \in \mathcal{E}}$ with $Q(o, e) \in \mathbb{N} \cup \{0\}$ captures which employees are qualified for which operations. An entry $Q(o, e) = 0$ means that employee e cannot be assigned to operation o and the value of an entry $Q(o, e) > 0$ indicates the level of qualification. Different levels of qualification can be used to model differences in the quality of the output resulting from an assignment: If e_1 and e_2 are to employees that are qualified for an operation $o \in \mathcal{O}$, $Q(o, e_1) > Q(o, e_2) > 0$ means that assigning e_1 to this operation leads to an output of superior quality than assigning e_2 to this operation.

For every bucket-employee-operation triple (b, o, e) with $b \in \mathcal{B}$, $o \in \mathcal{O}$ and $e \in \mathcal{E}$, a non-negative value $A(b, o, e) \in \mathbb{N} \cup \{0\}$ needs to be assigned. This value $A(b, o, e)$ corresponds to the number of time units employee e works on operation o within time bucket e. The constraints that need to be satisfied by the assignments $A(b, o, e)$ for all

Employees $e \in \mathcal{E}$	$\mathrm{sc}(b,e)$	supplied capacity	Constraint (3)
	$\mathrm{ep}(b,e)$	employee priority	Objective (1b)
Operations $o \in \mathcal{O}$	$\mathrm{dc}(b,o)$	demanded capacity	Constraint (3)
	$\mathrm{op}(b,o)$	operation priority	Objective (1c)
	$\mathrm{mc}(b,o)$	min. assigned capacity	Constraint (5)
	$\mathrm{mp}(b,o)$	max. parallel operations	Constraint (6)
Qualifications	$Q(o,e)$	qualification level	Constraint (2)
			Objective (1d)

Table 1: Overview of the instance parameters of the ETDP for a time bucket $b \in \mathcal{B}$

$b \in \mathcal{B}$, $o \in \mathcal{O}$ and $e \in \mathcal{E}$ as well as the objectives that should be minimized are described in what follows.

2.1 Constraints

A feasible set of assignments $A(b,o,e) \in \mathbb{N} \cup \{0\}$ for all time buckets b, operations o and employees e needs to respect the following constraints. Setting $A(b,o,e) = 0$ is always admissible; a positive assignment $A(b,o,e) > 0$ may however only be made if $Q(o,e) > 0$, i.e., employee e is qualified for operation o. For employees and operations, the supplied resp. demanded capacity levels may not be exceeded for each time bucket. That is, the sum of all assignments to a given operation o (resp. employee e) within a time bucket b may not exceed the demanded capacity $\mathrm{dc}(b,o)$ (resp. supplied capacity $\mathrm{sc}(b,e)$). Moreover, for operations with minimum assigned capacity $\mathrm{mc}(b,o) > 0$, the sum of all assignments must be equal to 0 or at least equal to $\mathrm{mc}(b,o)$. The maximum-parallel-operations constraint enforces that any employee e assigned to an operation o at time bucket b with $\mathrm{mp}(b,o) > 0$ may not be assigned to more than $\mathrm{mp}(b,o)$ many operations in total. Both the minimum-assigned-capacity and the maximum-parallel-operations constraints alone cause the ETDP to be $\mathcal{NP}$-hard. For a formal statement and proof of this result, see Section 4.

2.2 Objectives

A multitude of different practical use cases give rise to different optimization objectives for the ETDP. These use cases can occur independently of each other within a specific application. Often, a trade-off between several, potentially conflicting, objectives needs to be found in practice. Finding such a trade-off among is the topic of multiobjective optimization and many different methods have been suggested to approach this goal [7,19]. For our specific application, we wanted an approach that allows high flexibility and allows the users to select a combination of objectives, each with a certain priority and weight. Priorities can be used to set the order of lexicographic optimization and weights can additionally be used to balance objectives with the same priority level.
Basic objective: Maximize the sum of assignments The basic objective of the ETDP is to maximize the overall satisfied capacity demands over all time buckets. Equivalently,

the objective is to minimize the amount of unsatisfied capacity demands, i.e., the cost is defined as the difference between the theoretical maximum assignment level and the sum of all assignments.

Employee prioritization If capacity supplies exceed capacity demands, i.e., not all capacity supplies are fully used, the distribution of used capacity per employee should be done so that permanent staff are lexicographically more important than others, e.g. leased or temporary staff. With the usage of $\mathrm{ep}(b, e) \in \mathbb{N}$, many levels of employee prioritization are possible.

Operation prioritization If capacity demands cannot be met fully, the distribution of assigned capacity per operation should be such that bottleneck operations are lexicographically more important than others. Levels of operation prioritization are specified with $\mathrm{op}(b, o) \in \mathbb{N}$ for $o \in O$.

Maximize qualification score The quality of the output might depend on the qualification level of employees assigned to operations. In this case, the goal is to maximize the weighted sum of assignments, where the qualification levels give the weights. Note that this objective is equivalent to the basic objective if the qualification matrix Q is a binary matrix. Qualification levels can also be used to model employees' preferences for certain operations. In this case, this objective corresponds to maximizing employees' satisfaction with the assignments.

Time bucket change objective If multiple time buckets are considered, another criterion for the output's quality is the degree of continuity in the tasks assigned to employees. More precisely, if employee e is assigned to operation o in time bucket b, i.e. $A(b, o, e) > 0$, it can be beneficial to assign this employee to the same operation in the consecutive time bucket $b+1$, i.e. to set $A(b+1, o, e) > 0$ as well. Maximizing the overall continuity across all employees is achieved by minimizing the number of times this is not the case for every employee, i.e. minimizing the number of operations and time buckets for which $A(b, o, e) > 0$ but $A(b + 1, o, e) = 0$.

Concentrated assignments = minimize assignment count objective In practice, solutions in which the assignments are concentrated are often more favorable than those with assignments spread across the operations and employees. It is better for the employees and the outcome of their work if they are assigned to fewer operations within the same shift. For instance, a solution in which $A(b_1, o_1, e_1) = 2$ and $A(b_1, o_2, e_2) = 2$ is often clearly better than a solution with $A(b_1, o_1, e_1) = A(b_1, o_1, e_2) = A(b_1, o_2, e_1) = A(b_1, o_2, e_2) = 1$ even if all other objective values are equal.

The sole goal of minimizing the number of assignments would always lead to a null assignment. Therefore, it only makes sense in combination with one of the other objectives, e.g. minimizing the number of assignments while at the same time maximizing the sum of assignments. While all other objectives are linear in the assignment values $A(b, o, e)$, the "minimize-assignment-count" and the "time-bucket-change"-objective are not. Using the "minimize-assignment-count" objective combined with any of the other ones causes the ETDP to be $\mathcal{NP}$-hard, as we show in Section 4.

See the mathematical model introduced in the following section for a formal definition of these objective functions and an explanation of how they are combined into a single function using weights. Note that we formulate all objective functions as cost functions, i.e., we formulate the ETDP as a minimization problem.

3 Mathematical Model

We use decision variables

$$A(b,o,e) \in \mathbb{N} \cup \{0\} \quad \forall b \in \mathcal{B} \quad \forall o \in \mathcal{O} \quad \forall e \in \mathcal{E}$$

to indicate the amount of capacity assigned to a bucket, operation, and employee triple. In the following, we have underlined all constants involved. For their calculation, we refer the reader to Section 3.3 and the technical appendix[15].

$$\text{Min.} \quad obj = \underline{\lambda_a} \cdot a + \underline{\lambda_e} \cdot e + \underline{\lambda_o} \cdot o + \underline{\lambda_q} \cdot q + \underline{\lambda_r} \cdot r + \underline{\lambda_c} \cdot c, \text{ where} \tag{1}$$

$$a = \underline{max_a} - \sum_{b\in\mathcal{B}, o\in\mathcal{O}, e\in\mathcal{E}} A(b,o,e) \tag{1a}$$

$$e = \underline{max_e} - \sum_{\substack{p\in\{1,\dots,p_E\},\\ b\in\mathcal{B}}} \underline{\text{bigM}^{\text{e}}(b,p)} \left(\sum_{\substack{e\in\mathcal{E}:\, \text{ep}(b,e)=p\\ o\in\mathcal{O}}} A(b,o,e) \right) \tag{1b}$$

$$o = \underline{max_o} - \sum_{\substack{p\in\{1,\dots,p_O\},\\ b\in\mathcal{B}}} \underline{\text{bigM}^{\text{o}}(b,p)} \left(\sum_{\substack{o\in\mathcal{O}:\, \text{op}(b,o)=p\\ e\in\mathcal{E}}} A(b,o,e) \right) \tag{1c}$$

$$q = \underline{max_q} - \sum_{b\in\mathcal{B}, o\in\mathcal{O}, e\in\mathcal{E}} (A(b,o,e) \cdot Q(o,e)) \tag{1d}$$

$$r = \sum_{\substack{b\in\{1,\dots,u-1\},\\ o\in\mathcal{O},\\ e\in\mathcal{E}}} |\mathbf{1}_{>0}(A(b,o,e)) - \mathbf{1}_{>0}(A(b+1,o,e))| \tag{1e}$$

$$c = |\{b \in \mathcal{B}, o \in \mathcal{O}, e \in \mathcal{E} : A(b,o,e) > 0\}| \tag{1f}$$

$$\text{s.t. } A(b,o,e) \le Q(o,e) \cdot \underline{cap} \qquad \forall b \in \mathcal{B}, \forall o \in \mathcal{O}, \forall e \in \mathcal{E} \tag{2}$$

$$\sum_{o\in\mathcal{O}} A(b,o,e) \le \text{sc}(b,e) \qquad \forall b \in \mathcal{B}, \forall e \in \mathcal{E} \tag{3}$$

$$\sum_{e\in\mathcal{E}} A(b,o,e) \le \text{dc}(b,o) \qquad \forall b \in \mathcal{B}, \forall o \in \mathcal{O} \tag{4}$$

$$\sum_{e\in\mathcal{E}} A(b,o,e) = 0 \vee \sum_{e\in\mathcal{E}} A(b,o,e) \ge \text{mc}(b,o) \qquad \forall b \in \mathcal{B}, \forall o \in \mathcal{O} \tag{5}$$

$\forall b \in \mathcal{B}, \forall o \in \mathcal{O}$ with $\text{mp}(b,o) > 0$,

$\forall e \in \mathcal{E}$ with $A(b,o,e) > 0$:

$$|\{i \in \mathcal{O} : A(b,i,e) > 0\}| \le \text{mp}(b,o) \tag{6}$$

3.1 Objective function

The objective function in equation (1) consists of six components combined in a weighted sum. These weights $\underline{\lambda_i}$ are calculated on the basis of the priorities and weights of every objective component and are calculated as "big M" values. The use of large constants that are referred to as "big Ms" is very common when solving multi-objective optimization problems, particularly in practical applications. Note that all solutions that are optimal with respect to a weighted sum of the objective components are Pareto-optimal [5] with respect to the three objective components. All objective components are modeled as cost functions, i.e., we formulate the ETDP as a minimization problem. The objective component a corresponds to the basic objective, e to employee prioritization, o to operation prioritization, q to maximizing the qualification score, r the time bucket change objective with indicator function $\mathbf{1}_{>0}(x)$, which is one if $x > 0$ and zero otherwise, and c is the number of non-zero assignments. Note that all objective functions are linear in the decision variables except for r and c. The constants $\underline{max_a}$, $\underline{max_e}$, $\underline{max_o}$ and $\overline{max_q}$ are chosen in such a way for every instance that a solution with cost value equal to 0 corresponds to a theoretical optimum. We refer the reader to the online appendix for detailed calculations [15].

3.2 Constraints

The constraint in equation (2) ensures that an assignment of employee e to operation o at time bucket b, i.e. $A(b, o, e) > 0$, is only possible if employee e is qualified for operation o, i.e. $Q(o, e) > 0$. The constant $\underline{cap}$ is the minimum of the overall maximum supplied capacity and the overall maximum demanded capacity. Equations (3) and (4) encode the capacity constraints of employees and operations. Equation (5) encodes the minimum-assigned-capacity constraint: The sum of all assignments to operation o needs to be equal to 0 or greater than or equal to $\text{mc}(b, o)$ for bucket b. This constraint is thus always fulfilled for operations with $\text{mc}(b, o) = 0$. Finally, the maximum-parallel-operations constraint is modeled by Equation (6) and has to be fulfilled for all operations $o \in \mathcal{O}$ and buckets $b \in \mathcal{B}$ with $\text{mp}(b, o) > 0$: Every employee e assigned to such an operation can in total be assigned to no more than $\text{mp}(b, o)$ many different operations.

3.3 Weights of the objective components

The weights of objective components $\underline{\lambda_a}$, $\underline{\lambda_e}$, $\underline{\lambda_o}$, $\underline{\lambda_q}$, $\underline{\lambda_r}$ and $\underline{\lambda_c}$ are set in order to respect the priorities and weights of objective components in the aggregated objective function obj as defined in equation (1).

For this purpose, we introduce the following notation. The objectives a, e, o, q, r and c are first grouped into objectives with same priority level p. Note that we assume that the objective priorities take all values between 1 and the maximum objective priority k. Within a priority level p, the aggregated objective function obj_p is calculated as a weighted sum of the normalized objective components $obj_{p,1}, \ldots, obj_{p,k_p}$ using the respective weights $w_{p,1}, \ldots, w_{p,k_p}$, where k_p is the number of objective functions with priority level p. The overall aggregated objective function obj is then created as a linear combination of the functions $obj_1, \ldots, obj_k$ where the coefficients in this sum

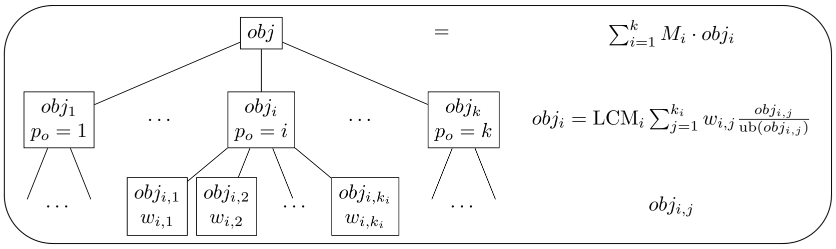


Fig. 1: Aggregated objective function with weights and priorities

are chosen as big M constants such that a lexicographic optimization is achieved (obj_1 is lexicographically more important than obj_2, aso.). For a visualization of the structure of the aggregated objective function, see Figure 1.

The objective function obj is defined as follows:

$$obj = \sum_{i=1}^{k} M_i \cdot obj_i \qquad \text{with}$$

$$M_i = \prod_{j=i+1}^{k} (1 + \mathrm{ub}(obj_i))$$

$$= \prod_{j=i+1}^{k} \left(1 + \mathrm{LCM}_i \cdot \sum_{j=1}^{k_i} w_{i,j}\right)$$

$$obj_i = \mathrm{LCM}_i \sum_{j=1}^{k_i} w_{i,j} \frac{obj_{i,j}}{\mathrm{ub}(obj_{i,j})} \leq \mathrm{LCM}_i \cdot \sum_{j=1}^{k_i} w_{i,j}$$

$$\mathrm{LCM}_i = \frac{\mathrm{LCM}\left(\mathrm{ub}(obj_{i,j}) : j \in [1, k_i]\right)}{\mathrm{GCD}\left(w_{i,j} : j \in [1, k_i]\right)}$$

The *least common multiple* LCM and the *greatest common divisor* GCD are used to ensure that the objective function is an integer to make it applicable for constraint programming solvers. The functions $\mathrm{ub}(obj_{i,j})$ are upper bounds on the respective objectives $obj_{i,j}$ which can be one of the cost functions a, e, o, q, r, and c. Their values are given by the constants defined in the previous section:

$$\mathrm{ub}(a) = \underline{max_a} \text{ as in equation (1a)}$$
$$\mathrm{ub}(e) = \underline{max_e} \text{ as in equation (1b)}$$
$$\mathrm{ub}(o) = \underline{max_o} \text{ as in equation (1c)}$$
$$\mathrm{ub}(q) = \underline{max_q} \text{ as in equation (1d)}$$
$$\mathrm{ub}(r) = |\{o \in \mathcal{O}, e \in \mathcal{E} : Q(o,e) > 0\}| \cdot (u-1).$$
$$\mathrm{ub}(c) = |\{o \in \mathcal{O}, e \in \mathcal{E} : Q(o,e) > 0\}|.$$

We can also write the objective function as follows:

$$obj = \sum_{f\in\mathcal{F}} \underline{\lambda_f} \cdot f = \sum_{f\in\mathcal{F}} \left(M_{p_f} \cdot \frac{\mathrm{LCM}_{p_f}}{\mathrm{ub}(f)} \cdot w_f \cdot f \right),$$

where p_f is the priority and w_f is the weight of objective function (name) f from the set of objective functions $\mathcal{F} = \{a, e, o, q, r, c\}$. The weights $\underline{\lambda_f}$ for $f \in \mathcal{F}$ are the weights used in the mathematical model.

4 Complexity Results

The Employee Task Distribution Problem with the basic objective (1a), the employee prioritization (1b), operation prioritization (1c) or qualification score objective (1d) but without max-parallel-operations-constraints (6) and without min-assigned-capacity-constraints (5) can be formulated as a classical transportation problem [25,8]. These special cases of the ETDP can thus also be solved in polynomial time.

However, very simplified versions of the ETDP involving the other objectives and the specialized max-parallel-operations- and min-assigned-capacity-constraints are $\mathcal{NP}$-hard. In the following, we formally state these $\mathcal{NP}$-hardness results for the ETDP. Note that some proofs can be found in the technical appendix, which is available online [15].

Theorem 1. *The Employee Assignment optimization problem without max-parallel-operations- and min-assigned-capacity-constraints and with the objective of minimizing the number of assignments as defined in equation* (1f) *and the quality score objective (equation* (1d)*) or sum of assignments (equation* (1a)*) is strongly $\mathcal{NP}$-hard.*

Proof. The ETDP without max-parallel-operations- and min-assigned-capacity-constraints and with the objective of minimizing the number of assignments and the quality score objective (or sum of assignments) is equivalent to a special case of the *Fixed-Charge Transportation Problem* (FCTP). The FCTP is a generalization of the transportation problem and was introduced by Hirsch and Dantzig [13]. The following description of the FCTP is given by Kowalski [18]:

The fixed-charge transportation problem consists of m suppliers and n customers. Each of the m suppliers can ship to any of the n customers at a shipping cost per unit $c_{i,j}$ (unit cost for shipping from supplier i to customer j), plus a fixed cost $f_{i,j}$, assumed for opening this route. The objective is then to determine which routes are to be opened and the size of the shipment on those routes so that the total cost of meeting demand, given the supply constraints, is minimized.

The FCTP can be formalized as follows, where the decision variables $x_{i,j}$ model the amount shipped from supplier i to customer j and the $\{0, 1\}$-variables $y_{i,j}$ model whether a positive amount is shipped from supplier i to customer j or not:

$$\text{minimize } Z = \sum_{i=1}^{m} \sum_{j=1}^{n} (c_{i,j} x_{i,j} + f_{i,j} y_{i,j})$$

$$\text{subject to } \sum_{j=1}^{n} x_{i,j} = a_i$$
$$\sum_{i=1}^{m} x_{i,j} = b_j$$
$$x_{i,j} \geq 0 \quad \forall i, j$$
$$\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_i$$
$$y_{i,j} = 0 \text{ if } x_{i,j} = 0$$
$$y_{i,j} = 1 \text{ if } x_{i,j} > 0$$

Note that setting $f_{i,j}$ to some fixed weight corresponding to the weight w_c of the objective "minimizing the number of assignments" allows us to model the ETDP (without max-parallel-operations and min-assigned-capacity constraints) as FCTP. The costs $c_{i,j}$ need to be set accordingly to model the qualification levels, setting them to large "big M" values in case of absent qualifications. Moreover, dummy operations or tasks must be introduced if the sum of demanded capacities is not equal to the sum of supplied capacities.

Angulo et al. [1] showed that the Fixed Charge Transportation Problem is strongly $\mathcal{NP}$-hard by a reduction from the strongly $\mathcal{NP}$-complete 3-Partition problem. The $\mathcal{NP}$-hardness of the problem holds even if the fixed costs $f_{i,j}$ are constant across all suppliers i and customers j. This shows the $\mathcal{NP}$-hardness of this version of the ETDP.

Theorem 2. *The Employee Assignment optimization problem with min-assigned-capacity-constraint and the sole objective of maximizing the sum of assignments is strongly $\mathcal{NP}$-hard.*

Proof. We prove the $\mathcal{NP}$-hardness of the ETDP with minimum-assigned-capacity-per-operation-Constraint by providing a polynomial time reduction from the $\mathcal{NP}$-hard 3-dimensional Matching problem [10]. See the technical appendix [15] for the complete proof.

Theorem 3. *The Employee Assignment optimization problem with maximum-parallel-operations-constraint and the sole objective of maximizing the sum of assignments is strongly $\mathcal{NP}$-hard.*

Proof. We prove the $\mathcal{NP}$-hardness of the ETDP with maximum-parallel-operations-constraint by providing a polynomial time reduction from the strongly $\mathcal{NP}$-complete 3-Partition problem [10]. See the technical appendix [15] for the complete proof.

Note however that if the max-parallel-operations-constraint is set to $\text{mp}(o) = 1$ for all operations $o \in \mathcal{O}$, the ETDP corresponds to a Single-Source Transportation Problem [21], which is equivalent to a Generalized Assignment Problem [4,23] and is solvable in polynomial time.

5 Experimental Evaluation

We implemented the mathematical model presented earlier using the high-level constraint modeling language MiniZinc [22]. In order to evaluate the performance of our model, we conducted experiments with the following state-of-the-art MIP and CP solvers: Gurobi, CPLEX, Cbc, Chuffed and OR-Tools. For the MIP solvers, MiniZinc automatically converts the given constraint model into an MIP model [2].

The benchmark set used for these experiments consists of a total of 216 instances and is publicly available online [15]. It is based on twelve relatively small real-life instances that were provided to us by our industrial partner. These instances have between 9 and 23 employees and between 5 and 16 operations; at most 448 capacity units need to be distributed among the employees and operations. For every one of these instances, six different settings for the objective function were evaluated in order to model different practical use cases:

- UCOpPrio: the sole objective is operation prioritization
- UCEmpPrio: the sole objective is employee prioritization
- UCAss: the objective is to maximize the sum of assignments with first priority and to minimize the assignment count with second priority
- UCQuali: the sole objective is to maximize the qualification score
- UC3Buckets: three time buckets are considered, a combined objective with the first priority being the maximization of the qualification score and the second priority being the minimization of changes between time buckets with weight 1 and the minimization of the assignment count with weight 2
- UC4Buckets: four time buckets are considered with the same combined objective as for the UC3Buckets use case

In order to test the scalability of our model, we created larger instances from real-life instances by copying the employees and operations and randomly perturbing their properties such as capacities and constraints. Qualifications were also perturbed. This resulted in a set of 72 medium-sized instances for which the number of employees and operations from the real-life instances was doubled and a set of 72 large instances for which they were multiplied by five. The large instances thus have up to 115 employees and up to 80 operations.

In practice, high-quality solutions to the ETDP need to be found very quickly. Indeed, even though the employee-operation-assignments are planned in advance over a large time horizon of several weeks or months, the actual capacities of employees and operations are subject to short-time changes due to sickness, material shortage, order

	feasible solutions found (in %)				opt. solved	proven opt.	opt. gap (in %)	
solver	total	small	medium	large	(in %)	(in %)	average	std
Gurobi	**100.0**	**100.0**	**100.0**	**100.0**	**84.7**	**84.7**	**2.1**	**12.0**
CPLEX	**100.0**	**100.0**	**100.0**	**100.0**	83.3	79.6	2.8	14.2
Cbc	97.2	**100.0**	**100.0**	91.7	65.7	64.4	11.1	29.2
Chuffed	86.6	98.6	88.9	72.2	10.6	6.9	68.5	39.2
OR-Tools	**100.0**	**100.0**	**100.0**	**100.0**	3.2	1.9	79.2	31.2

Table 2: Overview of the experimental results achieved with a time limit of 60 seconds.

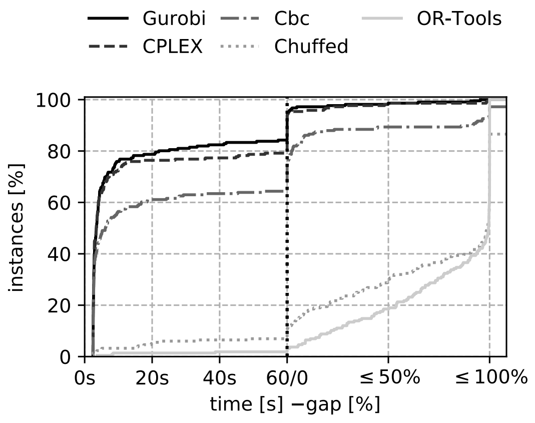


Fig. 2: Performance plot over all 216 instances, comparing solvers Gurobi, CPLEX, Cbc, Chuffed, and OR-Tools.

cancellations etc. Therefore, (updated) instances of the ETDP need to be solved often and quickly. We thus conducted experiments with a runtime limit of 60 seconds, which is a realistic time-bound for practical purposes. We also experimented with a 3-minute time limit but do not report these results here in detail as these only revealed minor qualitative differences in the solvers' performance. All experiments were run on single cores, using a computing cluster with ten identical nodes, each having 24 cores, an Intel(R) Xeon(R) CPU E5–2650 v4 @ 2.20GHz and 252 GB RAM.

5.1 Experimental Results

An overview of our experimental results with a runtime limit of 60 seconds can be found in Table 2. For all solvers, we display the percentages of instances for which (i) a feasible solution (in total, for small, medium and large instances), (ii) an optimal solution, and (iii) an optimal solution, including an optimality proof, could be found within the time limit. Note that the difference between (ii) and (iii) is that (iii) reports the percentage of instances that the corresponding approach could solve to proven optimality, whereas (ii) also includes instances where the approach was able to find the optimal solution, but not necessarily with an optimality proof (but we know that the solution is optimal due to a lower bound obtained from one of the other approaches). Moreover, the average optimality gaps (in percentage) with the corresponding standard deviations are reported. The optimality gap for a given instance i is defined as follows: $g(i) = (c(i) - b(i))/c(i) \cdot 100$, where $c(i)$ is the cost of the found solution and $b(i)$ is the best lower bound on the solution cost found by any of the evaluated solvers. In this table, the best results are highlighted in bold font.

A first observation is that some solvers were not capable of finding feasible solutions for all 216 benchmark instances, even though the trivial null assignment—assigning 0

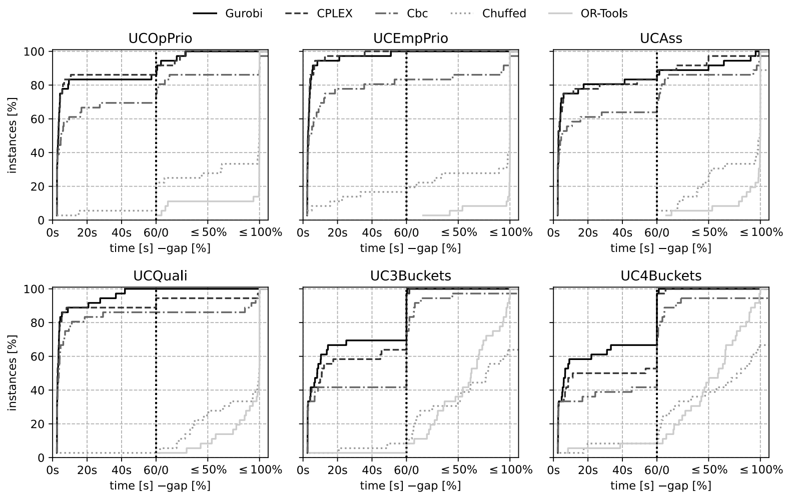


Fig. 3: Performance plot over all 216 instances grouped by the use cases. Use cases with a linear objective function can be solved more efficiently than use cases with non-linear objectives.

to every bucket-operation-employee-triple—is always a feasible solution. Indeed, only Gurobi, CPLEX, and OR-Tools found solutions for all instances, but Cbc failed to find a feasible solution for some large instances. Chuffed was overall less successful at finding solutions. Almost all solutions found by Gurobi were of very high quality, as optimality proofs could be delivered for more than 84 % of all instances. Moreover, for those instances that were not provably solved to optimality within the time limit of 60 seconds, the solutions found were also close to the optimum, as the average optimality gap is 2.1 %.

In terms of optimality, the second-best results could be achieved by CPLEX (roughly 83 % of optimally solved instances), followed by Cbc (slightly less than 66 % of optimally solved instances). Both Gurobi and CPLEX were capable of finding optimal solutions for almost all real-life instances and all use cases. Chuffed and OR-Tools only managed to solve very few instances to optimality and are clearly not competitive with the MIP solvers Gurobi, CPLEX and Cbc for this model. For all solvers except Gurobi and CPLEX, the optimality gaps get large, with average values between 11 and 80 %. Note that for those instances where some solver could find no feasible solution, we used the solution cost of the null assignment to compute the optimality gap.

These results are also presented graphically as a performance plot in Figure 2. The plot is divided into two parts: The left part shows the number of instances solved to proven optimality within a certain number of seconds. For instances that could not be solved to proven optimality within 60 seconds, the right part shows the remaining optimality gap for the number of instances.

Figure 2 also indicates that almost all optimality proofs are delivered by Gurobi, CPLEX and Cbc within the first 20 seconds. More precisely, Gurobi, CPLEX and Cbc were able to prove optimality of 93 %, 96 %, and 95 % of the optimality solved instances, respectively. Moreover, the median proof times of the instances solved to proven optimality are 2.95, 2.83, and 2.86 seconds, for Gurobi, CPLEX and Cbc, respectively. The slowest median proof times are obtained from Chuffed with 15.14 seconds, followed by OR-Tools with 8.24 seconds.

We also conducted experiments with a runtime limit of five minutes. Increasing the time limit allowed the weaker solvers, Chuffed and OR-Tools, to solve more instances and all solvers to deliver some more optimality proofs.

Moreover, we evaluated the results for the six different use cases separately. The results are shown in Figure 3 where for each use case a performance plot is drawn. All solvers could achieve similar results for the use cases UCOpPrio, UCEmpPrio, and UCQuali, with the highest number of optimally solved instances for these three use cases. The other three use cases UCAss, UC3Buckets, and UC4Buckets were harder to solve. This can be explained as follows: UCOpPrio, UCEmpPrio and UCQuali consist of a single, linear objective function that can be handled efficiently by the evaluated solvers. The other use cases were harder to solve. For the other three use cases UCAss, UC3Buckets, and UC5Buckets, Gurobi delivered the overall best results but could not provide optimality proofs within the time limit for 17 %, 28 %, and 33 % of the instances, respectively. Use case UCAss has a non-linear component (counting the number of positive assignments) and use cases UC3Buckets and UC4Buckets consist of a combination of several objectives with two non-linear components (counting the number of positive assignments and the number of employee-operation changes between time buckets), which slows down the solution process. These experimental results also reflect the complexity results from Section 4. Indeed, the objective functions for use cases UCOpPrio, UCEmpPrio and UCQuali can also be expressed as cost functions in a classical transportation problem, which can be solved in polynomial time. However, computational complexity remains also for these use cases due to the presence of max-parallel-operations- and min-assigned-capacity-constraints. Moreover, the use case UCAss corresponds to an $\mathcal{NP}$-hard variant of the ETDP.

6 Conclusion

In this paper, we introduced and formally defined the Employee Task Distribution Problem, a planning problem that arises in practice and has similarities with classical transportation problems. In order to handle a variety of different objective functions and specialized constraints that distinguish the ETDP from transportation problems, we propose a new solver-independent mathematical model. Our experiments conducted with five state-of-the-art solvers on a benchmark set of 216 instances showed that overall best results could be achieved using the MIP solver Gurobi which could provide optimality proofs within less than 60 seconds for more than 84 % of all instances. The MIP solvers CPLEX and Cbc could also achieve high-quality results. All real-life instances provided by our industry partner could be solved optimally within this very short time bound by Gurobi and CPLEX. Overall, the MIP solvers found better solutions for the large majority of instances in our experiments compared to the evaluated CP solvers, indicating that exploiting the linear structure that lies at the core of the problem is crucial for solving the ETDP efficiently.

As a next step, we plan to develop metaheuristic solution methods for the ETDP independent of external solvers. The challenge is to design algorithms that are capable of finding near-optimal solutions within a few seconds. This would significantly increase the practical applicability of our approach. Developing such solutions would allow our industrial partner to include them in a cloud-based web service and offer them to an even more diverse set of customers.

Acknowledgements The financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development and the Christian Doppler Research Association is gratefully acknowledged.

References

1. Angulo, G., Van Vyve, M.: Fixed-charge transportation problems on trees. Operations Research Letters **45**(3), 275–281 (2017)
2. Belov, G., Stuckey, P.J., Tack, G., Wallace, M.: Improved linearization of constraint programming models. In: CP. Lecture Notes in Computer Science, vol. 9892, pp. 49–65. Springer (2016)
3. Bertsekas, D.P., Castanon, D.A.: The auction algorithm for the transportation problem. Annals of Operations Research **20**(1), 67–96 (1989)
4. Cattrysse, D.G., Van Wassenhove, L.N.: A survey of algorithms for the generalized assignment problem. European journal of operational research **60**(3), 260–272 (1992)
5. Chiandussi, G., Codegone, M., Ferrero, S., Varesio, F.: Comparison of multi-objective optimization methodologies for engineering applications. Computers & Mathematics with Applications **63**(5), 912–942 (2012)
6. Dantzig, G.B.: Application of the simplex method to a transportation problem. Chapter XXIII in Activity analysis and production and allocation, Cowles Commission Monograph No. 13, (ed.) T. C. Koopmans (1951)
7. Deb, K.: Multi-objective optimization. In: Search methodologies, pp. 403–449. Springer, Boston, MA (2014)
8. Díaz-Parra, O., Ruiz-Vanoye, J.A., Bernábe Loranca, B., Fuentes-Penna, A., Barrera-Cámara, R.A.: A survey of transportation problems. Journal of Applied Mathematics (2014)
9. Ford Jr, L.R., Fulkerson, D.R.: Solving the transportation problem. Management Science **3**(1), 24–32 (1956)
10. Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman (1979)
11. Hartmann, S., Briskorn, D.: A survey of variants and extensions of the resource-constrained project scheduling problem. European Journal of Operational Research **207**(1), 1–14 (2010)
12. Hartmann, S., Briskorn, D.: An updated survey of variants and extensions of the resource-constrained project scheduling problem. European Journal of Operational Research **297**(1), 1–14 (2022)
13. Hirsch, W.M., Dantzig, G.B.: The fixed charge problem. Naval Research Logistics Quarterly **15**(3), 413–424 (1968)
14. Hitchcock, F.L.: The distribution of a product from several sources to numerous localities. Journal of mathematics and physics **20**(1-4), 224–230 (1941)

15. Horn, M., Lackner, M.L., Mrkvicka, C., Musliu, N., Walkiewicz, D., Winter, F.: Benchmark instances, models and technical appendix for the Employee Task Distribution Problem [Data Set] (Dec 2023), https://owncloud.tuwien.ac.at/index.php/s/JltgU8OiozNaTKf
16. Juman, Z., Hoque, M.: An efficient heuristic to obtain a better initial feasible solution to the transportation problem. Applied Soft Computing **34**, 813–826 (2015)
17. Kantorovich, L.V.: Mathematical methods of organizing and planning production. Publication House of the Leningrad State University (1939). Translated in Management science **6**(4), 366–422 (1960)
18. Kowalski, K.: On the structure of the fixed charge transportation problem. International journal of mathematical education in science and technology **36**(8), 879–888 (2005)
19. Miettinen, K.: Nonlinear multiobjective optimization, vol. 12. Springer Science & Business Media, New York (2012)
20. Monge, G.: Mémoire sur la théorie des déblais et des remblais. Histoire de l'Académie Royale des Sciences de Paris, avec les Mémoires de Mathématique et de Physique pour la même année pp. 666–704 (1781)
21. Nagelhout, R.V., Thompson, G.L.: A single source transportation algorithm. Computers & Operations Research **7**(3), 185–198 (1980)
22. 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.) Principles and Practice of Constraint Programming – CP 2007. pp. 529–543. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2007)
23. Öncan, T.: A survey of the generalized assignment problem and its applications. INFOR: Information Systems and Operational Research **45**(3), 123–141 (2007)
24. Pentico, D.W.: Assignment problems: A golden anniversary survey. European Journal of Operational Research **176**(2), 774–793 (2007)
25. Raju, N.: Transportation problem. In: Operations Research: Theory and Practice, chap. 5, pp. 193–290. CRC Press (2019)
26. Vasko, F.J., Storozhyshina, N.: Balancing a transportation problem: Is it really that simple? OR insight **24**(3), 205–214 (2011)

The Integrated Healthcare Timetabling Competition 2024
— Problem description and rules —

Sara Ceschia[1], Roberto Maria Rosati[1], Andrea Schaerf[1], Pieter Smet[2], Greet Vanden Berghe[2], and Eugenia Zanazzo[1]

[1] Polytechnic Department of Engineering and Architecture, University of Udine, Italy
{sara.ceschia,robertomaria.rosati,andrea.schaerf,eugenia.zanazzo}@uniud.it
[2] Department of Computer Science, KU Leuven, Gent, Belgium
{greet.vanden.berghe,pieter.smet}@kuleuven.be

Abstract. Research in timetabling often focuses on clear-cut academic problems. Real-world healthcare optimization problems, however, encompass additional challenges due to various decision and optimization problems being intertwined. Moreover, general timetabling methodologies are not necessarily suitable for addressing such integrated problems. In the interest of stimulating research on the specifics of integrated scheduling problems in healthcare, this paper introduces the Integrated Healthcare Timetabling Competition 2024 We begin by describing the problem formulation, which integrates three critical problems in healthcare: surgical case planning, patient admission scheduling and nurse-to-room assignment. Next, we discuss the data sets and file formats, along with the solution checker (validator) that we provide for the participants. Finally, we state the rules of the competition and explain how participants will be ranked. All up-to-date information concerning the competition is available at the competition's website https://ihtc2024.github.io.

Keywords: Healthcare, Integrated optimization problem, Competition.

1 Introduction

Integrated healthcare scheduling deals with the coordination of resources related to various services within a single healthcare system. It aims to streamline and optimize the flow of patients across different departments and facilities of the hospital. The benefits of integrated healthcare optimization are manifold: enhanced patient experience, improved operational efficiency, and optimized resource utilization across the entire hospital system.

Contributions to integrated healthcare applications have been surveyed by Rachuba et al. [6]. They identify three levels of increasing integration, ranging from solving a single problem while incorporating the constraints coming from the other problems (level 1), to sequentially solving two or more problems using the output of one problem as input for the next one (level 2), and finally to simultaneously solving two or more problems at once in a single stage (level 3).

A recent proposal for level 3 integration by Brandt et al. [1] addresses the simultaneous resolution of two operational problems that are critical in hospitals: the Patient-to-Room Assignment (PRA) and the Nurse-to-Patient Assignment (NPA) problems.

The Integrated Healthcare Timetabling Competition 2024 (IHTC) revises the problem introduced by Brandt et al. and generalizes it by incorporating a third important optimization problem in hospitals, namely Surgical Case Planning (SCP)[7]. The resulting integrated problem, which we call the *Integrated Healthcare Timetabling Problem* (IHTP), brings together three $\mathcal{NP}$-hard problems and requires the following decisions: (i) the admission date for each patient (or admission postponement to the next scheduling period), (ii) the room for each admitted patient for the duration of their stay, (iii) the nurse for each room during each shift of the scheduling period, and (iv) the operating theater (OT) for each admitted patient.

The IHTP is subject to many hard and soft constraints. Some of these constraints relate to a specific subproblem, while others arise from their interactions. The IHTP is a special case of real-world timetabling at hospitals, which are often subject to additional constraints. Furthermore, we consider the *static, deterministic* variant of the IHTC, in which all information for a fixed scheduling period is known at the time of solving.

The remainder of this paper is organized as follows. Section 2 provides the problem definition. Section 3 introduces the datasets and the validator made available to participants for evaluating their solutions. Finally, Section 4 describes the rules of the competition. Appendix A is also included as supplementary material, which describes the file formats. All up-to-date information concerning the competition is available at the competition's website https://ihtc2024.github.io.

2 Problem definition

After first introducing the basic concepts of the IHTP, we will define the hard and soft constraints of the problem and explain how they must be evaluated throughout the entire scheduling horizon.

2.1 Basic concepts

We begin by introducing the time horizon and physical resources involved in the IHTP:

Scheduling period: The scheduling period is defined as a number D of consecutive days. D is always a multiple of seven, and can vary from 14 (two weeks) to 28 (four weeks).

Shifts: A shift denotes a nurse's working period during a day. We assume three non-overlapping shifts per day: *early*, *late*, and *night*. The entire scheduling period thus consists of $3D$ shifts. Each shift is denoted by an integer ranging from 0 to $3D - 1$. The early, late and night shifts on the first day are numbered 0, 1, and 2, respectively. For the second day, the shifts are numbered 3, 4, and 5. This pattern continues until the end of the scheduling period.

Operating theaters: All OTs are identical in that they are suitable for accommodating any type of surgery. Each OT has a daily maximum capacity, expressed in minutes.

Some OTs might be unavailable on specific days, indicated by a maximum capacity of 0 minutes on those days.

Rooms: Rooms host the patients during their recovery. These rooms are characterized by their capacity, expressed in terms of the number of beds. Room equipment is not explicitly taken into account. However, as will be outlined in what follows, some rooms might be declared unsuitable for some patients.

Next, we describe the human resources that are involved in the IHTP:

Nurses: Each nurse has a skill level. Levels are strictly ordered (hierarchical) and represented by an integer that ranges from 0 (lowest) to $L - 1$ (highest), where L is the number of skill levels. Furthermore, each nurse has a predetermined roster, which is defined as a set of shifts that the nurse has been assigned to, along with the maximum workload the nurse can accommodate in each shift. This roster is fixed and cannot be changed.

Surgeons: Each surgeon has a maximum operating time on each day, which is 0 when the surgeon is unavailable on that day. If a surgeon is available, we assume their surgical team is also available. In other words, the surgeon and their team form an atomic indivisible resource (called *surgeon* for simplicity).

Note that the maximum nurse workload is shift-dependent as nurses can carry out auxiliary activities during some specific working shifts, thereby reducing their availability. Also note that we assume an *open scheduling policy* [3], which means that all surgeons can operate in all OTs.

The patient is the central entity of the problem. The following information is provided for each patient:

– **mandatory/optional**: mandatory patients must be admitted during the scheduling period, while the admission of optional patients can be postponed until a future scheduling period.
– **release date**: earliest possible admission date for the patient.
– **due date**: latest possible admission date, provided only for mandatory patients.
– **age group**: the age group of the patient (e.g., infant, youth, adult, elder). The list of age groups is fully ordered.
– **gender**: the gender of the patient.
– **length of stay**: duration of the hospitalization in days.
– **incompatible rooms**: set of rooms that must not be allocated to the patient because, for example, they do not have the specific equipment or the necessary isolation.
– **surgery duration**: the expected duration of the patient's surgery, which is assumed to always take place on the day of admission.
– **surgeon**: the surgeon who carries out the patient's surgery.
– **workload**: the workload profile generated by the patient, which is described by a vector, starts at the early shift of the admission day and ends at the night shift of the discharge day. The length of the vector equals 3 times the patient's length of stay.
– **minimum skill level**: the minimum nurse skill level required by the patient for each shift they are staying in the hospital; described by a vector similar to the patient workload vector.

Note that both the workload and the minimum skill level required for a patient can vary based on the shift and how long the patient has been in the hospital, as these factors are related to the patient's treatments and stage of recovery. Both values are usually lower during night shifts and higher during the initial days of the stay.

2.2 Solution

The solution of an IHTP instance consists of the following decisions:

i. the admission date for patients, or, in the case of optional patients, potentially their postponement to the next scheduling period;
ii. the allocation of a room for each admitted patient;
iii. the assignment of a single nurse to each occupied room, for each shift within the scheduling period;
iv. the assignment of patient surgeries to OTs, for each day of the scheduling period.

We assume that patients are always admitted and discharged after the night shift and before the early shift. Note that a patient stays in only one room during the entire length of their stay, meaning a patient cannot be transferred from one room to another.

We also assume that all patients undergo surgery, and that this takes place on the day of admission. In addition, as each patient's surgeon is predetermined, the day of admission automatically determines the total surgery time of each surgeon on each day. By contrast, the OT must be selected. This assignment does not include the precise operating time, only the date. The IHTP does not consider the order of surgeries in an OT.

Finally, note that it is necessary to assign a nurse to a room on a given shift only if that room contains patients on the day to which the shift belongs. Nevertheless, assigning nurses to empty rooms would be feasible and does not incur additional costs.

2.3 Constraints

We divide the constraints into four sets: (i) those related to the PAS problem, (ii) those related to the NRA problem, (iii) those related related to the SCP problem, and finally (iv) those related to the integration of the three problems. In addition, constraints are categorized as either hard (starting with **H**) or soft (starting with **S**). The former must always be satisfied, while the latter contribute to the objective function. Violations of soft constraint $\mathbf{S}_i$ are multiplied by weight W_i. Note that the soft constraint weights are instance-specific and thus given in each input file.

Constraints on Patient Admission Scheduling

H1 No gender mix: Patients of different genders may not share a room on any day.
H2 Compatible rooms: Patients can only be assigned to one of their compatible rooms.
S1 Age groups: For each day of the scheduling period and for each room, the maximum difference between age groups of patients sharing the room should be minimized.

Constraints on Nurse-to-Room Assignment

While the IHTP does not explicitly require the assignment of nurses to patients, the combination of patient-to-room assignments and nurse-to-room assignments determines which nurses are responsible for which patients. The following constraints (**S2**, **S3**, and **S4**) depend on the resulting nurse-patient assignment.

S2 Minimum skill level: The minimum skill level a nurse must have to provide the required care for a patient during each shift of their stay should be met. If the skill level of the nurse assigned to a patient's room in a shift does not reach the minimum level required by that patient, a penalty is incurred equal to the difference between the two skill levels. Note that a nurse with a skill level greater than the minimum required can be assigned to the room at no additional cost.

S3 Continuity of care: To ensure continuity of care, the total number of distinct nurses providing care to a patient during their entire stay should be minimized. The given rosters assume maximum one shift per day for each nurse, hence the number of different nurses who take care of a patient is at least 3.

S4 Maximum workload: For each shift, the total workload induced by patients staying in rooms assigned to a nurse should not exceed the maximum workload of that nurse in that shift.

Constraints on Surgical Case Planning

H3 Surgeon overtime: The maximum daily surgery time of a surgeon must not be exceeded.

H4 OT overtime: The duration of all surgeries allocated to an OT on a day must not exceed its maximum capacity.

S5 Open OTs: The number of OTs opened on each day should be minimized. Note that if an OT has no patients assigned for a particular day, it should not open on that day.

S6 Surgeon transfer: The number of different OTs a surgeon is assigned to per working day should be minimized.

Global constraints

H5 Mandatory versus optional patients: All mandatory patients must be admitted within the scheduling period, whereas optional patients may be postponed to future scheduling periods.

H6 Admission day: A patient can be admitted on any day from their release date to their due date. Given that optional patients do not have a due date, they can be admitted on any day after their release date.

S7 Admission delay: The number of days between a patient's release date and their actual date of admission should be minimized.

S8 Unscheduled patients: The number of optional patients who are not admitted in the current scheduling period should be minimized.

2.4 Boundary data

We assume that some patients are already present in the hospital on the first day of the scheduling period. We use the term *occupants* to refer to these special patients. While these occupants contribute to the occupancy of the rooms and to all related constraints, their admission date and room assignment are fixed. Occupants do not contribute to the OTs' occupancy because their surgery occurred during the preceding scheduling period.

For patients admitted during the current scheduling period and who stay after the end of it, no penalties are incurred after the end of the horizon.

3 Datasets and validator

Problem instances are supplied as JSON files following the structure outlined in Appendix A. Each instance is contained within a single file. We provide a *public* dataset composed of 30 instances, named `i01`, . . . `i30`, with a scheduling period ranging from two to four weeks and a number of patients ranging from approximately 50 to 500. In addition, we provide five instances, `test01`, . . . , `test05`, for testing and debugging purposes. We also provide a solution for each test instance. We will employ a different *hidden* dataset to evaluate the participants' submissions. This dataset will be shared with the participants at the end of the competition. Both the *public* and *hidden* datasets are generated using the same instance generator, which utilizes realistic patterns and distributions.

Generated solutions must be saved as JSON files adhering to the format described in Appendix A. The validator, which certifies the feasibility and quality of a given solution, is provided as a C++ source code and should be compiled using, for example, the GNU compiler g++. The validator receives the instance and solution files as command line parameters, as demonstrated in the following example.

```
> ./IHTP_Validator.exe input_file.json sol_file.json
```

The command line output of the validator appears as follows:

```
VIOLATIONS:
RoomGenderMix.....................0
PatientRoomCompatibility..........0
SurgeonOvertime...................0
OperatingTheaterOvertime..........0
MandatoryUnscheduledPatients......0
AdmissionDay......................0
RoomCapacity......................0
NursePresence.....................0
UncoveredRoom.....................0
Total violations = 0

COSTS (weight X cost):
RoomAgeMix............................5 (  5 X   1)
RoomSkillLevel.......................21 (  1 X  21)
```

```
ContinuityOfCare.....................43 (  1 X  43)
ExcessiveNurseWorkload................0 (  1 X   0)
OpenOperatingRoom...................100 ( 50 X   2)
SurgeonTransfer.......................0 (  5 X   0)
PatientDelay.........................50 ( 10 X   5)
ElectiveUnscheduledPatients...........0 (300 X   0)
Total cost = 219
```

If `verbose` is added as a third parameter, the details of each single cost element are also printed:

```
Room r0 is age-mixed 1/2 in day 1
Nurse n5 is underqualified for occupant a1 in room r0 in shift 3 (day1@early)
Nurse n6 is underqualified for patient p5 in room r0 in shift 4 (day1@late)
...
6 distinct nurses for occupant a0
4 distinct nurses for occupant a1
...
Operating theater t0 is open on day 1
Operating theater t0 is open on day 4
Patient p0 has been delayed for 1 days
Patient p1 has been delayed for 2 days
...
```

4 Competition rules

This competition seeks to encourage research into automated timetabling and scheduling methods for solving an integrated healthcare problem, with prizes offered for the most successful methods. As with any set of rules for any competition it is possible to work within the letter of rules but outside their spirit. We, as organizers, ask all participants to respect both the letter and spirit of these rules. Failing to do so will result in disqualification.

Rule 1: We reserve the right to update the rules at any time if they believe it is necessary for the sake of ensuring the correct operation of the competition. Any change of rules will be notified in the repository.

Rule 2: The competition has deadlines concerning when all submissions must be uploaded. These deadlines are strict and no extensions will be given under any circumstances.

Rule 3: Participants may use any programming language. The use of third-party software is allowed under the following restrictions:
- either it is open source (https://opensource.org/osd) or it provides a free, unlimited academic license;
- its behavior is (reasonably) documented;
- it runs under a commonly-used operating system (Unix/Linux, Windows, or MAC OS).

Rule 4: The solution method should take as input a file in the format described, and produce as output a solution file in the correct format. The algorithm must stop after 10 minutes wall time. Parallel computing is allowed, using up to 4 threads.

Rule 5: The solution method may be either deterministic or stochastic. In both cases, participants must be prepared to show that the results are repeatable within the given computational time. In particular, participants using a stochastic algorithm should do their utmost best to code their program in such a way that the run producing each submitted solution can be replicated by reusing the same random seed.

Rule 6: Participants must submit (i) solutions for all instances from the public dataset and (ii) a clear and concise description of their algorithm before the first competition deadline. A set of 5 finalists will be determined by ranking the participants on each public instance.
If the first 5 finalists all use licensed software, the number of finalists will be increased to 6 by adding the best-ranked solver using only open-source software.
An infeasible or missing solution will equate to the last position in the ranking for that particular instance. The mean average of the ranks across all instances will produce the participant ordering, of which the first 5 are then selected as finalists. Section 4.2 provides additional details on how the ordering will be established.

Rule 7: We will rerun the finalists' solution methods on the hidden dataset using the same time limit specified in Rule 4. The official PC will be a AMD Ryzen Threadripper PRO 3975WX, 3.50 GHz, running Ubuntu Linux 22.4. A different operating environment might be used in exceptional cases if necessary. It is the responsibility of competition participants to ensure that all files and information needed to run their code is provided to us.

Rule 8: The final ranking of the finalists will be based on the ranks obtained for each instance for a set of trials on hidden instances. Section 4.2 provides an explanation of the procedures to be used.

4.1 Dates

The competition will be announced at different conferences during the summer of 2024, including PATAT 2024 and ORAHS 2024. The competition will then officially begin on September 1, 2024. On this date, we will release the public dataset, the specifications, and the validator. The deadline for submission of participants' best solutions and a description of their solution method is March 1, 2025. Notifications of the finalists will be sent out on April 1, 2025. The winners will be announced at the EURO 2025 conference in Leeds, UK (June 22-25, 2025).

4.2 Adjudication procedure

We follow the same adjudication procedure used in the First and Second International Nurse Rostering Competitions (INRC-I, INRC-II) [4,2], which was originally imported from the Second International Timetabling Competition (ITC-2007) [5]. The procedure is repeated here for the sake of completeness.

Let m be the total number of problem instances and k the number of participants who produce a solution for all m instances. Let X_{ij} be the result supplied (and verified) by participant i for instance j. Each X_{ij} is the value of the objective function s, for participant i on instance j. In case participant i is unable to provide a feasible solution for instance j, X_{ij} is assigned a conventional value M larger than the result supplied by any other participant for that instance.

The matrix X of results is transformed into a matrix of ranks R by assigning to each R_{ij} a value from 1 to k. That is, for instance j the supplied X_{1j}, X_{2j}, $\ldots$,X_{kj} are compared with each other and rank 1 is assigned to the smallest value observed, rank 2 to the second smallest, and so on to rank k, which is assigned to the largest value for instance j. Ranks are assigned for all the instances. We use average ranks in case of ties. If a solution method produces an infeasible solution, it will be assigned the highest rank for the corresponding instance. The rule of average ranks for tie-breaking is not applied in case of infeasibility: solution methods that generate infeasible solutions or fail to generate solutions at all are assigned rank k for the corresponding instance, in which k is the total number of participating solution methods.

Consider the example with $m = 6$ instances and $k = 7$ participants in Table 1. Table 2 shows the ranks.

Table 1: An example of submitted solution scores.

Instance	1	2	3	4	5	6
Solution method 1	34	35	42	32	10	12
Solution method 2	32	24	44	33	13	15
Solution method 3	33	36	30	12	10	17
Solution method 4	36	32	46	32	12	13
Solution method 5	37	30	43	29	9	4
Solution method 6	68	29	41	55	10	5
Solution method 7	36	30	43	58	10	4

Table 2: Corresponding solution ranks for the example.

Instance	1	2	3	4	5	6
Solution method 1	3	6	3	3.5	3.5	4
Solution method 2	1	1	6	5	7	6
Solution method 3	2	7	1	1	3.5	7
Solution method 4	4.5	5	7	3.5	6	5
Solution method 5	6	3.5	4.5	2	1	1.5
Solution method 6	7	2	2	6	3.5	3
Solution method 7	4.5	3.5	4.5	7	3.5	1.5

We define for each solution method the mean of the ranks. The finalists of the competition will be the 5 solution methods with the lowest mean ranks. In case of a tie for the last position, all tying methods will be included in the final (in this case the number of finalists will be more than 5). Table 3 shows the mean ranks for the example.

Table 3: Mean ranks.

Solution method 1	3.83
Solution method 2	4.33
Solution method 3	3.58
Solution method 4	5.17
Solution method 5	3.08
Solution method 6	3.92
Solution method 7	4.08

In this case, the finalists would be solution methods 1, 3, 5, 6 and 7.

During the final phase of the competition, the evaluation process is repeated for the finalists with the following new elements:

1. The hidden dataset will be used.
2. We will run the solution methods of the finalists. We expect the finalists to offer support in the process of compiling and running their solution method.
3. For each problem instance, we will run 10 independent trials with random seeds. For each trial, we will compute the ranks and average them over all trials on all instances.

The winner is the participant with the lowest mean rank. In case of a tie, an additional trial will be run for all instances until a single winner is found.

4.3 Prizes

The top three teams will receive a cash prize (first prize € 1100, second € 700, third € 400), and be offered one non-transferable free registration to EURO 2025, which will host a special track dedicated to the competition.

The best open-source finalist will receive a special prize of € 200, which can be awarded in addition to the regular prize.

Acknowledgements The group at University of Udine has been funded by the project “Models and algorithms for the optimization of integrated healthcare management” (no. 2020LNEZYC) supported by the Italian Ministry of University and Research (MUR) under the PRIN-2020 program. This research was also supported by the project “Human-centred decision support based on a new theory for personnel rostering", KU Leuven-IOF (C2E/23/012).

Appendix A – File formats

Input and solution files are written in JSON. The input file contains a header part in addition to separate sections for nurses, rooms, operating theaters, surgeons, patients, and occupants. What follows is an example of the header part, containing the general data and the weights of the cost components.

```
{
  "days": 28,
  "skill_levels": 3,
  "shift_types": [
    "early",
    "late",
    "night"
  ],
  "age_groups": [
    "infant",
    "adult",
    "elderly"
  ],
  "weights": {
    "room_mixed_age": 5,
    "room_nurse_skill": 10,
    "continuity_of_care": 5,
    "nurse_eccessive_workload": 10,
    "open_operating_theater": 20,
    "surgeon_transfer": 1,
    "patient_delay": 5,
    "unscheduled_optional": 350
  }
 ...
}
```

What follows is a fragment of an example for the section about nurses. For each nurse, we have a unique identifier (id), the skill level and a list of working shifts with their respective maximum workloads.

```
"nurses": [
    {
      "id": "n00",
      "skill_level": 0,
      "working_shifts": [
        {
          "day": 0,
          "shift": "early",
          "max_load": 10
        },
        {
          "day": 1,
          "shift": "night",
          "max_load": 5
        },
        ...
    ]},
    ...
    ]
```

The structure of the sections concerning rooms, OTs, and surgeons is straightforward and shown in the following fragment.

```
 "surgeons": [
    {
      "id": "s0",
      "max_surgery_time": [0, 360, 0, 600, 480, 0, 0, 600, ...]
    },
    ...
  ],
  "operating_theaters": [
    {
      "id": "t0",
      "availability": [0, 600, 720, 600, 600, 720, 720, ...]
    },
    ...
  ],
  "rooms": [
    {
      "id": "r0",
      "capacity": 2
    },
    {
      "id": "r1",
      "capacity": 3
    },
   ....
   ]
```

Finally, we introduce the structure of the patient data, divided into occupants (present at the beginning of the scheduling period) and regular patients.

```
 "occupants": [
    {
      "id": "a0",
      "gender": "B",
      "age_group": "elderly",
      "length_of_stay": 2,
      "workload_produced": [2, 1, 1, 2, 3, 2],
      "skill_level_required": [1, 2, 0, 0, 0, 0],
      "room_id": "r21"
    },
    ...
    ]
 "patients": [
    {
      "id": "p28",
      "mandatory": true,
      "gender": "A",
      "age_group": "elderly",
      "length_of_stay": 3,
      "surgery_release_day": 3,
      "surgery_due_day": 17,
      "surgery_duration": 90,
      "surgeon_id": "s0",
      "incompatible_room_ids": ["r2"],
      "workload_produced": [1, 1, 1, 2, 1, 1, 1, 2, 1],
      "skill_level_required": [1, 2, 0, 2, 0, 0, 2, 1, 1]
    }
    ...
    }
```

The solution file format is divided into two sections: one concerning patients and one concerning nurses. The following fragment illustrates both sections.

```
{
  "patients": [
    {
      "id": "p00",
      "admission_day": 4,
      "room": "r3",
      "operating_theater": "t0"
    },
    ...
    ],
  "nurses": [
    {
      "id": "n00",
      "assignments": [
        {
          "day": 0,
          "shift": "early",
          "rooms": ["r1", "r4"]
        },
        {
          "day": 1,
          "shift": "night",
          "rooms": ["r4", "r5"]
        },
        ...
        ]
    },
    ...
    ]
}
```

References

1. Brandt, T., Klein, T.L., Reuter-Oppermann, M., Schäfer, F., Thielen, C., van de Vrugt, M., Viana, J.: Integrated patient-to-room and nurse-to-patient assignment in hospital wards. arXiv preprint 2309.10739 (2023)
2. Ceschia, S., Dang, N., De Causmaecker, P., Haspeslagh, S., Schaerf, A.: The second International Nurse Rostering Competition. Annals of Operations Research **274**(1), 171–186 (2019)
3. Guerriero, F., Guido, R.: Operational research in the management of the operating theatre: a survey. Health care management science **14**, 89–114 (2011)
4. Haspeslagh, S., De Causmaecker, P., Schaerf, A., Stølevik, M.: The first International Nurse Rostering Competition 2010. Annals of Operations Research **218**, 221–236 (2014)
5. McCollum, B., Schaerf, A., Paechter, B., McMullan, P., Lewis, R., Parkes, A.J., Di Gaspero, L., Qu, R., Burke, E.K.: Setting the research agenda in automated timetabling: The second International Timetabling Competition. INFORMS Journal on Computing **22**(1), 120–130 (2010)
6. Rachuba, S., Reuter-Oppermann, M., Thielen, C.: Integrated planning in hospitals: A review. arXiv preprint 2307.05258 (2023)
7. Zhu, S., Fan, W., Yang, S., Pei, J., Pardalos, P.M.: Operating room planning and surgical case scheduling: a review of literature. Journal of Combinatorial Optimization **37**, 757–805 (2019)

A Rule Language and Feature Model for Educational Timetabling*

Corentin Behuet, Vincent Barichard, David Genest, Marc Legeay, and David Lesaint

Univ Angers, LERIA, SFR MATHSTIC, F-49000 Angers, France
`{firstname.lastname}@univ-angers.fr`

Abstract. Educational timetabling subsumes core problems (student sectioning, course scheduling, etc.) which are challenging from a modeling and computational perspective. In this paper, we expand on the University Timetabling Problem framework (UTP) designed to address a wide range of university timetabling problems. The framework combines a rich data schema with a rule language and comes with a tool chain to compile instances into constraint satisfaction problems. We present the UTP modeling language and a feature model to capture the problem classes that are expressible. The feature model provides a simple problem classifier which we use in our literature review. We also present a timetabling instance generator and report on experiments carried out with Constraint Programming, Answer-Set Programming and Mixed Integer Linear Programming solvers.

Keywords: Timetabling, Domain-Specific Modeling Language, Feature Model, Exact Methods, Timetable Dataset Generation

1 Introduction

Various problem formulations, data formats and algorithms have been proposed to tackle specific aspects of university timetabling ranging from curriculum balancing [16,18,50], student sectioning [42,52], examination timetabling [14,10,40], curriculum-based or post-enrollment-based course timetabling [40,12,37,13,26,17], tutor allocation [15], to minimal timetabling perturbation [38,36]. Modeling languages have also been developed, notably the XHSTT language [48], the ITC language used in the 2019 international timetabling competition [41,29] and the UTP language introduced in [9].

UTP is a modeling language for educational timetabling problems which is built on a structured domain model coupled with a rules language. The model supports sessions requiring a single resource and those needing multiple resources capturing essential limitations related to the timing of sessions and distribution of resources. It operates under the presumption that resources can overlap (i.e. rooms, teachers, and students can be involved in simultaneous sessions), though this approach can be adjusted through specific scheduling rules that prevent such overlaps. Given a UTP instance, the objective is to assign time slots and allocate resources to class sessions so that core constraints and rules are satisfied.

We first introduce the UTP schema which has been extended to broaden the range of problems that can be modeled. We then present a feature model to classify educational

*Supported by a research grant from Université d'Angers.

timetabling problems and compare UTP with other modeling languages. Lastly, we report on experiments carried out with 3 UTP solvers - CP, ASP, MIP - on instances created with a custom generator.

2 The UTP Schema

The UTP schema combines a schema to model timetabling entities and solutions, and a rule language. The entity schema models the entities of a UTP instance - scheduling horizon, resources, and course elements including course sessions -, their properties and relationships. The rule language is user-oriented and serves to concisely express constraints over any set of entities on the different facets of a problem (e.g., session scheduling, capacity planning, resource allocation). Rules are formulated using a catalog of timetabling constraint predicates and a query language to select, filter and bind entities to sessions.

A rule-based UTP instance is converted to a constraint-based instance that is readily processable by solvers. The conversion translates the entity schema as decision variables and core constraints, and then flattens rules as additional constraints. A UTP instance is thus cast as a hard constraint satisfaction problems. Solving an instance involves scheduling sessions and assigning them resources so that the core constraints and the rule constraints are satisfied. The solution schema allows to represent any timetabling solution computed for an instance, be it incomplete or inconsistent.

This section introduces the components of the schema. The abstract syntax of the entity schema is given in Table 1, its constraint-based modeling in Table 6 and Table 7 (Appendix), and the syntax of the rule language and constraint predicates in Table 8, Table 9 and Table 10 (Appendix).

2.1 Entity Schema

The entity schema of a UTP instance combines a hierarchy of course elements (i.e., courses, course parts, part classes and class sessions) a scheduling horizon over which sessions are to be scheduled, and 4 types of resources to which sessions must be allocated to (i.e., rooms, teachers, students and student groups). The schema encodes the nesting of course elements and various properties and constraints concerning session scheduling, resource availability, resource eligibility, teaching service, room capacity, and student sectioning.

The scheduling horizon is a range of integers denoting time points. The time points are the start and end times allowed for sessions and any duration (i.e., session length, travel time and break time) is measured as a number of time points. The horizon is defined using 3 instance fields: the number of weeks w dividing the horizon, the number of weekdays d making a week and the number of daily slots m making a 24-hour day. The time points correspond to all possible triplets combining a week, a weekday, and a daily slot. Note that daily slots may have any granularity (e.g., 1 minute, 2 hours) and the scheduling horizon may be sparse (e.g., if weeks i and $i + 1$ are not consecutive calendar weeks for some $1 \leq i < w$ or if weekdays are dropped, i.e., $d < 7$).

Course elements follow a hierarchical structure. Each course (e.g., Algorithms) consists of one or more parts (e.g., Lecture and Lab), each part is taught to one or more classes (e.g., lecture classes A and B), and all classes of a part have the same number of sessions (e.g., sessions 1 to 10 for each lecture class). The schema requires that all sessions in a class have the same duration and be chronologically ranked, i.e., session of rank $i + 1$ in a class must start after session of rank $i - 1$ ends in any solution. These constraints are paramount to model course plans that rely on clear-cut sessions (e.g., starting lab classes after 2 lecture sessions, synchronizing the 5^{th} sessions of lab classes for a joint examination). Besides, the schema allows to restrict the possible time slots for the sessions of a part by setting allowed and forbidden ranges using the time format. Note that sessions must start and end on the same day, and cannot be interrupted. The schema also specifies a set of possible resources for each session. As for students, a sectioning plan is assumed and hard-coded together with group-to-class assignments. Specifically, students are partitioned into groups, and groups aggregated and assigned to classes with no group being assigned to more than one class per course part. The schema encodes group and class headcounts as well as class headcount thresholds used for sectioning. Other sectioning data and constraints are compiled away. The implicit constraint to satisfy is that a group must attend all the sessions of a class it is bound to.

Teacher-to-session assignments are not fixed but subject to domain and cardinality constraints. To meet practical needs, the schema allows multiple teachers per session (e.g., joint supervision of a lab session) and teacher-less sessions (e.g., unsupervised project work). The number of teachers per session is specific to each course part and is lower- and upper-bounded, possibly fixed. Each part is also associated with a set of required teachers and a superset of allowed teachers. Hence two sessions of a class may be allocated different teachers and numbers of teachers. A part also sets the fixed number of sessions a teacher is committed to. Overall, various demand and capacity requirements relating to teaching service can be addressed on course parts. If needed, finer-grained rules may be imposed (e.g., requiring the same staff for a class, naming a lecturer for a session).

Similarly, each part sets the required and possible rooms for its sessions and their number. This caters for the case of multi-room sessions (e.g., for hybrid teaching) and room-less sessions (e.g., field trips). In addition, each part casts its sessions as room-exclusive or room-inclusive which entails different allocation constraints. A session is room-exclusive if none of the room(s) hosting it may simultaneously host another session. Conversely, a room-inclusive session allows for its room(s) to be shared from start to finish. While single-room sessions may be cast as exclusive or inclusive, multi-room sessions may only be cast as exclusive. That is, every session of a part whose room upper-bound is greater than 1 is considered exclusive. The rationale is that multi-room inclusive sessions have arguably little practical interest and they also burden the computational model with decisions to make on the distribution of groups in shared rooms.

All resources enforce capacity constraints w.r.t. their utilization. Students, teachers and rooms are considered cumulative resources in this respect. That is, they may attend, teach or host simultaneous sessions. A cumulative model is paramount to satisfy flexible attendance requirements (e.g., students attending tutoring sessions overlapping with compulsory courses) and multi-class events (e.g., an amphitheater hosting a joint conference for different classes). Again, rules may be used to impose disjunctive resources or to ban session overlapping. No limit is set on the number of parallel sessions teachers and students may attend. Session hosting however is subject to capacity constraints and the schema encodes the capacity of each room, allowing for infinite capacity to handle virtual rooms. As discussed above, at any point in time, an allocated room will either co-host a multi-room (exclusive) session or host one or more single-room sessions (one only if a session is exclusive). The schema hence enforces two kinds of capacity constraints. The single-room case involves checking if the total headcount of the session(s) falls below the room capacity. The multi-room case involves ensuring the total capacity of the rooms envisaged for the session exceeds its headcount. If so, no restriction is imposed as to the distribution of students in rooms and whether it preserves group structure or not.

Lastly, the schema provides users with the ability to define their own classes of entities, mixing course elements and resources as needed with no limit on classification (e.g., a block of rooms, the lecturers of a faculty department). This is achieved by labeling entities. Labels, built-in entity types and ids are the building blocks of the query language to forge rules for any group of entities.

Table 1 provides a formal specification of the schema elements. Resources and course elements, except sessions, are referred to as *entities*. Entities are typed, the set of sessions is cast as distinct type, and each type is modeled as a finite set. The course element hierarchy defines 1-to-many *composition relations* over the pair of types (X, Y) corresponding to parent and child types in the course element hierarchy. Each relation is modeled by a function $d^{X,Y} : X \rightarrow 2^Y$ mapping each object i of type X to the set $d_i^{X,Y}$ of its constitutive objects of type Y. For instance, $d^{P,K}$ models the classes of each part. Each *compatibility relation* defining the allowed or assigned resources of a course element object for a given resource type and course element type defines a many-to-many relation which we model the same way. For instance, $d^{P,R}$ models the allowed rooms per part and $d^{K,G}$ the set of groups assigned to classes.

For notational convenience, the table also defines the maps resulting from the symmetric and transitive closure of the binary relation merging the composition and compatibility maps. This includes the maps computed over the course tree. For instance, $d^{K,P}$ models the (singleton) part of each class, and $d^{C,S}$ the sessions of a course. This also includes the inverse compatibility constraints and those inherited along the course tree. For instance, $d^{S,R}$ models the rooms allowed for a session which results from the composition of $d^{S,K}$, $d^{K,P}$ and $d^{P,R}$. Lastly, the table defines the constants (e.g., number of weeks), scalar properties (e.g., room capacity), and remaining relations and sets (e.g. required resources, labels).

2.2 Solutions

The solution schema is used to encode any solution pre-computed for an instance. Such a solution needs not be complete, nor consistent with the instance constraints. An instance may hence be associated with any kind of input solution based on the computational task, e.g. no solution at all when generating a timetable from scratch, a resource allocation solution to extend into a complete timetable, a seed solution to

w	number of weeks dividing the scheduling horizon
d	number of weekdays making a week
m	number of daily slots making a 24-hour day
$W = \{1, \ldots w\}$	range of weeks
$D = \{1, \ldots d\}$	range of weekdays
$M = \{1, \ldots m\}$	range of daily slots
$H = \{1, \ldots w \times d \times m\}$	range of time points (schedule horizon)
C	courses
P	course parts
K	part classes
R	rooms
T	teachers
U	students
G	groups of students
$\Gamma = \{C\}$	course domain
$\mathcal{E} = \{\Gamma, C, P, K, R, T, U, G\}$	types of entities
$E = \cup_{X \in \mathcal{E}} X$	set of entities
$d_i^{X,Y} \subseteq Y, X \in \{\Gamma, C, P, K\}$	set of entities of type Y tied to entity i of type X
$d_i^{X,Y} \subseteq Y, X \in \{R, T, U, G\}$	set of entities of type Y associated with entity i of type X
$\mathcal{L} \subseteq 2^E$	labels
$S^{(e)}$	exclusive class sessions
$S^{(i)}$	inclusive class sessions
$S = S^{(e)} \uplus S^{(i)}$	class sessions
$d_s^{S,H} \subseteq H$	start times allowed for session s
$d_s^{S,X} \subseteq X$	set of entities of type X tied to session s
$d_i^{X,S} \subseteq S$	set of sessions tied to entity i of type X
$\underline{d}_i^{X,S} \subseteq S$	set of sessions required by resource entity i of type X
$min\_rooms_p^P \in \mathbb{N}$	min number of rooms usable by each session of part p
$max\_rooms_p^P \in \mathbb{N}$	max number of rooms usable by each session of part p
$min\_lecturer_p^P \in \mathbb{N}$	min number of lecturers usable by each session of part p
$max\_lecturer_p^P \in \mathbb{N}$	max number of lecturers usable by each session of part p
$size_g^G \in \mathbb{N}$	headcount of group g
$size_k^K \in \mathbb{N}$	headcount of class k
$capacity_r^R \in \mathbb{N}$	capacity of room r
$length_s^S \in H$	duration of session s
$rank_s^S \in \mathbb{N}^*$	rank of session s in its class
$service_{t,p}^{T \times P} \in \mathbb{N}$	number of sessions required by teacher t in part p

Table 1: Core data model.

improve, or an inconsistent solution to repair. Formally, the solution schema supports the representation of any decision made for a session as to its start time, its set of rooms and its set of teachers.

2.3 Predicates, Constraints and Rules

The UTP schema comes with a rule language to formulate instance-specific constraints. Rule constraints add to the built-in constraints of the schema and all must be checked when evaluating a solution. The rule language is designed to target groups of entities, or individual entities, and constrain the scheduling of their sessions from any standpoint (e.g., an institutional rule imposing a time structure on curricula, a disjunctive scheduling rule applied to student groups, a rule modeling the service plan within a faculty department, a rule for a lecturer's agenda). The schema comes with a catalog of timetabling predicates to build rules and compile them into constraints. It also includes a query language to select entities and sessions on which rules should apply.

All these components are designed around the concept of *e-map*. Formally, an e-map is a pair (e_i, S_i) mapping an entity e_i to a set S_i of sessions. The query language is used to forge queries that retrieve sets of e-maps. Each query selects, filters and binds entities to sessions from instance data in order to extract one or more sets of e-maps. Each rule is bound to a predicate and scoped by a query. At flattening time, the query is performed to retrieve a fixed number of sets of e-maps. The rule is then compiled into a conjunction of constraints by computing the cross-product of the extracted sets and applying the predicate to each tuple of e-maps in the cross-product. Constraint e-maps act as guards when checking solutions and they also narrow the scope of interpretation. The rationale is to discard constraints that are irrelevant (e.g., a teacher's constraint forbidding afternoon lectures while the solution only assigns him lab sessions) and, more generally, to limit constraint checks to the proposed assignments (e.g., checking the above lecturer's constraint on the actual lectures the solution assigns him).

As mentioned above, each constraint applies a predicate to a tuple of e-maps. UTP predicates either accept a fixed number of e-maps or are variadic. Their semantics may be indifferent to the ordering of their arguments or not, and some accept parameters. Besides, each predicate may be used indistinctly with course e-maps or resource e-maps (i.e., e-maps pairing course elements or resources), and any n-ary constraint may freely mix the two types (e.g., a constraint booking rooms for sessions involving different classes). Let $F = E \times 2^S$ denote the domain of e-maps, the general form of a constraint is $c((e_1, S_1), \ldots, (e_n, S_n), p_1, \ldots, p_m)$ where c is a predicate of arity n, $(e_1, S_1), \ldots, (e_n, S_n)$ are e-maps ($(e_i, S_i) \in F$ for $i = 1 \ldots n$) and $p_1, \ldots, p_m$ are values for the parameters of c ($m \geq 0$).

The semantics of constraints relies on a join operation between constraint e-maps and solutions. Note first that any solution may be cast as a tuple of e-maps by converting the session-to-resource assignments into resource e-maps and re-encoding the fixed maps binding course elements to their sessions. We say an e-map is *null* if it pairs an entity with an empty set of sessions, and, by extension, a tuple of e-maps is null if it includes a null e-map. Given a solution and an e-map for some entity, we call *joint e-map* the pairing of the entity with the set of sessions on which the solution and the e-map agree, i.e., the set-intersection of the sessions of the e-map and those assigned/bound to the entity in the solution encoding. We say a solution is *inconsistent* with an e-map if their joint e-map is null. The join operation extends to tuples by performing the operation component-wise and a solution is said to be inconsistent with a tuple of e-maps if its is inconsistent with at least one e-map in the tuple.

The evaluation of a solution against a constraint is conditioned by the tuple of e-maps joining those of the solution and the constraint. If the joint tuple is null, the constraint is considered satisfied (i.e., it is deemed irrelevant and discarded). Otherwise, the predicate is evaluated on the joint e-map and the result depends on its built-in semantics. Specifically, the predicate is assessed on the tuple of sets obtained by substituting each set of sessions in the joint e-map either by the set of their assigned start times, or the set of their assigned resources of a given type (rooms, etc). Which type (time or resource type) to pick per e-map is fixed and predicate-specific (e.g., a temporal predicate will substitute any e-map argument by start times). Note that entities play no role in the evaluation once the join and substitution operations are over: each predicate is ultimately evaluated on sets made of start times or sets of resources. Note also that join operations leave course e-maps unchanged unlike resource e-maps. This means constraints applying exclusively to course e-maps are de facto unconditional.

The `UTP` catalog provides predicates to cover the various dimensions of time–tabling problems. Some only address scheduling (i.e., start times), others room allocation, and so on. Table 9 and Table 10 given in Appendix describe the predicates of the catalog and provide their semantics. Syntactically, each rule binds a predicate to a query and denotes the conjunction of constraints obtained by applying the predicate to each tuple of e-maps extracted by the query. A rule has the form $c\langle Q, p_1, \ldots, p_m\rangle$ and is interpreted by the formula

$$\forall (e_1, S_1) \in [\![Q_1]\!], \ldots, (e_n, S_n) \in [\![Q_n]\!] : c((e_1, S_1), \ldots, (e_n, S_n), p_1, \ldots, p_m)$$

where c is a predicate of arity n accepting m parameters ($m \geq 0$), Q is a query sized to extract n sets of e-maps, $[\![Q_i]\!]$ denotes the i-th set of e-maps extracted with Q ($i = 1 \ldots n$), and $p_1, \ldots p_m$ are values for the parameters of c.

3 A Feature Model

This section introduces a feature model for educational timetabling problems based on the `UTP` schema. The model is not meant to be exhaustive, nor stable, but is a first attempt to capture the key variability points (the features) in the family of instances that can be expressed with the schema. Some features are plain flags characterizing the compliance of an instance to the schema (e.g., whether courses are hierarchically structured or not) while others are logical assertions on instance data (e.g., whether the number of weeks is set to 1 or not). In either case, each feature may be checked against any instance and, in turn, instances classified into different classes based on the features they satisfy.

The feature model hence decomposes the space of `UTP` problems which serves different purposes. One is to quickly assess whether the schema is applicable to a particular setting. Another is to provide a straightforward characterization of problem classes, similarly to the way 3-field notation is used in other scheduling domains [27,3,1].The aim is also to facilitate the comparison of `UTP` with competing schemas, possibly paving the way for formal reductions between problems and conversions between schemas. Lastly, the feature model can guide the configuration of efficient computational models by using features to reformulate or optimize built-in constraints and predicate implementations.

We first recall the basic notations and definitions commonly used in feature modeling languages [31,19,44]. A feature model is a tree-like structure connecting features and factoring in different feature configurations. A configuration is a subset of features selected from the model. The configuration process is subject to constraints that primarily capture dependencies that exist between a feature and its children (a.k.a., sub-features). These fall into 4 categories: *mandatory sub-feature* (it must be selected if the parent is) labeled by $^\bullet$, *optional sub-feature* (it may be selected if the parent is) labeled by $^\circ$, *or-feature* (at least one of the sub-features must be selected) labeled by $+$, and *xor-feature* (exactly one sub-feature must be selected) labeled by 1. Finer-grained cardinality constraints may apply as well as cross-tree constraints modeling dependencies or incompatibilities between features that sit in different branches.

Table 2 details our feature model. The feature-tree (rotated anticlockwise by 90°) has 3 levels: the root node (not shown), its sub-features and their labels shown respectively on the 2nd and 1st columns and their variants shown on the next 2 columns. For instance, selecting feature hosting in a configuration requires selecting at least one of no-room, single-room or multi-room. The last column provides the formal or informal characterization of each leaf feature. The sub-features of the root characterize core structural elements (course and time structure), orthogonal decision layers (scheduling, room allocation, etc.), and cross-cutting concerns (session planning, resource availability, etc.). The latter is tagged optional and so are hosting and teaching as these decision layers may be out of scope in an instance. We explain next the variants of these sub-features.

course-hierarchy applies to instances whose course elements are nested hierarchically. event applies when events unrelated to courses (e.g., staff meetings) must be scheduled too. The next 3 features characterize the sparsity and scope of the time horizon. full-period indicates if it is built on consecutive calendar weeks and full-week if a weekday is missing. single-week checks whether the instance is restricted to a single week which is typical of timetabling practices in high schools. The next 3 characterize the temporal structure imposed on sessions from "time grids" in high-schools to free-flow timetables for higher grade curricula. no-overlap holds true if sessions can never overlap if they start at different times, same-duration if all sessions have the same duration, and modular if every session length, break time included, breaks down to a unit session length (e.g., some sessions are 1h long and any other session is measured in hours).

The next features characterize room utilization. no-room, single-room, multi-room, hold true if the instance includes a session that demands no room, a single room or more than 1 room, respectively. Similar features are introduced for the demand on teachers. all-exclusive, none-exclusive, some-exclusive, indicate if the instance includes only room-exclusive sessions, only inclusive sessions or a mix, respectively. room-capacity, service, and sectioning apply if resp. room capacity, teaching service and student sectioning are in scope. As for teaching, session-overlap indicates if teachers are cast as disjunctive resources (the counterpart is introduced for students). Lastly, the sub-features of crosscutting capture cross-cutting concerns and we simply list examples of constraints taken from the UTP catalog to convey the meaning.

●	courses	○	course-hierarchy	*courses are decomposed hierarchically into sessions*
			event	*events unrelated to courses must be scheduled*
●	timing	○	full-period	*weeks are consecutive calendar weeks*
			full-week	$d = 7$
			single-week	$w = 1$
●	scheduling	○	no-overlap	$\forall s_i, s_j \in S, s_i \neq s_j, \forall h_i \in d_{s_i}^{S,H}, \forall h_j \in d_{s_j}^{S,H}$ $h_i < h_j \wedge h_i \div m = h_j \div m : \ h_i + length_{s_i}^{S} \leq h_j$
			same-duration	$\forall s_i, s_i \in S, length_{s_i}^{S} = length_{s_i}^{S}$
			modular	Let $A = \{h_j - h_i \mid s_i, s_j \in S, h_i \in d_{s_i}^{S,H}, h_j \in d_{s_j}^{S,H},$ $s_i \neq s_j, \ h_i < h_j\} : gcd(A) = \min(A) \wedge gcd(A) > 1$
○	hosting	+	no-room	$\exists p \in P, \ min\_rooms_p^P = max\_rooms_p^P = 0$
			single-room	$\exists p \in P, \ min\_rooms_p^P = max\_rooms_p^P = 1$
			multi-room	$\exists p \in P, \ min\_rooms_p^P \geq 1 \wedge max\_rooms_p^P > 1$
		○	room-capacity	$\forall r \in R, \ capacity_r^R \neq \emptyset$
		1	all-exclusive	$S^{(e)} = S$
			none-exclusive	$S^{(i)} = S$ *(Not compatible with “multi-room”)*
			some-exclusive	$S^{(e)} \neq \emptyset \wedge S^{(i)} \neq \emptyset$
○	teaching	+	no-teacher	$\exists p \in P, \ min\_lecturer_p^P = max\_lecturer_p^P = 0$
			single-teacher	$\exists p \in P, \ min\_lecturer_p^P = max\_lecturer_p^P = 1$
			multi-teacher	$\exists p \in P, \ min\_lecturer_p^P \geq 1 \wedge max\_lecturer_p^P > 1$
		○	session-overlap	$\forall t \in T, \forall s_i, s_j \in d_t^{T,S}, s_i + length_{s_i}^{S} \leq s_j \vee s_i \geq s_j + length_{s_j}^{S}$
			service	*service constraints apply to teachers*
●	attending	○	session-overlap	$\forall g \in G, \forall s_i, s_j \in d_g^{G,S}, s_i + length_{s_i}^{S} \leq s_j \vee$ $s_i \geq s_j + length_{s_j}^{S}$
			sectioning	*student groups must be fixed and pre-assigned to classes*
○	crosscutting	○	calendar	*allowed_slots, forbidden_slots, allowed_grids, ...*
			regularity	*periodic, allowed_grids,same_rooms, different_teachers, ...*
			orchestration	*same_start, different_day, sequenced, no_overlap*
			workload	*compactness, gap, ...*
			logistics	*same_rooms, adjacent_rooms, different_teachers, ...*
			resourcing	*allowed_rooms, required_teachers, ...*

Table 2: A feature model for UTP.

4 Related Work

The design of timetables is a widely studied problem. Given the multitude of situations encountered, simpler, specialized variants of the general problem have been created in order to produce solutions within an acceptable time frame. The best-known variants include ETT (Exam Timetabling) [11,25] which focuses on exams, PE-TT (Post-Enrolment-based Timetabling) [43,51] in which students register for the courses they wish to take, CB-TT (Curriculum-Based Timetabling) [39,5,32], in which students enroll for a curriculum that includes all the courses they have to take, TAP (Tutor Allocation

Problem) [15], which manages the allocation of teachers after the course slots have been set, and `HTT` (Highschool Timetabling) [33,22], which deals with timetables for high schools.

A timetable design problem is broader than simply scheduling lessons. It depends, for example, on student sectioning [20,6] which consists in dividing students into different groups. But it can also be the starting point for other problems such as `BACP` [50,18] which seeks to balance teaching periods. Given the difficulty of finding a solution, these ancillary problems are often solved beforehand. Simplification assumptions and resource management differ from problem to problem. Table 3 uses the feature model to compare the scope of the different problems, highlighting the common features and differences.

Although widely studied, the problem of timetable design is often dealt with on an ad-hoc basis. It is a crucial problem in the management of certain institutions which seek above all to produce a solution to their specific problem. This explains the heterogeneity of approaches, making it difficult to evaluate and compare work in the field.

Features	Problems	ETT	CB-TT	PE-TT	HTT	TAP
courses	course-hierarchy		✓			
	event		✓		✓	✓
timing	full-period				✓	
	full-week	✓	✓	✓		
	single-week	✓			✓	
scheduling	no-overlap	✓	✓	✓	✓	
	same-duration	✓	✓	✓	✓	
	modular	✓	✓	✓	✓	
hosting	no-room					NA
	single-room	✓	✓	✓	✓	NA
	multi-room	✓	✓			NA
	room-capacity	✓	✓	✓		NA
	none-exclusive	✓				NA
	all-exclusive	✓	✓	✓	✓	NA
	some-exclusive	✓	✓	✓	✓	NA
teaching	no-teacher		✓	✓		
	single-teacher	✓	✓	✓	✓	✓
	multi-teacher	✓	✓			✓
	session-overlap		✓	✓	✓	✓
	service		✓	✓		
attending	session-overlap	✓	✓		✓	✓
	sectioning		✓		✓	✓
crosscutting	calendar	✓	✓		✓	✓
	regularity	✓	✓	✓		
	orchestration		✓	✓	✓	
	workload	✓	✓	✓	✓	
	logistics	✓				✓
	resourcing		✓	✓	✓	

Table 3: Problem features: a comparison.

The emergence of competitions such as ITC (International Timetabling Competition) has led to the creation of standardized formats, making it easier to compare approaches. ITC-2007, one of the most studied schemas, provides a simplified representation of ETT, PE-TT, and CB-TT. In this schema, the aim is to assign one room and one teacher to each session (single-room,single-teacher). The description of academic courses is carried in CB-TT by the curricula which group the courses together, and in PE-TT by the students (session-overlap). The teachers service is assumed to have already been resolved upstream. A teacher assigned to a course does all the sessions of a course, and sessions are otherwise exclusive (session-overlap). Time is expressed in terms of relative slots, i.e., there is a standard duration of one lesson between 2 slots (no-overlap). Class sessions also all have the same duration (same-duration) and daily slots are repeated in the same pattern every day (synchronous).

The XHSTT-2014 [45,23,21] schema, based on the ITC schema, focuses mainly on modeling timetables for secondary schools. Ancillary problems are solved beforehand: generation of groups, breakdown of rooms, teacher services. In addition to the usual resources (rooms, teacher, students, etc.), it is possible to represent other types of resource (e.g. equipment, vehicles, etc.). However, it is possible to leave out a set of resources on which to make a choice of allocations when solving (single-room,single-teacher). A pre-fit is carried out upstream of the schema to reduce the set of rooms to those authorized according to the size of the groups of students (room-capacity, group). The schema generally contains a single time grid, but there's nothing to stop having several. With this schema, the objective of the solver is to build a typical week (single-week,periodicity). The model proposes a catalog of constraints: hard constraints are interpreted as core constraints, while soft constraints have a violation score to minimize (session-distribution). Constraints can be imposed on resources (resource-distribution).

The ITC-2019 [41,38,30] model focuses specifically on university timetables, more specifically anglo-saxon universities. The ITC-2019 schema addresses scheduling as a combinatorial optimization problem, with a cost function that takes into account 4 criteria. The criteria concern the choice of time slots for sessions, rooms for sessions, violations of soft constraints and the overlap of sessions per student (session overlap). This model takes into account a time horizon of several weeks (full-period,full-week. Timetables are defined as the repetition over a set of weeks (multi-week) of one or more sessions of the same duration starting on specific days of the week at the same predefined time (periodicity). Each room has a penalty score for a session. This has an impact on the choice of room (single-room, exclusive-room). The choice has been made not to represent teachers, nor groups of students. A problem expressed in this model comprises a constraint catalog made up of flexible constraints with a penalty score. The catalog of constraints is used to ensure quality and to express the different needs of the timetable (session-distribution, availability).

The UTP schema [9] has been designed to represent problems in which students enrol on courses. As with the other schemas, simplifications are made. For example, it is assumed that students are divided up into groups beforehand, just like the teachers (the allocation of a teacher to a group and a session is done during the design process). It allows problems to be represented over a modular time horizon and clearly identifies teachers. It also allows resources (rooms, teachers, etc.) to be treated disjunctively or

Features	Schemas	ITC-2007	ITC-2019	XHSTT-14	UTP
courses	course-hierarchy		✓		✓
	event	✓		✓	✓
timing	full-period	✓	✓		✓
	full-week	✓	✓	✓	✓
	single-week	✓		✓	
scheduling	same-duration	✓	✓	✓	✓
	no-overlap	✓	✓	✓	✓
	modular	✓	✓	✓	✓
hosting	no-room			✓	✓
	single-room	✓	✓	✓	✓
	multi-room				✓
	room-capacity	✓	✓		✓
	all-exclusive	✓	✓	✓	✓
	none-exclusive				✓
	some-exclusive				✓
teaching	no-teacher		NA	✓	✓
	single-teacher	✓	NA	✓	✓
	multi-teacher		NA		✓
	session-overlap		NA	✓	✓
	service		NA		✓
attending	session-overlap		✓	✓	✓
	sectioning			✓	✓
crosscutting	calendar	✓	✓	✓	✓
	regularity	✓	✓	✓	✓
	orchestration	✓	✓	✓	✓
	workload		✓		
	logistics				✓
	resourcing		✓	✓	✓

Table 4: Schema features.

cumulatively according to need. A few changes have been made since [9]. In [9], the problem of groups sectioning is dealt in conjunction with that of designing the timetable. However, a timetable is often designed on the basis of provisional enrolments, as the definitive enrolments are not yet closed. It is therefore not possible to set up the actual groups at such an early stage. It is interesting to be able to dissociate these two problems and, as with the other schemas, sectioning is considered to have been resolved upstream. The UTP schema takes as input the list of groups formed. In [9], the time grid is identical whatever the week. In the current version, this can be adapted for a particular day or set of days in the entire time horizon.

Most of the schemas presented above stem from a desire to abstract and generalize a real variant of the problem. Thus, certain assumptions and simplifications are made, limiting the expressiveness of the schema, in particular to express other variants orthogonal to the initial assumptions. By analyzing Table 4, we can cite 3 cases where these simplifying assumptions prevent the representation of other variants: the management of the time horizon, the management of teacher services and the management of resources. As far as time horizon management is concerned, only ITC-2019 and UTP can represent a problem with a time horizon longer than a week. Managing the timetable on a weekly basis is incompatible with institutions where each week is different. With the exception of UTP, no schema takes into account the representation of a teacher's service. They consider that teachers are assigned upstream and cannot be exchanged. However, when several groups follow the same course and several teachers are involved, this removes flexibility and prevents certain solutions that could be of high quality from being achieved. Finally, the management of resources also differs from one schema to another. XHSTT and UTP allow teachers to be represented as such, whereas the ITC models do not explicitly include them. In addition, whether for rooms or teachers, the various schemas, apart from UTP, do not allow the resource to be shared over several sessions. For example, it is not possible to represent a problem where one teacher supervises several practical sessions. Nor is it possible to represent a problem in which a session must be hosted in several rooms (adjacent or not). Only UTP can represent problems in which the resources are disjunctive or cumulative.

Competitions are regularly organized [29] and provide an opportunity to make available a set of real or fictitious instances, enabling any new algorithm to be compared with existing approaches. Whether for simulation or comparison purposes, it is useful to have a means of generating new instances. Only the ITC-2007 schema has an instance generator. Developed in 2008, this generator was improved in 2010 and again in 2022 to produce more realistic instances and better cover the range of possible configurations (available on [2]).

5 Instance Generator and Experiments

This section introduces a generator of pseudo-random UTP instances and reports on experiments carried out with three models, namely, CP, ASP and MIP. The objective is to assess the scalability of the models and their applicability to real-life instances. The complete list of instances and the models may be found in Appendix.

5.1 Instance Generator

To generate a UTP instance involves generating a course structure, groups of students, teacher services, and rules. All those generators can be configured thanks to XML files to select the features we want our instance to fit in.

In our generator, we define curricula associated with faculty departments. Curricula enable us to associate a set of courses with a set of students and a set of teachers. Each department is associated with a set of courses, teachers and specific rooms (*i.e.* rooms that can be used only by courses of the department). The courses are divided into several parts. The number of parts usually varies between 1 (only lectures) and 4 (lectures, tutorials, practices and evaluations).

Student sectioning is the problem of assigning students to groups. The UTP schema takes groups, rather than individual students, necessitating that the generator supply groups. The input of the generator is the number of students enrolled in a curriculum. A CSP model is used by the generator to create groups. The different sizes of groups

name	$\lvert R\rvert$	$\lvert T\rvert$	$\lvert U\rvert$	$\lvert S\rvert$	#ru	CP		ASP		CASP		MIP	
						BT	ST	BT	ST	BT	ST	BT	ST
gi8201	82	4	33	118	23	0.86	1.86	15.71	0.08	0.30	0.23	0.17	0.26
gi5301	99	96	55	139	33	0.85	483.59	68.70	0.36	0.40	0.12	0.22	0.49
gi4389	65	6	174	662	33	1.47	4.62	693.40	210.58	14.91	4.93	7.66	10.23
gi5567	94	94	1770	6180	562	4.75	381.78			122.89	3548.58		
gi2767	92	21	501	2003	78	2.02	30.48			128.21	57.11	130.51	7995.32
real	117	183	768	2625	520	2.21	8.71			219.97	411.79	-	-

Table 5: Selected list of instances. $|R|$ is the number of rooms, $|T|$ the number of teachers, $|U|$ the number of students, $|S|$ the number of sessions, #ru the number of rules; BT is the building time (s) and ST the solving time (s).

(lecture, tutorial, practice) should be given. It is possible to change the size of a specific group for a curriculum (e.g., a specific curriculum with group sizes different from the standard ones). The sectioning CSP can create groups with a fixed size, or create courses with a limited number of groups, to fix the total number of hours. Teaching service is a problem where we know how many hours a teacher has to do, how many hours each part of a class lasts, and we want to assign each teacher with a number of classes in each part. This will give us all the course parts a teacher has to teach. Note that we just know how many classes a teacher is assigned to, not to which class, which is another problem known as the tutor allocation problem. The generator uses a CSP model to tackle this problem.

There are various rules, often used together with some being more common. We defined three rule packs: 1) light: some classes have `same_rooms`, `same_teachers`, `periodic`, and a `sequenced` rule between two parts of the course; 2) medium: all classes have `same_rooms` and `same_teachers`, with some also having `periodic` and `sequenced`; 3) heavy: like medium, but with additional `same_teachers` and `same_start` rules for classes in the same part.

5.2 Instances and Results

We carried out experiments on pseudo-random instances built with our generator and on a real-life instance from our Computer Sciences department. The pseudo-random instances are listed in Appendix 6 and may be downloaded from [4]. A selected subset is given in Table 5.

The generated instances were built by varying the number of rooms, groups and sessions. All are single-room and single-teacher and uses the "medium" rules pack, meaning that all classes have `same_rooms` and `same_teachers`, and some have `periodic` and a `sequenced` constraint between 2 different parts of the same course. The real-life instance consists of the 3 years of bachelor and the 2 years of master in Computer Sciences at the University of Angers in 2023. The instance is reduced only to courses that occur at the first time period of each curriculum.

The experiments were performed on a computer with a processor Intel-Xeon E7-4850 v4, 2.1 GHz, 40MB of cache. The CP solver is Choco-solver [49] 4.10.12. The ASP solver is Clingo [24] 5.6.2. The CASP solver is Clingcon [46] 5.2.0. The MIP solver is Gurobi Optimizer [28] 10.0.3. The building (BT) and solving (ST) times can be found in Appendix 6. Table 5 shows selected instances: an instance where all solvers performed well (gi8201) and the worst-case instances in run-time for CP (gi5301), ASP (gi4389), CASP (gi5567) and MIP (gi2767). Those instances show the limits of the solvers. In particular, the more sessions there are, the more difficult it is for ASP and MIP to solve instances to completion. Some instances could not be solved by ASP as it may need more than 80GB when MIP instances use up to 15GB, and CP and CASP only need 8GB. Some instances could not be solved by MIP due to time outs with a time limit set to 5 hours.

6 Conclusion

In this article, we focused on a class of timetabling problems (UTP), proposing a framework that can adapt to different types of institutions, whether they operate like high schools or universities, and to account for regular classes, exams, meetings, or special events. Our current work addresses several aspects. Firstly, we aim to experiment on larger real-world instances and are developing a set of software applications for this purpose. We are also working on incorporating soft constraints and priorities to propose a solution in cases where there is no solution that satisfies all expressed constraints. Finally, we are working on timetable revisions to accommodate unforeseen events such as teacher absences or room unavailability.

Appendix A – Core Computational Model

UTP instances are cast as hard constraint satisfaction problems. We present here a formal specification of the constraint-based model for the entity schema. This core model only formulates the built-in constraints that apply to any instance, leaving out the constraints generated from instance-specific rules.

Table 6 lists the core decision variables of the model and includes auxiliary variables for notational convenience. All, except temporal variables, are cast as set variables.

$x_s^{S,H} \in H$	the start time assigned to session s
$x_s^{S,R} \subseteq R$	the set of rooms assigned to session s
$x_s^{S,T} \subseteq T$	the set of lecturers assigned to session s
$x_s^{S,W} \in W$	(auxiliary variable) the week of the start time assigned to session s
$x_s^{S,D} \in D$	(auxiliary variable) the weekday of the start time assigned to session s
$x_s^{S,M} \in M$	(auxiliary variable) the daily slot of the start time assigned to session s

Table 6: Core and auxiliary decision variables.

Table 7 formulates the core constraints of the model. Note that some statements reify primitive constraints (e.g., set memberhsip) as implicit pseudo-boolean variables.

Constraint (1) establishes the relationship between the start time of a session and its start points on the 3 time scales. (2) restricts the start time of a session to the allowed start points. Constraint (3) ensures that every session starts and ends on the same day. Constraint (4) sequences the sessions of a class based on ranks. (5) models the sets of possible and required rooms for a session as set inclusion constraints. (6) is the counterpart for teachers. (7) and (8) set the bounds on the number of rooms and teachers per session. (9) models the service constraint for each teacher measured in number of sessions. Constraints (10) and (11) model the cumulative capacity of rooms and address the 3 hosting scenarios discussed in Section 2.1 (multi-room exclusive, single-room exclusive and single-room inclusive sessions). (10) ensures any (single- or multi-room) exclusive session allocated to a room virtually fulfills the room capacity and hence has exclusive use of it. Otherwise, the total headcount of the inclusive session(s) occupying the room at any time must not exceed its capacity. Constraint (11) models the capacity demand of each session, be it single- or multi-rooms, and ensures its allocated room(s) meet the demand.

$\forall s \in S$	$x_s^{S,H} = x_s^{S,M} + m * (x_s^{S,D} - 1) + m * d * (x_s^{S,W} - 1)$	(1)
$\forall s \in S$	$x_s^{S,H} \in d_s^{S,H}$	(2)
$\forall s \in S$	$x_s^{S,D} + (x_s^{S,W} - 1) * d = (x_s^{S,H} + length_s^S) \div m$	(3)
$\forall c \in K,\ \forall s_1, s_2 \in d_c^{K,S}$	$rank_{s_1}^S < rank_{s_2}^S \rightarrow (x_{s_1}^{S,H} + length_{s_1}^S) \leq x_{s_2}^{S,H}$	(4)
$\forall s \in S$	$\underline{d}_s^{S,R} \subseteq x_s^{S,R} \subseteq \overline{d}_s^{S,R}$	(5)
$\forall s \in S$	$\underline{d}_s^{S,T} \subseteq x_s^{S,T} \subseteq \overline{d}_s^{S,T}$	(6)
$\forall p \in P, \forall s \in d_p^{P,S}$	$min\_rooms_p^P \leq \lvert x_s^{S,R} \rvert \leq max\_rooms_p^P$	(7)
$\forall p \in P, \forall s \in d_p^{P,S}$	$min\_lecturer_p^P \leq \lvert x_s^{S,T} \rvert \leq max\_lecturer_p^P$	(8)
$\forall t \in T,\ \forall p \in P$	$service_{l,p}^{T \times P} = \sum_{s \in d_p^{P,S}} (l \in x_s^{S,T})$	(9)
$\forall r \in R, \forall h \in H$	$capacity_r^R \geq \sum_{s \in S^{(i)}} (r \in x_s^{S,R}) * (h = x_s^{S,H}) * (size_{d_s^{S,K}}^K) + \sum_{s \in S^{(e)}} (r \in x_s^{S,R}) * (h = x_s^{S,H}) * (capacity_r^R)$	(10)
$\forall s \in S, \forall k \in d_s^{S,K}$	$d_k^{K,R} \neq \emptyset \rightarrow size_k^K \leq \sum_{r \in x_s^{S,R}} capacity_r^R$	(11)

Table 7: Core constraints.

Appendix B – Rules Syntax and Constraint Predicate Catalog

Table 8 provides the syntax of the rules language: e-maps, constraints, queries and rules.

(e_i, S_i)	e-map mapping entity e_i to the set of sessions S_i
$F = E \times 2^S$	the domain of e-maps
$c((e_1, S_1), .., (e_n, S_n), p_1, .., p_m)$	constraint of predicate c, arity n, parameters $p_1, .., p_m$ and e-map arguments $(e_1, S_1), .., (e_n, S_n) \in F^n$
$O = 1.. \max\limits_{s \in S} rank_s^S$	the range of session ranks
$\mathcal{L}^* = \mathcal{L} \cup \{E\} \cup \{\{e\} \mid e \in E\}$	the set of labels
$Q = \cup_{n \geq 1} (\mathcal{E} \times \mathcal{L}^* \times 2^O)^n$	the language of queries
$c\langle Q, p_1, \ldots, p_m \rangle$	rule of predicate c, query Q and parameters $p_1, \ldots p_m$ - c predicate of arity n and number of parameters m - Q query sized to extract n sets of e-maps - $p_1, \ldots p_m$ values for the parameters of c

Table 8: Predicates, constraints, queries and rules.

Table 9 lists and describes the predicates of the catalog.

Predicate	**Description**
adjacent_rooms	Sessions must be adjacent in the given room(s)
allowed_grids	Sessions may only start in the given time grid(s)
allowed_rooms	Sessions may only be hosted in the given room(s)
allowed_slots	Sessions may only run in the given time slots
allowed_teachers	Sessions may only be taught by the given teacher(s)
assign_rooms	Sessions are hosted in the given room(s)
assign_start	Sessions start at the given time
assign_teachers	Sessions are taught by the given teacher(s)
compactness	The sessions makespan is bounded
different_daily_start	Sessions start on different daily slots
different_day	Sessions start on different days
different_rooms	Sessions are hosted in different rooms
different_starts	Sessions start at different times
different_teachers	Sessions are taught by different teachers
different_week	Sessions start on different week
different_weekday	Sessions start on different weekday
different_weekly_start	Sessions start on different weekly time points
forbidden_rooms	Sessions cannot be hosted in the given room(s)
forbidden_slots	Sessions cannot run in the given time slots
forbidden_teachers	Sessions cannot be taught by the given teacher(s)
gap	Gaps between sessions are bounded
no_overlap	Sessions in the given set cannot overlap
pairwise_no_overlap	Sessions cannot overlap if in different sets
periodic	Sessions are periodic
required_rooms	Sessions must be hosted in the given room(s)
required_teachers	Sessions must be taught by the given teacher(s)
same_daily_start	Sessions start on the same daily slot
same_day	Sessions start on the same day
same_rooms	Sessions are hosted in the same room(s)
same_start	Sessions start at the same time
same_teachers	Sessions are taught by the same teacher(s)
same_weekday	Sessions start on the same weekday
same_weekly_start	Sessions start on the same weekly time point
same_week	Sessions start on the same week
sequenced	Sessions run sequentially
sessions_workload	The number of sessions per time frame is bounded
times_workload	The total duration of sessions per time frame is bounded

Table 9: Catalog of UTP constraint predicates.

Table 10 provides the semantics of each predicate once the scope is restricted to a tuple of sets of sessions (obtained after joining a solution and a constraint built with the predicate). Given a n-ary predicate c accepting m parameters ($m \geq 0$) and given a n-uple $(S'_1, \ldots, S'_n) \in S^n$, we provide the semantics for $c(S'_1, \ldots, S'_n, p_1, \ldots, p_m)$ which denotes the evaluation of the predicate on the tuple.

Predicate / **Semantics**	**Parameters**	
adjacent_rooms(S',K$_1$,...,K$_n$)	$K_i \subseteq R, i : 1..n$	
$\forall s_1, s_2 \in S', x_{s_1}^{S,H} = x_{s_2}^{S,H}\ \forall i \in \{1..n\} : C_i = \{x_s^{S,R} \mid s \in S', x_s^{S,R} \in K_i\} \wedge$ $\forall r, r' \in C_i : \exists path(r, r') \wedge \lvert\{C_i \mid C_i \neq \emptyset, i = 1..n\}\rvert \leq \sigma$		(1)
allowed_grids(S',G)	$G \in (W' \times D' \times M')^n, n \in \mathbb{N}^*,$ $S' \subseteq S, W' \subseteq W, D' \subseteq D, M' \subseteq M$	
$\forall s \in S', \exists k \in \{1..n\} : x_s^{S,W} \in W'_k \wedge x_s^{S,D} \in D'_k \wedge x_s^{S,M} \in M'_k$		(2)
allowed_rooms(S',R')	$S' \subseteq S, R' \subseteq R$	
$\forall s \in S', x_s^{S,R} \subseteq R'$		(3)
allowed_slots(S',H')	$H' \subseteq H,$	
$\forall s \in S'[x_s^{S,H}, x_s^{S,H} + length_s^S] \subseteq H'$		(4)
allowed_teachers((e,T'),R')	$S' \subseteq S, T' \subseteq T$	
$\forall s \in S', x_s^{S,T} \subseteq T'$		(5)
assign_rooms(S',R')	$R' \subseteq R$	
$\forall s \in S', x_s^{S,R} = R'$		(6)
assign_start(S',H')	$h \in H$	
$\forall s \in S', x_s^{S,H} = h$		(7)
assign_teachers S',T')	$T' \subseteq T$	
$\forall s \in S', x_s^{S,T} = T'$		(8)
compactness(S',σ)	$\sigma \in 0..\lvert H'\rvert$	
$\forall d \in D : \exists S'' = \{s \in S' : x_s^{S,D}\} \wedge$ $((\max\limits_{s \in S''}(x_s^{S,H} + length_s^S) - \min\limits_{s \in S'}(x_s^{S,H})) - \sum_{s \in S''} length_s^S))/(\lvert S''\rvert - 1) \leq \sigma$		(9)
different_daily_start(S')		
$\forall s_1, s_2 \in S', x_{s_1}^{S,M} \neq x_{s_2}^{S,M}$		(10)
different_day(S')		
$\forall s_1, s_2 \in S', x_{s_1}^{S,D} \neq x_{s_2}^{S,D} \vee x_{s_1}^{S,W} \neq x_{s_2}^{S,W}$		(11)
different_rooms(S')		
$\forall s_1, s_2 \in S', x_{s_1}^{S,R} \cap x_{s_2}^{S,R} = \emptyset$		(12)
different_starts(S')		
$\forall s_1, s_2 \in S', x_{s_1}^{S,H} \neq x_{s_2}^{S,H}$		(13)
different_teachers(S')		
$\forall s_1, s_2 \in S', x_{s_1}^{S,T} \cap x_{s_2}^{S,T} = \emptyset$		(14)
different_week(S')		
$\forall s_1, s_2 \in S', x_{s_1}^{S,W} \neq x_{s_2}^{S,W}$		(15)
different_weekday(S')		

$\forall s_1, s_2 \in S', x_{s_1}^{S,D} \neq x_{s_2}^{S,D}$ (16)

different_weekly_start(S')
$\forall s_1, s_2 \in S', x_{s_1}^{S,M} \neq x_{s_2}^{S,M} \vee x_{s_1}^{S,D} \neq x_{s_2}^{S,D}$ (17)

forbidden_rooms(S',R') $R' \subseteq R$
$\forall s \in S', x_s^{S,R} \subseteq R \setminus R'$ (18)

forbidden_slots(S',H') $H' \subseteq H$
$\forall s \in S' [x_s^{S,H}, x_s^{S,H} + length_s^S [\cap H' = \emptyset$ (19)

forbidden_teachers(S',T') $T' \subseteq T$
$\forall s \in S', x_s^{S,T} \subseteq T \setminus T'$ (20)

min_slot_gap(S',σ_{min}) $\sigma_{min} \in 0..|H|,$
$\exists! \pi : S' \rightarrow [[S']] : x^{S,H}_{\pi^{-1}(i)} < x^{S,H}_{\pi^{-1}(j)} \quad {\scriptstyle (1 \leq i < j \leq |S'|)}$
$\forall i \in 1..|S'| - 1, x^{S,H}_{\pi^{-1}(i+1)} - (x^{S,H}_{\pi^{-1}(i)} + length^S_{\pi^{-1}(i)}) \geq \sigma_{min}$ (21)

min_day_gap(S',σ_{min}) $\sigma_{min} \in 0..|D| * |W|,$
$\exists! \pi : S' \rightarrow [[S']] : x^{S,H}_{\pi^{-1}(i)} < x^{S,H}_{\pi^{-1}(j)} \quad {\scriptstyle (1 \leq i < j \leq |S'|)}$
$\wedge \forall i \in 1..|S'| - 1, x^{S,D}_{\pi^{-1}(i+1)} - (x^{S,D}_{\pi^{-1}(i)}) \geq \sigma_{min}$ (22)

min_week_gap(S',σ_{min}) $\sigma_{min} \in 0..|W|,$
$\exists! \pi : S' \rightarrow [[S']] : x^{S,H}_{\pi^{-1}(i)} < x^{S,H}_{\pi^{-1}(j)} \quad {\scriptstyle (1 \leq i < j \leq |S'|)}$
$\wedge \forall i \in 1..|S'| - 1, x^{S,W}_{\pi^{-1}(i+1)} - (x^{S,W}_{\pi^{-1}(i)}) \geq \sigma_{min}$ (23)

max_slot_gap(S',σ_{max}) $\sigma_{max} \in 0..|H|,$
$s_1 = \min_{s \in S'}(x_s^{S,H}), s_2 = \max_{s \in S'}(x_s^{S,H} + length_s^S), x_{s_2}^{S,H} - (x_{s_1}^{S,H} + length_s^S) \leq \sigma_{max}$ (24)

max_day_gap(S',σ_{max}) $\sigma_{max} \in 0..|D| * |W|,$
$s_1 = \min_{s \in S'}(x_s^{S,H}), s_2 = \max_{s \in S'}(x_s^{S,H}), x_{s_2}^{S,D} - x_{s_1}^{S,D} \leq \sigma_{max}$ (25)

max_week_gap(S',σ_{max}) $\sigma_{max} \in 0..|W|,$
$s_1 = \min_{s \in S'}(x_s^{S,H}), s_2 = \max_{s \in S'}(x_s^{S,H}), x_{s_2}^{S,W} - x_{s_1}^{S,W} \leq \sigma_{max}$ (26)

last_first_slot_gap$(S'_1 \ldots, S'_n, \sigma_{min}, \sigma_{max})$ $\sigma_{min}, \sigma_{max} \in 0..|H|,$
$\forall i \in 1..n-1, \; s_i = \max_{s \in S_i}(x_s^{S,H} + length_s^S), \; s_{i+1} = \min_{s \in S_{i+1}}(x_s^{S,H})$
$\sigma_{min} \leq s_{i+1} - s_i \leq \sigma_{max}$ (27)

last_first_day_gap$(S'_1 \ldots, S'_n, \sigma_{min}, \sigma_{max})$ $\sigma_{min}, \sigma_{max} \in 0..|H|,$
$\forall i \in 1..n-1, \; s_i = \max_{s \in S_i}(x_s^{S,D}), \; s_{i+1} = \min_{s \in S_{i+1}}(x_s^{S,D})$
$\sigma_{min} \leq s_{i+1} - s_i \leq \sigma_{max}$ (28)

last_first_week_gap$(S'_1 \ldots, S'_n, \sigma_{min}, \sigma_{max})$ $\sigma_{min}, \sigma_{max} \in 0..|H|,$
$\forall i \in 1..n-1, \; s_i = \max_{s \in S_i}(x_s^{S,W}), \; s_{i+1} = \min_{s \in S_{i+1}}(x_s^{S,W})$
$\sigma_{min} \leq s_{i+1} - s_i \leq \sigma_{max}$ (29)

first_last_slot_gap$(S'_1 \ldots, S'_n, \sigma_{min}, \sigma_{max})$ $\sigma_{min}, \sigma_{max} \in 0..|H|,$
$\forall i \in 1..n-1, \; s_i = \min_{s \in S_i}(x_s^{S,H} + length_s^S), \; s_{i+1} = \max_{s \in S_{i+1}}(x_s^{S,H})$

$\sigma_{min} \leq s_{i+1} - s_i \leq \sigma_{max}$		(30)
first_last_day_gap($S'_1 \ldots, S'_n, \sigma_{min}, \sigma_{max}$)	$\sigma_{min}, \sigma_{max} \in 0..\lvert H \rvert$,	
$\forall i \in 1..n-1,\ s_i = \min\limits_{s \in S_i}(x_s^{S,D}),\ s_{i+1} = \max\limits_{s \in S_{i+1}}(x_s^{S,D})$		
$\sigma_{min} \leq s_{i+1} - s_i \leq \sigma_{max}$		(31)
first_last_week_gap($S'_1 \ldots, S'_n, \sigma_{min}, \sigma_{max}$)	$\sigma_{min}, \sigma_{max} \in 0..\lvert H \rvert$,	
$\forall i \in 1..n-1,\ s_i = \min\limits_{s \in S_i}(x_s^{S,W}),\ s_{i+1} = \max\limits_{s \in S_{i+1}}(x_s^{S,W})$		
$\sigma_{min} \leq s_{i+1} - s_i \leq \sigma_{max}$		(32)
no_overlap(S')		
$\bigwedge\limits_{\substack{s_1, s_2 \in S' \\ s_1 \neq s_2}} (x_{s_1}^{S,H} + length_{s_1}^S \leq x_{s_2}^{S,H}) \vee (x_{s_2}^{S,H} + length_{s_2}^S \leq x_{s_1}^{S,H})$		(33)
pairwise_no_overlap(S'_1, S'_2)		
$\bigwedge\limits_{\substack{s_1 \in S'_1, s_2 \in S'_2 \\ s_1 \neq s_2}} (x_{s_1}^{S,H} + length_{s_1}^S \leq x_{s_2}^{S,H}) \vee (x_{s_2}^{S,H} + length_{s_2}^S \leq x_{s_1}^{S,H})$		(34)
periodic(S', n)	$n \in \mathbb{N}$	
$\exists \pi : S' \rightarrow 1..\lvert S' \rvert, \forall i \in 1..\lvert S' \rvert - 1, x_{\pi^{-1}(i)}^{S,H} + n = x_{\pi^{-1}(i+1)}^{S,H}$		(35)
required_rooms(S', R')	$R' \subseteq R$	
$\forall s \in S',\ R' \subseteq x_s^{S,R}$		(36)
required_teachers($S', T', \Delta*$)	$T' \subseteq T, \Delta = \{\forall t \in T' \mid \delta_{1,t}, \delta_{2,t}\}$, $\forall t \in T', \forall i \in \{1, 2\},\ \delta_{i,t} \in \mathbb{N}$	
$\forall s \in S',\ (x_s^{S,T} \subseteq T' \wedge \forall t \in T',\ \delta_{1,t} \leq \sum\limits_{s \in S'} (t \in x_s^{S,T}) \leq \delta_{2,t})$		(37)
same_daily_start (S')		
$\forall s_1, s_2 \in S', x_{s_1}^{S,M} = x_{s_2}^{S,M}$		(38)
same_day(S')		
$\forall s_1, s_2 \in S', x_{s_1}^{S,D} = x_{s_2}^{S,D} \wedge x_{s_1}^{S,W} = x_{s_2}^{S,W}$		(39)
same_rooms(S')		
$\forall s_1, s_2 \in S', x_{s_1}^{S,R} = x_{s_2}^{S,R}$		(40)
same_start(S')		
$\forall s_1, s_2 \in S', x_{s_1}^{S,H} = x_{s_2}^{S,H}$		(41)
same_teachers(S')		
$\forall s_1, s_2 \in S', x_{s_1}^{S,T} = x_{s_2}^{S,T}$		(42)
same_week(S')		
$\forall s_1, s_2 \in S', x_{s_1}^{S,W} = x_{s_2}^{S,W}$		(43)
same_weekday(S')		
$\forall s_1, s_2 \in S', x_{s_1}^{S,D} = x_{s_2}^{S,D}$		(44)
same_weekly_start(S')		
$\forall s_1, s_2 \in S', x_{s_1}^{S,M} = x_{s_2}^{S,M} \wedge x_{s_1}^{S,D} = x_{s_2}^{S,D}$		(45)
sequenced($S'_1 \ldots, S'_n$)		
$\forall s_i \in S'_i, \forall s_{i+1} \in S'_{i+1},\ x_{s_{i+1}}^{S,H} \geq x_{s_i}^{S,H} + length_{s_i}^S$		(46)
time_workload(S', w_1, w_2)	$w_1, w_2 \in H \cup \{0\}$	
$\forall d \in D,\ w_1 \leq \sum\limits_{s \in S'} length_s^S \times (x_s^{S,D} = d) \leq w_2$		(47)

session_workload(S',w_1, w_2)	$w_1, w_2 \in 0..\lvert S\rvert$
$\forall d \in D,\ w_1 \leq \lvert\{s \in S' : x_s^{S,D} = d\}\rvert \leq w_2$	(48)

Table 10: Semantics of UTP constraint predicates

Appendix C – Models

The CP model (see table 11 in appendix 6) for UTP is based on decisions variables listed in table 6. For the constraint related to teacher service (requiredTeacher), we using the global cardinality constraint (gcc), which enables element counting. It is also feasible to add counting constraints to better distribute the workload or limit the number of hours per day for a room, aiming to achieve more robust solutions. For conditional constraints, we can apply the reification pattern (checking the feasibility and consumption of potential values and use possible sink state). For constraints of equality, we employ the global constraint all_equal, which ensures for input variables, all have the same value (the implemented propagator is similar to gcc). For the no_overlap constraint and for the main room usage constraint, we employ the global cumulative constraint to ensure that, when sessions utilizing a resource, its maximum capacity is not exceeded, thereby preventing multiple class sessions from overlapping. For difference/disjoint constraints, we used the global n-ary constraints all_different for integer variables and all_disjoint for set variables.

ASP [8] is a form of declarative programming for solving difficult search problems. It operates by defining problems in terms of rules and constraints, then computing the "answer sets" which are collections of assumptions that satisfy rules and constraints without contradiction. The programmer specifies the desired properties of the solution in a high-level language, and the ASP system automatically searches for all solutions that meet these criteria, making it a powerful tool for knowledge representation and reasoning tasks. ASP has been used to address timetabling problems proposed in the ITC-2007 competition [7]. The ASP program we propose for UTP (see appendix 6 for the full program) has been developed with clingo - an ASP solver - and clingcon - a constraint answer set programing solver.

In the state of the art [47,35], mixed-integer linear programming (MIP) models are usually used to solve timetable scheduling problems. In the literature, MIP models are presented in a pseudo-Boolean format, where time is represented in a time-indexed representation. This implies that time is discretized into intervals from 0 to 1 for each time slot. For the UTP problem, where the time horizon is extended, classical representations are not very efficient. Indeed, time-indexed representations exhibit exponential growth. In other scheduling problems such as RCPCSP, representations can be time-indexed, or as in CP in continuous time slots (integer value), or even an event-based approach. Continuous time variables in MIP offer advantages in terms of variable economy, but they require techniques such as big-M to express disjunctions. In some articles [34], it has been demonstrated that event-driven approaches are more effective than continuous or time-indexed approaches for extended time horizons. Here, we present a MIP program for UTP. The program will be used in our experimental study at Section 5(see Appendix 6 for the full program).

C.1 – CP model

$$\forall s \in S \quad x_s^{S,R} \subseteq d_{d_s^{S,P}}^{P,R} \tag{1}$$

$$\forall s \in S \quad x_s^{S,T} \subseteq d_{d_s^{S,P}}^{P,T} \tag{2}$$

$$\forall p \in P,\ \forall s \in d_p^{P,S} \quad min\_rooms_p^P \leq |x_s^{S,R}| \leq max\_rooms_p^P \tag{3}$$

$$\forall p \in P,\ \forall s \in d_p^{P,S} \quad min\_lecturer_p^P \leq |x_s^{S,T}| \leq max\_lecturer_p^P \tag{4}$$

$$\forall p \in P,\ \forall k \in K \quad size_k^K \leq \sum_{r \in R} (r \in x_s^{S,R}) * capacity_r^R \tag{5}$$

$$\forall s \in S: \quad x_s^{S,M} + (x_s^{S,D} - 1) * |M| + (x_s^{S,W} - 1) * |D| * |M| = x_s^{S,H} \tag{6}$$

$$\forall r \in R: \quad cumulative(\{\forall s \in S \mid (r \in x_s^{S,R}) * x_s^{S,H}\}, \{\forall s \in S \mid length_s^S\}, 1) \tag{7}$$

$$\forall p \in P,\ \forall r \in d_p^{P,R}: \quad cumulative(\{\forall s \in d_p^{P,S} \mid (r \in x_s^{S,R}) * x_s^{S,H}\}, \{\forall s \in d_p^{P,S} \mid length_s^S\}, 1) \tag{8}$$

$$\forall p \in P: \quad gcc(\{\forall s \in d_p^{P,S} \mid x_s^{S,R}\}, \{\forall s \in d_p^{P,S} \mid length_s^S\}, 1) \tag{9}$$

$$forbidden\_period(S') = \forall s \in S' \quad (x_s^{S,H} + length_s^S \leq h_1) \vee (x_s^{S,H} \geq h_2) \tag{10}$$

$$same\_weekday(S') = \forall s_1, s_2 \in S', s_1 < s_2 \quad x_{s_1}^{S,D} = x_{s_2}^{S,D} \tag{11}$$

$$same\_rooms(S') = \forall s_1, s_2 \in S', s_1 < s_2 \quad x_{s_1}^{S,R} = x_{s_2}^{S,R} \tag{12}$$

$$different\_rooms(S') = all\_disjoint(\{\forall s \in S' \mid x_s^{S,R}\}) \tag{13}$$

$$different\_starts(S') = all\_different(\{\forall s \in S' \mid x_s^{S,H}\}) \tag{14}$$

$$sequenced(S_1, S_2) = \forall s_1 \in S_1,\ \forall s_2 \in S_2 \quad x_{s_1}^{S,H} + length_{s_1}^S \leq x_{s_2}^{S,H} \tag{15}$$

$$no\_overlap(S') = \text{cumulative}(\{\forall s \in S' \mid x_s^{S,H}\}, \{\forall s \in S' \mid length_s^S\}, 1) \tag{16}$$

Table 11: Constraints and predicates of the CP model.

C.2 – ASP model

The ASP model is split in two parts: the first part lists the facts with a small example (Listing 10.1), the second part is the declaration of rules to solve the given UTP problem (Listing 10.2).

```
weeks(4). days(12). slot_per_day(1440).grid(480,90,7).
courses(1). parts(2).classes(3). sessions(12).
room(Salle-1,40). room(Salle-2,20). room(Salle-3,20).
course(math). teacher(teacher1).teacher(teacher2).
part(math-CM,12,120,5,12,2,1). class(math-CM-1,80).
    course_part(math,math-CM).
part_class(math-CM,math-CM-1). class_sessions(math-CM-1,1..12).
part_teacher(math-CM,(teacher-1;teacher-2)).
part_room(math-CM,(salle-1;salle-2;salle-3)).
part_days(math-CM,1..5). part_weeks(math-CM,1..12).
part_slots(math-CM, (480;570;660;750;840;930)).
part_grids("cours-1-pCM",1,1,6).
group(group-1,20).group(group-2,20).
    class_group(math-CM-1,(group-1;group-2)).
session_duration(S,D) :- session(S),session_part(S,P),
    part_grids(P,_,D,_).
session_group(S,G) :- class_sessions(C,S),class_group(C,G).
session_part(S,P) :- session(S), class_session(C,S),
    part_class(P,C).
session_teacher(S,T) :- session(S), session_part(S,P),
    part_teacher(P,T).
session_room(S,R) :- session(S), session_part(S,P),
    part_room(P,R).
sequenced(3,(6;7)).
periodic(S1,S2,7200) :- session(S1),session(S2),S2 = S1+1 .
disjunctive_room((1..12),R):- room(R,_).
disjunctive_teacher((1..12),T):- teacher(T).
```

Listing 10.1: ASP facts

```
1{assigned(S,SL) : session_part(S,P), part_slot(P,SL)}1 :-
    session(S).
nrPositionRoom(S,1..N) :- session(S), nrRoomMax(S,N).
nrRoomMax(S,N) :- session(S), session_part(S,P),
    part(P,_,_,_,_,N,_), N > 1.
sessionRoomFix(S) :- session(S), not nrRoomMax(S,_).
partRoomFix(P) :- part_sessions(P,S), sessionRoomFix(S).
partRoomMulti(P) :- part_sessions(P,S), nrRoomMax(S,N).
K{assignedrk(S,SL,I) : nrPositionRoom(S,I)}K :- assigned(S,SL),
    nrRoomMax(S,K).
1{assignedr(S,SL,R,K) : session_room(S,R)}1 :-
    assignedrk(S,SL,K).
1{assignedr(S,SL,R,1) : session_room(S,R)}1 :- assigned(S,SL),
    sessionRoomFix(S).
1{assigned(S,SL) : session_part(S,P), part_slot(P,SL)}1 :-
    session(S).
nrPositionTeacher(S,1..N) :- session(S), nrteacherMax(S,N).
nrteacherMax(S,N) :- session(S), session_part(S,P),
    part(P,_,_,_,_,_,N), N > 1.
sessionTeacherFix(S) :- session(S), not nrteacherMax(S,_).
partTeacherFix(P) :- part_sessions(P,S), sessionTeacherFix(S).
partTeacherMulti(P) :- part_sessions(P,S), nrteacherMax(S,N).
K{assignedtk(S,SL,I) : nrPositionTeacher(S,I)}K :-
    assigned(S,SL), nrteacherMax(S,K).
:- assignedt(S,_,T,K2), assignedt(S,_,T,K1), K1 != K2.

:- not {assignedt(S,SL,T,K) : session(S),
    disjunctive_teacher(T,S), nrPositionTeacher(S,K) } 1,
    slots(SL), teacher(T).
:- not {assignedt(S,SL,T,1) : session(S),
    disjunctive_teacher(T,S) } 1, slots(SL), teacher(T).
:- nrRoomMax(S,_), session_class(S,C), class_headcount(C,N), N
    > #sum{V:assignedr(S,_,R,_),room(R,V)}.
:- assignedr(S,SL,R,1), room(R,C1),session_class(S,C),
    class_headcount(C,N), N > C1.
:- not {assigned(S,SL) : session(S),disjunctive_group(S,G)}1,
    group(G,_), slots(SL).
:- not {assignedr(S,SL,R,K) : session(S),
    disjunctive_room(R,S), nrPositionRoom(S,K) } 1, slots(SL),
    room(R,_).
:- not {assignedr(S,SL,R,1) : session(S), disjunctive_room(R,S)
    } 1, slots(SL), room(R,_).
:- assignedr(S,_,R,K2), assignedr(S,_,R,K1), K1 != K2.
:- periodic(S1,S2,N), assigned(S1,SL1), assigned(S2,SL2), SL2
    != SL1+N.
:- sequenced(S1,S2), assigned(S1,SL1), session_part(S1,P),
    part_grids(P,_,N,_), assigned(S2,SL2), SL1+N > SL2.
sequenced(S1,S2) :- part(P,_,_,_,_,_,_), part_class(P,C),
    class_sessions(C,S1), class_sessions(C,S2), S1+1 = S2.
```

```
:- same_slot(S1,S2), assigned(S1,SL1), assigned(S2,SL2), SL1 !=
    SL2.
:- same_teachers(S1,S2), assignedt(S1,_,T1,K),
    assignedt(S2,_,T2,K), T1 != T2.
:- same_rooms(S1,S2), assignedr(S1,_,R1,K),
    assignedr(S2,_,R2,K), R1 != R2.
:- same_rooms(S,S2), nrRoomMax(S,_), not nrRoomMax(S2,_).
:- assign_rooms(S1,R1), assignedr(S1,_,R2,_), R1 != R2.
:- assign_teachers(S1,T1), assignedt(S1,_,T2,_), T1 != T2.
:- serviceTeacher(T,P,N), #count { S,T
    :assignedt(S,_,T,_),part_sessions(P,S)} != N.
```

Listing 10.2: ASP model

C.3 – MIP model

$$\begin{aligned}&\forall p \in P, R' = d_p^{P,R},\\ &\qquad \forall s \in d_p^{P,S},\ \forall r \in R \setminus R',\ x_{s,r}^{S,R} = 0\end{aligned} \tag{1}$$

$$\begin{aligned}&\forall p \in P, T' = d_p^{P,T},\\ &\qquad \forall s \in d_p^{P,S},\ \forall t \in T \setminus T',\ x_{s,t}^{S,T} = 0\end{aligned} \tag{2}$$

$$\begin{aligned}&\forall p \in P,\ \forall s \in d_p^{P,S}\\ &\qquad min\_rooms_p^P \le \sum_{\forall r \in R} x_{s,r}^{S,R} \le max\_rooms_p^P\end{aligned} \tag{3}$$

$$\begin{aligned}&\forall p \in P,\ \forall s \in d_p^{P,S}\\ &\qquad min\_lecturer_p^P \le \sum_{\forall t \in T} x_{s,t}^{S,T} \le max\_lecturer_p^P\end{aligned} \tag{4}$$

$$\begin{aligned}&\forall p \in P,\ \forall k \in K\\ &\qquad size_k^K \le \sum_{r \in R} x_{s,r}^{S,R} * capacity_r^R\end{aligned} \tag{5}$$

$$\begin{aligned}&\forall s \in S:\\ &\qquad x_s^{S,M} + (x_s^{S,D} - 1) * |M| + (x_s^{S,W} - 1) * |D| * |M| = x_s^{S,H}\end{aligned} \tag{6}$$

$$\begin{aligned}&forbidden\_period(S', H') = \forall s \in S',\ h = min(H'),\\ &\qquad s_h \in \{0, 1\}\\ &\qquad (x_s^{S,H} - h) \ge ((length_s^S + M) * s_h - M)\\ &\qquad (h - x_s^{S,H}) \ge ((|H'| + M) * s_h - M)\end{aligned} \tag{7}$$

$$\begin{aligned}&forbidden\_rooms(S', R') = \forall s \in S'\\ &\qquad \forall r \in R',\ x_{s,r}^{S,R} = 0\end{aligned} \tag{8}$$

$$\begin{aligned}&same\_weekday(S') = \forall s_1, s_2 \in S', s_1 < s_2\\ &\qquad x_{s_1}^{S,D} - x_{s_2}^{S,D} = 0\end{aligned} \tag{9}$$

$$\begin{aligned}&same\_start(S') = \forall s_1, s_2 \in S', s_1 < s_2\\ &\qquad (x_{s_1}^{S,H} - x_{s_2}^{S,H}) = 0\end{aligned} \tag{10}$$

$$\begin{aligned}&same\_rooms(S') = \forall s_1, s_2 \in S', s_1 < s_2\\ &\qquad x_{s_1}^{S,R} = x_{s_2}^{S,R}\end{aligned} \tag{11}$$

$$\begin{aligned}&assign\_rooms(S', R') = \forall s \in S',\\ &\qquad \forall r \in R, x_{s,r}^{S,R} = 1\end{aligned} \tag{12}$$

$$\begin{aligned}&different\_rooms(S') = \forall s_1, s_2 \in S',\\ &\qquad \forall r \in R,\ x_{s_1,r}^{S,R} + x_{s_2,r}^{S,R} \le 1\end{aligned} \tag{13}$$

$$\begin{aligned}&periodic(S') = \forall s_1, s_2 \in S', s_1 < s_2,\\ &\qquad (x_{s_1}^{S,H} + n) = x_{s_2}^{S,H}\end{aligned} \tag{14}$$

$$\begin{aligned}&sequenced(S_1, S_2) = \forall s_1 \in S_1,\ s_2 \in S_2\\ &\qquad (x_{s_1}^{S,H} + length_{s_1}^S) \le x_{s_2}^{S,H}\end{aligned} \tag{15}$$

$$\begin{aligned}&required\_teachers(S', LG) = \forall s S',\ \forall t \in T,\ \forall \sigma \in \mathbb{N}\\ &\qquad (\textstyle\sum_{s \in P} x_{s,t}^{S,T}) \le \sigma\end{aligned} \tag{16}$$

$$\begin{aligned}&no\_overlap\_room(S') = \forall s_1, s_2 \in S',\\ &\qquad ord_{s_1,s_2}, ord_{s_2,s_1} \in \{0, 1\}, r_1 r_2 \in \{0, 1\}\\ &\qquad r_1 r_2 \le x_{s_1}^{S,R}, r_1 r_2 \le x_{s_2}^{S,R},\ r_1 r_2 \ge (x_{s_1}^{S,R} + x_{s_2}^{S,R} - 1)\\ &\qquad r_1 r_2 \ge ord_{s_1,s_2}, r_1 r_2 \ge ord_{s_2,s_1}\\ &\qquad (x_{s_1}^{S,H} - x_{s_2}^{S,H}) \ge ((length_{s_2}^S + M) * ord_{s_1,s_2} - M)\\ &\qquad (x_{s_2}^{S,H} - x_{s_1}^{S,H}) \ge ((length_{s_1}^S + M) * ord_{s_2,s_1} - M)\end{aligned} \tag{17}$$

$$\begin{aligned}&no\_overlap\_group(S') = \forall s_1, s_2 \in S',\\ &\qquad ord_{s_1,s_2}, ord_{s_2,s_1} \in \{0, 1\}, ord_{s_1,s_2} + ord_{s_2,s_1} = 1\\ &\qquad (x_{s_1}^{S,H} - x_{s_2}^{S,H}) \ge ((length_{s_2}^S + M) * ord_{s_1,s_2} - M)\\ &\qquad (x_{s_2}^{S,H} - x_{s_1}^{S,H}) \ge ((length_{s_1}^S + M) * ord_{s_2,s_1} - M)\end{aligned} \tag{18}$$

Appendix D – Complete list of instances

name	$\lvert R\rvert$	$\lvert T\rvert$	$\lvert U\rvert$	$\lvert S\rvert$	#ru	CP		ASP		CASP		MIP	
						BT	ST	BT	ST	BT	ST	BT	ST
gi5167	87	4	33	81	19	0.86	0.97	55.41	1.93	0.34	0.06	0.04	0.11
gi678	60	101	30	117	22	0.86	0.89	48.04	2.61	0.60	0.10	0.13	0.21
gi8201	82	4	33	118	23	0.86	1.86	15.71	0.08	0.30	0.23	0.17	0.26
gi8445	83	12	45	110	25	0.93	2.44	54.78	1.39	0.62	0.05	0.21	0.33
gi5301	99	96	55	139	33	0.85	483.59	68.70	0.36	0.40	0.12	0.22	0.49
gi9456	102	15	73	186	27	1.02	2.74	190.18	15.01	2.35	0.28	0.29	0.52
gi42910	84	96	110	163	30	1.03	1.87	146.45	1.25	0.49	0.18	0.16	0.30
gi29910	100	33	49	163	33	1.03	11.14	129.40	4.17	1.10	0.14	0.43	0.83
gi8601	104	97	143	200	16	0.99	0.98	42.49	4.93	0.40	0.17	0.08	0.14
gi6012	125	97	65	175	36	1.06	2.66	296.48	11.96	1.38	0.66	0.25	0.94
gi6101	85	100	48	212	40	0.99	1.47	131.63	8.63	0.78	0.38	0.18	0.36
gi4867	96	21	62	211	30	0.94	2.10	214.09	7.74	2.38	0.23	0.97	1.91
gi1045	108	99	50	175	44	0.93	7.60	84.57	2.49	0.98	0.20	0.26	0.44
gi2123	128	98	61	240	29	1.03	1.87	333.32	36.93	6.69	0.37	1.09	1.37
gi8056	104	4	66	223	37	1.06	5.01	221.33	7.81	3.31	0.57	1.61	2.52
gi5534	87	22	194	189	24	1.07	3.90	71.02	2.00	0.87	0.12	0.73	1.72
gi2501	123	98	151	245	23	1.00	1.50	143.26	28.74	1.53	0.49	0.22	0.38
gi223	102	6	135	219	30	1.01	4.09	87.73	1.52	0.89	0.19	1.88	2.79
gi1378	95	37	68	244	46	1.15	3.98	227.07	4.96	2.03	0.43	3.21	4.64
gi8467	115	24	67	264	46	1.30	3.33	297.07	10.62	4.23	0.57	2.48	3.87
gi7967	114	101	206	192	15	1.16	4.37	406.36	24.56	3.42	0.12	0.78	1.44
gi9123	78	4	66	325	38	1.15	213.82	212.85	17.56	1.51	0.78	1.81	2.81
gi9434	99	4	65	334	38	0.93	1.73	208.71	29.71	1.77	1.22	0.59	1.30
gi2245	110	105	132	313	30	1.08	2.25	134.36	16.68	2.01	0.26	1.16	1.94
gi8478	90	96	138	295	27	1.09	3.63	475.86	19.95	4.69	0.23	4.18	6.00
gi9334	107	14	86	341	30	1.03	2.24	162.71	30.17	4.54	1.43	1.63	2.64
gi745	107	105	91	352	38	1.10	1.95	60.29	7.10	0.99	3.89	0.34	0.54
gi1245	132	5	152	275	30	1.19	4.44	287.89	13.60	3.30	0.81	3.93	5.81
gi27910	97	100	114	369	34	1.07	59.67	131.10	15.43	1.53	1.01	0.98	1.74
gi5501	83	17	123	361	31	1.03	2.20	71.03	3.26	0.81	1.03	1.39	2.62
gi1867	86	39	91	405	47	1.08	5.22	77.95	12.20	1.99	4.64	2.07	2.95
gi4067	84	95	154	363	21	1.17	3.25	118.31	12.65	1.01	0.94	0.39	0.70
gi5856	78	10	114	324	50	1.12	5.15	182.23	6.41	2.02	0.40	5.19	8.20
gi9323	85	8	194	335	39	1.25	3.81	331.39	11.30	3.40	0.44	6.08	9.02
gi8956	102	97	162	419	26	1.25	11.72	527.65	33.91	1.26	1.99	1.79	2.84
gi8023	93	30	150	392	38	1.09	5.48	116.28	15.60	0.84	1.54	0.56	1.20
gi6889	94	98	66	522	55	1.24	3.64			4.33	10.41	5.76	6.45
gi3167	89	95	194	422	27	1.14	3.42	283.10	32.00	3.94	1.41	2.93	3.85
gi878	87	98	97	437	41	1.18	3.01	346.90	72.95	4.74	1.75	2.05	2.69
gi9189	82	27	319	362	52	1.36	7.84	395.76	14.18	3.58	3.39	6.88	10.05
gi1578	127	99	195	461	31	1.14	2.35	319.65	90.44	2.56	2.16	0.80	1.44
gi8678	83	24	261	369	28	1.24	8.08	525.14	38.13	3.69	4.99	10.32	15.92
gi7445	104	7	196	423	58	1.24	5.02	599.88	41.23	12.32	1.21	19.15	28.57
gi3934	104	96	159	623	64	1.32	4.56	613.68	142.05	5.51	2.02	5.84	8.20
gi5378	88	21	158	665	59	1.32	4.68	726.97	37.24	2.96	3.19	3.73	6.07
gi6745	98	98	150	624	65	1.35	7.20	634.56	125.36	13.92	4.53	10.29	14.84
gi4389	65	6	174	662	33	1.47	4.62	693.40	210.58	14.91	4.93	7.66	10.23
gi6278	99	8	192	588	40	1.22	17.32	491.72	183.93	13.94	3.68	6.37	9.01
gi7923	96	96	155	590	45	1.32	6.38	528.75	76.06	3.51	2.98	5.72	8.36
gi7867	118	101	211	617	59	1.51	8.89	1058.61	100.78	22.35	1.43	15.72	22.05
gi30910	104	9	190	523	49	1.22	8.45	472.29	22.97	7.81	1.21	21.71	32.06
gi4323	120	23	170	629	39	1.40	14.42	498.26	117.77	15.58	2.82	11.04	16.85
gi8745	108	105	249	632	34	1.38	8.38	610.86	156.25	20.13	3.22	13.08	20.08
gi2401	122	104	152	696	29	1.41	4.97			6.31	4.81	5.45	7.72
gi2867	102	11	154	783	38	1.46	4.74			6.41	11.81	6.53	10.06
gi4245	67	46	187	707	53	1.34	8.32	386.84	23.77	3.00	3.73	12.49	17.63
gi6134	113	105	152	748	35	1.40	8.56	437.44	109.94	3.17	13.31	4.95	7.36
gi8145	96	100	268	477	42	1.21	36.84	196.38	8.18	3.59	2.05	11.62	18.11
gi4278	95	47	251	811	20	1.30	4.93	645.57	25.87	1.75	8.60	1.08	2.09
gi7156	125	98	486	711	23	1.41	9.81	491.04	206.26	19.09	2.32	13.48	14.97
gi5145	99	96	154	1017	101	1.43	6.75			2.86	36.03	0.07	0.14

(. . .)

name	$\|R\|$	$\|T\|$	$\|U\|$	$\|S\|$	#ru	CP		ASP		CASP		MIP	
						BT	ST	BT	ST	BT	ST	BT	ST
(continued)													
gi1067	113	56	376	716	56	1.39	9.23	255.67	22.65	7.37	4.12	22.45	33.70
gi3367	98	45	539	836	99	1.48	8.44	372.95	16.04	5.09	6.10	22.19	94.02
gi30910	104	95	361	810	78	1.62	13.50	825.29	36.13	6.77	5.40	24.78	29.16
gi1778	112	10	400	880	109	1.66	11.05			46.16	15.64	65.96	107.49
gi7267	89	98	298	1295	30	1.67	10.51			33.20	29.72	35.02	258.28
gi77910	103	97	422	2120	102	2.17	12.58			114.51	94.53	96.28	141.20
gi9912	84	103	562	1956	172	2.08	47.84			10.48	1426.66		
gi2767	92	21	501	2003	78	2.02	30.48			128.21	57.11	130.51	7995.32
gi7034	102	100	454	2268	162	2.07	20.72			10.70	104.37	30.92	52.49
gi9023	144	103	1247	1980	200	2.49	24.85			142.35	41.64	258.11	799.06
gi7723	89	55	432	2745	152	2.50	28.02			79.88	218.58	157.51	405.68
gi5567	94	94	1770	6180	562	4.75	381.78			122.89	3548.58		
gi467	90	61	1982	8886	383	4.61	33.59			92.65	2609.97		
gi2934	106	73	2111	9168	515	5.34	33.87			92.25	2469.69		

Table 12: List of instances.

References

1. Scheduling zoo, http://schedulingzoo.lip6.fr/
2. Dataset Generator ITC-2007 (2022), https://cdlab-artis.dbai.tuwien.ac.at/papers/cb-ctt/
3. RobinX three field (2022), https://robinxval.ugent.be/RobinX/threeField.php
4. UTP dataset (2024), https://ua-usp.github.io/timetabling/instances
5. Abdullah, S., Turabieh, H.: On the use of multi neighbourhood structures within a tabu-based memetic approach to university timetabling problems. Information Sciences **191**, 146–168 (2012). https://doi.org/https://doi.org/10.1016/j.ins.2011.12.018, https://www.sciencedirect.com/science/article/pii/S0020025511006670, data Mining for Software Trustworthiness
6. Amintoosi, M., Haddadnia, J.: Feature selection in a fuzzy student sectioning algorithm. In: 4th International Conference on the Practice and Theory of Automated (PATAT) 2004. pp. 147–160 (08 2004). https://doi.org/10.1007/11593577_9
7. Banbara, M., Inoue, K., Kaufmann, B., Okimoto, T., Schaub, T., Soh, T., Tamura, N., Wanko, P.: teaspoon: solving the curriculum-based course timetabling problems with answer set programming. Ann. Oper. Res. **275**(1), 3–37 (2019). https://doi.org/10.1007/s10479-018-2757-7
8. Baral, C.: Knowledge representation, reasoning and declarative problem solving. Cambridge university press (2003)
9. Barichard, V., Behuet, C., Genest, D., Legeay, M., Lesaint, D.: A Constraint Language For University Timetabling Problems. In: 13th International Conference on the Practice and Theory of Automated (PATAT) 2022. Louvain, Belgium (Aug 2022), https://hal.science/hal-04073981
10. Battistutta, M., Ceschia, S., De Cesco, F., Di Gaspero, L., Schaerf, A., Topan, E.: Local search and constraint programming for a real-world examination timetabling problem. In: Hebrard, E., Musliu, N. (eds.) Integration of Constraint Programming, Artificial Intelligence, and Operations Research. pp. 69–81. Springer International Publishing, Cham (2020)
11. Bellio, R., Ceschia, S., Di Gaspero, L., Schaerf, A.: Two-stage multi-neighborhood simulated annealing for uncapacitated examination timetabling. Computers & Operations Research **132**, 105300 (2021). https://doi.org/https://doi.org/10.1016/j.cor.2021.105300, https://www.sciencedirect.com/science/article/pii/S0305054821000927
12. Bettinelli, A., Cacchiani, V., Roberti, R., toth, P.: An overview of curriculum-based course timetabling. TOP **23**, 313–349 (2015). https://doi.org/https://doi.org/10.1007/s11750-015-0366-z

13. Cambazard, H., Hebrard, E., O'Sullivan, B., Papadopoulos, A.: Local search and constraint programming for the post enrolment-based course timetabling problem. Ann. Oper. Res. **194**(1), 111–135 (2012). https://doi.org/10.1007/s10479-010-0737-7
14. Carter, M.W., Laporte, G., Lee, S.Y.: Examination timetabling: Algorithmic strategies and applications. The Journal of the Operational Research Society **47**(3), 373–383 (1996), http://www.jstor.org/stable/3010580
15. Caselli, G., Delorme, M., Iori, M.: Integer linear programming for the tutor allocation problem: A practical case in a british university. Expert Systems with Applications **187**, 115967 (2022). https://doi.org/https://doi.org/10.1016/j.eswa.2021.115967, https://www.sciencedirect.com/science/article/pii/S095741742101318X
16. Castro, C., Manzano, S.: Variable and Value Ordering When Solving Balanced Academic Curriculum Problems. ARXIV (Nov 2001)
17. Chen, M., Sze, S., Goh, S.L., Sabar, N., Kendall, G.: A Survey of University Course Timetabling Problem: Perspectives, Trends and Opportunities. IEEE Access **PP**, 1–1 (Jul 2021). https://doi.org/10.1109/ACCESS.2021.3100613
18. Chiarandini, M., Di Gaspero, L., Gualandi, S., Schaerf, A.: The balanced academic curriculum problem revisited. Journal of Heuristics **18**(1), 119–148 (Feb 2012). https://doi.org/10.1007/s10732-011-9158-2, https://doi.org/10.1007/s10732-011-9158-2
19. Czarnecki, K., Østerbye, K., Völter, M.: Generative programming. In: Núñez, J.H., Moreira, A.M.D. (eds.) Object-Oriented Technology, ECOOP 2002 Workshops and Posters, Málaga, Spain, June 10-14, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2548, pp. 15–29. Springer (2002). https://doi.org/10.1007/3-540-36208-8_2, https://doi.org/10.1007/3-540-36208-8_2
20. David, S.: Optimal student sectioning on mandatory courses with various sections numbers. Annals of operations research **275**(1), 209–221 (2019)
21. Demirovic, E., Musliu, N.: Modeling high school timetabling as partial weighted maxsat. In: LaSh 2014: The 4th Workshop on Logic and Search (a SAT/ICLP workshop at FLoC 2014), July 18, Vienna, Austria (2014)
22. Demirovic, E., Stuckey, P.J.: Constraint programming for high school timetabling: A scheduling-based model with hot starts. In: van Hoeve, W.J. (ed.) Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 15th International Conference, CPAIOR 2018, Delft, The Netherlands, June 26-29, 2018, Proceedings. Lecture Notes in Computer Science, vol. 10848, pp. 135–152. Springer (2018). https://doi.org/10.1007/978-3-319-93031-2_10, https://doi.org/10.1007/978-3-319-93031-2_10
23. Demirović, E., Musliu, N.: Maxsat-based large neighborhood search for high school timetabling. Computers & Operations Research **78** (02 2017). https://doi.org/10.1016/j.cor.2016.08.004
24. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Multi-shot asp solving with clingo. Theory and Practice of Logic Programming **19**(1), 27–82 (2019). https://doi.org/10.1017/S1471068418000054
25. Gogos, C., Dimitsas, A., Nastos, V., Valouxis, C.: Some insights about the uncapacitated examination timetabling problem. In: 2021 6th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM). pp. 1–7 (2021). https://doi.org/10.1109/SEEDA-CECNSM53056.2021.9566258
26. Goh, S.L., Kendall, G., Sabar, N.R.: Improved local search approaches to solve the post enrolment course timetabling problem. European Journal of Operational Research **261**(1), 17–29 (2017). https://doi.org/https://doi.org/10.1016/j.ejor.2017.01.040, https://www.sciencedirect.com/science/article/pii/S0377221717300759
27. Graham, R.L., Lawler, E.L., Lenstra, J.K., Kan, A.R.: Optimization and approximation in deterministic sequencing and scheduling: a survey. In: Annals of discrete mathematics, vol. 5, pp. 287–326. Elsevier (1979). https://doi.org/10.1016/S0167-5060(08)70356-X

28. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2023), https://www.gurobi.com
29. ITC19: International Timetabling Competition (2019), https://www.itc2019.org/
30. Jawa Bendi, K., Junaidi, H.: simulated annealing approach for university timetable problem. Jurnal Ilmiah Matrik **21** (12 2019). https://doi.org/10.33557/jurnalmatrik.v21i3.723
31. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A.: Feature-oriented domain analysis (foda) feasibility study (01 1990)
32. Kiefer, A., Hartl, R.F., Schnell, A.: Adaptive large neighborhood search for the curriculum-based course timetabling problem. Annals of Operations Research **252**, 255 – 282 (2016), https://api.semanticscholar.org/CorpusID:20405903
33. Kingston, J.H.: Repairing high school timetables with polymorphic ejection chains. Annals of Operations Research **239**, 119–134 (2016)
34. Koné, O., Artigues, C., Lopez, P., Mongeau, M.: Event-based milp models for resource-constrained project scheduling problems. Computers & Operations Research **38**(1), 3–13 (2011)
35. Kristiansen, S., Sørensen, M., Stidsen, T.: Integer programming for the generalized high school timetabling problem. Journal of Scheduling **18** (08 2015). https://doi.org/10.1007/s10951-014-0405-x
36. Lemos, A., Monteiro, P., Lynce, I.: Disruptions in timetables: A case study at universidade de lisboa. Journal of Scheduling (02 2021). https://doi.org/10.1007/s10951-020-00666-3
37. Lewis, R., Paechter, B., Mccollum, B.: Post enrolment based course timetabling: A description of the problem model used for track two of the second international timetabling competition. Cardiff University, Cardiff Business School, Accounting and Finance Section, Cardiff Accounting and Finance Working Papers (01 2007)
38. Lindahl, M., Stidsen, T., Sørensen, M.: Quality recovering of university timetables. European Journal of Operational Research **276**(2), 422 – 435 (2019). https://doi.org/https://doi.org/10.1016/j.ejor.2019.01.026, http://www.sciencedirect.com/science/article/pii/S0377221719300451
39. Lü, Z., Hao, J.K.: Adaptive tabu search for course timetabling. European Journal of Operational Research **200**, 235–244 (01 2010). https://doi.org/10.1016/j.ejor.2008.12.007
40. Mccollum, B., McMullan, P., Paechter, B., Lewis, R., Schaerf, A., Di Gaspero, L., Parkes, A., Qu, R., Burke, E.: Setting the research agenda in automated timetabling: The second international timetabling competition. INFORMS Journal on Computing **22**, 120–130 (02 2010). https://doi.org/10.1287/ijoc.1090.0320
41. Müller, T., Rudová, H., Müllerová, Z.: University course timetabling and International Timetabling Competition 2019. In: Burke, E.K., Di Gaspero, L., McCollum, B., Musliu, N., Özcan, E. (eds.) Proceedings of the 12th International Conference on the Practice and Theory of Automated Timetabling (PATAT-2018). pp. 5–31 (2018)
42. Müller, T., Murray, K.: Comprehensive approach to student sectioning. Annals of Operations Research **181**, 249–269 (Dec 2010). https://doi.org/10.1007/s10479-010-0735-9
43. Nagata, Y.: Random partial neighborhood search for the post-enrollment course timetabling problem. Computers & Operations Research **90**, 84–96 (2018). https://doi.org/https://doi.org/10.1016/j.cor.2017.09.014, https://www.sciencedirect.com/science/article/pii/S0305054817302447
44. Nešić, D., Krüger, J., Stănciulescu, u., Berger, T.: Principles of feature modeling. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. p. 62–73. ESEC/FSE 2019, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3338906.3338974, https://doi.org/10.1145/3338906.3338974
45. Nysret Musliu, E.D.: Solving high school timetabling with satisability modulo theories (08 2014), https://api.semanticscholar.org/CorpusID:14768396

46. Ostrowski, M.: Modern constraint answer set solving. Ph.D. thesis, Universität Potsdam (2018)
47. Árton P. Dorneles, de Araújo, O.C., Buriol, L.S.: A fix-and-optimize heuristic for the high school timetabling problem. Computers & Operations Research **52**, 29–38 (2014). https://doi.org/https://doi.org/10.1016/j.cor.2014.06.023, https://www.sciencedirect.com/science/article/pii/S0305054814001816
48. Post, G., Ahmadi, S., Daskalaki, S., Kingston, J., Kyngas, J., Nurmi, K., Post, G., Ahmadi, S., Daskalaki, S., Kyngas, J., Ranson, D.: An XML format for benchmarks in High School Timetabling. Annals of Operations Research **194**, 385–397 (Apr 2012). https://doi.org/10.1007/s10479-010-0699-9
49. Prud'homme, C., Fages, J.G.: Choco-solver: A java library for constraint programming. Journal of Open Source Software **7**(78), 4708 (2022). https://doi.org/10.21105/joss.04708, https://doi.org/10.21105/joss.04708
50. Rubio, J.M., Palma, W., Rodriguez, N., Soto, R., Crawford, B., Paredes, F., Cabrera, G.: Solving the Balanced Academic Curriculum Problem Using the ACO Metaheuristic. Mathematical Problems in Engineering **2013**, e793671 (Dec 2013). https://doi.org/10.1155/2013/793671, https://www.hindawi.com/journals/mpe/2013/793671/, publisher: Hindawi
51. Say Leng Goh, G.K., Sabar, N.R.: Simulated annealing with improved reheating and learning for the post enrolment course timetabling problem. Journal of the Operational Research Society **70**(6), 873–888 (2019). https://doi.org/10.1080/01605682.2018.1468862, https://doi.org/10.1080/01605682.2018.1468862
52. Schindl, D.: Optimal student sectioning on mandatory courses with various sections numbers. Annals of Operations Research **275** (04 2019). https://doi.org/10.1007/s10479-017-2621-1

The Meaning of Permutation in Reentrant Permutation Flow Shop Problems

Itamar Segal[1], Tal Grinshpoun[1][0000−0002−4106−3169], Hagai Ilani[2][0000−0003−3548−1572], and Elad Shufan[2][0000−0001−7960−5798]

[1] Ariel University, Ariel, Israel
itamar.segal@msmail.ariel.ac.il, talgr@ariel.ac.il
[2] Shamoon College of Engineering, Ashdod, Israel
hagai@sce.ac.il, elads@sce.ac.il

Abstract. In a flow shop, jobs are serially processed on a set of machines, and the machine order is the same for all the jobs. In a permutation flow shop, there is an additional assumption that the order in which jobs enter the machines is the same on each machine. While the meaning of "permutation" is clear for a flow shop, it is more ambiguous for a reentrant flow shop. In a reentrant flow shop, jobs are processed on some machines more than once. Then, there are several ways to understand the meaning of permutation. We indicate that different researchers use the term "permutation" for different assumptions. Our effort is to clear up this ambiguity. This is significant for definition clarity when studying the reentrant scenario. Moreover, the various definitions influence proposed heuristics for solving the problem and, by that, also influence the quality of the resulting solutions. We show this through basic examples as a first step towards more extensive experiments.

Keywords: Permutation Flow shop, Reentrant Flow Shop, Heuristics

1 Introduction

Scheduling in the manufacturing industry involves assigning jobs to machines and determining their order to optimize some criteria [1,22]. Many manufacturing layouts take the form of flow shops in which jobs progress from one machine to the next machine in a serial order without ever visiting the same machine twice [22]. Flow shop problems that do not allow sequence changes between machines are called **permutation flow shops** (PFS). In this class of problems, jobs are processed by a series of machines in precisely the same order [25]. A PFS problem is thus characterized by the same machine order for all jobs (flow shop) and the same job order for each machine (permutation).

In classical scheduling, it is typically assumed that each job visits any given machine at most once [1]. Contrary to that, many practical scenarios are of **reentrant shops**, in which a job may be processed by some of the machines several times [9]. In some industries, including signal processing and semiconductor wafer manufacturing, product design may call for jobs to recirculate or revisit a stage in the manufacturing process [24]. Reentrant shops can also be found in manufacturing facilities such as textile fabric manufacturing processes and mirror manufacturing systems [30]. A **reentrant flow**

shop (RFS) problem is distinguished from the classical flow shop problem by allowing jobs to be processed repeatedly at some machines.

An RFS is characterized by a reentrant pattern ϕ noting the machine order for the jobs. Explicitly, let ϕ_i be the i-th stage visited by each job, where $\phi_i \in \{1, 2, \ldots, m\}$. The reentrant pattern $\phi = (\phi_1,\phi_2,...,\phi_{|\phi|})$ is the sequence in which jobs visit the machines in the shop. The *reentrant property* occurs when at two stages in the sequence, $\phi_i = \phi_j$ for $i \neq j$. In maximal generality, the reentrant pattern determined by the manufacturing process may be arbitrary, provided all jobs follow the same sequence (i.e., flow shop) and at least one machine is revisited. Nonetheless, there are some special reentrant patterns that commonly appear in real-life manufacturing processes. One such pattern is the cyclic pattern [10], also called RFS with full loops [30]. In a cyclic-reentrant flow shop, all the jobs make l passes $(l > 1)$ through all the machines, and each pass goes through all the machines in order $1, 2, \ldots, m$. For $l = 2$ as an example, the obtained pattern is $\phi = (1, 2, \ldots, m, 1, 2, \ldots, m)$. This work assumes a cyclic pattern for the RFS. An example of cyclic RFS is in the assembly of electronic circuits stacked on top of each other [10]; each time a new circuit is connected, the same set of machines is visited. The cyclic pattern is extensively studied also because it is possible to describe any reentrant pattern by setting the processing time on some machines in some passes to be zero. Nevertheless, for reasons of clarity, we hereby consider non-zero processing times in each machine at each pass, that is, a true cyclic RFS.

An RFS problem with permutation characteristics is known as a **reentrant permutation flow shop** (RPFS) problem. The main contribution of this article regards the notion of **permutation** within the RPFS context. We claim that the use of the word "permutation" in RPFS takes on several different meanings, which naturally may create confusion. We offer a clear way to describe the different definitions of RPFS. We do not propose to abolish the various definitions but rather propose to address several permutation types. In the proposed approach, four different types of RPFS are obtained; in the literature, we found that researchers use three of them under the name RPFS.

The four permutation types are described in Section 2, together with a literature review mainly concerning RPFS with a cyclic pattern. In Section 3, we show that the different permutation types have meaning not only in terms of the clarity of representation but also from the point of view of heuristic construction. We consider Palmer's slope heuristic [20] for each of the four presented types and demonstrate it through simple examples. A discussion in Section 4 concludes the paper.

2 RPFS Permutation Types

Consider an RFS problem with a cyclic reentrant pattern. It consists of n jobs, m machines, and l levels. A level in a cyclic pattern is a pass of a job in all the machines in the order $1, 2, \ldots, m$. A level is also called a cycle or loop. A specific level of a specific job is also called a sub-job [8]. To show the ambiguity regarding the meaning of "permutation" in the RPFS literature, we present the following quotes that use contradicting definitions.

Definition 1: "Every job can be decomposed into several layers each of which starts on M_1 and finishes on M_m. In the RFS case, if the job ordering is the same on any machine at each layer, then no passing is said to be allowed, since no job is allowed to pass any former job. The RFS scheduling problem in which no passing is allowed, is called a RPFS problem." — quote taken from [6].

Definition 2: "Our considered m-machine reentrant permutation flow shop scheduling problem (MRPFSSP) can be viewed as a special kind of non reentrant flow shop scheduling problem (FSSP). If each job is decomposed into L sub-jobs and the reentrant-based precedence constraints among all sub-jobs are satisfied, MRPFSSP can be treated as an imaginary FSSP with nL sub-jobs." — quote taken from [23].

The first definition is more restrictive than the second. There is a third, intermediate definition used for general reentrant patterns.

Definition 3: "Pan and Chen (2003) developed three mixed integer models for the reentrant flow-shop problem . . . In these models, the job sequence for every machine is the same in each level and it is not allowed that higher levels preempt lower ones . . . Lee et al. (2011) relaxed the constraints of level permutation set by Pan and Chen (2003) in order to get better objective values. This relaxation makes it possible to mix job levels, i.e., jobs on higher levels can be processed on a machine k before jobs on a lower level." — quote taken from [15].

Table 1 presents a summary of the literature regarding RPFS with the cyclic pattern over the last two decades. The table shows that the definitions are used interchangeably over the years and are all referred to as RPFS. This creates an ambiguity that we aim to correct. It is noteworthy that:

- The majority of studies used Definition 1.
- Most studies that applied Definition 2 were related to the specific case of two-machine problems.
- Only a few studies used Definition 3.

A key insight that can be deduced from the above definitions, is that to properly handle the permutation characteristic in RFS problems, two issues should be considered:

1. Job passing: Is it allowed for the job order to be different at different levels?
2. Level passing: Is it allowed for level k of job j to be scheduled before level k' of another job j' where $k' < k$?

For Definition 1, the answer to both questions is negative: there is a single job order and this order is kept in all the levels. In addition, level passing is forbidden, i.e., level $k + 1$ of a job does not precede level k of a different job. We suggest to term this RPFS type as **Passing Prohibited** (PP). This term has a similar meaning to the "no passing" term used with respect to RPFS, for example, in [21,6].

For Definition 2, the answer to both questions is positive: the jobs are practically divided into sub-jobs, and the order of sub-jobs is not restricted as long as precedence constraints are kept, i.e., a level $k + 1$ of a job does not precede level k of the same job. We suggest to term this RPFS type as **Passing Allowed** (PA).

For the intermediate Definition 3, the answer to the first question is negative; a single job order is kept in all the levels. However, the answer to the second question is positive, i.e., level passing is allowed. We suggest to term this RPFS type as **Job Passing Prohibited** (JPP).

Year	**Reference**	**Permutation definition**	**Objective**	**Notes**
2003	[21]	Definition 1, PP	Makespan	
2006	[3]	Definition 1, PP	Makespan	
2007	[5]	Definition 1, PP	Makespan	
2007	[7]	Definition 2, PA	Makespan	Two-machine problem
2008	[6]	Definition 1, PP	Makespan	
2008	[17]	Definition 2, PA	Makespan	Two-machine problem
2008	[8]	Definition 2, PA	Makespan	
2009	[4]	Definition 1, PP	Makespan	
2013	[15]	Definition 3, JPP	Makespan	Not specific to cyclic RFS
2014	[16]	Definition 2, PA	Total tardiness	Two-machine problem
2014	[29]	Definition 1, PP	Makespan	
2016	[28]	Definition 1, PP	Makespan	
2017	[23]	Definition 1, PP	Makespan	
2018	[27]	Definition 1, PP	Makespan	
2021	[24]	Definition 1, PP	Makespan, average completion times, total tardiness	Multi-objective performance measure
2023	[11]	Definition 1, PP	Makespan, maximum tardiness	Bi-objective performance measure
2023	[18]	Definition 2, PA	Value-at-risk of the makespan	Two-machine problem, stochastic processing times
2023	[26]	Definition 3, JPP	Makespan	Not specific to cyclic RFS, identical jobs

Table 1: Summary of the cyclic RPFS literature review.

There is also a fourth definition, for which the answers to Questions 1 and 2 are positive and negative, respectively. For this type, level passing is forbidden, but the job order may be different at different levels. We suggest to term this RPFS type as **Level Passing Prohibited** (LPP). Table 2 summarizes the four types of RPFS.

Permutation Type	Job Passing (Question 1)	Level Passing (Question 2)
Passing Prohibited (PP)	Not Allowed	Not Allowed
Level Passing Prohibited (LPP)	Allowed	Not Allowed
Job Passing Prohibited (JPP)	Not Allowed	Allowed
Passing Allowed (PA)	Allowed	Allowed

Table 2: Four possible permutation types in cyclic RPFS.

Example 1. An RFS problem with $n = 3$ jobs, $m = 3$ machines, $l = 2$ levels, and the following processing times:

$$P_1 = \begin{pmatrix} 4\,2\,4 & 2\,2\,8 \\ 4\,4\,2 & 4\,2\,8 \\ 8\,2\,2 & 6\,2\,2 \end{pmatrix}$$

Here, each line corresponds to a job, and each column to an operation according to its order in the flow. For example, the processing times of job j_1 in machines 1, 2, and 3 are $(4, 2, 4)$ at level l_1 and $(2, 2, 8)$ at level l_2.

Figure 1 shows potential schedules based on the four permutation types. Different colors represent different jobs (j_1 – red, j_2 – green, and j_3 – blue):

– PP type schedule with job order $(2, 1, 3)$ at the two levels.
– LPP type schedule with job order $(2, 1, 3)$ at the first level and $(1, 2, 3)$ at the second level.
– JPP type schedule with job order $(2, 1, 3)$ for both levels, but l_2 of j_2 (green) precedes l_1 of j_3 (blue).
– PA type schedule with job order $(2, 1, 3)$ at the first level and $(1, 2, 3)$ at the second level. In addition, l_2 of j_1 (red) precedes l_1 of j_3 (blue).

Figure 2 depicts a Venn diagram of the solution space of the different permutation types. As follows from the definitions and visually presented by the Venn diagram, the largest solution space is of the PA type with $(3 \cdot 2)!/(2!)^3 = 90$ possible permutations for Example 1. The smallest solution space is of the PP type with $3! = 6$ possible permutations. The intermediate types contain $(3!)^2 = 36$ and $5 \cdot 3! = 30$ possible permutations for the LPP and JPP types of Example 1, respectively.

3 Heuristic Issues

A classical three-machine permutation flow shop scheduling problem with the makespan objective is strongly NP-hard even without job reentrancy [14]. The problem Fm |

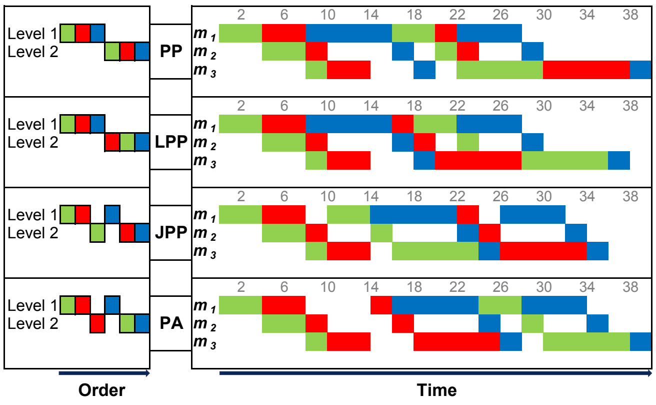


Fig. 1: Possible schedules of the four permutation types: passing prohibited (PP), level passing prohibited (LPP), job passing prohibited (JPP), passing allowed (PA).

reentrant | $C_{\max}$ is NP-hard, even for a two-machine problem [10]. Most studies on RFS problems thus focus on improving the computational efficiency of optimization algorithms (e.g., branch and bound) or developing efficient heuristics and metaheuristics. A common approach to devising a heuristic for RPFS is to adjust a (non-reentrant) flow shop heuristic to the RFS problem. Several studies applied this methodology while considering the PP [21] and PA [7] permutation types. The intermediate types – LPP and JPP – were not considered in this respect.

This section aims to show the relevance of the different permutation types to heuristics construction. We demonstrate this issue by examining Palmer's slope heuristic [20] as an initial step toward analyzing other commonly used PFS heuristics, such as CDS [2], NEH [19], and others. Palmer's slope heuristic has already been generalized for solving an RFS problem with makespan objective [21]. The obtained solution is of a PP permutation type. The generalization to the other permutation types is obtained by treating the processing times of each job as the processing times of a job in a non-reentrant flow shop (as if a machine at each level is a machine in itself). For each job, the slope index is calculated, and the jobs are arranged in descending order of the slope. When passing is prohibited, the job order is sufficient to determine the schedule. This simple procedure can be naturally generalized to the other permutation types by the calculation of slope indices for each of the sub-jobs. Treating sub-jobs is acceptable in generalizing flow shop heuristics to the reentrant with PA permutation type [7]. The generalization we consider here is to define an order of the sub-jobs according to their slopes, keeping in mind the constraints each type induces:

– For the LPP permutation type, we first arrange the sub-jobs of all the jobs of the first level according to its slope indices, from the largest to the smallest. We then arrange the sub-jobs of the second level according to its slope. We continue this, level after level, until the last level is reached and arranged.

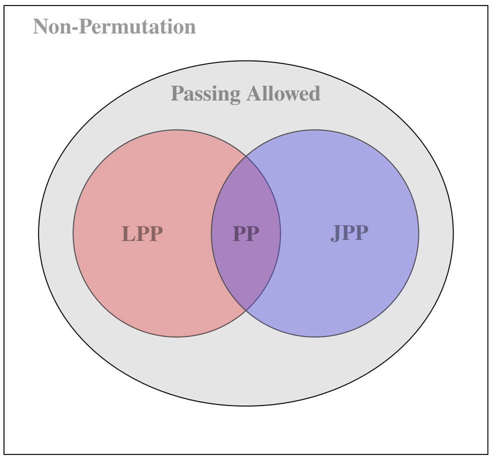


Fig. 2: A Venn diagram of the RPFS permutation types, including non-permutation RFS.

- For the PA permutation type, we arrange all the sub-jobs according to the slope indices while taking care of operation precedence, i.e., for each job, the sub-jobs of that job will be arranged in ascending order of levels.
- For the JPP permutation type, we regard a combination of the two types of slope indices. A job order is first determined according to the rule established for the PP type (considering the job slope indices). Then, the sub-jobs are arranged according to the sub-job indices while taking care of level precedence (like in the PA type), i.e., ensuring that the sub-job order is maintained within each job.

Once the rule for sub-job order (described above for each of the permutation types) is applied, a schedule is constructed by selecting each sub-job according to the order and inserting it after the previous sub-job. Another step that is generally considered in heuristics is that of solution improvement [13]. A standard improvement is obtained by sub-job swaps. The described heuristic, including some swap improvements, is shown in the following examples.

Consider Example 1 of Section 2. For this small example, following the above heuristic steps results in only two schedules for the four permutation types. The PP-type and LPP-type schedules are identical, with a makespan of 38. The JPP-type and the PA-type schedules are identical, with a makespan of 50. The example shows a major challenge in the generalization of heuristics based on the division into sub-jobs: how to avoid, or at least reduce, cases in which sub-jobs that belong to the same job are sequentially scheduled. In the second schedule, two such sequences greatly increase the makespan. By applying two simple swaps to avoid these sequences, a JPP-type schedule with a makespan of 36 can be obtained, see Figure 3.

Another challenge related to the construction of heuristics is related to tie-breaking, as demonstrated by Example 2.

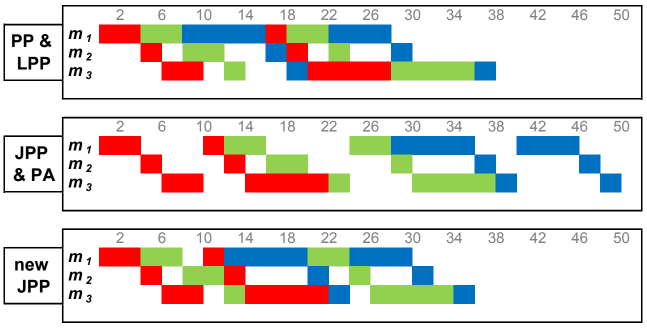


Fig. 3: Schedules obtained by the adjusted slope heuristic. The "new JPP" schedule is obtained from the JPP&PA schedule by two swaps: $j_1l_2 \leftrightarrow j_2l_1$ and $j_2l_2 \leftrightarrow j_3l_1$.

Example 2. An RFS problem with $n = 4$ jobs, $m = 3$ machines, $l = 3$ levels, and the following processing times:

$$P_2 = \begin{pmatrix} 6\,4\,2 & 8\,6\,4 & 6\,2\,2 \\ 8\,6\,8 & 6\,4\,2 & 2\,2\,6 \\ 4\,2\,4 & 6\,4\,6 & 4\,4\,8 \\ 8\,6\,8 & 4\,6\,4 & 8\,6\,4 \end{pmatrix}$$

Recall that each line corresponds to a job, and each column corresponds to an ordered operation.

In Example 2, several sub-jobs have an identical slope index. For the LPP type, 96 different sub-job orders can be obtained depending on the rule that breaks the tie. Finding a good tie-breaking rule is a known problem in flow shops [12] and can be significant for the quality of the solution. Figure 4 shows two possible LPP-type schedules. There is a tie between the schedules regarding the sub-job indices but the makespan is significantly different.

For both Examples 1 and 2, better solutions were obtained by allowing passing. In other scenarios, the strict PP type may be dominant. It is clear that a solution that allows passing is always at least as good as a solution that prohibits it, and a substantial challenge remains in finding good solutions by efficiently examining only a part of the solution space.

4 Discussion

It is known that there are scenarios for which the optimal solution is not a permutation schedule. This is true for $Fm||C_{max}$ compared to $Fm|prmu|C_{max}$ if $m > 3$ [10]. For the RFS problem, this is true even for two machines and the PA less-restricting permutation type [7]. Nevertheless, many real-world RFS manufacturing systems prefer to use permutation schedules because they offer greater ease of operation and control.

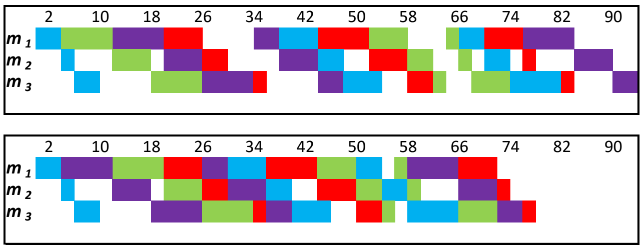


Fig. 4: Two LPP type schedules with ties regarding the sub-job slopes (j_1 – red, j_2 – green, j_3 – blue. and j_4 – purple)

In some cases, only permutation schedules are feasible because of the inflexibility of material handling systems or limited buffer space [16]. Theoreticians may prefer the permutation version as an algorithmically simplifying assumption. The permutation scenario was also adjusted to the reentrant case. However, the meaning of "permutation" has had several interpretations. We suggest to clarify the definition using the four permutation types.

Several heuristics have been proposed over the years to achieve a high-quality RPFS solution [21,7,8,17]. The types of permutations were only partially considered in previous work on RFS heuristics. We have shown a possible approach to extend Palmer's slope heuristic demonstrating the potential in regarding the four permutation types, as well as the challenges ahead. This only serves as a preview for further comprehensive research that will involve addressing larger problems and exploring other heuristics. A generalization of existing heuristic methods to each permutation type may provide a flexible and efficient framework for scheduling in RFS environments.

References

1. Baker, K.R.: Introduction to sequencing and scheduling (1974)
2. Campbell, H.G., Dudek, R.A., Smith, M.L.: A heuristic algorithm for the n job, m machine sequencing problem. Management science **16**(10), B–630 (1970)
3. Chen, J.S.: A branch and bound procedure for the reentrant permutation flow-shop scheduling problem. The International Journal of Advanced Manufacturing Technology **29**, 1186–1193 (2006)
4. Chen, J.S., Pan, J.C.H., Lin, C.M., et al.: Solving the reentrant permutation flow-shop scheduling problem with a hybrid genetic algorithm. International Journal of Industrial Engineering **16**(1), 23–31 (2009)
5. Chen, J.S., Pan, J.C.H., Wu, C.K.: Minimizing makespan in reentrant flow-shops using hybrid tabu search. The International Journal of Advanced Manufacturing Technology **34**, 353–361 (2007)
6. Chen, J.S., Pan, J.C.H., Wu, C.K.: Hybrid tabu search for re-entrant permutation flow-shop scheduling problem. Expert Systems with Applications **34**(3), 1924–1930 (2008)
7. Choi, S.W., Kim, Y.D.: Minimizing makespan on a two-machine re-entrant flowshop. Journal of the Operational Research Society **58**(7), 972–981 (2007)

8. Choi, S.W., Kim, Y.D.: Minimizing makespan on an m-machine re-entrant flowshop. Computers & Operations Research **35**(5), 1684–1696 (2008)
9. Danping, L., Lee, C.K.: A review of the research methodology for the re-entrant scheduling problem. International Journal of Production Research **49**(8), 2221–2242 (2011)
10. Emmons, H., Vairaktarakis, G.: Flow shop scheduling: theoretical results, algorithms, and applications, vol. 182. Springer Science & Business Media (2012)
11. Fasihi, M., Tavakkoli-Moghaddam, R., Jolai, F.: A bi-objective re-entrant permutation flow shop scheduling problem: minimizing the makespan and maximum tardiness. Operational Research **23**(2), 29 (2023)
12. Fernandez-Viagas, V., Framinan, J.M.: On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem. Computers & Operations Research **45**, 60–67 (2014)
13. Framinan, J.M., Gupta, J.N., Leisten, R.: A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. Journal of the Operational Research Society **55**, 1243–1255 (2004)
14. Garey, M.R., Johnson, D.S., Sethi, R.: The complexity of flowshop and jobshop scheduling. Mathematics of operations research **1**(2), 117–129 (1976)
15. Hinze, R., Sackmann, D., Buscher, U., Aust, G.: A contribution to the reentrant flow-shop scheduling problem. IFAC Proceedings Volumes **46**(9), 718–723 (2013)
16. Jeong, B., Kim, Y.D.: Minimizing total tardiness in a two-machine re-entrant flowshop with sequence-dependent setup times. Computers & operations research **47**, 72–80 (2014)
17. Jing, C., Tang, G., Qian, X.: Heuristic algorithms for two machine re-entrant flow shop. Theoretical Computer Science **400**(1-3), 137–143 (2008)
18. Liu, L., Urgo, M.: Robust scheduling in a two-machine re-entrant flow shop to minimise the value-at-risk of the makespan: branch-and-bound and heuristic algorithms based on markovian activity networks and phase-type distributions. Annals of Operations Research pp. 1–24 (2023)
19. Nawaz, M., Enscore Jr, E.E., Ham, I.: A heuristic algorithm for the m-machine, n-job flowshop sequencing problem. Omega **11**(1), 91–95 (1983)
20. Palmer, D.S.: Sequencing jobs through a multi-stage process in the minimum total time—a quick method of obtaining a near optimum. Journal of the Operational Research Society **16**(1), 101–107 (1965)
21. Pan, J.H., Chen, J.S.: Minimizing makespan in re-entrant permutation flow-shops. Journal of the operational Research Society **54**, 642–653 (2003)
22. Pinedo, M.: Scheduling. theory, algorithms and systems. ISBN0-13-706757-7 (1995)
23. Qian, B., Li, Z.c., Hu, R.: A copula-based hybrid estimation of distribution algorithm for m-machine reentrant permutation flow-shop scheduling problem. Applied Soft Computing **61**, 921–934 (2017)
24. Rifai, A.P., Kusumastuti, P.A., Mara, S.T.W., Norcahy, R., Dawal, S.: Multi-operator hybrid genetic algorithm-simulated annealing for reentrant permutation flow-shop scheduling. ASEAN Engineering Journal **11**(3), 109–126 (2021)
25. Sauvey, C., Sauer, N.: Two neh heuristic improvements for flowshop scheduling problem with makespan criterion. Algorithms **13**(5), 112 (2020)
26. Shufan, E., Grinshpoun, T., Ikar, E., Ilani, H.: Reentrant flow shop with identical jobs and makespan criterion. International Journal of Production Research **61**(1), 183–197 (2023)
27. Wu, C.C., Liu, S.C., Cheng, T., Cheng, Y., Liu, S.Y., Lin, W.C.: Re-entrant flowshop scheduling with learning considerations to minimize the makespan. Iranian Journal of Science and Technology, Transactions A: Science **42**, 727–744 (2018)
28. Xu, J., Lin, W.C., Wu, J., Cheng, S.R., Wang, Z.L., Wu, C.C.: Heuristic based genetic algorithms for the re-entrant total completion time flowshop scheduling with learning con-

sideration. International Journal of Computational Intelligence Systems **9**(6), 1082–1100 (2016)
29. Xu, J., Yin, Y., Cheng, T.C.E., Wu, C.C., Gu, S.: A memetic algorithm for the re-entrant permutation flowshop scheduling problem to minimize the makespan. Applied Soft Computing **24**, 277–283 (2014)
30. Yu, T.S., Pinedo, M.: Flow shops with reentry: Reversibility properties and makespan optimal schedules. European Journal of Operational Research **282**(2), 478–490 (2020)

iMOPSE library for benchmarking Multi-Skill Resource-Constrained Project Scheduling Problem

Konrad Gmyrek[1] [0009−0000−7206−3674], Michał Antkiewicz[1] [0000−0002−6249−4507], Paweł Borys Myszkowski[1] [0000−0003−2861−7240], and Jose Luis Calvo-Rolle[2] [0000−0002−2333−8405]

[1] Wrocław University of Science and Technology, Faculty of Information and Communication Technology, Wrocław, Poland
`{konrad.gmyrek,michal.antkiewicz,pawel.myszkowski}@pwr.edu.pl`
[2] University of A Coruña - Department of Industrial Engineering, Coruña, Spain
`jlcalvo@udc.es`

Abstract. This paper presents an open-source iMOPSE (Intelligent Multi-Objective Problem Solving Environment) library for (meta)heuristic optimization for benchmarking Multi-Skill Resource-Constrained Project Scheduling Problem considered as single-, multi-, and many-objective optimization problems. The library is implemented in C++ and is designed to support researchers, students, and practitioners. The library includes several sets of benchmark instances, implementation of NP-hard problems, and (meta)heuristics, like Genetic Algorithm, Tabu Search, and state-of-the-art multi-objective NSGA-II, SPEA2, or MOEA/D. Additionally, supporting software tools are included, which are helpful in solution validation, visualization, and research automatization. All data and provided code is freely published as open source repository on GitHub.

Keywords: benchmark, scheduling, Multi-Skill Resource-Constrained Project Scheduling Problem.

1 Introduction

The Multi-Skill Resource-Constrained Project Scheduling (MS-RCPSP) is a combinatorial NP-hard scheduling problem related to real-world problems, e.g., the Software Project Scheduling Problem in software development used in Volvo IT company. The tasks that need to be executed are connected in a precedence graph, so the MS-RCPSP problem is overconstrained. Moreover, in MS-RCPSP, the RCPSP problem is extended by resource (human) skills at various levels, introducing additional domain constraints, making the problem more difficult to solve but flexible in management. The MS-RCPSP problem benchmark was originally defined [1][2] and could be considered a single- and many-objective optimization problem. MS-RCPSP problem is commonly used and cited (e.g., surveys [13][14][20]) – according to GoogleScholar nearly 80 scientific papers (years 2015-2024) reference or use MS-RCPSP iMOPSE dataset. Additionally, 13 papers that define the MS-RCPSP problem (and solving methods) are cited 605 times.

Initially, the MS-RCPSP problem was defined as a single-objective optimization problem and presented in [4], where a hybrid of Ant Colony Optimisation (HantCo) and the greedy algorithm was proposed. The greedy Randomized Adaptive Search Procedure (GRASP) method gained a solution about 15.8% more efficient than HantCo [6]. The above methods solve MS-RCPSP as a single objective using project duration time *Makespan* as the only objective function. Additionally, some of the papers explore the effect of autonomous team role selection in flexible projects and investigate synergies between employees using the iMOPSE dataset [19]. Hybridized Differential Evolution and Greedy (DEGR)[7] can also be used to solve MS-RCPSP, and like all the above methods, is based on a greedy algorithm (Schedule Generator Scheme) to get feasible solutions. Additionally, the extra research presented in [8] presents the influence of how coevolution and solution representation could be effective in solving MS-RCPSP.

The MS-RCPSP can also be solved considering two objectives: project total *Cost* and duration (*Makespan*). In this context, bi-objective optimization in MS-RCPSP should be applied – in [9] results of Non-dominated Sorting Genetic Algorithm (NSGA-II) are presented. However, classical NSGA-II has some limitations, and the new NTGA (Non-dominated Tournament Genetic Algorithm)[10] method has been proposed – NTGA focuses on the diversity of the population, what makes it more effective in solving bi-objective MS-RCPSP. Next, the NTGA2 [11] method has been proposed, which extends the NTGA by GAP selection method and extra mechanism to manage the archive actively. It makes NTGA2 a robust and effective method of multi- and many-objective optimization in solving MS-RCPSP.

To compare the results of NTGA2, NTGA, and classical NSGA-II with other methods, a survey of quality that could be applied directly to MS-RCPSP has been published [12]. A set of complementary measures has been proposed – dedicated and verified in application to MS-RCPSP. In work [11] NTGA2 is investigated in solving MS-RCPSP with 5 objectives and compared to state-of-the-art many-objective methods (e.g. U-NSGA-III or Theta-DEA). Recently, the new *balanced* B-NTGA [18] method has been published – it actively balances the exploitation/exploration in the solution landscape – it dominates the results of other state-of-the-art methods in multi- and many-objective MS-RCPSP.

There are several methods presented in the literature that use iMOPSE MS-RCPSP dataset [2] as a benchmark – they could divided by optimization types: single objective (1 objective), bi–objective (or multi-), and many–objectives (5 objectives). Methods are based on various metaheuristics, like Differential Evolution [7], Ant Colony Optimisation [4], Fruit Fly Optimisation [15] or Teaching-Learning Optimisation [16]. Moreover, hyperheuristics like Genetic Programming Hyper-heuristic [17] are used. Several methods solve MS-RCPSP as a bi-objective problem (e.g., fruit fly MOFOA [21], genetic program. hyper-heuristic MOGP-HH-D [22], NTGA). However, to our best knowledge, only two methods solve MS-RCPSP with 5 objectives: NTGA2 [11] and B-NTGA [18].

The main **contribution** of this paper is (1) to summarise and extend the MS-RCPSP research: all benchmark data [1][2][3] to define a standard set of instances, (2) to provide iMOPSE *open-source* and publish code for state-of-the-art solving MS-RCPSP methods to make method comparison more accessible, and to develop a (3) software tools specialized for MS-RCPSP, like validators or visualizers.

The rest of the article is structured as follows. The definition of MS–RCPSP problem is given in section 2. The MS-RCPSP instances for the investigations are presented in section 3. Section 4 contains the description of iMOPSE and concludes with a case study showcasing an example of an experiment workflow. The last section (5) concludes the work and highlights potential directions for further research.

2 Multi-Skill Resource-Constrained Project Scheduling Problem

The MS-RCPSP is a combinatorial NP-hard problem within the domain of scheduling problems based on model used in Volvo IT company. The discussed problem comprises two interconnected sub-problems: task sequencing, which involves placing tasks on a timeline, and resource assignments. MS-RCPSP is defined by a list of tasks and resources where each task requires a resource with a specified skill level to be executed. The goal is to find a optimal and a *feasible* schedule PS, meaning a solution that satisfies all **constraints**.

Each resource is linked to a corresponding salary, adhering to the constraint defined in Eq. 1, ensuring that for each resource r, no salary (r_{salary}) value can assume a negative value. Additionally, it dictates that each resource must be associated with a non-empty set of skills, as resources and tasks are linked to specific skill sets.

$$\forall_{r \in R} r_{salary} \geq 0, \forall_{r \in R} S^r \neq \emptyset \tag{1}$$

where S^r is the set of skills possessed by resource $r \in R$.
The duration and finish time of each task cannot be negative (see in Eq. 2.)

$$\forall_{t \in T} F_t \geq 0; \forall_{t \in T} d_t \geq 0 \tag{2}$$

where F_t denotes the finish time, and d_t represents the duration of task t.
Eq. 3 introduces constraints related to task precedence, stating that a task can only start after all its predecessors are completed.

$$\forall_{t \in T, p \in t_p} F_p \leq F_t - d_t \tag{3}$$

where t_p denotes the predecessors of task t.
Eq. 4 addresses the skill requirements in MS-RCPSP, ensuring a resource allocated to a task possesses the requisite skill at an appropriate level.

$$\forall_{t \in T^r} \exists_{s_r \in S^r} \; h_{s_t} = h_{s_r} \wedge l_{s_t} \leq l_{s_r} \tag{4}$$

where T^r is a set of tasks assigned to a resource r, s_t is the skill required by the task t, S^r is the set of skills possessed by the resource r, h and l are the type and level of the skill respectively.
A constraint ensuring at most one resource is assigned to any task at any given time is presented in Eq. 5.

$$\forall_{r \in R} \forall_{t \in \tau} \sum_{i=1}^{n} U_{i,r}^{t} \leq 1 \tag{5}$$

where τ is the time domain, n represents the total number of tasks, and $U_{i,r}^{t}$ is a binary variable, equal to 1 if resource r is assigned to task i at time t.

The final constraint (see Eq. 6) ensures that all tasks must be finished by ensuring that all tasks have a resource assigned at some time slot.

$$\forall_{i \in T} \exists_{t \in \tau, r \in R} U^t_{i,r} = 1 \tag{6}$$

where τ and $U^t_{i,r}$ are defined as in Eq. 5.

2.1 objectives in MS-RCPSP

The MS-RCPSP can be defined as single- multi- and ultimately as a many-objective optimization framework, accommodating up to five objectives [3].

The optimal schedule is the one with the minimal objective function – for multi-objective optimization, an approximation of Pareto Front is investigated. The feasible schedule satisfies all constraints related to tasks, resources, skills, and precedence relations. Formally, the MS–RCPSP optimization problem can be defined as follows:

$$f : \Omega \longrightarrow \mathbb{R}, min(f) \tag{7}$$

where Ω is the feasible schedule space, while the f is the given objective function(s). In many-objective MS–RCPSP there are five defined objectives as follows:

- the project schedule duration (makespan) – f_τ (see Eq. 8),
- schedule's cost – f_C (see Eq.9),
- skill overuse – f_S (see Eq.10),
- and average use of resources – f_R (see Eq.11).
- average cash flow – (f_F) (see Eq.12),

The two most commonly described objectives in literature consist of schedule *Makespan* (or Duration) and total *Cost*. Additional MS-RCPSP objective aims to describe a specific schedule aspect: *Average Cash Flow*, *Skill Overuse*, and the *Average Use of Resources*. The MS-RSPSP optimization objectives are defined below.

The **Makespan** $\mathbf{f}_\tau(\mathbf{PS})$ of the project schedule PS is given as Eq.8.

$$f_\tau(PS) = \max_{t \in T} t_{finish} \tag{8}$$

where T is a set of all tasks, t_{finish} is the finish time of the task t. The **Cost** of the schedule is $\mathbf{f_C}(\mathbf{PS})$ defined as Eq.9.

$$f_C(PS) = \sum_{i=1}^{n} R_i^{salary} * T_i^{duration} \tag{9}$$

where n is the number of all task-resource assignments, R_i^{salary} is the salary of a resource of the i-th assignment, $T_i^{duration}$ is the duration of the task of the i-th assignment.

Skill Overuse aims to minimize the difference between the skill level of a resource and the required skill. **Skill Overuse** $\mathbf{f_S}(\mathbf{PS})$ – see Eq.10) – ensure that the resources assigned to the task are not overqualified, which could be essential in the practical applications.

$$f_S(PS) = \sum_{i=1}^{n} R_s^l - T_s^l \tag{10}$$

where n is the number of task-resource assignments, R_s^l is the skill level of a resource R, and T_s^l is the skill level required for the task T.

Some resources are assigned to the project - and must receive a salary, even if they are not assigned to any task. The **Average Use of Resources** (see Eq. 11) objective gives the distribution of tasks and ensures the efficient use of resources. It aims to minimize the deviation of the number of task-resource assignments.

$$f_R(PS) = \frac{1}{r} \sum_{i=1}^{r} (R_i^n - R_{avg}^n) \tag{11}$$

where r is the number of resources, R_i^n is the number of tasks assigned to the i-th resource, R_{avg}^n is the expected average number of assignments.

The **Average Cash Flow $\mathbf{f_F(PS)}$** (see Eq. 12) measures the deviation of costs over the entire duration of the project and allows for more effective budget management.

$$f_F(PS) = \frac{1}{f_\tau(PS)} \sum_{t=1}^{f_\tau} (C_t - C_{avg}) \tag{12}$$

where C_t is the cost of the project in a single time slot t, $f_\tau(PS)$ is the makespan of the project, C_{avg} is the average cost of the project in a time unit and can be defined by the Eq.13.

$$C_{avg} = \frac{C}{f_\tau} \tag{13}$$

where C and f_τ are the total $cost$ and $makespan$ of the project respectively.

Although the MS-RCPSP, in its nature is multi-objective, but could also be considered in a simplified version as **single-objective MS-RCPSP**[7], where the evaluation function is formulated as follows:

$$\min f(PS) = w_\tau f_\tau(PS) + (1 - w_\tau) f_c(PS) \tag{14}$$

where: w_τ – weight of duration component, where $f_\tau(PS)$ and $f_C(PS)$ is normalised and $w_\tau \in [0; 1]$.

The above five objectives $(f_\tau, f_C, f_S, f_R, f_F)$ are already implemented in iMOPSE library for MS-RCPSP, but there are no limits and this set could be redefined as iMOPSE is opensource.

3 MS-RCPSP instances

The provided dataset emerged during a research cycle and was gradually expanded to accommodate the needs. It consists of 265 unique test instances prepared using the iMOPSE generator with different parameters to show the influence of constraints (e.g., introducing extremely low and high values for precedence relations or no skill requirements), as well as the task and resource quantity.

All instances are defined using the coherent format and follow the naming convention, encoded as <*task count*>_<*resource count*>_<*precedence relation count*>_<*skill count*>_<*postfix*>. For example, 200_10_84_9 consists of 200 tasks with 84 precedence relations and 10 resources with up to 9 unique skills. The <*postfix*> at the end is optional and can be used for marking specific variations of the instance. With that said, instance names should not be decoded to access the exact values, as those are nominal values, and slight deviations can be found in the data. Therefore, the exact quantities are included inside the instance definition.

All instances are located within iMOPSE directory *configurations/problems/MSR-CPSP/all* and have been divided into the following groups:

– **Small (6)** - small toy-size instances with 10-15 tasks and 3-9 resources; good for validation and visualization as optimal solutions are easy to find
– **Regular (36)** - regular instances with 100/200 tasks, and 5-40 resources, with varying numbers of skills and relatively small numbers of constraints; instances with postfix 'D' come from the real-world scenarios and might be considered more difficult due to bottlenecks and project milestones; the other instances are generated to imitate the same characteristics
– **RegularGenerated (128)** - regular randomly generated instances with 100/200 tasks, 5-40 resources, 5/10 skills, and varying numbers of constraints; generated to increase the standard instances set providing more combinations to investigate the impact of task, resource, skill, and precedence relation number
– **Dense (7)** - randomly generated instances with 100 tasks, 10-40 resources, 9/15 skills, and a high density of precedence relations
– **NoConstr (8)** - randomly generated instances with 100/200/500/1000 tasks, 20/40 resources, and no constraints (every resource-task connection is valid; no precedence relations)
– **Big (80)** - randomly generated instances with 500/1000 tasks, 10 - 40 resources, 5/10 skills, and varying numbers of constraints

4 iMOPSE library – a general idea

iMOPSE is an advanced, open-source C++ toolkit designed for solving NP-hard problems through a suite of optimization algorithms. Tailored for academic research and practical applications, iMOPSE streamlines the process of addressing complex optimization tasks. The library's modular architecture allows easy extensibility in method and problem implementation (Fig. 1). This section provides an in-depth exploration of iMOPSE capabilities that support research related to the MS-RCPSP problem and methods for solving it.

The iMOPSE library includes several evolutionary-based state-of-the-art multi-objective **methods**, such as NSGA-II [23], Multi-objective Evolutionary Algorithm Based on Decomposition (MOEA/D) [24], Strength Pareto Evolutionary Algorithm

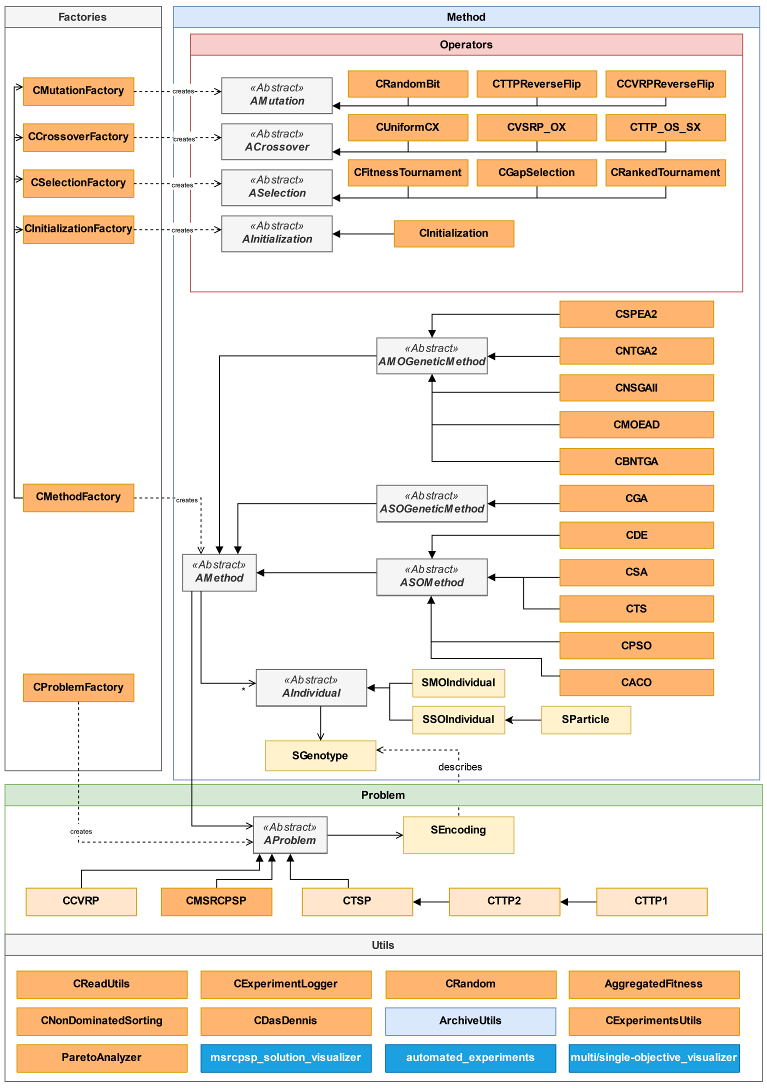


Fig. 1: iMOPSE library – a general schema

(SPEA2) [25], NTGA2 [11] and experimental balanced B-NTGA [18]. Additionally, iMOPSE offers a set of algorithms for single-objective optimization, including Genetic Algorithms (GA), Differential Evolution (DE), Ant Colony Optimization (ACO), Tabu Search (TS), Simulated Annealing (SA), and Particle Swarm Optimization (PSO), therefore providing a set of ready-to-use methods that can be easily extended.

The above methods are known as effective in solving multi-objective NP-hard **problems** with constraints, such as MS-RCPSP [3] and Traveling Thief Problem (TTP). Beyond MS-RCPSP, iMOPSE is capable of handling various classical NP-hard problems, such as the Traveling Salesman Problem (TSP), TTP, and Capacitated Vehicle Routing Problem (cVRP).

Each metaheuristic to search solution space needs a solution representation and defined genetic operators (or neighborhoods). The following two sections describe that.

4.1 representation for MS-RCPSP

For metaheuristics, to enable an effective search in the solution space, a **representation** (solution) should be specialized to a given problem. The iMOPSE library supports three types of representation (permutation, binary, and real-coded), which enable coding NP-hard problems as combinatorial (for example TSP) or priority-based in MS-RCPSP resource assignment (see Fig. 2).

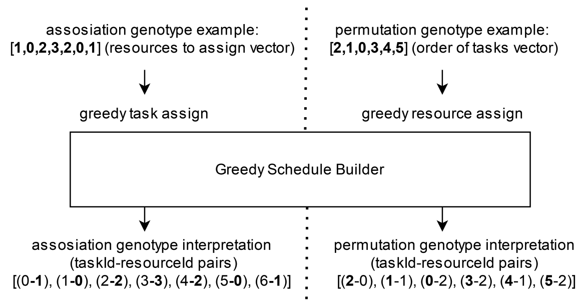

assosiation genotype example:
[**1,0,2,3,2,0,1**] (resources to assign vector)
permutation genotype example:
[**2,1,0,3,4,5**] (order of tasks vector)
greedy task assign
greedy resource assign
Greedy Schedule Builder
assosiation genotype interpretation
(taskId-resourceId pairs)
[(0-**1**), (1-**0**), (2-**2**), (3-**3**), (4-**2**), (5-**0**), (6-**1**)]
permutation genotype interpretation
(taskId-resourceId pairs)
[(**2**-0), (**1**-1), (**0**-2), (**3**-2), (**4**-1), (**5**-2)]

Fig. 2: An example of two types of representation for MS-RCPSP

In the iMOPSE framework, two distinct genotype encodings for the MS-RCPSP are implemented. The first encoding utilizes a vector of task-to-resource associations, wherein each vector element represents a unique resource identifier, and the element's index corresponds to the task identifier. The second encoding adopts a permutation-based approach, wherein the vector describes the sequence of tasks, with each element denoting a task identifier. A dedicated schedule builder uses both encoding methods to construct valid solutions from the genotypes. For the association-based approach, the schedule builder assigns resources by iterating through the genotype vector, assigning the i-th task to the resource indicated by the i-th value in the genotype vector. On the other hand, the permutation-based approach iterates through the sequence of tasks described by the genotype and assigns each task to the resource that will be available in the shortest time. In both scenarios, the greedy schedule builder takes precedence relations into account and automatically adjusts the schedule to ensure validity (see Fig. 2).

Not all solutions can be regarded as *feasible* schedules, as certain constraints might remain unsatisfied. Each investigated method presented in the following sections, capable of acquiring *feasible* schedule uses a **greedy–based algorithm as Schedule Builder** (i.e.[7][11]) to construct a feasible solution.

4.2 Genetic operators and neighborhoods

Each metaheuristic utilizes **operators** to explore the solution landscape effectively. In iMOPSE, there are several implemented operators: neighborhood operators (for TS or SA), mutations (like *RandomBit*, *ReverseFlip*, *GaussMutation*), or crossovers (e.g., *Ordering Crossover OX*, *Cycle Crossover CX*). Each operator is designed to work with specific encoding types; therefore, users must ensure they are using the correct operator for their chosen encoding method.

To direct metaheuristic in a global search, especially for evolutionary computation, **selection** operators also are needed – *random* (semi-blind, without selection pressure, as reference), classic *tournament* or *gapSelection* [11] for multi-objective optimization. It is worth mentioning that a predefined set of representations, operators, and selections can be easily extended as the iMOPSE library C++ interfaces are given for implementation.

Although the operator architecture in the proposed system already includes specific operations common across various methods, it's entirely feasible to write the entire code for a method within a single class. Nevertheless, we recommend enhancing and building upon the existing operator's architecture or extending it when developing new methods. This approach facilitates more robust and flexible method implementation.

4.3 An additional tools (utils)

The iMOPSE library is equipped with utils that support computation and provide researchers with ready-to-use tools for automation, analysis, and visualization. In this subsection, we describe the most relevant utils tools present in the discussed library (see Fig. 1).

iMOPSE main framework utilizes *ExperimentUtils* and *ExperimentLogger* to support the process of collecting and saving data from experiments, enhancing data management. Additionally, *ArchiveUtils* aids in archive operations for multi-objective methods. For sorting and decomposing optimization problems, Non-dominated sorting is utilized by NSGAII and NTGA2.

External utils – Python scripts The C++ programming language is known as very effective in computation; however, for data analysis and visualization, more useful is *Pyton*. That's why, in the iMOPSE library, several external tools have been added. Our collection of Python scripts is crafted to augment scientific research, each designed for a specific use case and supplemented with descriptive comments for ease of use. *automated_experiments.py* automates the concurrent execution of iMOPSE, offering a robust solution for efficient experimenting. This solution allows researchers to focus more on analysis and less on the operational aspects of their experiments. Furthering our support for scientific analysis of methods, *msrcpsp_solution_visualizer.py* validates the given MS-RCPSP solution against constraints and visualizes it, highlighting broken constraints and discrepancies (see Fig. 3). This not only aids in improving the understanding of complex optimization solutions but also in refining and enhancing these methods through iterative feedback.

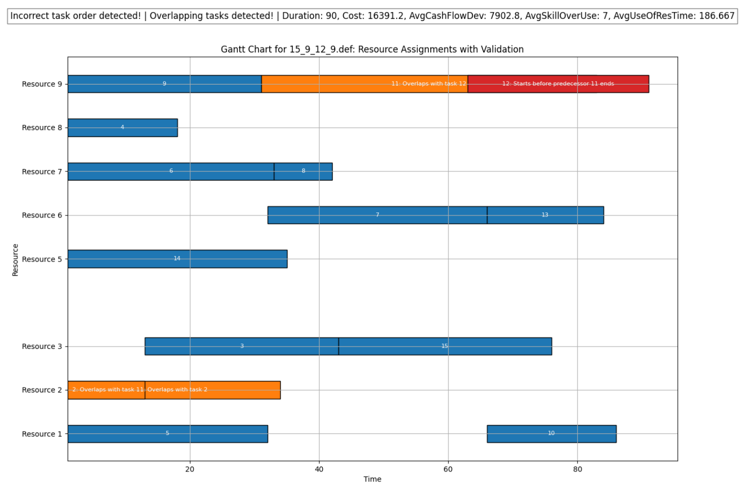

Incorrect task order detected! | Overlapping tasks detected! | Duration: 90, Cost: 16391.2, AvgCashFlowDev: 7902.8, AvgSkillOverUse: 7, AvgUseOfResTime: 186.667

Fig. 3: Example of (invalid) visualized MS-RCPSP solution for instance 15_9_12_9

For researchers working on multi-objective optimization, *multi-objective_visualizer.py* offers visualization of PFA to elucidate the trade-offs between competing objectives. Meanwhile, *single-objective_visualizer.py* provides a graphical overview of the best, worst, and mean fitness values throughout an experiment, enriching the analysis of optimization processes.

Together, these scripts furnish researchers with tools for conducting experiments that are not only more efficient and insightful but also more impactful, thereby enriching the quality and depth of scientific investigations. These scripts serve as rapid, flexible solutions and are planned to be integrated with the main C++ codebase in the future, enhancing the robustness and scalability of the software for broader scientific applications and research.

Pareto Analyzer tool In multi- and many-objective optimization, the output of each method is a set of non-dominated points.A point is dominated if any other point has at least one better (lower, considering the MS-RCPSP) objective value and no worse objective value. All quality measures (QMs) used for multi-objective MS-RCPSP solution are calculated based on the returned set, called Pareto Front Approximation (PFA). The true Pareto Front (TPF) could be defined as a set of all non-dominated solutions and can be considered the best available PFA. However, in practical real-world problems, TPF is usually unknown. *NadirPoint* is a point with the worst possible values for all objectives. For the MS-RCPSP, the worst value of makespan is the total sum of all tasks' duration - all tasks are serial. The worst cost value is the cost of schedule, where the most expensive resource performs all tasks. Worst skill overuse is achieved by assigning tasks to the resources with the highest required skill level. The worst average use of resources is the maximum makespan multiplied by the number of resources - 1 and divided by the number of resources. The worst value of the average cash flow is the same as the maximum cost.

Commonly used QMs are [12]: *HyperVolume* (HV), *Inverted Generative Distance* (IGD) and *Purity*. All the QMs implemented in the Pareto Analyzer are described below.

HyperVolume ↑ (HV) quantifies the volume of the objective space dominated by a set of solutions. It reflects the spread and coverage of a PF. The higher the hypervolume, the more comprehensive coverage of the objective space. It can be formally defined by Eq. 15.

$$HV(PF) = \Lambda(\bigcup_{s \in PF} \{s' | s \prec s' \prec s^{nadir}\}) \tag{15}$$

where PF is an approximation of PF, s is the point of approximated PF, s^{nadir} is a $NadirPoint$, Λ is a Lebesgue measure, which is the generalization of a volume, $\prec$ is a domination relation.

Inverted Generative Distance ↓ (IGD) captures both convergence and diversity. It is an average distance from each TPF point to the closest point in PF as presented in Eq.16, where d_i is the Euclidean distance for the i-th point. Lower IGD values signify that solutions are closer to the ideal Pareto front. As objectives vary in scale, using absolute values for IGD calculation might favor certain objectives. For that reason, points in PF are normalized beforehand, using minimum and maximum values from the TPF.

$$IGD(PF, TPF) = \frac{\sqrt{\sum_{i=1}^{|TPF|} d_i^2}}{|TPF|} \tag{16}$$

IGD is a relative metric that uses TPF as a reference point. As the real TPF is unknown, it is constructed using results generated by all runs of all compared methods.

Purity ↑ defined as in Eq.17, where ND is the number of solutions (aggregated from all runs) not dominated by the "True Pareto Front approximation" (TPFa), where TPFa is constructed by merging a PFa from each method and removing dominated solutions. $Purity$ calculated for a single method returns the value from 0 to 1 and could be interpreted as the part of TPFa that the given method resulted in. However, the same points (solutions) can be found by different methods. Therefore the sum of $Purity$ for all investigated methods could exceed the value of 1.

$$Purity(PF, TPF) = \frac{|ND(PF, TPF)|}{|TPF|} \tag{17}$$

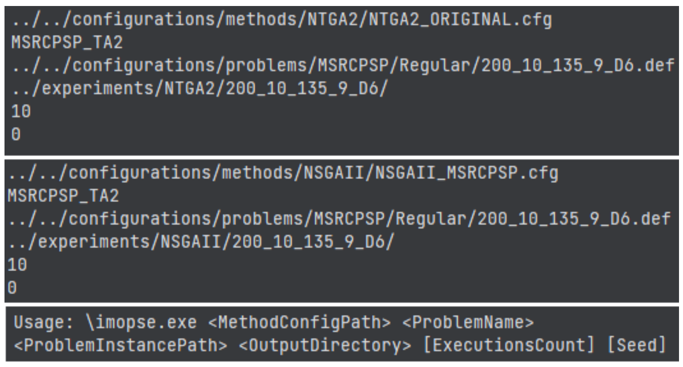
As each method is evaluated multiple times, the Analyzer repeats the process: merges the results returned by all the methods in the n'-th run, and calculates the $Purity$ per run. At the end, it averages the final $Purity$ value.

A crucial requirement for running the Analyzer is specifying the path to a configuration file as well as the instance name as a mandatory parameters. Configuration file must be prepared in advance and should list paths to the experiment output directory on separate lines. This structured format enables the Analyzer to systematically process and analyze the outcomes for each method listed for a specified instance, ensuring a thorough and organized evaluation process.

iMOPSE supports experiment **reproducibility** through the usage of seed parameters. The first experiment is conducted under a seed provided by the user, and for each next experiment, the seed is achieved by adding one to the initial seed value.

4.4 Case study - performing experiments with iMOPSE

In this section, we introduce an example experiment that users can conduct using iMOPSE, designed to demonstrate its capabilities and help users become acquainted with its operational workflow.
For the case study, we have selected the NSGAII and NTGA2 multi-objective optimization methods to find PFAs for the 200_10_135_9_D6 MS-RCPSP instance. For each method, the experiment will be repeated ten times. Thanks to iMOPSE being outfitted with pre-loaded instances and pre-configured methods, conducting experiments is straightforward. To examine two methods, users can execute the iMOPSE program twice by inputting different parameters (see Fig. 4), or they can utilize the *automated_experiments.py* script, which requires the path to the executable and in this case, two sets of previously mentioned input parameters to run the experiments.



```
../../configurations/methods/NTGA2/NTGA2_ORIGINAL.cfg
MSRCPSP_TA2
../../configurations/problems/MSRCPSP/Regular/200_10_135_9_D6.def
../experiments/NTGA2/200_10_135_9_D6/
10
0
```

```
../../configurations/methods/NSGAII/NSGAII_MSRCPSP.cfg
MSRCPSP_TA2
../../configurations/problems/MSRCPSP/Regular/200_10_135_9_D6.def
../experiments/NSGAII/200_10_135_9_D6/
10
0
```

```
Usage: \imopse.exe <MethodConfigPath> <ProblemName>
<ProblemInstancePath> <OutputDirectory> [ExecutionsCount] [Seed]
```

Fig. 4: iMOPSE input parameters for running NTGA2 and NSGAII

iMOPSE is designed to store the results of each run in a specified output directory. If the directory does not exist, it will automatically create one. Should the output directory already contain data from previous experiments, iMOPSE will halt its operation and notify the user, preventing any loss or accidental overwriting of experiment data due to an incorrect output directory path being provided.

The generated output data can subsequently be analyzed with the aid of additional Python scripts and the *Pareto Analyzer*. Nonetheless, users have the flexibility to employ alternative analysis software or methods according to their preferences.

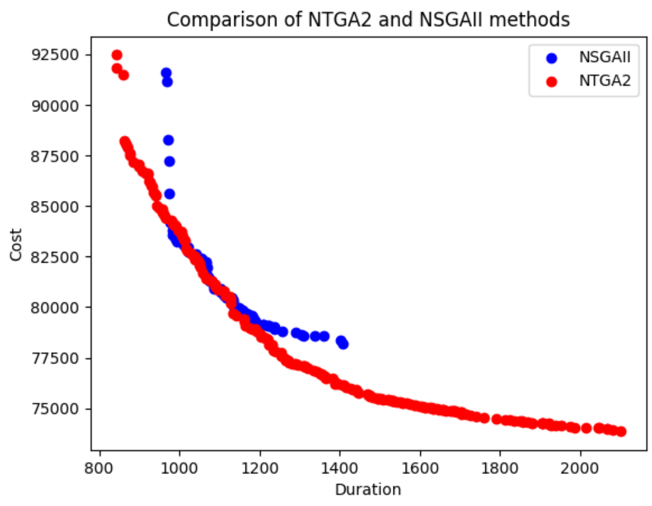


Fig. 5: PFAs comparison for NTGA2 and NSGAII (200_10_135_9_D6 MS-RCPSP)

In the context of our case study, we will utilize the *Pareto Analyzer* to compute metrics and generate a TPF approximation, which will be saved in the same output directory. To run the Pareto Analyzer, the user has to provide a path to the configuration file and the name of the examined instance, in this case, the configuration file contains paths to the output directories of analyzed methods. *Pareto Analyzer* merges results for each method and calculates TPF approximation by taking non-dominated solutions from all methods as reference. QMs acquired by *Pareto Analyzer* in this case study: NSGAII - $HV = 0.56 \pm 0.05$, $IGD = 0.02 \pm 0.003$, $Purity = 0.1$ and for NTGA2 - $HV = 0.76 \pm 0.01$, $IGD = 0.002 \pm 0.0007$, $Purity = 0.9$. The results show that NTGA2 generates approximately 90% of PFA and strongly dominates NSGA-II. Moreover, the PFAs can be visualized and compared by employing the *multi-objective_visualizer.py* script, facilitating a visual comparison of the multi-objective optimization outcomes (see Fig. 5).

5 Summary and future work

This article proposes a new open-source iMOPSE C++ library to support MS-RCPSP researchers, students, and practitioners. The iMOPSE library consists of methods for solving single- and multi-objective NP-hard combinatorial problems, especially for MS-RCPSP. Additionally, tools are added to validate and visualize MS-RCPSP results. However, the iMOPSE schema is flexible and could be extended easily by adding new methods and/or problems. Thus, the code of iMOPSE is open-source and published on GitHub [5]. The future directions of iMOPSE library development could be connected to support parallel computation. A multi-thread computation and a GPU-based computation of metaheuristics should be considered to speed up computations.

References

1. Myszkowski, P.B., Skowroński M.E., and Sikora K. "A new benchmark dataset for multi-skill resource-constrained project scheduling problem." 2015 Fed.Conf. on Comp. Sci. and Inf. Systems (FedCSIS). IEEE, 2015.
2. Myszkowski P.B., Laszczyk M., Nikulin I., and Skowroński M., "iMOPSE: a library for bicriteria optimization in Multi-Skill Resource-Constrained Project Scheduling Problem", Soft Computing vol.23, (2019), pp.3397–3410.
3. Myszkowski P.B., Laszczyk M., "Investigation of benchmark dataset for many-objective Multi-Skill Resource Constrained Project Scheduling Problem", Applied Soft Computing, Vol 127, (2022), 109253.
4. Myszkowski P.B., Skowroński M.E., Olech L., K Oślizło, "Hybrid ant colony optimization in solving multi-skill resource-constrained project scheduling problem", Soft Comp. 19 (12), 2015, pp.3599-3619.
5. –, http://imopse.ii.pwr.edu.pl – official homepage of iMOPSE project, includes MS-RCPSP resources, GitHub – https://github.com/imopse/iMOPSE
6. Myszkowski P.B., Siemienski J.J., "GRASP Applied to Multi–Skill Resource–Constrained Project Scheduling Problem", Inter. Conf. on Comp. Collective Intelligence, ICCCI 2016, pp.402–411.
7. Myszkowski, P.B., et al. "Hybrid differential evolution and greedy algorithm (DEGR) for solving multi-skill resource-constrained project scheduling problem." Applied Soft Computing 62 (2018): 1-14.
8. Myszkowski P.B., Kalinowski D., and Laszczyk M., "Co-Evolutionary Algorithm solving Multi-Skill Resource-Constrained Project Scheduling Problem", ACSIS 2017, Vol. 11, pages 75–82.
9. Myszkowski P.B., Laszczyk M., Lichodij J., "Efficient selection operators in NSGA-II for solving bi-objective multi-skill resource-constrained project scheduling problem", ACSIS 2017, Vol. 11, pp. 83–86
10. Laszczyk M., and Myszkowski P.B., "Improved selection in evolutionary multi–objective optimization of multi–skill resource–constrained project scheduling problem.", Information Sciences 481 (2019): 412-431.
11. Myszkowski, P.B., and Laszczyk M., "Diversity based selection for many-objective evolutionary optimisation problems with constraints." Inf. Sci. 546 (2021): 665-700.
12. Laszczyk M., and Myszkowski P.B. "Survey of quality measures for multi–objective optimisation. Construction of complementary set of multi-objective quality measures." Swarm and Evolutionary Computation, 2019
13. Hartmann S., Briskorn D., "An updated survey of variants and extensions of the resource-constrained project scheduling problem", European Journal of Operational Research (2022), 297(1), pp.1-14.
14. Verma S., Snasel V., "A Comprehensive Review on NSGA-II for Multi-Objective Combinatorial Optimization Problems", IEEE Access (2021) vol.9., pp. 57757–57791.

15. Zheng Xiaolong, Wang Ling, Zheng Huany, ”A knowledge-based fruit fly optimization algorithm for multi-skill resource-constrained project scheduling problem“, 2015 34th Chinese Control Conf. (CCC), pp.2615–2620.
16. Huan-yu Zheng, Ling Wang, Xiao-long Zheng, ”Teaching–learning-based optimization algorithm for multi–skill resource constrained project scheduling problem“, Soft Computing 21, 2017, pp.1537–1548.
17. Jian Lin, Lei Zhu, Kaizhou Gao, ”A genetic programming hyper-heuristic approach for the multi-skill resource constrained project scheduling problem“, Expert Systems with Applications, vol.140 (2020), 112915.
18. M.Antkiewicz, P.B.Myszkowski, Balancing Pareto Front exploration of Non-dominated Tournament Genetic Algorithm (B-NTGA) in solving multi-objective NP-hard problems with constraints, Information Sciences, (2024), Volume 667.
19. Z.T.Kosztyán, P.Harta, I.Szalkai, The effect of autonomous team role selection in flexible projects, Computers & Industrial Engineering, vol 190, (2024), 110079,
20. J.Snauwaert, M.Vanhoucke, A classification and new benchmark instances for the multi-skilled resource-constrained project scheduling problem, European Journal of Oper. Research, Vol 307 (1), (2023), pp.1-19.
21. Ling Wang, Xiao-long Zheng, A knowledge-guided multi-objective fruit fly optimization algorithm for the multi-skill resource constrained project scheduling problem, Swarm Evol. Comput. 38 (2018) 54–63.
22. Lei Zhua, Jian Lin, Yang-Yuan Li, Zhou-Jing Wang, ”A decomposition-based multi-objective genetic programming hyper-heuristic approach for the multi-skill resource constrained project scheduling problem”, Knowl.-Based Syst. 225 (2021) 107099.
23. Deb, K. and Pratap, A. and Agarwal, S. and Meyarivan, T., A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. on Evol. Comp. (2002).
24. Q.Zhang and H. Li, MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition, IEEE Trans. on Evol. Comp., vol 11(6), pp.712-731, (2007).
25. V.J.Amuso and J.Enslin, The Strength Pareto Evolutionary Algorithm 2 (SPEA2) applied to simultaneous multi-mission waveform design, 2007 Inter. Waveform Diversity and Design Conf., (2007), pp.407-417.

A Simulated Annealing Approach to the Multi-Activity Multi-Day Shift Scheduling Problem

László Kálmán Trautsch$^{1\,[0000-0002-1589-3265]}$ and Bence Kovari$^{2\,[0000-0003-1555-640X]}$

[1] Department of Automation and Applied Informatics, Faculty of Electrical Engineering and Informatics, Budapest University of Technology and Economics, Műegyetem rkp. 3., H-1111 Budapest, Hungary. trautschl@edu.bme.hu
[2] Department of Automation and Applied Informatics, Faculty of Electrical Engineering and Informatics, Budapest University of Technology and Economics, Műegyetem rkp. 3., H-1111 Budapest, Hungary. kovari@aut.bme.hu

Abstract. This paper addresses the multi-activity multi-day shift scheduling problem with a homogeneous workforce and quadratic cost function for overstaffing. The objective of this problem is to assign shifts to employees and activities within these shifts based on short time intervals, respecting numerous hard constraints and minimizing overstaffing. We propose a multi-neighborhood Simulated Annealing algorithm as a solution method, for which we introduce eight neighborhood relations. The search space and neighborhood relations are designed so that the search algorithm can be executed efficiently even on large problem instances. The method is evaluated on a benchmark dataset consisting of problem instances with varying complexity. The results show that our approach can handle even the most complex tasks and is able to find feasible solutions for 201 out of the 225 total problem instances, of which 99 were previously unsolved. Our method outperforms the solver that produced the previous best known solutions for the benchmark dataset and finds new best solutions for 190 of the instances. The algorithm can create good schedules in a matter of a few seconds, using limited computing resources.

Keywords: Shift Scheduling, Multi-Activity, Simulated Annealing

1 Introduction

The multi-activity shift scheduling problem occurs in many industries, particularly in the service sector, commonly in retail environments. Making effective use of the workforce while satisfying various organizational and social constraints is an important task that could yield substantial cost savings. In these problems, an activity represents an interruptible operation, which can be assigned to several employees at the same time. There is a minimum required workforce for the activities at each period, to ensure an acceptable service quality. This demand may fluctuate throughout the planning horizon.

Personnel scheduling has been a widely studied problem in the literature for a long time, as shown by the numerous references given in the surveys of Ernst et al. [1, 2]. On the other hand, multi-activity shift scheduling problems were relatively under-studied until recently. One of the earliest works addressing this problem was by Loucks and

Jacobs [3]. Since then, different versions of the problem emerged with vastly different constraints, and various solving methods have been employed for solving these. Some works consider anonymous shifts, where it is not specified which employee will be assigned to a given shift. Dahmen, Rekik and Soumis [4] proposed an implicit model for this task. Other works address a variation of the problem in which interruptible activities and uninterruptible tasks should be scheduled at the same time. Lequy, Desaulniers and Solomon [5] used a two-stage heuristic for this problem. Solving shift construction and activity assignment simultaneously on a multi-day planning horizon is a challenging task, which to the best of our knowledge, has been addressed only by a few papers. The qualification to perform certain activities can also differ within the workforce in some problems, such as in the work of Dahmen and Rekik [6], where they proposed a hybrid heuristic for solving a multi-activity multi-day shift scheduling problem with a heterogenous workforce. Most recently a mathematical programming-based approach has been used for variants of multi-activity shift scheduling problems with anonymous shifts by Römer [7], who proposed block-based state-expanded network models.

This paper addresses the multi-activity shift scheduling problem with a homogeneous workforce in a multi-day environment as described in the formal description [8] of the associated benchmark problem [9]. The task is to assign shifts to employees on the given days, and to schedule the shifts and the activities within them, based on short time intervals, in a way that respects all the various hard constraints. The cost function in this problem is the quadratic penalization for overstaffing at each period for every activity. There is one existing work addressing this exact problem, by Qu and Curtois [10], in which they use Variable Neighborhood Search as a solution method.

We propose a multi-neighborhood Simulated Annealing approach for this problem. Simulated Annealing was first introduced by Kirkpatrick, Gelatt and Vecchi [11], and since then it has been successfully applied for many scheduling tasks in the literature, such as for sports timetabling [12], nurse rostering [13], course timetabling [14, 15] and most recently examination timetabling [16]. Our proposed approach for the multi-activity multi-day shift scheduling problem is based on a mathematical model which enables the efficient inspection of the various hard and soft constraints, and our introduced eight different neighborhood relations allow for an effective traversal of the state space.

The organization of this paper is as follows. Section 2 overviews the multi-activity multi-day shift scheduling problem addressed in this paper. Section 3 describes the proposed local search method and the proposed neighborhood relations in detail. In Section 4, we report and discuss the experimental results obtained on the benchmark dataset. Section 5 provides concluding remarks and future plans.

2 Problem Definition

This paper addresses the Multi-Activity Multi-Day Shift Scheduling Problem, as described in the formal description [8]. The mathematical model of the problem with the formalization of the exact hard and soft constraints are available in the formal description. For clarity, we briefly summarize the key aspects of the problem. The goal is to assign shifts to employees on the given days, and activities within these shifts. An employee can work on one or more tasks during a shift, therefore activities should be scheduled within the shifts, each with an assigned task. In this context, activities and tasks are considered equivalent. Therefore, when we refer to an activity, we are indicating the duration during which an employee works on one of the specified tasks. An employee must work on exactly one task at each interval of a shift, which means that there can be no overlap between the different activities. Employees are assumed to be homogeneous in the sense that they are all qualified to perform any of the different tasks. The planning horizon is divided into 15-minute intervals and the scheduling has to be done based on these time slots. The planning horizon always starts at 6:00 a.m. on the first day and finishes at 6:00 a.m. on the last day. Therefore, if the planning horizon is 7 days long, then it runs from 6:00 a.m. on day 1 to 6:00 a.m. on day 8.

There are various hard constraints for the problem, all of which must be respected for a schedule to be considered feasible. An employee cannot start more than one shift on a day, and a shift can only start at one of the time intervals between the following times on each day: 0:00-0:00, 6:00-10:00, 14:00-18:00 and 20:00-23:45. Each shift duration should be between 6 and 10 hours. After a shift finishes, an employee cannot start another shift until at least 14 hours later. An employee cannot start shifts on more than 5 consecutive days, in other words at least one day off must be taken on each 6 consecutive days. There are no limitations on the number of activities a shift can hold or on the number of activity changes within a shift, however, every activity must be at least 1 hour long before an activity change occurs or the shift ends.

In the different problem instances, it is specified for each employee how many total minutes that employee should work at minimum and at maximum during the whole planning horizon. The minimum cover requirement is also specified for each task at each time interval, which is the minimum number of required staff to work on that task at that interval.

The objective is to minimize assigning more staff than the maximum specified for each task at each time interval. When there is overstaffing for a given task at a specific interval, the penalty is the squared difference between the maximum required number of staff and the actual number of staff. The total cost of a solution is the sum of all the penalties for every time interval and task. Thus, the cost function is quadratic to ensure that overstaffing is spread out over the planning horizon rather than occurring in a small number of tasks and intervals, as the penalty for each additional unit of overstaffing for a task at an interval increases more rapidly than linearly.

3 Solution Method

Our solution method is based on the Simulated Annealing [11] local search, for which we designed a multi-neighborhood consisting of eight different neighborhoods. The key components of the proposed method are described in this section.

3.1 Search Space

A state in the search space is the direct representation of all the shifts and activities assigned to each employee, with their respective schedules based on the time intervals

Table 1: Decision variables

Symbol	Definition
$s_{h,e} \in \{0, \ldots, \lvert H\rvert * \lvert E\rvert - 1\}$	Shift index of employee e on the h-th 24-hour period of the planning horizon. $\lvert H\rvert$ is the total number of 24-hour periods and $\lvert E\rvert$ is the total number of employees in the given problem instance.
$b_s \in \mathbb{Z}^+$	Start time of shift s.
$a_s \in \{0, \ldots, n\}$	Activity count of shift s. The maximum number of activities per shift (n) is a selectable parameter.
$l_{s,a} \in \mathbb{Z}^+$	Length of the a-th activity of shift s.
$t_{s,a} \in \{0, \ldots, \lvert T\rvert\}$	Task of the a-th activity of shift s. $\lvert T\rvert$ is the total number of tasks in the given problem instance.
Auxiliary variables	
$w_e \in \mathbb{Z}^+$	Workload of employee e.
$c_{t,i} \in \mathbb{Z}^+$	Cover of task t at time interval i.
$f_{d,e} \in \{0, 1, 2\}$	Count of non-empty shifts of employee e on day d.

of the planning horizon. The decision variables of our model with their descriptions are shown in Table 1.

Although there are demand requirements for each time interval of the planning horizon, we do not directly model the assignment of employees at each interval, as this would imply an unnecessarily large model for our approach, because only the sum of the workforce is relevant at each interval. Rather, we use variables to specify which workers are assigned to which shifts at the given 24-hour periods, when these shifts start, how many activities the shifts contain, how long these activities are, and which tasks are assigned to them. Given these variables, a complete schedule can be composed, and all the different constraints can be inspected. To enable the efficient inspection of the various constraints, different auxiliary variables can be introduced, offering the necessary aggregated information directly. Our key auxiliary variables are presented in Table 1.

In our model, each employee must have one shift assigned to them at each 24-hour period, but a shift can be empty meaning that it should be ignored from the complete schedule and the relevant employee does not start a working shift at that period. A shift must start at one of the time intervals of the relevant 24-hour period, but it can extend beyond that period. We base our shifts on 24-hour periods of the planning horizon rather than on days, because this way fewer variables are needed for modeling the shifts, and all the shifts can be handled uniformly. If shifts were created for each day, then the start and the end of the planning horizon would cut into the shifts on the first and the last day respectively, making allocation to time intervals completely different on those days. The first 24-hour period starts at the beginning of the planning horizon, which is at 6:00 a.m. on the first day, and ends on the next day at 6:00 a.m. The number of 24-hour periods is one less than the number of days in a problem instance.

The problem constraints can be reformulated for our decision variables with a straightforward matching between them. The only difficulty emerges because a shift can start at 0:00 on a given day, which time interval is part of the 24-hour period of the previous day. Thus, two shifts could start on the same day, which is a constraint violation. In order to restrict this, we modified the length of the minimum rest time to 24 hours for a shift starting at 0:00. The auxiliary variables for the number of non-empty shifts of each employee on each day are created to help the inspection of the constraint on the number of consecutive working shifts. An example of two shifts of an employee starting on the same day would be that the first shift starts at 0:00 and the second one starts at 20:00. In that case the corresponding "$f_{d,e}$" value would be 2, indicating that two shifts start on that day for the employee.

To make the search space more connected, certain hard constraints are relaxed and made soft constraints, but with high weights applied to the cost for violating them. Thus, the cost function of a state in the search space is the sum of the cost induced by the soft and the hard constraints. The actual weights of the hard constraints are set by parameters associated with them. The workload constraints, the minimum cover constraint, the maximum number of consecutive shifts constraint, the minimum rest time constraint, and the constraints regarding the length of shifts and activities are relaxed. Linear cost functions are used, except for the minimum and maximum shift length constraints, for which the deviation from the minimum and maximum length is penalized quadratically.

A parameter can be set to control the maximum number of activities per shift. The variables associated with the activities are created based on this parameter, for each shift as many as the parameter specifies. Based on the maximum shift length and minimum activity length hard constraints, at maximum 10 activities per shift are needed to create any feasible solution. A higher value can be set for this parameter if we want the search algorithm to move more freely in the search space by adding more activities, but the problem constraints need to be modified in this case. A value lower than 10 can speed up the search for problem instances with few tasks, although finding the optimal solution might become theoretically impossible.

The activity count variable specifies how many activities are actually relevant from all the activities of a shift. When the activity count of a shift is lower than the maximum number of activities per shift, that means that the following activities are empty, and their tasks and lengths should be ignored. A shift is empty when its activity count is zero.

3.2 Initial Solution

For the initial solution of the search, an empty schedule is created based on the number of days and employees of the given problem instance, where each employee has an empty shift assigned to them on each 24-hour period of the planning horizon. The activity count of each shift is zero, which means that the other decision variables associated with the shift are irrelevant until an activity is assigned by a move from the neighborhood relations. The auxiliary variables also have zero values in this initial state. The solution is infeasible, and the total cost is calculated and used as the initial cost. This empty schedule is used as the initial state, which is populated with working shifts by the neighborhood relations during the search.

3.3 Neighborhood Relations

We propose a multi-neighborhood on the previously described search space, composed of the union of eight neighborhoods:

- **AddShiftActivity:** A random empty shift is selected, and an activity is added to it with a random task. A random start is also assigned to the shift and a random length to the activity. The start of the shift is selected from the valid shift start times of the day. The length is selected from the possible shift lengths that do not extend beyond the planning horizon, given the already selected shift start time.
- **RemoveShiftActivities:** A random non-empty shift is selected, and all its activities are removed.
- **ChangeShiftStart:** A random non-empty shift is selected, and its start is changed to a different, random start. The start is selected from those valid shift start times of the day that would not make the shift extend beyond the end of the planning horizon, given its current length.
- **SwapShifts:** A random non-empty shift is selected, and its assignment is swapped between its original employee and the employee of an other random shift from the same day. The other shift can be either empty or not.
- **ChangeActivityLength:** A random non-empty activity is selected, and its length is changed to a different, random length. The length is selected so that the shift would not extend beyond the planning horizon, and the length of the shift up to the end of the selected activity would not be longer than the maximum shift length and shorter than the minimum shift length. The minimum activity length is also respected when the previous criteria enable it.
- **ChangeActivityTask:** A random non-empty activity is selected, and its task is changed to a different, random task.
- **AddLastActivity:** A random non-empty and non-full shift is selected, and a new activity is added to its end with a random task, which is different than the task of the previous activity. A random length is also assigned to the activity, and it is selected so that the shift would not extend beyond the planning horizon, and the shift would not be longer than the maximum shift length. When the previous criteria enable it, the minimum activity length is also respected.
- **RemoveLastActivity:** A random shift is selected from the shifts that have at least two activities, and the last activity of that shift is removed.

At each iteration step of the search, one of the eight neighborhood types is selected with probabilities specified by associated parameters, then a move is randomly drawn from the selected neighborhood. When the random selection of a variable cannot be made during a move, the move is instantly rejected. For example, if there are no shifts with at least one activity assigned to them, then moves from the *RemoveShiftActivities* neighborhood are rejected.

3.4 Simulated Annealing

As the metaheuristic to guide the search, we implemented the Simulated Annealing algorithm [11]. The method starts from an initial random state and at each iteration selects a random move from its neighborhood as explained above. Calling Δf the change in cost induced by the selected move, the move is always accepted if $\Delta f \leq 0$, and it is accepted with probability (1) when $\Delta f > 0$, where T_a is the temperature parameter controlled by the algorithm.

$$e^{-\Delta f/T_a} \tag{1}$$

We implemented the Fast Simulated Annealing [17] cooling scheme to determine the temperature at each iteration, based on the number of the current iteration (t) and the initial temperature (T_0), as described by equation (2).

$$T_a(t) = \frac{T_0}{(1+t)} \tag{2}$$

The search is repeated for a set number of iterations, which number is a parameter of the metaheuristic.

3.5 Efficient Implementation

The neighborhood relations and decision variables were designed to enable efficient implementation of the search algorithm, so that a high number of iterations could be executed even on large problem instances. The change in cost induced by new candidate moves should be calculated only based on the constraints and variables directly relevant to the actual neighborhood type and the exact move, and the state should be modified only if the move is accepted. The proposed auxiliary variables are used for inspecting the relevant constraints of the actual neighborhood relation. Shift indexes were introduced for achieving low computational complexity when executing a *SwapShifts* move between two employees. We also used other auxiliary variables and structures to help select random variables and calculate the changes in cost during the search, but these are not reported in this paper for the sake of brevity.

4 Experimental Results

4.1 Problem Instances

The algorithm was tested on the instances of the publicly available multi-activity shift scheduling benchmark dataset [9]. The benchmark contains 225 different problem instances, with varying difficulty. There are instances with lengths of 7, 14, and 28 days. The number of staff varies from 10 to 150, and the number of tasks varies from 1 to 19. The problem size tends to increase with the instances. The features of the instances are shown in Table 4. in the Appendix. It is known that every instance has a feasible solution, due to the way the instances were created [10].

4.2 Parameter Settings

A single parameter configuration was tested during the experiments on the different problem instances, which is shown in Table 2. The table includes the parameters for the Simulated Annealing metaheuristic and the weights assigned to the various hard constraints.

Table 2. Parameter configuration

Parameter	**Assigned value**
Initial temperature	800,000
Number of iterations	10,000,000
Maximum number of activities per shift	10
Each neighborhood relation probability	0.125
Weight for minimum rest time constraint	15,000,000
Weight for minimum workload constraint	1,500,000
Weight for maximum number of consecutive working days constraint	1,000,000
Weight for shift length constraint	225,000
Weight for maximum workload constraint	150,000
Weight for minimum activity length constraint	150,000
Weight for minimum activity cover demand constraint	10,000

The hard constraint weights are based on the problem instance files in XML format found in the benchmark dataset, except for the weight for violating the minimum rest time, for which we assigned a weight higher than the others. The neighborhood relation probabilities were selected uniformly. The number of iterations was set so that the runtime of the search on even the hardest problem instance would take no longer than 5 seconds. The initial temperature was chosen intuitively, based on trial runs on the hardest problem instance. It is important to note that the presented method could significantly benefit from parameter tuning, and employing a different cooling scheme or stopping criterion might further improve the results.

4.3 Experimental Setup

The solution method was implemented in C++ and compiled using g++. The experiments were run on a machine with 16 GB of RAM and a 3.3 GHz Intel Core i5-4590 processor, using a single core during the tests. The number of iterations was selected so that the search on each instance would take no longer than 5 seconds. A single run of our solution method was performed on each problem instance of the dataset.

4.4 Results

The results of our solution method on each problem instance are shown in Table 4. in the Appendix. We compare our results achieved by Simulated Annealing (SA) to the solver that produced the existing best known solutions for the benchmark dataset, the method by Qu and Curtois [10], which uses Variable Neighbourhood Search (VNS). The best result found for a problem instance is highlighted in bold and underlined. Only feasible solutions are reported, in which none of the hard constraints are violated. A cell contains "-" if no feasible solution was found by a method under its time limit.

The authors of the VNS method used a time limit of 10 minutes for their experiments on each instance, and they conducted their tests on a comparable machine (Intel Core i5-4690K CPU 3.50GHz) to the one used in our tests.

For our solution method, we selected the number of iterations so that the runtime of the search on each instance would not take longer than 5 seconds. The actual runtimes ranged from 2.079 seconds on the simplest instance to 4.073 seconds on the most complex one. It should be noted that the runtime scales well with the problem complexity when using a fixed number of iterations for the search. Less than twice as much time was needed for a problem instance with a four times longer planning period, fifteen times as many staff, and nineteen times as many tasks, and it should be also considered that the simplest instances had one task to be scheduled, meaning that moves from three neighborhood types were always rejected in those cases, reducing the runtime. The memory usage of the algorithm was also efficient, less than 4 MB of memory was needed during the searches.

Table 3. shows an overview of the results on the benchmark dataset, comparing our approach to the VNS method. Simulated Annealing was able to find feasible solutions for most of the problem instances (201 out of the 225 total), of which 190 are the best solutions found so far. It was able to find feasible solutions for many previously unsolved problem instances, even for the most complex ones.

Table 3. Overview of the comparative results on the benchmark dataset

	VNS [10]	**SA**
Time limit	10 minutes	5 seconds
Number of instances for which feasible solutions are found	107	**<u>201</u>**
Number of instances for which best solutions are found	16	**<u>190</u>**

On the other hand, our approach could not find feasible solutions for some simpler instances, 5 of which have been solved by VNS. The size of a problem instance does not seem to affect finding a feasible solution for the Simulated Annealing. The search randomly gets stuck in an infeasible local optimum in some cases, which could be due to the selected cooling scheme or the parameters of the metaheuristic. Only a single parameter configuration with equal probabilities for all neighborhood types was tested, which implies that applying parameter tuning could greatly improve the results. Improving the neighborhood relations and introducing new ones could also help escaping the local optima, and different selection of weights for the hard constraints should also be inspected. A single search was conducted on each problem instance, but the method could benefit from selecting the best solution from more searches, run both in parallel and by applying restarts.

The reported solutions were all validated using a verification software, which is available for the benchmark dataset [9]. The software can be used to view and verify the solutions created for the problem instances. It is able to identify any hard constraint violations and calculate the cost function of a complete schedule. The accuracy of our new computational results was ensured by using this validation.

5 Conclusions and Future Work

In this paper, we presented a multi-neighborhood Simulated Annealing method for addressing a multi-activity multi-day shift scheduling problem. We introduced eight different neighborhood relations for the search algorithm. We tested our approach on a benchmark dataset and compared the results with the best existing solution available for the problem. Our method was able to outperform the previously developed algorithm on most of the problem instances and was able to find feasible solutions for many of the unsolved ones. The results show that our approach can produce good schedules in a matter of a few seconds, using limited computing resources. The method would be able to find solutions for even larger problems than the most complex instances of the benchmark dataset, as the results suggest. However, in some cases, it was not able to produce feasible solutions even for some simpler instances, therefore there is still room for improvement.

As part of our future work, first, we will investigate the possibility of improving our proposed multi-neighborhood by introducing new types of relations and modifying the existing ones. Secondly, we plan to configure our algorithm by using automated parameter tuning to find better values for the probabilities of the neighborhood types, the hard constraint weights, the initial temperature, and finally for the maximum number of activities within the shifts. Applying a different cooling scheme and stopping criterium might also improve the results. We will conduct further evaluations of our approach, including experiments with longer runtimes, enabling the restart of the search method multiple times on each problem instance, from which the best solution can be selected. We also plan to extend our method to allow running searches on multiple threads in parallel. Finally, we will create an iterative procedure for populating the initial state of the search. This procedure would take every constraint into account for creating a good initial solution in a short timeframe, thus speeding up the search method and possibly enabling the production of better solutions.

Acknowledgements This research was supported by the Ministry of Culture and Innovation and the National Research, Development and Innovation Office within the Cooperative Technologies National Laboratory of Hungary (Grant No. 2022- 2.1.1-NL-2022-00012).

Appendix

Table 4. Features of the problem instances and comparative results

Inst.	**Days**	**Staff**	**Tasks**	**VNS [10]**	**SA**
1	7	10	1	387	**383**
2	7	10	1	176	**140**
3	7	10	1	317	**290**
4	7	10	1	328	**304**
5	7	10	2	115	**37**
6	7	20	1	900	**779**
7	7	20	1	818	**783**
8	7	20	2	884	**775**
9	7	20	2	500	**353**
10	7	20	3	268	**59**
11	7	30	1	844	**788**
12	7	30	2	1541	**1501**
13	7	30	2	**1440**	-
14	7	30	3	**1469**	-
15	7	30	4	553	**270**
16	7	40	2	1883	**1580**
17	7	40	2	1831	**1713**
18	7	40	3	1737	**1457**
19	7	40	4	1437	**1034**
20	7	40	5	955	**457**
21	7	50	2	1740	**1647**
22	7	50	3	2646	**2596**
23	7	50	4	2446	**2115**
24	7	50	5	1795	**1395**
25	7	50	7	1344	**758**
26	7	60	2	1734	**1594**
27	7	60	3	2904	**2622**
28	7	60	4	3248	**2836**
29	7	60	6	2463	**1918**
30	7	60	8	-	**1121**
31	7	70	3	2574	**2466**
32	7	70	4	3288	**3182**
33	7	70	5	3170	**3025**
34	7	70	7	-	-
35	7	70	9	-	**1386**
36	7	80	3	2709	**2536**
37	7	80	4	**3335**	3422
38	7	80	6	3894	**3610**
39	7	80	8	-	**2709**
40	7	80	10	-	**1787**
41	7	90	3	**2575**	2643
42	7	90	5	4317	**4302**
43	7	90	6	4877	**4463**
114	14	80	8	-	**5799**
115	14	80	10	-	**4006**
116	14	90	3	5598	**5214**
117	14	90	5	8818	**7985**
118	14	90	6	-	**8663**
119	14	90	9	-	**5399**
120	14	90	12	-	**3175**
121	14	100	4	-	**6946**
122	14	100	5	-	**9009**
123	14	100	7	-	**9779**
124	14	100	10	-	**7867**
125	14	100	13	-	**4061**
126	14	110	4	7573	**7151**
127	14	110	6	-	**9879**
128	14	110	8	-	**10015**
129	14	110	11	-	**7898**
130	14	110	14	-	**3758**
131	14	120	4	7475	**6877**
132	14	120	6	-	**11057**
133	14	120	8	-	**11847**
134	14	120	12	-	**7340**
135	14	120	15	-	-
136	14	130	5	-	**8764**
137	14	130	7	-	**12460**
138	14	130	9	-	**11958**
139	14	130	13	-	**7345**
140	14	130	17	-	**5093**
141	14	140	5	**8013**	8859
142	14	140	7	-	**12725**
143	14	140	10	-	**13013**
144	14	140	14	-	**9022**
145	14	140	18	-	**5706**
146	14	150	5	-	**8763**
147	14	150	8	-	**14321**
148	14	150	10	-	**14824**
149	14	150	15	-	-
150	14	150	19	-	**6012**
151	28	10	1	1677	**1486**
152	28	10	1	1509	**1341**
153	28	10	1	1729	**1597**
154	28	10	1	1535	**1299**
155	28	10	2	-	**255**
156	28	20	1	3766	**3565**

				VNS						VNS	
Inst.	**Days**	**Staff**	**Tasks**	**[10]**	**SA**	**Inst.**	**Days**	**Staff**	**Tasks**	**[10]**	**SA**
44	7	90	9	-	**3109**	157	28	20	1	3523	**3312**
45	7	90	12	-	**1539**	158	28	20	2	3327	**2663**
46	7	100	4	**3471**	3635	159	28	20	2	2989	**2071**
47	7	100	5	**4837**	-	160	28	20	3	1803	**696**
48	7	100	7	5302	**4331**	161	28	30	1	3505	**3264**
49	7	100	10	-	**3662**	162	28	30	2	6551	**5499**
50	7	100	13	-	**2171**	163	28	30	2	6209	**5370**
51	7	110	4	**3338**	3553	164	28	30	3	-	**4163**
52	7	110	6	**5084**	5460	165	28	30	4	-	-
53	7	110	8	6237	**4980**	166	28	40	2	7613	**7173**
54	7	110	11	-	**3388**	167	28	40	2	7317	**7177**
55	7	110	14	-	**2694**	168	28	40	3	8270	**6710**
56	7	120	4	3486	**3410**	169	28	40	4	-	**5940**
57	7	120	6	5991	**5267**	170	28	40	5	-	**2960**
58	7	120	8	6749	**5931**	171	28	50	2	**6843**	-
59	7	120	12	-	**4643**	172	28	50	3	-	**8896**
60	7	120	15	-	**2714**	173	28	50	4	-	**7765**
61	7	130	5	4932	**4485**	174	28	50	5	-	**6374**
62	7	130	7	6720	**6366**	175	28	50	7	-	**3293**
63	7	130	9	7086	**6264**	176	28	60	2	7179	**6861**
64	7	130	13	-	**4449**	177	28	60	3	-	-
65	7	130	17	-	-	178	28	60	4	-	-
66	7	140	5	**4057**	4432	179	28	60	6	-	**8477**
67	7	140	7	**6009**	6370	180	28	60	8	-	**4513**
68	7	140	10	-	**6719**	181	28	70	3	-	**10393**
69	7	140	14	-	**4462**	182	28	70	4	-	-
70	7	140	18	-	**2685**	183	28	70	5	-	**13913**
71	7	150	5	**4063**	4419	184	28	70	7	-	**10329**
72	7	150	8	7590	**7367**	185	28	70	9	-	**5941**
73	7	150	10	-	**7330**	186	28	80	3	11181	**10544**
74	7	150	15	-	**5493**	187	28	80	4	-	**14658**
75	7	150	19	-	**2955**	188	28	80	6	-	**15811**
76	14	10	1	598	**550**	189	28	80	8	-	**10153**
77	14	10	1	814	**775**	190	28	80	10	-	**6690**
78	14	10	1	634	**581**	191	28	90	3	-	**10314**
79	14	10	1	607	**509**	192	28	90	5	-	-
80	14	10	2	292	**88**	193	28	90	6	-	**16767**
81	14	20	1	1659	**1580**	194	28	90	9	-	**13170**
82	14	20	1	1643	**1561**	195	28	90	12	-	**5338**
83	14	20	2	1387	**1053**	196	28	100	4	-	**14039**
84	14	20	2	1168	**906**	197	28	100	5	-	-
85	14	20	3	520	**123**	198	28	100	7	-	**20037**
86	14	30	1	1738	**1725**	199	28	100	10	-	**13458**

				VNS						VNS	
Inst.	**Days**	**Staff**	**Tasks**	**[10]**	**SA**	**Inst.**	**Days**	**Staff**	**Tasks**	**[10]**	**SA**
87	14	30	2	2672	**2541**	200	28	100	13	-	-
88	14	30	2	2780	**2539**	201	28	110	4	-	**14678**
89	14	30	3	**2551**	-	202	28	110	6	-	-
90	14	30	4	-	**1145**	203	28	110	8	-	**22346**
91	14	40	2	3514	**3324**	204	28	110	11	-	**15528**
92	14	40	2	3767	**3588**	205	28	110	14	-	**8984**
93	14	40	3	3820	**3232**	206	28	120	4	-	**14038**
94	14	40	4	3980	**3417**	207	28	120	6	-	**22210**
95	14	40	5	-	**1264**	208	28	120	8	-	-
96	14	50	2	3666	**3390**	209	28	120	12	-	**15592**
97	14	50	3	4921	**4278**	210	28	120	15	-	**12832**
98	14	50	4	4802	**4095**	211	28	130	5	-	**17786**
99	14	50	5	-	**3602**	212	28	130	7	-	-
100	14	50	7	-	**947**	213	28	130	9	-	**25974**
101	14	60	2	3419	**3327**	214	28	130	13	-	**15203**
102	14	60	3	5473	**5309**	215	28	130	17	-	-
103	14	60	4	5942	**5914**	216	28	140	5	-	**18010**
104	14	60	6	5620	**4278**	217	28	140	7	-	-
105	14	60	8	-	-	218	28	140	10	-	**25463**
106	14	70	3	**5137**	5170	219	28	140	14	-	**19802**
107	14	70	4	6892	**6546**	220	28	140	18	-	-
108	14	70	5	-	**5705**	221	28	150	5	-	-
109	14	70	7	-	**4684**	222	28	150	8	-	**29983**
110	14	70	9	-	**3434**	223	28	150	10	-	**28523**
111	14	80	3	5510	**5310**	224	28	150	15	-	**19259**
112	14	80	4	**6748**	7113	225	28	150	19	-	**13429**
113	14	80	6	8124	**7034**						

References

1. Ernst, A.T., Jiang, H., Krishnamoorthy, M., Owens, B., Sier, D.: An annotated bibliography of personnel scheduling and rostering. Annals of Operations Research 127(1), 21–144 (2004).
2. Ernst, A.T., Jiang, H., Krishnamoorthy, M., Sier, D.: Staff scheduling and rostering: A review of applications, methods and models. European Journal of Operational Research 153(1), 3–27 (2004).
3. Loucks, J.S., Jacobs, F.R.: Tour scheduling and task assignment of a heterogeneous work force: A heuristic approach. Decision Sciences 22(4), 719–738 (1991).
4. Dahmen, S., Rekik, M., Soumis, F.: An implicit model for multi-activity shift scheduling problems. Journal of Scheduling 21(3), 285–304 (2018).
5. Lequy, Q., Desaulniers, G., Solomon, M.M.: A two-stage heuristic for multi-activity and task assignment to work shifts. Computers & Industrial Engineering 63(4), 831–841 (2012).

6. Dahmen, S., Rekik, M.: Solving multi-activity multi-day shift scheduling problems with a hybrid heuristic. Journal of Scheduling 18(2), 207-223 (2015).
7. Römer, M: Block-based state-expanded network models for multi-activity shift scheduling. Journal of Scheduling, 1-21 (2023).
8. Qu, Y., Curtois, T.: Multi-Activity, Multi-Day Shift Scheduling Problem Definition, http://www.schedulingbenchmarks.org/matsp/ProblemDefinition.pdf, last accessed 2024/03/13.
9. Multi-activity, Multi-day Shift Scheduling Benchmark Instances, http://www.schedulingbenchmarks.org/matsp/, last accessed 2024/03/13.
10. Qu, Y., Curtois, T.: Solving the Multi-Activity Shift Scheduling Problem Using Variable Neighbourhood Search. In: Proceedings of the 9th International Conference on Operations Research and Enterprise Systems, pp. 227-232. SCITEPRESS, Valetta (2020).
11. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by Simulated Annealing. Science 220(4598), 671-680 (1983).
12. Rosati, R.M., Petris, M., Di Gaspero, L., Schaerf, A.: Multi-neighborhood simulated annealing for the sports timetabling competition ITC2021. Journal of Scheduling 25(2), 1-19 (2022).
13. Ceschia, S., Guido, R., Schaerf, A.: Solving the static inrc-ii nurse rostering problem by simulated annealing based on large neighborhoods. Annals of Operations Research 288(1), 95–113 (2020).
14. Cambazard, H., Hebrard, E., O'Sullivan, B., Papadopoulos, A.: Local search and constraint programming for the post enrolment-based timetabling problem. Annals of Operational Research 194(1), 111–135 (2012).
15. Lewis, R., Thompson, J.: Analysing the Effects of Solution Space Connectivity with an Effective Metaheuristic for the Course Timetabling Problem. European Journal of Operational Research 240(3), 637-648 (2015).
16. Van Bulck, D., Goossens D., Schaerf, A.: Multi-neighbourhood simulated annealing for the ITC-2007 capacitated examination timetabling problem. Journal of Scheduling, 1-16 (2023).
17. Szu, H., Hartley, R.: Fast simulated annealing. Physics Letters A 122(3-4), 157-162 (1987).

Introducing Individuality into Students' High School Timetables

Andreas Krystallidis[1] [0009-0007-0183-3038] and Rubén Ruiz-Torrubiano[1] [0000-0001-9314-0739]

IMC Krems University of Applied Sciences, Piaristengasse 1, Krems, 3500, Austria
andreas.krystallidis@fh-krems.ac.at
ruben.ruiz@fh-krems.ac.at

Abstract. In a perfect world, each high school student could pursue their interests through a personalized timetable that supports their strengths, weaknesses, and curiosities. While recent research has shown that school systems are evolving to support those developments by strengthening modularity in their curricula, there is often a hurdle that prevents the complete success of such a system: the scheduling process is too complex. While there are many tools that assist with scheduling timetables in an effective way, they usually arrange students into groups and classes with similar interests instead of handling each student individually. In this paper, we propose an extension of the popular XHSTT framework that adds two new constraints to model the individual student choices as well as the requirements for group formation that arise from them. Those two constraints were identified through extensive interviews with school administrators and other school timetabling experts from six European countries. We propose a corresponding ILP formulation and show first optimization results for real-world instances from schools in Germany.

Keywords: Educational Timetabling, High School Timetabling, Integer Linear Programming, XHSTT.

1 Introduction

As educational systems continuously evolve, crafting close-to-optimal high school timetables continues to pose a major challenge. An important aspect that contributes to this phenomenon is the increasing demand for personalized and flexible learning experiences by all stakeholders of the educational system (policy makers, teachers, students and society in general). This demand results in modular educational systems, where students can choose parts of their own curriculum individually. However, conventional approaches to the creation of timetables can not fulfill the diverse needs of students. This paper addresses the need for innovation in the encoding of constraints for high school timetables, aiming to support the flexibility that modern modular educational systems afford to students.

While traditional formats of constraint encoding do support most of the requirements that schools have on timetables including the possibility to work with individual students instead of classes, they are not able to represent the requirements of each student pursuing diverse individual academic interests in an adaptive manner. In this paper, we present an extension of the XHSTT format [11] (introduced in the Third International Timetabling competition [10]) that incorporates two new constraints, making possible to encode flexible student course choices and class formation requirements while still supporting all previous instances developed for this format. The need for those two new constraint types became apparent through recent research that explored the timetabling demands for schools across central Europe [12] which shows that there is a trend of enabling students to follow their individual interests. Even though this new formulation features students choosing their own respective courses, the problem is still very different from University Course Timetabling Problems [8,7], which were already found to be solvable when transformed to the XHSTT format [4].

While, our new formulation has some similarities to the student choices featured in the Post Enrolment based Course Timetabling (PE-CTT) Problem, which was first presented in the second track of the ITC 2007 [6], the problem we are modeling here has some key differences: In the PE-CTT Problem students select a set of events that they want to attend without providing alternatives or preferences. It is a hard constraint that they visit all their selected events, while in our newly formulated Constraints it is possible to specify that only a subset of a flexible quantity should be attended. Some additional flexibility is provided in the University Course Timetabling Problem featured at the ITC 2019 [9]. While the students still attend a fixed amount of selected courses, the courses themselves can be split into structured subparts that could be used to model distributions of students to equivalent courses (e.g. Math_1_1 and Math_1_2). However, to the best of our knowledge, there is no constraint in any Educational Timetabling format that would enable us to model student choices on the individual level in the flexible manner that is desired in modular educational systems. Finally, since we are solving High School Instances and not University Instances, we use almost all the different constraints of XHSTT, so it is much more efficient to extend this format instead of an University Timetabling format that focuses on other qualities that we mostly do not need (e.g. differentiating between different weeks of the semester and complex orderings and structures of events).

We hope that by extending the standard high school timetable format we will be able to spur new original research that adapts and improves methods of automatic timetabling for modular high schools. We support these developments explicitly by providing an ILP formulation that extends one of the state-of-the-art ILP formulations by Kristiansen et al. [5] as well as 18 publicly available instances that include the new constraints. Additionally, we present first results for upper bounds using our ILP.

This paper is structured as follows: In Section 2, we introduce the new constraints as well as how they can be used and provide examples of how to encode various situations that may occur in modular school timetables. In Section 3, we provide a corresponding extension to the well-known ILP formulation by Kristiansen et al. [5]. In Section 4, we describe the new instances in our format and provide some first results. The instances are made publicly available as a benchmark set for modular high school timetabling. Finally, in Section 5, we provide an overview of our findings and describe possible directions of future research.

2 An Extension to the XHSTT Format

The XHSTT format [11] is the most widely used format for encoding the High School Timetabling Problem. It is versatile enough to encode many real-world instances of timetable requirements from various schools around the world accurately. However, in recent years there has been a trend of giving students more choices in what individual courses they wish to attend. While some of the requirements that arise from those choices can be modeled using the existing constraints, others are not supported by the XHSTT format. Since the goal of the XHSTT format is to provide a way to encode timetable requirements in a unified way, we find it important that those recent developments reflect themselves in the form of an extension of the format. This extension should be as small as possible while still being able to accurately encode the new constraints for timetables. Furthermore, we find that everything that can be encoded with the existing set of constraints should still be encoded using them (even when it is in a slightly roundabout way) in order not to put an unnecessary strain upon those who maintain and possibly want to extend existing methods and solutions for solving the High School Timetabling Problem. Finally, it is also important that the format only includes constraints that are actually useful for those in charge of creating the timetables. That is why the constraints proposed in this paper are chosen based on a study [12] where experts across Europe were asked about the challenges they face when creating timetables for high schools. The result of all of those requirements are three new constraint categories that encode student choices, class size requirements and class size balance. Of those three requirements, the class size requirements will be encoded using the existing constraints of the XHSTT format while the other two require one new constraint type each.

2.1 Student Choices

From the interviews conducted in the paper by Ruiz-Torrubiano et al. [12] it becomes apparent that many schools offer course choices to students in one form or another, especially in the respective upper cycles. Such choices can range from choosing a general direction (profile) for their studies, which usually results in scheduling all students with a given profile together, to individual course choices subscribing them to specific courses together with other students that made the same election. However, once there is a certain level of modularity it is usually in practice infeasible to schedule the courses in such a way that every student can attend exactly the courses they have selected. To deal with that problem schools have adopted two different methods to manage student course choices. Either they create a timetable first, and students choose lectures that fit into their individual schedules, or students give priorities and/or alternatives for their choices and the timetabler tries to fulfill those preferences to the best of their ability. The first possibility is already supported in XHSTT by simply creating courses with no classroom assignment together with some time preference constraints for said courses. However, the second method requires a possibility to model how many from a pool of courses can be attended on an individual resource level. So for example, a student that wishes to learn another language might choose to attend a Spanish course, if they do not get into that course they would like to learn Italian and if there is also no more room in the Italian course they might want to learn French. It is a hard requirement for that student that they will exactly attend one of those courses. Another student may want to specialize in natural sciences, and they want to attend biology, physics and chemistry eventually while not caring exactly how many of those subjects they will attend in the following year as long as it is at least one. An example for how this constraint may look like can be seen in Figure 1. Additionally, the school might also support that students provide weights (preferences) to their choices. Note that while we designed this constraint to model course choices for students, it can also be used to model teaching preferences, which are also a common theme for many high schools.

All those requirements can be unified into a new constraint type which we call Student Choice Constraint. The constraint has the standard children that all XHSTT constraints share (Id, Name, Required, Weight, CostFunction) the AppliesTo tag consists of Resources and ResourceGroups children. Additionally, the constraint has the child categories EventGroups, Minimum and Maximum.

```
<StudentChoiceConstraint Id="StudentChoice_ST_Bob">
    <Name>StudentChoice_ST_Bob</Name>
    <Required>true</Required>
    <Weight>50</Weight>
    <CostFunction>Linear</CostFunction>
    <AppliesTo>
        <ResourceGroups/>
        <Resources>
            <Resource Reference="ST_Bob"/>
        </Resources>
    </AppliesTo>
    <EventGroups>
        <EventGroup Reference="Biology_10"/>
        <EventGroup Reference="Physics_10"/>
        <EventGroup Reference="Chemistry_10"/>
    </EventGroups>
    <Minimum>1</Minimum>
    <Maximum>3</Maximum>
</StudentChoiceConstraint>
```

Fig. 1: Student Choice Example

- **AppliesTo:** Each resource that is either part of the Resources child or is part of a resource group, which is mentioned in the ResourceGroups child, is a point of application for this constraint.
- **EventGroups:** All Event Groups mentioned in this child are relevant to the constraint.
- **Minimum:** Each resource this constraint applies to has to attend at least Minimum Event Groups from the relevant Event Groups for this constraint.

– **Maximum:** Each resource this constraint applies to can, at most, attend Maximum Event Groups from the relevant Event Groups for this constraint.

The deviation of the constraint is described as follows: For each resource part of the Resources and ResourceGroups the deviation is equal to the number of Event Groups that the resource attends, which exceed the Maximum or fall short of the Minimum. Note that we define attendance as visiting any subevent of a given Event Group.

This constraint can also be used to encode weighted preferences. Imagine a student who wants to attend one out of 3 courses A, B, and C but has a preference order of $A \succ B \succ C$. First, a hard constraint can be added where the EventGroups child contains all 3 courses, and the Minimum and Maximum are set to 1. We then add a soft constraint containing only Events A and B and another constraint containing only event A. Again we set the Minimum and Maximum to 1 for both soft constraints. Depending on the weights of the Soft constraints we can now adjust the importance of the student getting his first or second choice.

2.2 Class Sizes

Whether it is due to room limitations, pedagogic restrictions, or legal reasons (supervision duties), schools usually have limits on how large the classes for each course can be at most. In a system without student choices, this is usually enforced when the classes are put together before scheduling the individual lessons. Events where the classes are split and mixed are often modeled using one main Event (to which the whole class is assigned) and multiple subevents that are all linked to the main event under the assumption that none of the students attend multiple of those subevents. This works, for example, if one wants to split all students from one class level into two math groups, and every student has to choose a second foreign language. However, this method quickly becomes more complex the more individual the student choices become since it most likely won't be possible to build sets of subjects that have no student overlaps while also not creating many idle periods in student timetables and giving all of the students their preferred subjects, which means that there is a need to find some optimal balance between those constraints. Usually there are restrictions that don't allow for any idle time in student timetables during certain periods, while other periods are more flexible (e.g., in the afternoon). Building the classes as part of the optimization problem allows the solver to find which classes to group dynamically based on when it schedules them, also taking into account how important it is to fulfill each individual student's choice. We can model those restrictions using the existing XHSTT constraints, which makes it easier for existing approaches to adapt to those changes. In the following, we describe exactly how to model this class size problem because the translation into XHSTT constraints is not completely trivial. However, first, we want to recap how the XHSTT format works on a high level.

An instance in the XHSTT format consists of Times, Resources, Events, and Constraints. The Events have a duration, which specifies how many time slots must be assigned to them. How exactly those times are distributed over the week is specified as part of the Constraints. We say that each block of consecutive time slots that is part of the final schedule builds a SolutionEvent (or subevent). Each of those SolutionEvents specifies some resource requirements that can either come in the form of a fixed AssignedResource or a flexible UnassignedResource. In the case of an UnassignedResource, there are often some constraints that restrict the pool of possible concrete resource assignments. Finally, the set of Constraints also consists of other types of Constraints that further impose limits on when, how often, and in what constellations Events, Resources, and groups of Events and Resources are scheduled.

With that basic understanding of the XHSTT format, we can now get into how we modeled the class sizes. First of all, we assume that three resource types exist (but it is possible to arbitrarily add more): Teachers, Rooms, and Individual Students. If fixed classes still exist for certain courses they can easily be modeled by assigning all students of that class directly to that course. For those courses that should be built by the solver, we create three event types:

1. One main event that will be used in all constraints that handle the time assignment of the lessons. This main event will also be used to either directly assign a room and teacher or model the resource preferences for those two resource types. Any restrictions on how the event should be split and distributed over the week will also be applied here
2. As a next step, we create one event for each student resource that is required to fulfill the minimum student number of the course s_{min} (e.g., if it needs at least 10 students to build a language class we create 10 events). They only have one student event resource which is usually not preassigned (can optionally be preassigned to a specific student if attendance is mandatory). We also need a hard Assign and Prefer Resource Constraint so that only those students who choose the course can be assigned and all events must have a student assigned.
3. Next, we need to add a hard Avoid Split Assignments constraint for each Event (modeled with an Event Group that only contains one student Event) which ensures that two subevents of the same Event can't have different student assignments.
4. Afterwards, we add a hard Link Events Constraint to the main Events so that all subevents must have the exact same time assignments as the main Event. Through hard Avoid Clashes Constraints on the individual students this also guarantees that each subevent must have a different student assigned. We will henceforth call Events of this type "minimum requirement events".
5. As a final step, we create Events that are very similar to the previous ones but contain those students that are optional from the Event perspective. We create a total of $s_{max} - s_{min}$ events of this type. All constraints are the same except that we do not use any Assign Resource Constraints since it is fine if no students are assigned to the Event (The Prefer Resource constraints have to stay so that if a student is assigned it must be one that chose the class). We will henceforth call events of this type "maximum requirement events"

Note that it would be possible to combine the student requirement events into a single event from a modeling perspective (using a separate Role for each student). We decided against this approach in case that some existing solvers might enumerate all combinations of resource assignments for each Event, which would lead to an exponential amount of such combinations. In all other aspects the approaches are to the best of our knowledge equivalent, except that it would be possible to directly quantify over *b* variables in the two new proposed constraint types. However, quantifying over EventGroups instead of individual Events gives the advantage that we can use more "high level" constraints, for example if we go back to the example provided in Listing 1 Biology_10, Physics_10 and Chemistry_10 could each be an EventGroup that represents multiple courses (e.g. Biology_10_1, Biology_10_2 and Biology_10_3). We would then add three more Student Choice Constraints (one for each subject) for ST_Bob each with a Minimum set to 0 and Maximum set to 1. The result is that on the high level Bob will visit between 1 and 3 of his selected choices and on the lower level he will be assigned to exactly one course corresponding to the assigned subjects.

2.3 Class Size Balance

Sometimes one subject is taught in multiple courses handling exactly the same school material because the number of students is too big for one single classroom and teacher. One way to handle this using existing constraints is to simply use two teachers and rooms for the class while also scaling the student requirements. However, this has the restriction that both of those courses would need to happen in parallel, which takes away some flexibility, especially when the student schedules are very individual.

A better alternative would be to have two completely separate courses. With the help of Student Choice constraints, we can then model that a student can or must attend one of them. However, this could result in the undesirable property of possibly very unbalanced class sizes (e.g., two math classes, one with the bare minimum assignment of 10 students while the other is fully booked with 30 students). To prevent this, we introduce Balance Class Size constraints that, aside from the standard children that all XHSTT constraints share (Id, Name, Required, Weight, CostFunction), have the tags AppliesTo with the child EventGroups, Role and MaximumDifference. An example for how this constraint would be modeled in the case of two equivalent math classes can be found in Figure 2.

- **AppliesTo:** Each Event Group that is mentioned in the EventGroups child is relevant for this constraint.
- **MaximumDifference:** An integer that sets a limit of how much difference between the number of assigned resources can be between the Event Groups without causing a deviation.
- **Type:** Optional child. If a Type is given only resources with this type are counted towards the assigned resources of each event.

The deviation of this constraint is described as follows: For each Event Group that is part of the EventGroups child, the deviation is calculated as the Maximum Difference to the Event Group (part of the Event Groups Child) with either the most or the least assigned resources (depending on which difference is higher) minus the allowed MaximumDifference.

```
<BalanceClassSizeConstraint Id="BalanceClassSize_Math_5a">
    <Name>BalanceClassSize_Math_5a</Name>
    <Required>false</Required>
    <Weight>1</Weight>
    <CostFunction>Linear</CostFunction>
    <AppliesTo>
        <EventGroups>
            <EventGroup Reference="Math_1_5A"/>
            <EventGroup Reference="Math_2_5A"/>
        </EventGroups>
    </AppliesTo>
    <MaximumDifference>2</MaximumDifference>
    <Type>Student</Type>
</BalanceClassSizeConstraint>
```

Fig. 2: Balance Class Size Example

3 ILP Formulation

In this section we introduce an ILP model that can be used to solve an instance of our extension to the High School Timetabling Problem. For this purpose we will extend the formulation from Kristiansen et al. [5] by our two new constraints as well as the relevant variables and linkings. In this section, we will only describe the new constraints. The full ILP formulation can be found in the Appendix. Note that for the moment our new instances and format only support linear and quadratic cost functions.

3.1 Sets

First, we introduce some sets of entities relevant to the extended modular XHSTT problem which are the same as used by the model for the original problem [5].

$t \in T$	ordered set of times	(1)
$tg \in TG$	set of time groups	(2)
$r \in R$	set of resources	(3)
$e \in E$	set of events	(4)
$eg \in EG$	set of event groups	(5)
$er \in e$	set of event resources of an event e	(6)
$se \in e$	set of subevents of an event e	(7)
$c \in C$	set of constraints	(8)
$p \in c$	points of application of constraint c	(9)
$d \in p$	deviations of a point of application p	(10)
$i \in I$	possible deviation values of deviations d	(11)

$j \in J$	possible deviation sum values at points p	(12)
$T^{\text{start}}_{se,t}$	possible start times for se that occupy t	(13)

3.2 Further Notation

Next, we need some further notation to express some of the constraints.

r_D	dummy-resource with meaning no resource assigned	(14)
t_D	dummy-time with meaning no time assigned	(15)
D_e	duration of event e	(16)
D_{se}	duration of subevent se	(17)
$e \in c$	constraint c applies to event e	(18)
$r \in c$	constraint c applies to resource r	(19)
$eg \in c$	constraint c applies to event group eg	(20)
w_c	weight of constraint c	(21)
$\rho(t)$	index of time t in ordered set T	(22)
PA_{er}	is 1 if event resource er has a preassigned resource otherwise 0	(23)
$\overline{B}_c$	upper limit of constraint c	(24)
$\underline{B}_c$	lower limit of constraint c	(25)

3.3 Variables

Compared to the formulation by Kristiansen et al. [5], we added the variables b and c which are required to model if a resource is participating in Events or Event Groups.

$x_{se,t,er,r}$	binary, indicates if se starts at t and r is assigned to er	(26)
$y_{se,t}$	binary, indicates that se starts at t	(27)
$v_{t,r}$	integer, indicates how often r is used at t by any se	(28)
$w_{se,er,r}$	binary, indicates if se is assigned r for er	(29)
$b_{e,r}$	binary, indicates if r is assigned to any se of e	(30)
$c_{eg,r}$	binary, indicates if r is assigned to any e of eg	(31)
$s_{c,p,d}$	integer, deviation d at point of application p of c	(32)
$s_{c,p,d,i}$	binary, indicates that d has value i at p of c	(33)
$u^{\text{SquareSum}}_{c,p,j}$	binary, indicates that the sum of deviations at p is j	(34)
u_{se}	binary, indicates whether se is active or not	(35)
$q_{r,t}$	binary, indicates if r is busy at t	(36)
$p_{r,tg}$	binary, indicates if r is busy at some t in tg	(37)

3.4 Functions

We also need to introduce some functions that will mainly be used to express the objective value of the problem.

$$f(s_{c,p,d}) = w_c \cdot \text{CostFunction}(s_{c,p,d}) \qquad \text{cost of constraint } c \quad (38)$$

$$CF^{\text{Sum}} = \sum_{p\in c, d\in p} s_{c,p,d} \qquad \text{Sum cost function} \quad (39)$$

$$CF^{\text{SumSquare}} = \sum_{p\in c, d\in p, i\in I} i^2 \cdot s_{c,p,d,i} \qquad \text{SumSquare cost function} \quad (40)$$

$$CF^{\text{SquareSum}} = \sum_{p\in c, j\in J} j^2 \cdot u^{\text{SquareSum}}_{c,p,j} \qquad \text{SquareSum cost function} \quad (41)$$

Some constraints have an upper limit and a lower limit. In this case we define the value of deviation V using the function $U_{\underline{B}_c,\overline{B}_c} V$ as follows:

$$s \geq U_{\underline{B}_c,\overline{B}_c} V \rightarrow \begin{cases} s \geq V - \overline{B}_c \\ s \geq \underline{B}_c - V \end{cases} \qquad \text{deviation with upper and lower limit} \quad (42)$$

3.5 Updated Objective Function

The objective function consists of the sum of all cost functions of individual constraints. We can also split this objective function into separate values z_{hard} and z_{soft} to denote the costs of hard and soft constraints respectively. Compared to the objective function by Kristiansen et al. [5], we simply extended the function by adding the terms describing the deviation of our new constraint types.

$$\begin{aligned} \min z = & f(s^{\text{assignres}}_{c,er}) + f(s^{\text{assigntime}}_{c,er}) + f(s^{\text{spliteventamount}}_{c,e} + s^{\text{spliteventdur}}_{c,e}) \\ & + f(s^{\text{distsplitevent}}_{c,e,er}) + f(s^{\text{preferres}}_{c,er}) + f(s^{\text{prefertime}}_{c,e}) + f(s^{\text{avoidsplit}}_{c,eg}) \\ & + f(s^{\text{spreadevent}}_{c,eg,tg}) + f(s^{\text{linkevent}}_{c,eg,t}) + f(s^{\text{avoidclashes}}_{c,r,t}) + f(s^{\text{unavailabletimes}}_{c,r}) \\ & + f(s^{\text{idletimes}}_{c,r}) + f(s^{\text{clusterbusy}}_{c,r}) + f(s^{\text{limitbusy}}_{c,r,tg}) + f(s^{\text{limitworkload}}_{c,r}) \\ & + f(s^{\text{balancesize}}_{c,eg}) + f(s^{\text{studentchoice}}_{c,r}) \end{aligned} \quad (43)$$

3.6 Constraints

Added General Constraints

The model needs several linking constraints and other general constraints to make the variables express the above-described properties. We will only describe those general constraints that were added to the model of Kristiansen et al. [5] the remaining constraints can be found in the Appendix.

The following new constraints link variables $w_{se,er,r}$ and our new variables $b_{e,r}$:

$$w_{se,er,r} \leq b_{e,r} \qquad \forall e \in E, se \in e, er \in e, r \in R \quad (44)$$

$$\sum_{se \in e} w_{se,er,r} \geq b_{e,r} \qquad \forall e \in E, er \in e, r \in R \quad (45)$$

To link our new variables $c_{eg,r}$ to $b_{e,r}$ we need two more new constraints:

$$b_{e,r} \leq c_{eg,r} \qquad \forall eg \in EG, e \in eg \quad (46)$$

$$\sum_{e \in eg} b_{e,r} \geq c_{eg,r} \qquad \forall eg \in EG \quad (47)$$

Balance Class Size Constraint

Applies to: Event Groups
Point-of-application: Event Group
We use the parameter $\overline{B}_c$ to denote the maximum class size difference specified in constraint $c \in \overline{C}$. The role parameter is optional and denotes that only resources of a specific role in the event should be considered We use the variable $mr_{c,eg}$ to denote the number of resources allocated to the specified Event Group

$$\sum_{\substack{e \in eg, er \in e, \\ r \in er, type_r = type_c \setminus \{r_D\}}} c_{eg,r} = mr_{c,eg} \qquad \forall c \in \overline{C}, eg \in \overline{C} \quad (48)$$

$$mr_{c,eg} - mr_{c,eg2} - \overline{B}_c \leq s^{\text{balancesize}}_{c,eg} \qquad \forall c \in \overline{C}, eg \in c, eg2 \in c, eg \neq eg2 \quad (49)$$

$$mr_{c,eg2} - mr_{c,eg} - \overline{B}_c \leq s^{\text{balancesize}}_{c,eg} \qquad \forall c \in \overline{C}, eg \in c, eg2 \in c, eg \neq eg2 \quad (50)$$

Student Choice Constraint

Applies to: Resources
Point-of-application: Resource
We use parameters $\underline{B}_c$ and $\overline{B}_c$ to denote the minimum and maximum values specified in constraint $c \in \overline{C}$.

$$U_{\underline{B}_c,\overline{B}_c} \sum_{eg \in c, e \in eg, er \in e, r \in er, r = r_c} c_{eg,r} \leq s^{\text{studentchoice}}_{c,r} \qquad \forall c \in \overline{C}, r_c \in c \quad (51)$$

3.7 Model Size Reductions

Using the model described above without any additions results in models that are too big to handle on our computing cluster with 64 GB of RAM for most of the instances. For that reason, we eliminated some variables that would never be used for an acceptable solution. Our variable eliminations consist of the following list:

– We eliminated all variables $x_{se,t,er,r}$ and $w_{se,er,r}$ for students r that did not select an Event e, $se \in e$. This means that we do not permit solutions where students who did not select an event are assigned to one of its subevents (which is supported by practice).
– We only generate subevents with a feasible duration if there is a hard split events constraint restricting the duration and/or amount of generated subevents. [3] This is the same technique that was used by Fonseca et al. [2] to reduce model sizes.

4 Evaluation

4.1 Instances

As a first benchmarking set[1], we chose 18 high schools with modular school systems from 6 different federal states in Germany. Note that the secondary educational systems in Germany can vary greatly depending on the particular federal state [12], which makes this set more diverse than instance groups from most other countries. The original anonymized instances were provided by Untis GmbH[2], an Austrian company that specializes in software that assists schools with their various scheduling problems. We implemented methods to automatically translate their format for encoding constraints to the new extended XHSTT format, which enables us to provide many more instances in the future (Untis collaborates with over 26,000 schools worldwide). However, it is important to note that the XHSTT instances are not a one to one match semantically with the original instances provided by Untis. This is due to the complex nature of the Untis specification that uses a lot of empirical experience to evaluate timetables on factors that can't be represented in a standardized format. We still managed to achieve an extended XHSTT formulation that matches the Untis formulation closely. Some statistics of the instances can be found in Tables 1 and 2. Table 1 describes how many resources of each resource type are used in each instance. Note that we do not include the minimum/maximum requirement events in the event count since we categorize them as part of the original event and they will always be scheduled together. However, we list the number of requirement events as well as other quantifiable properties that describe the modularity of each instance in Table 2. Note that the amount of requirement events is equal to the number of student assignments that can (but don't necessarily have to) happen to modular events. Table 2 lists how many Student Choice and Balance Class Size constraints each instance uses. The column Modular Events describes the cardinality of the subset of events that have minimum and/or maximum requirement events associated with them. However, there are many more factors that can have an impact on how complex the resulting instance will be. One factor of complexity is the number and type of constraints from the original XHSTT problem definition. Another factor that has a significant impact is the size of the student pool that is feasible for each requirement event. An instance will be much harder if it features some events where a high percentage of the total amount of students wants to participate in certain events. It is also noteworthy that compared to most other benchmark instances of the original XHSTT

[1]https://github.com/IMC-UAS-Krems/modularXHSTT
[2]https://www.untis.at

problem, this instance set is more restrictive on the possible times for events and the available times for students and teachers. Specifically, there are constraints restricting how many primary subjects a student can attend per day/in a row, how many lessons a teacher may teach in a row without a break, minimum and maximum amounts of idle times per week for teachers, global constraints that define a time-window when a lunch break can/must happen for both students and teachers and hard restrictions that allow no student idle times in the hours before lunch. We made sure to choose schools of varying sizes and proportions of modular events so that it will be possible in the future to find out where the complexities of the problem lie. We also plan to extend this set of instances in the future with other modular schools from across Europe to cover more of the possible instance space.

Table 1: Amount of resources present in each of the new instances (by resource type).

Instance	Events	Students	Classes	Teachers	Rooms
GermanyRHPF1	1430	143	141	142	208
GermanyHAMB1	554	847	10	104	80
GermanyNRWE1	627	118	52	185	186
GermanyHAMB2	636	215	43	106	73
GermanyNRWE2	329	252	16	40	69
GermanyRHPF2	414	134	25	88	74
GermanyRHPF3	454	167	25	92	84
GermanyNRWE3	226	252	16	40	69
GermanyNRWE4	1045	331	112	185	310
GermanyRHPF4	545	894	0	83	106
GermanySAAR1	428	169	25	75	65
GermanyRHPF5	375	182	21	75	61
GermanyRHPF6	526	360	30	107	67
GermanyBAWU1	832	157	76	201	176
GermanyBAWU2	976	272	59	122	173
GermanyBAWU3	241	228	22	92	71
GermanyBAWU4	762	205	24	106	70
GermanyHESS1	705	181	49	177	225

4.2 ILP Evaluation

Based on the size of the new instances and the added complexity from the new constraints we expect the new instances to be more difficult to solve than previous benchmark sets for the XHSTT. Previous experiments [5] using ILP as an exact method have shown that the problem is still too challenging for modern ILP solvers when using bigger instances. Therefore, we expect to only produce weak upper bounds using the above-described ILP model. Nevertheless, we find it important to provide first results for the newly introduced problem, which should show whether the problem is trivial to solve or not.

Table 2: Quantifyable properties describing the modularity of the new instances.

Instance	Choice Constraints	Balance Constraints	Modular Events	Requ. Events
GermanyRHPF1	2378	34	142	2440
GermanyHAMB1	911	42	46	970
GermanyNRWE1	1254	60	73	1265
GermanyHAMB2	2053	62	112	2206
GermanyNRWE2	2408	56	106	2893
GermanyRHPF2	1476	40	93	1385
GermanyRHPF3	1718	50	109	1892
GermanyNRWE3	1677	35	73	1942
GermanyNRWE4	3433	79	179	3828
GermanyRHPF4	3984	100	241	4063
GermanySAAR1	1421	57	81	1596
GermanyRHPF5	1904	41	124	2622
GermanyRHPF6	2567	21	131	3255
GermanyBAWU1	1452	45	76	1764
GermanyBAWU2	3743	84	176	4112
GermanyBAWU3	2473	75	162	2496
GermanyBAWU4	2309	61	160	2347
GermanyHESS1	2947	24	47	1189

While employing ILP as an exact method might not yield practical solutions directly, past research suggests its potential when integrated into metaheuristic or matheuristic approaches for tackling the High School Timetabling Problem [1,2].

Table 3 displays the outcomes of executing the 18 instances, with a Memory Limit set at 64 GB, on an AMD EPYC 7252 processor utilizing the commercial ILP solver Gurobi version 10.0.1. Each instance was allotted a time limit of 6 hours for computation. This time frame notably exceeds the 1000-second constraint imposed during the ITC2011 competition. The objective value is split into two components of the form (hard, soft) constraint deviations.

It's worth noting that in practical scenarios, schools typically face fewer time constraints when devising their timetables. They are often willing to invest several hours, or even days, in computational time without significant pressure. However, it's crucial to strike a balance between computational resources and research accessibility. Excessive resource consumption could potentially limit accessibility to the problem, which runs counter to the goal of fostering open research.

The experiment reveals the complexity inherent in the problem, suggesting that exact methods may struggle to provide satisfactory solutions. Out of the 18 instances studied, solutions were only found for 10.

Furthermore, the integral solutions obtained from the experiment did not meet the criteria necessary for a viable school timetable. Despite leveraging Gurobi's optimization capabilities, the solutions fell short, underscoring the intricacies of the problem and the limitations of current methodologies.

Table 3: ILP results and statistics on 6-hour run.

Instance	Variables	Constraints	Objective Value
GermanyRHPF1	9007909	67145239	—
GermanyHAMB1	22276651	61355937	—
GermanyNRWE1	5282717	22736030	(152422, 2330)
GermanyHAMB2	6268589	19983356	—
GermanyNRWE2	12183297	41285751	—
GermanyRHPF2	3013943	9066705	(125430, 4663)
GermanyRHPF3	6422201	21112371	(157904, 86000)
GermanyNRWE3	8581837	30750272	(75295, 206299)
GermanyNRWE4	23744735	88931169	—
GermanyRHPF4	42657262	87422983	—
GermanySAAR1	4695284	14566661	(142059, 23465)
GermanyRHPF5	5870336	14030401	(117175, 379454)
GermanyRHPF6	20876398	53277518	—
GermanyBAWU1	7567498	27892830	(231653, 70076)
GermanyBAWU2	11729612	31872156	(257726, 348709)
GermanyBAWU3	8464935	18952959	(81008, 3416)
GermanyBAWU4	7740808	19141384	(193313, 8665)
GermanyHESS1	5737775	29194865	—

Additionally, Gurobi's inability to provide lower bounds within the designated timeframe prevents us from assessing the optimality gap. However, there is potential for progress as the schools that provided the instances have successfully created their own timetables. In general it is not always possible to find feasible solutions for the requirements encoded by schools. Untis deals with this problem by either leaving some hours unscheduled or reporting the problems with the final timetable that will then be manually resolved by the administrator, which then has to decide which hard constraints can be softened. Nevertheless those solutions may enable us to establish tighter upper bounds in the future, enhancing our understanding of the problem and potentially guiding optimization strategies.

5 Conclusion and Future Work

In this paper, we proposed an extension to the XHSTT format to address the constraints that are present in the ever-growing number of modular high schools. We explained the reasons that make those changes meaningful and how exactly they can be incorporated into the XHSTT format. Furthermore, we provided 18 new real-world instances from modular high schools of different regions in Germany together with an ILP model that can be used to find feasible schedules. Our experiments showed that using our ILP model as an exact method for finding solutions is not very effective, and even after 6 hours of runtime, it could only find solutions that are nowhere near satisfactory. However, based on the data from previous benchmarks those results were expected and should not discourage us from finding more efficient methods for creating timetables for modular high schools. We believe that the real strength of the presented ILP will be revealed once it is used as part of a heuristic approach like Large Neighborhood Search (LNS).

In future work, we want to use different exact methods, like a SAT solver to inspect if the problem is really as hard as it seems to be or if ILP is simply not the right approach for this problem. There has already been previous work into possible cuts for the original ILP model and we plan to look into possible new cuts for our extension as well. It will also be interesting to see how the various heuristics developed for the original High School Timetabling Problem perform on this extension and if new heuristics can be found that might work even better for this extension. Specifically, we want to look into a more adaptive LNS-based approach to see if methods from Reinforcement Learning can be used to improve the process of finding good schedules. We will support those developments by providing more benchmark instances from all across Europe together with a publicly available tool for validation. Finally, we will also compare future findings with the timetables produced for actual schools to see if there is any gap between theory and practice.

Acknowledgements A. Krystallidis and R. Ruiz-Torrubiano acknowledge financial support from the Research Promotion Agency of Lower Austria (GFF) under project grant FTI21-A-002.

Appendix

Complete ILP Formulation

The variables, sets, functions and general further notation of the model can be found in Section 3. Here we provide the full set of constraints used in the model (with some repetition from Section 3 to avoid confusion), as well as some further additions to guarantee exact objective values.

Constraints

General Constraints Link variables $s_{c,p,d}$ and $s_{c,p,d,i}$:

$$\sum_{i \in I} i * s_{c,p,d,i} = s_{c,p,d} \qquad \forall c \in C, p \in c, d \in c \quad (52)$$

Only one deviation indicator can be set per deviation:

$$\sum_{i \in I} s_{c,p,d,i} = 1 \qquad \forall c \in C, p \in c, d \in c \quad (53)$$

Link variables $s_{c,p,d}$ and $u^{\text{SquareSum}}_{c,p,j}$:

$$\sum_{j \in J} j * u^{\text{SquareSum}}_{c,p,j} = \sum_{d \in p} s_{c,p,d} \qquad \forall c \in C, p \in c \quad (54)$$

Only one deviation indicator can be set per point of application:

$$\sum_{j \in J} u^{\text{SquareSum}}_{c,p,j} = 1 \qquad \forall c \in C, p \in c \quad (55)$$

Link variables $s_{c,p,d}$ and u^{StepSum}_{c}:

$$M \cdot u^{\text{StepSum}}_{c} \geq s_{c,p,d} \qquad \forall c \in C, p \in c, d \in p \quad (56)$$

A subevent is assigned exactly one starting time and the number of assigned resources equals the number of event resources ($|er|_{se}$):

$$\sum_{t \in T, r \in er} x_{se,t,er,r} = 1 \qquad \forall se \in SE, er \in se \quad (57)$$

$$\sum_{er \in se, r \in er} x_{se,t,er,r} = |er|_{se} \cdot y_{se,t} \qquad \forall se \in SE, t \in T \quad (58)$$

Link variables $v_{t,r}$ and $w_{se,er,r}$:

$$\sum_{se \in SE, er \in se, t' \in T^{\text{start}}_{se,t}} x_{se,t',er,r} = v_{t,r} \qquad \forall t \in T \backslash \{t_D\}, r \in R \quad (59)$$

$$\sum_{t \in T} x_{se,t,er,r} = w_{se,er,r} \qquad \forall se \in SE, er \in se, r \in er \quad (60)$$

The following new constraints link variables $w_{se,er,r}$ and our new variables $b_{e,r}$:

$$w_{se,er,r} \leq b_{e,r} \qquad \forall e \in E, se \in e, er \in e, r \in R \quad (61)$$

$$\sum_{se \in e} w_{se,er,r} \geq b_{e,r} \qquad \forall e \in E, er \in e, r \in R \quad (62)$$

To link our new variables $c_{eg,r}$ to $b_{e,r}$ we need two more new constraints:

$$b_{e,r} \leq c_{eg,r} \qquad \forall eg \in EG, e \in eg \quad (63)$$

$$\sum_{e \in eg} b_{e,r} \geq c_{eg,r} \qquad \forall eg \in EG \quad (64)$$

A subevent can not be assigned a start time that does not have enough times after it to fit its duration:

$$y_{se,t} = 0 \qquad \forall se \in SE, t \in T \setminus \{t_D\}, \rho(t) + D_{se} - 1 > |T| \quad (65)$$

Only a subset of the subevents are active at a time (since we create all possible subevents). A subevent is considered active if it has a starting time or resource assigned:

$$\sum_{r \in er \setminus \{r_D\}} w_{se,er,r} \leq u_{se} \qquad \forall se \in SE, er \in se, PA_{er} = 0 \quad (66)$$

$$\sum_{t \in T \setminus \{t_D\}} y_{se,t} \leq u_{se} \qquad \forall se \in SE \quad (67)$$

$$\sum_{t \in T \setminus \{t_D\}} y_{se,t} + \sum_{\substack{r \in er \setminus \{r_D\}, \\ er \in se, PA_{er}=0}} w_{se,er,r} \geq u_{se} \qquad \forall se \in SE \quad (68)$$

The sum of the durations of the subevents of an event must equal the total duration of the event:

$$\sum_{se \in e} D_{se} * u_{se} = D_e \qquad \forall e \in E \quad (69)$$

Linking the variables $q_{r,t}$ and $p_{r,tg}$ that indicate if a resource is busy:

$$|SE| \cdot q_{r,t} \geq v_{t,r} \qquad \forall r \in R, t \in T \setminus \{t_D\} \quad (70)$$

$$q_{r,t} \leq v_{t,r} \qquad \forall r \in R, t \in T \setminus \{t_D\} \quad (71)$$

$$p_{r,tg} \geq q_{r,t} \qquad \forall r \in R, tg \in TG, t \in tg \quad (72)$$

$$p_{r,tg} \leq \sum_{t \in tg} q_{r,t} \qquad \forall r \in R, tg \in TG \quad (73)$$

$$(74)$$

Events that have a given start time must have that time assigned ($se*$, represents an arbitrarily chosen subevent that has the same duration as the event e):

$$y_{se*,e_{\text{Time}}} = 1 \qquad \forall e \in E, e_{\text{Time}} \neq None, se*_{\text{Duration}} = e_{\text{Duration}} \quad (75)$$

Assign Resource Constraint

Applies to: Events
Point-of-application: Event resource

$$D_e - \sum_{\substack{se \in e,\\ r \in er \setminus \{r_D\}}} D_{se} \cdot w_{se,er,r} = s^{\text{assignres}}_{c,er} \qquad \forall c \in \overline{C}, e \in c, er \in e, role_{er} = role_c \quad (76)$$

Assign Time Constraint

Applies to: Events
Point-of-application: Event

$$D_e - \sum_{t \in T \setminus \{t_D\}, se \in e} D_{se} \cdot y_{se,t} = s^{\text{assigntime}}_{c,e} \qquad \forall c \in \overline{C}, e \in c \quad (77)$$

Split Events Constraint

Applies to: Events
Point-of-application: Event
We use the parameters $\underline{B}^{\text{amount}}_c$ and $\overline{B}^{\text{amount}}_c$ to denote the minimum and maximum amount of events respectively. Likewise, we use the parameters $\underline{B}^{dur}_c$ and $\overline{B}^{dur}_c$ to denote the minimum and maximum duration of a subevent that is part of a given event, respectively. The full deviation of a constraint $c \in \overline{C}$ is given by $s^{\text{spliteventamount}}_{c,e} + s^{\text{spliteventdur}}_{c,e}$

$$U_{\underline{B}^{\text{amount}}_c, \overline{B}^{\text{amount}}_c} \sum_{se \in e} u_{se} \leq s^{\text{spliteventamount}}_{c,e} \qquad \forall c \in \overline{C}, e \in c \quad (78)$$

$$\sum_{se \in e, \underline{B}^{dur}_c > D_{se}, \overline{B}^{dur}_c < D_{se}} u_{se} = s^{\text{spliteventdur}}_{c,e} \qquad \forall c \in \overline{C}, e \in c \quad (79)$$

Distribute Split Events Constraint

Applies to: Events
Point-of-application: Event
We use the parameters $\underline{B}_c$ and $\overline{B}_c$ to denote the minimum and maximum number of subevents respectively. D_c denotes the duration for which the constraint applies.

$$U_{\underline{B}_c, \overline{B}_c} \sum_{se \in e, D_{se} = D_c} u_{se} \leq s^{\text{distsplitevent}}_{c,e,er} \qquad \forall c \in \overline{C}, e \in c \quad (80)$$

Prefer Resources Constraint

Applies to: Events

Point-of-application: Event resource

$$\sum_{\substack{se\in e, r\notin c,\\ r\in R\setminus\{r_D\}}} D_{se}\cdot w_{se,er,r} = s^{\text{preferres}}_{c,er} \qquad \forall c\in\overline{C}, e\in c, er\in e, PA_{er}=0, role_{er}=role_c \tag{81}$$

Prefer Times Constraint

Applies to: Events
Point-of-application: Event
If D_c is given only sub-events of Duration D_c are considered. Otherwise, all sub-events are considered ($D_c = D_se$ is removed from sum).

$$\sum_{\substack{se\in e, t\notin c,\\ t\in T\setminus\{t_D\}, D_c=D_se}} D_{se}\cdot y_{se,t} = s^{\text{prefertime}}_{c,e} \qquad \forall c\in\overline{C}, e\in c \tag{82}$$

Avoid Split Assignments Constraint

Applies to: Event Groups
Point-of-application: Event Group
We slightly simplify this constraint compared to the original formulation, since we can make use of our new c variables.

$$\sum_{\substack{er\in e, PA_{er}=0,\\ role_c=role_{er}}} w_{se,er,r} \le k_{c,eg,r} \qquad \forall c\in\overline{C}, r\in R, eg\in c, e\in eg, se\in e \tag{83}$$

$$\sum_{r\in R} k_{c,eg,r} - 1 \le s^{\text{avoidsplit}}_{c,eg} \qquad \forall c\in\overline{C}, eg\in c \tag{84}$$

Spread Events Constraint

Applies to: Event Groups
Point-of-application: Event Group
We use parameters $\underline{B}_{c,tg}$ and $\overline{B}_{c,tg}$ to denote the minimum and maximum number of sub-events of a given event that can be placed in time group tg of a constraint $c\in\overline{C}$

$$U_{\underline{B}_{c,tg},\overline{B}_{c,tg}} \sum_{se\in e\in eg, t\in tg} y_{se,t} \le s^{\text{spreadevent}}_{c,eg,tg} \qquad \forall c\in\overline{C}, eg\in c, tg\in c \tag{85}$$

Link Events Constraint

Applies to: Event Groups
Point-of-application: Event Group
We define the binary variable $o_{e,t}$ that takes the value 1 if at least one sub-event of event

e is scheduled at time t, and 0 otherwise. We define the binary variable $l_{eg,t}$ that takes the value 1 if at least one event in event group eg is scheduled at time t, and 0 otherwise.

$$\sum_{t' \in T^{\text{start}}_{se,t}} y_{se,t'} \leq o_{e,t} \qquad \forall e \in E, se \in e, t \in T \backslash \{t_D\} \quad (86)$$

$$\sum_{se \in e, t' \in T^{\text{start}}_{se,t}} y_{se,t'} \geq o_{e,t} \qquad \forall e \in E, t \in T \backslash \{t_D\} \quad (87)$$

$$l_{eg,t} \geq o_{e,t} \qquad \forall eg \in EG, e \in eg, t \in T \backslash \{t_D\} \quad (88)$$

$$l_{eg,t} - o_{e,t} \leq s^{\text{linkevent}}_{c,eg,t} \qquad \forall c \in \overline{C}, eg \in c, e \in eg, t \in T \backslash \{t_D\} \quad (89)$$

Order Events Constraint

Applies to: Pairs of Events
Point-of-application: Pair of Events
The variables h^{first}_e and h^{last}_e represent the first and last time assigned to any subevent of event e. We use parameters $\underline{B}_c$ and $\overline{B}_c$ to denote the minimum and maximum number of times to separate the pair of events (e, e') which are specified in constraint $c \in \overline{C}$.

$$\rho(t) \cdot y_{se,t} + D_{se} \leq h^{\text{last}}_e \qquad \forall c \in \overline{C}, e \in c, se \in e, t \in T \quad (90)$$

$$|T| - (|T| - \rho(t)) \cdot y_{se,t} \leq h^{\text{first}}_e \qquad \forall c \in \overline{C}, e \in c, se \in e, t \in T \quad (91)$$

$$U_{\underline{B}_c, \overline{B}_c}(h^{\text{last}}_e - h^{\text{first}}_{e'}) \leq s^{\text{orderevent}}_{c,(e,e')} \qquad \forall c \in \overline{C}, (e, e') \in c \quad (92)$$

Avoid Clashes Constraint

Applies to: Resources
Point-of-application: Resource

$$v_{t,r} - 1 \leq s^{\text{avoidclashes}}_{c,r,t} \qquad \forall c \in \overline{C}, r \in c, t \in T \backslash \{t_D\} \quad (93)$$

Avoid Unavailable Times Constraint

Applies to: Resources
Point-of-application: Resource

$$\sum_{t \in c} q_{r,t} = s^{\text{unavailabletimes}}_{c,r} \qquad \forall c \in \overline{C}, r \in c \quad (94)$$

Limit Idle Times Constraint

Applies to: Resources
Point-of-application: Resource
We define the binary variables $h^{\text{before}}_{r,tg,t}$ and $h^{\text{after}}_{r,tg,t}$ to indicate if any events are happening

before and after time t in the timegroup tg, respectively. We define the binary variables $h^{\text{timeslot}}_{r,tg,t}$ to indicate if time t is an idle time. We define the integer variable $h^{\text{timegroup}}_{r,tg}$ to indicate the total amount of idle times in timegroup tg We use $|tg|$ to indicate the amount of times in a time group tg. We also use parameters $\underline{B}_c$ and $\overline{B}_c$ to denote the minimum and maximum values specified in constraint $c \in \overline{C}$.

$$q_{r,t_2} \leq h^{\text{before}}_{r,tg,t_1} \qquad \forall r \in C_R, tg \in C_{TG}, t_1, t_2 \in tg, \rho(t_1) > \rho(t_2) \tag{95}$$

$$\sum_{t_2 \in tg, \rho(t_1) > \rho(t_2)} q_{r,t_2} \geq h^{\text{before}}_{r,tg,t_1} \qquad \forall r \in C_R, tg \in C_{TG}, t_1 \in tg \tag{96}$$

$$q_{r,t_2} \leq h^{\text{after}}_{r,tg,t_1} \qquad \forall r \in C_R, tg \in C_{TG}, t_1, t_2 \in tg, \rho(t_1) < \rho(t_2) \tag{97}$$

$$\sum_{t_2 \in tg, \rho(t_1) < \rho(t_2)} q_{r,t_2} \geq h^{\text{after}}_{r,tg,t_1} \qquad \forall r \in C_R, tg \in C_{TG}, t_1 \in tg \tag{98}$$

$$h^{\text{before}}_{r,tg,t} - q_{r,t} + h^{\text{after}}_{r,tg,t} - 1 \leq h^{\text{timeslot}}_{r,tg,t} \qquad \forall r \in C_R, tg \in C_{TG}, t \in tg \tag{99}$$

$$- q_{r,t} + 1 \geq h^{\text{timeslot}}_{r,tg,t} \qquad \forall r \in C_R, tg \in C_{TG}, t \in tg \tag{100}$$

$$h^{\text{before}}_{r,tg,t} \geq h^{\text{timeslot}}_{r,tg,t} \qquad \forall r \in C_R, tg \in C_{TG}, t \in tg \tag{101}$$

$$h^{\text{after}}_{r,tg,t} \geq h^{\text{timeslot}}_{r,tg,t} \qquad \forall r \in C_R, tg \in C_{TG}, t \in tg \tag{102}$$

$$\sum_{t \in tg} h^{\text{timeslot}}_{r,tg,t} = h^{\text{timegroup}}_{r,tg} \qquad \forall r \in C_R, tg \in C_{TG} \tag{103}$$

$$U_{\underline{B}_c, \overline{B}_c} \sum_{tg \in c} h^{\text{timegroup}}_{r,tg} \leq s^{\text{idletimes}}_{c,r} \qquad \forall c \in \overline{C}, r \in c \tag{104}$$

Cluster Busy Times Constraint

Applies to: Resources
Point-of-application: Resource
We use parameters $\underline{B}_c$ and $\overline{B}_c$ to denote the minimum and maximum values specified in constraint $c \in \overline{C}$.

$$U_{\underline{B}_c, \overline{B}_c} \sum_{tg \in c} p_{r,tg} \leq s^{\text{clusterbusy}}_{c,r} \qquad \forall c \in \overline{C}, r \in c \tag{105}$$

Limit Busy Times Constraint

Applies to: Resources
Point-of-application: Resource
We use parameters $\underline{B}_c$ and $\overline{B}_c$ to denote the minimum and maximum values specified in constraint $c \in \overline{C}$.

$$-|tg| \cdot (1 - p_{r,tg}) + U_{\underline{B}_c,\overline{B}_c} \sum_{t \in tg} q_{r,t} \leq s^{\text{limitbusy}}_{c,r,tg} \qquad \forall c \in \overline{C}, r \in c, tg \in c \quad (106)$$

Limit Workload Constraint

Applies to: Resources
Point-of-application: Resource
The workload of a solution resource is given by $w_{e,se,er} = \frac{D_{se} \cdot L_{er}}{D_e}$ where L_{er} is an integer denoting the workload of event resource er. We use parameters $\underline{B}_c$ and $\overline{B}_c$ to denote the minimum and maximum values specified in constraint $c \in \overline{C}$.

$$U_{\underline{B}_c,\overline{B}_c} \sum_{e \in c, t \in T \setminus \{t_D\}, se \in e, er \in e} w_{e,se,er} \cdot x_{se,t,er,r} \leq s^{\text{limitworkload}}_{c,r} \qquad \forall c \in \overline{C}, r \in c \quad (107)$$

Balance Class Size Constraint

Applies to: Event Groups
Point-of-application: Event Group
We use the parameter $\overline{B}_c$ to denote the maximum class size difference specified in constraint $c \in \overline{C}$. The role parameter is optional and denotes that only resources of a specific role in the event should be considered We use the variable $mr_{c,eg}$ to denote the number of resources allocated to the specified Event Group

$$\sum_{\substack{e \in eg, er \in e, \\ r \in er, type_r = type_c \setminus \{r_D\}}} c_{eg,r} = mr_{c,eg} \qquad \forall c \in \overline{C}, eg \in \overline{C} \quad (108)$$

$$mr_{c,eg} - mr_{c,eg2} - \overline{B}_c \leq s^{\text{balancesize}}_{c,eg} \qquad \forall c \in \overline{C}, eg \in c, eg2 \in c, eg \neq eg2 \quad (109)$$

$$mr_{c,eg2} - mr_{c,eg} - \overline{B}_c \leq s^{\text{balancesize}}_{c,eg} \qquad \forall c \in \overline{C}, eg \in c, eg2 \in c, eg \neq eg2 \quad (110)$$

Student Choice Constraint

Applies to: Resources
Point-of-application: Resource
We use parameters $\underline{B}_c$ and $\overline{B}_c$ to denote the minimum and maximum values specified in constraint $c \in \overline{C}$.

$$U_{\underline{B}_c,\overline{B}_c} \sum_{eg \in c, e \in eg, er \in e, r \in er, r = r_c} c_{eg,r} \leq s^{\text{studentchoice}}_{c,r} \qquad \forall c \in \overline{C}, r_c \in c \quad (111)$$

Further Additions for guaranteeing exact objective values

Using the model as described above will result in valid/optimal schedules if the ILP solver is run without a time limit. If there is a time limit the objective value provided by the ILP solver might be bigger than the actual objective value of the produced schedule. This is due to the formulation using the $\leq$ operator in combination with the deviation variables. So while it does lead to a worse objective value the ILP solver can save computational time by setting the deviation variables higher than necessary. To mitigate this issue for the constraints that feature an upper and lower bound we can add binary indicators that tell the ILP solver which constraint to use based on the current assignments (e.g. if the constraints have the form $U_{\underline{B}_c,\overline{B}_c} x \leq s$) we set a binary variable $bin_{max} = 1$ if $x > \overline{B}_c$ and a binary variable $bin_{min} = 1$ if $x < \underline{B}_c$. We then use those indicators to add the following constraints:

$$x - \overline{B}_c = s \qquad \text{if } bin_{max} \quad (112)$$
$$\underline{B}_c - x = s \qquad \text{if } bin_{min} \quad (113)$$

We proceed similarly for constraints of the form $x_1 \leq s, x_2 \leq s, \ldots, x_n \leq s$ (where x_i may represent an arbitrary linear expression). We can instead model them with the following constraint:

$$\max(x_1, \ldots, x_n) = s \quad (114)$$

Note that while we use so-called general constraints (indicator constraints, max constraints etc.) the ILP solver automatically transforms those into a set of linear constraints.

References

1. Fonseca, G.H.G., Santos, H.G., Carrano, E.G.: Integrating matheuristics and metaheuristics for timetabling. Comput. Oper. Res. **74**, 108–117 (2016). https://doi.org/10.1016/J.COR.2016.04.016, https://doi.org/10.1016/j.cor.2016.04.016
2. Fonseca, G.H.G., Santos, H.G., Carrano, E.G., Stidsen, T.R.: Integer programming techniques for educational timetabling. Eur. J. Oper. Res. **262**(1), 28–39 (2017). https://doi.org/10.1016/J.EJOR.2017.03.020, https://doi.org/10.1016/j.ejor.2017.03.020
3. da Fonseca, G.H.G., Santos, H.G., Toffolo, T.Â.M., Brito, S.S., Souza, M.J.F.: GOAL solver: a hybrid local search based solver for high school timetabling. Ann. Oper. Res. **239**(1), 77–97 (2016). https://doi.org/10.1007/S10479-014-1685-4, https://doi.org/10.1007/s10479-014-1685-4
4. Fonseca, G., Santos, H.G., Carrano, E.G., Stidsen, T.: Modelling and solving university course timetabling problems through xhstt. In: PATAT. vol. 16, pp. 127–138 (2016)
5. Kristiansen, S., Sørensen, M., Stidsen, T.R.: Integer programming for the generalized high school timetabling problem. Journal of Scheduling **18**, 377–392 (2015)
6. Lewis, R., Paechter, B., McCollum, B., et al.: Post enrolment based course timetabling: A description of the problem model used for track two of the second international timetabling competition. Cardiff Business School Cardiff, Wales (2007)

7. Mansour, N., El-Jazzar, H.: Curriculum based course timetabling. In: Wang, H., Yuen, S.Y., Wang, L., Shao, L., Wang, X. (eds.) Ninth International Conference on Natural Computation, ICNC 2013, Shenyang, China, July 23-25, 2013. pp. 787–792. IEEE (2013). https://doi.org/10.1109/ICNC.2013.6818082, https://doi.org/10.1109/ICNC.2013.6818082
8. McCollum, B., Schaerf, A., Paechter, B., McMullan, P., Lewis, R., Parkes, A.J., Gaspero, L.D., Qu, R., Burke, E.K.: Setting the research agenda in automated timetabling: The second international timetabling competition. INFORMS J. Comput. **22**(1), 120–130 (2010). https://doi.org/10.1287/IJOC.1090.0320, https://doi.org/10.1287/ijoc.1090.0320
9. Müller, T., Rudová, H., Müllerová, Z., et al.: University course timetabling and international timetabling competition 2019. In: Proceedings of the 12th International Conference on the Practice and Theory of Automated Timetabling (PATAT-2018). vol. 1, pp. 5–31 (2018)
10. Post, G., Gaspero, L.D., Kingston, J.H., McCollum, B., Schaerf, A.: The third international timetabling competition. Ann. Oper. Res. **239**(1), 69–75 (2016). https://doi.org/10.1007/S10479-013-1340-5, https://doi.org/10.1007/s10479-013-1340-5
11. Post, G., Kingston, J.H., Ahmadi, S., Daskalaki, S., Gogos, C., Kyngäs, J., Nurmi, C., Musliu, N., Pillay, N., Santos, H.G., Schaerf, A.: XHSTT: an XML archive for high school timetabling problems in different countries. Ann. Oper. Res. **218**(1), 295–301 (2014). https://doi.org/10.1007/S10479-011-1012-2, https://doi.org/10.1007/s10479-011-1012-2
12. Ruiz-Torrubiano, R., Knopp, S., Wolf, L.M., Krystallidis, A.: A scheduling perspective on modular educational systems in europe (2024), https://arxiv.org/abs/2403.05549

Theoretical Lower Bounds for the Oven Scheduling Problem

Francesca Da Ros[1] [0000-0001-7026-4165], Marie-Louise Lackner[2] [0000-0002-9916-9011], and Nysret Musliu[2] [0000-0002-3992-8637]

[1] DMIF, University of Udine, Italy francesca.daros@uniud.it
[2] Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling, Institute for Logic and Computation, TU Wien, Austria
{marie-louise.lackner, nysret.musliu}@tuwien.ac.at

Abstract. The Oven Scheduling Problem (OSP) is an NP-hard real-world parallel batch scheduling problem arising in the semiconductor industry. The objective of the problem is to schedule a set of jobs on ovens while minimizing several factors, namely total oven runtime, job tardiness, and setup costs. At the same time, it must adhere to various constraints such as oven eligibility and availability, job release dates, setup times between batches, and oven capacity limitations. The key to obtaining efficient schedules is to process compatible jobs simultaneously in batches. In this paper, we develop theoretical, problem-specific lower bounds for the OSP that can be computed very quickly. We thoroughly examine these lower bounds, evaluating their quality and exploring their integration into existing solution methods. Specifically, we investigate their contribution to exact methods and a metaheuristic local search approach using simulated annealing. Moreover, these problem-specific lower bounds enable us to assess the solution quality for large instances for which exact methods often fail to provide tight lower bounds.

Keywords: Oven scheduling problem, Parallel batch scheduling, Lower bounds, Exact methods, Simulated annealing

1 Introduction

The semiconductor manufacturing sector has been identified as one of the most energy-intensive industries [17], particularly in the context of hardening electronic components in specialized heat treatment ovens. To mitigate energy consumption, one strategy involves grouping and processing compatible jobs together in batches to optimize resource utilization. Such scheduling tasks that aim to increase efficiency by processing multiple jobs simultaneously in batches are known as batch scheduling problems.

Over the last three decades, the scientific community has extensively investigated batch scheduling problems, as witnessed by the surveys by [15,4]. A multitude of problem variants, in the single or parallel machine setting, and each with distinct constraints and objectives imposed by different industries [16,18] have been studied. One such formulation, the Oven Scheduling Problem (OSP), was recently introduced by [10] and is particularly pertinent to semiconductor manufacturing. The goal of this problem is to efficiently schedule jobs on multiple ovens, aiming to minimize total oven runtime, job tardiness, and setup costs simultaneously. In order to reach these goals, compatible jobs are grouped and processed together in batches. Schedules must adhere to various constraints, including oven eligibility and availability, job release dates, setup times between batches, oven capacity limitations, and compatibility of job processing times.

The OSP was initially addressed using exact methods as well as a heuristic construction method: [10] proposed two different modeling approaches, encompassing Constraint Programming (CP) and Integer Linear Programming (ILP) model formulations. The exact approaches successfully identified optimal solutions for 38 out of 80 benchmark instances. However, for larger instances, optimal solutions were rarely obtained within a time-bound of one hour. In a later extended abstract, a metaheuristic local search approach based on Simulated Annealing (SA) was suggested by [11]. This approach showed promising results, as optimal solutions could often be reached quickly and non-optimal solutions were improved for numerous instances.

In practical settings, it is most often desirable to obtain solutions of sufficiently good, albeit not necessarily optimal, quality within a short time frame. However, assessing the solution quality becomes challenging in the absence of a baseline, i.e., when exact methods are not employed or do not deliver tight enough lower bounds on the objective value. Providing problem-specific, efficiently computable lower bounds on the optimal solution cost can thus be very helpful in assessing the quality of a solution. Moreover, lower bounds can aid existing solution approaches and increase their performance: in exact methods, they can be used to bound the range of variables, and in (meta-)heuristic search methods, they can be included in stopping criteria. Theoretical, problem-specific lower bounds have been developed for batch scheduling problems in the literature. [1] proposed a SA approach in a parallel batch setting and presented a procedure for calculating lower bounds on the makespan. Additionally, lower bounds on the makespan and total completion time have been addressed by [7]. The maximum lateness has been tackled by [13,6]. While these lower bounds have been proposed for different batching problems, not all features of the OSP have been considered previously.

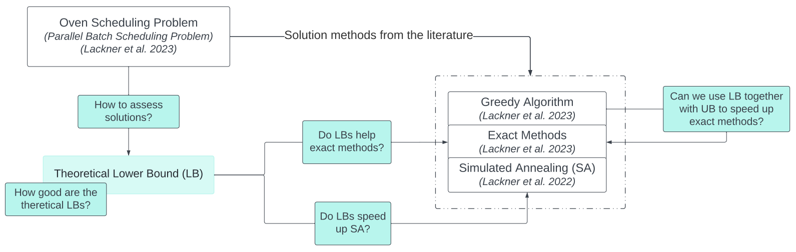

Oven Scheduling Problem
(Parallel Batch Scheduling Problem)
(Lackner et al. 2023)
Solution methods from the literature
How to assess solutions?
Theoretical Lower Bound (LB)
How good are the theretical LBs?
Do LBs help exact methods?
Do LBs speed up SA?
Greedy Algorithm
(Lackner et al. 2023)
Exact Methods
(Lackner et al. 2023)
Simulated Annealing (SA)
(Lackner et al. 2022)
Can we use LB together with UB to speed up exact methods?

Fig. 1: Overview of the goals targeted by this work.

In this paper, we present an approach to computing lower bounds for the objectives of the OSP, making use of the imposed machine eligibility and processing time constraints. The goals of this paper are visualized in 1. Our primary contributions are as follows:

- We introduce a procedure to compute theoretical lower bounds for the OSP, more specifically for the number of batches and the total oven runtime. These lower bound results can be adapted to tackle related (parallel) batch scheduling problems. Our approach differs from the existing literature as it considers machine eligibility and compatibility of processing times.
- We conduct a comprehensive evaluation of the tightness of our calculated lower bounds on a benchmark set consisting of 120 instances with up to 500 jobs. This evaluation encompasses the overall cost function and its individual components. We differentiate between instances where an optimal solution is available and those where it is not. Notably, for larger instances with 50 jobs or more, our calculated lower bounds provide a small gap w.r.t. the optimal solution value and very often outperform the lower bounds generated by commercial solvers (when the optimal solution value is not known).
- We integrate the derived lower bounds into state-of-the-art solution approaches and demonstrate that they can aid with solving the OSP. Our experiments explore to what extent exact methods benefit from being provided with the calculated lower bounds. Furthermore, we investigate whether lower bounds can speed up Local Search (LS) algorithms, such as SA. Using a 1% gap between the SA solution and the calculated lower bound as a stopping criterion, many of the benchmark instances can be solved very fast (50 of the 120 benchmark instances are solved in roughly 15 seconds on average).
- To encourage future contributions and enhance the replicability of results, we provide a software toolbox that enables the generation of instances and the calculation of lower bounds.

The remainder of this paper is structured as follows. 2 introduces the OSP. 3 elaborates on the theoretical calculations of lower bounds for the OSP. 4 displays proposals on how to integrate lower bounds in the solution methods. 5 details our experimental evaluation. Eventually, 6 draws some conclusions and suggests future research directions.

2 The Oven Scheduling Problem

The OSP aims to group compatible jobs into batches and devise an optimal schedule for these batches across a set of ovens. We report an abridged description of the problem and forward the interested reader to the rigorous mathematical formulation proposed in the original paper [10].

An instance of the OSP consists of a set $\mathcal{M} = \{1, \ldots, k\}$ of ovens (also referred to as machines) as well as a set $\mathcal{A} = \{1, \ldots, a\}$ of possible attributes (also known as job families in the literature). Each machine $m \in \mathcal{M}$ is associated with a maximal processing capacity c_m and an initial state $ia_m \in \mathcal{A}$. Each oven presents a set of availability intervals $[as(m,i), ae(m,i)]$, where $as(m,i)$ $(ae(m,i))$ indicates the start (end) of the i-th interval.

A set $\mathcal{J} = \{1, \ldots, n\}$ of jobs is given. Each job $j \in \mathcal{J}$ is described by an attribute $a_j \in \mathcal{A}$, a size $s_j \in \mathbb{N}$, an earliest start time (or release date) $et_j \in \mathbb{N}$, and a latest end time (or due date) $lt_j \in \mathbb{N}$. The processing of a job is constrained by its minimal and maximal processing times ($mint_j$ and $maxt_j \in \mathbb{N}$, respectively). Additionally, jobs have eligibility constraints, limiting their assignment to specific machines (indicated with the set $\mathcal{E}_j \subseteq \mathcal{M}$).

Setup times and costs are incurred between consecutive batches on the same machine and depend upon the attributes of the batches (attributes of jobs in the batch). They are indicated with two $(a \times a)$-matrices of setup times $st = (st(a_i, a_j))_{1 \leq a_i, a_j \leq a}$ and of setup costs $sc = (sc(a_i, a_j))_{1 \leq a_i, a_j \leq a}$ are given to denote the setup times (costs) incurred between a batch with attribute a_i and a subsequent one with attribute a_j.

The OSP aims to establish a feasible assignment of jobs to ovens, grouping them into batches, and to determine the schedule of batches on the ovens. A feasible batch construction and schedule must respect the following rules:

- **Attribute homogeneity**: Jobs in the same batch must share the attribute.
- **Release date**: A batch cannot start processing until the release date of the latest-released job assigned to it.
- **Processing time**: The processing time of a batch must be longer than or equal to the minimal processing time and shorter than or equal to the maximal processing time of any job in the batch. Jobs in the same batch start and finish processing at the same time and job-preemption is not allowed.
- **Setup time**: Batches on the same machine may not overlap, and setup times between consecutive batches need to be respected.
- **Machine eligibility**: Jobs can only be assigned to one of their eligible machines.
- **Machine availability**: For every batch, the entire processing time and the preceding setup time must be scheduled within a single availability interval.
- **Machine capacity**: The size of each batch cannot exceed the capacity of the machine it is assigned to.

The objective of the OSP is threefold: to minimize the cumulative batch processing time (p), the number of tardy jobs (t), and the cumulative setup costs (sc). Given a solution to the OSP, the three objective components are formally defined as follows:

$$p = \sum_{m \in \mathcal{M}} \sum_{b \in \mathcal{B}_m} P_{m,b}, \qquad t = \left|\left\{ j \in \mathcal{J} : C_j > lt_j \right\}\right| \quad \text{and } sc = \sum_{m \in \mathcal{M}} \sum_{b \in \mathcal{B}_m} sc_{m,b}$$

where $\mathcal{B}_m$ is the set of batches of machine $m \in \mathcal{M}$ that composes the solution. The processing time of batch $b \in \mathcal{B}_m$ on machine $m \in \mathcal{M}$ is indicated with $P_{m,b}$. The total tardiness is calculated as the number of jobs $j \in \mathcal{J}$ for which the completion time C_j is greater than their due date lt_j in the given solution. The setup cost of batch b on machine m is denoted by $sc_{m,b}$. Each component is then normalized and aggregated in a weighted sum to account for different real-world scenarios. The weights used throughout this paper are set as follows: $w_p = 4$, $w_t = 100$, and $w_{sc} = 1$ (these are also normalized by their sum, see Use Case 1 by [10]). To illustrate the problem, 6 reports an example instance for the OSP.

2.1 Solution methods for the OSP

In the literature, the OSP has been solved with a construction heuristic [10], exact methods [10], and a SA algorithm [11] which we very briefly describe here.

The construction heuristic introduced to solve the OSP [10] is a dispatching rule that prioritizes jobs based on their release dates and then on their due dates. The algorithm starts at time 0. At each time step, it compiles the list of currently available machines and currently released jobs that have not yet been scheduled. The algorithm then selects the job with the earliest due date from this pool and greedily assigns it to one of the eligible machines. Once a job is scheduled, other available jobs are included in the same batch, provided that the job's attribute, processing time, and the machine's capacity allow it. If no job can be scheduled, the time is incremented by one, and the process is repeated. This heuristic has been used to warm-start the exact methods with some of the solvers [10] and as an initial solution for the SA approach [11].

Two exact modeling approaches which were formulated as CP and ILP models were proposed by [10]. The first approach is based on batch positions: each job is assigned to one of the possible batches, which are uniquely characterized by their machine and the batch position on this machine. The constraints are formulated on the level of batches and an optimal schedule of the batches needs to be found. The second uses a unique representative job for each batch and seeks an optimal schedule for these jobs. These two modeling approaches are implemented both in the high-level solver-independent modeling language MiniZinc [22] and using interval variables in the Optimization Programming Language (OPL) [5] used by CP Optimizer. Moreover, different state-of-the-art solvers, search strategies, and a warm-start approach leveraging the construction heuristic were employed. Ultimately, the best results were achieved with `CP Optimizer` and the OPL-model using representative jobs as well as with `Gurobi` and the MiniZinc-model with batch positions. In what follows, we will refer to these two solution methods as "cpopt" and "mzn-gurobi" (as well as "cpopt-WS" and "mzn-gurobi-WS" for the variants with warmstart).

A SA algorithm for the OSP was proposed by [11]. In this algorithm, a solution to the OSP is represented by the assignments of jobs to ovens and by the processing order of the jobs on their respective machines. The schedule of the batches on the ovens is then deterministically constructed from this representation. The initial solution is retrieved from the construction heuristic previously presented. The algorithm relies on four neighborhood-moves: the Swap Consecutive Batches (SCB) move, which swaps consecutive batches on the same machine; the Insert Batch (IB) move, which inserts a given batch in a new position on the same machine; the Move Job to Existing Batch (MJEB) move, which inserts a job j in an existing batch; the Move Job to New Batch (MJNB) move, which inserts a job in a newly created batch. In the original work by [11], SA was proposed with a preliminary manual tuning, whereas we fine-tuned its parameters for this work.

3 Lower bounds on the optimal solution cost

In this section, we describe a procedure to calculate lower bounds on the optimal solution cost for a given instance of the OSP. Our main focus lies in bounding the number of batches required in any feasible solution. At the same time, we derive bounds on the cumulative batch processing time. These lower bounds serve as a basis for deriving lower bounds on the cumulative setup costs. Finally, we provide a brief discussion on the number of tardy jobs.

3.1 Minimum number of batches required and minimal cumulative batch processing time

Since jobs can only be combined in a batch if they share the same attribute, bounds on the number of batches required are calculated independently for all attributes. For a given attribute $r \in \mathcal{A}$, we denote by b_r the number of batches in a feasible solution and by p_r the minimal cumulative processing time of batches.

Bound based on machine capacities and job sizes. Due to the capacity constraints of machines, a simple bound on the number of batches required is

$$b_r \geq \left\lceil \frac{\sum_{j \in \mathcal{J}: a_j = r} s_j}{\max_{m \in \mathcal{M}}\{c_m\}} \right\rceil, \tag{1}$$

as stated by [7]. This corresponds to the minimal number of batches required if we assume that jobs can be split into smaller jobs of unit size and that all jobs can be scheduled on the machine with the largest machine capacity.

This bound can be tightened by distinguishing between “large” and “small” jobs (in a similar fashion as [1,12,13]). Large jobs are those jobs that are so large that they cannot accommodate any other jobs in the same batch and thus need to be processed in a batch of their own. All other jobs are referred to as small jobs. For a given attribute r, the sets of large jobs J_r^l and small jobs J_r^s with attribute r are thus defined as follows:

$$J_r^l = \left\{ j \in \mathcal{J} : a_j = r, s_j + s_i > \max_{m \in \mathcal{E}_j}(c_m) \quad \forall i \in \mathcal{J} \text{ with } i \neq j \text{ and } a_i = r \right\},$$
$$J_r^s = \left\{ j \in \mathcal{J} : a_j = r \right\} \setminus J_r^l$$

Instead of the bound in equation (1), we thus have the tighter bound:

$$b_r \geq |J_r^l| + \left\lceil \frac{\sum_{j \in J_r^s} s_j}{\max_{m \in \mathcal{M}}\{c_m\}} \right\rceil. \tag{2}$$

In the following, we refine these bounds from the literature by considering machine eligibility and compatibility of processing times.

Refinement of the bound for small jobs based on machine eligibility. Considering the small jobs of attribute $r \in \mathcal{A}$, we further distinguish them between those that can be processed on several machines and those with a single eligible machine. Given a machine $i \in \mathcal{M}$, we use the following notation:

$$b_{r,i} = \frac{\sum_{j \in J_r^s : \mathcal{E}_j = \{i\}} s_j}{c_i}, \text{ and } cap_i = (\lceil b_{r,i} \rceil - b_{r,i}) \cdot c_i$$

i.e., $\lceil b_{r,i} \rceil$ is the minimal number of batches with small jobs that need to be processed on machine i and cap_i is the total remaining capacity in these batches.

To schedule the small jobs of attribute r, we proceed as follows:

- All small jobs that need to be processed on a specific machine are scheduled on this machine.
- The remaining small jobs are used to fill up the previously created batches.
- If there are still jobs left, we assume that they can be split into unit-size jobs and can be scheduled on the machine with maximal capacity, creating b_r^* additional batches.

The bound in equation (2) can then be tightened as follows:

$$b_r \geq b_r^E = |J_r^l| + \sum_{i \in \mathcal{M}} \lceil b_{r,i} \rceil + \underbrace{\left\lceil \frac{\max\left(0, \sum_{j \in J_r^s : |\mathcal{E}_j| > 1} s_j - \sum_{i \in \mathcal{M}} cap_i\right)}{\max_{m \in \mathcal{M}}\{c_m\}} \right\rceil}_{=b_r^*} \tag{3}$$

In order to calculate a lower bound on the cumulative batch processing time p_r of these batches, note that all large jobs are processed in batches of their own which run for their respective minimal processing times. Thus

$$p_r = \sum_{j \in J_r^l} mint_j + p_r^E, \tag{4}$$

where p_r^E denotes the minimal cumulative processing time of batches consisting of small jobs with attribute r. A bound for p_r^E can be calculated as follows:

- For every machine i with $b_{r,i} > 0$, create the collection of minimal processing times of small jobs that need to be processed on i; create the sum of the $\lceil b_{r,i} \rceil$ smallest elements from this collection.
- From the collection of minimal processing times of small jobs that can be processed on multiple machines, create the sum of the b_r^* smallest elements.
- Among all small jobs, pick the one with the largest minimal processing time. The batch containing this job will necessarily have this job's minimal processing time. In the previous two sums, one can thus replace the overall largest processing time with this value.

Alternative refinement of the bound for small jobs based on compatible job processing times. Two jobs i and j with respective minimal and maximal processing times $mint_i, mint_j$ and $maxt_i, maxt_j$ may only be combined in a batch if the intervals of their processing times have a non-empty intersection:

$$[mint_i, maxt_i] \cap [mint_j, maxt_j] \neq \emptyset. \tag{5}$$

This compatibility relation between jobs can be represented with the help of a *compatibility graph* $G = (V, E)$, where V is the set of all jobs $\mathcal{I}$ and $(i, j) \in E$ if and only if the jobs i and j have compatible processing times. In this graph, a batch forms a (not necessarily maximal) clique. The problem of solving an OSP instance with unit-sized jobs and a single machine with capacity c is thus equivalent to covering the nodes of the compatibility graph with the smallest number of cliques with size no larger than c.

This problem is NP-complete for arbitrary graphs, but solvable in polynomial time for interval graphs. A simple greedy algorithm is provided by [3] and referred to as the algorithm GAC (greedy algorithm with compatibility). By adapting the order in which jobs are processed by the GAC algorithm, we obtain an algorithm that minimizes both the number of batches and the cumulative batch processing time. We call this algorithm GAC+.

Algorithm GAC+: Consider the jobs in non-increasing order $j_1, j_2, \dots, j_n$ of their minimal processing times $mint_j$, breaking ties arbitrarily. Construct one batch per iteration until all jobs have been placed into batches. In iteration i, open a new batch B_i and label it with the first job j^* that has not yet been placed in a batch. Starting with $j^* = [mint_{j^*}, maxt_{j^*}]$, place into B_i the first c not yet scheduled jobs j for which $mint_{j^*} \in [mint_j, maxt_j]$.

For a set $\mathcal{J}$ of jobs with arbitrary job sizes, let $GACb(\mathcal{J}, c)$ denote the number of batches returned by the GAC+ algorithm when replacing every job $j \in \mathcal{J}$ with s_j identical copies of unit size jobs. Similarly, let $GACp(\mathcal{J}, c)$ denote the minimal processing time returned by the GAC+ algorithm for this instance. With this notation, we obtain the following bounds:

$$b_r \geq |J_r^l| + b_r^C, \qquad \text{with } b_r^C = GACb(J_r^s, \max_{m \in \mathcal{M}}\{c_m\}), \tag{6}$$

$$p_r \geq \sum_{j \in J_r^l} mint_j + p_r^C, \qquad \text{with } p_r^C = GACp(J_r^s, \max_{m \in \mathcal{M}}\{c_m\}). \tag{7}$$

For a formal statement and proof of this result, see Section 6 of the appendix.

Overall bound on the number of batches and the minimal cumulative processing time. Combining the previously established bounds, we obtain:

$$b \geq \sum_{r=1}^{a}(|J_r^l| + \max(b_r^E, b_r^C)) \text{ and } p \geq \sum_{r=1}^{a}(\sum_{j \in J_r^l} mint_j + \max(p_r^E, p_r^C)),$$

where b_r^E is defined in equation (3) and b_r^C in equation (6), the procedure to calculate p_r^E is described right after equation (4) and p_r^C is defined in equation 7.

3.2 Bounds on the other components of the objective function

Setup costs. If we assume that the setup costs *before* batches of a given attribute are always minimal, we obtain the following bound on the setup costs:

$$sc \geq \sum_{r=1}^{a} b_r \cdot \min_{s \in \{1,\dots,a\}}\{sc(s, r)\}. \tag{8}$$

A similar bound can be derived assuming that the setup costs *after* batches are always minimal. For this case, we include initial setup costs for all machines to which batches are scheduled and ignore the last batch on every machine. Since a prior it is not known which machines are used in a schedule, we create the list `setup_costs` as follows. For every attribute r, we add b_r copies of $\min_{s \in \{1,\dots,a\}}\{sc(r, s)\}$ to `setup_costs`. Moreover, for every machine m, we add the element $\min_{s\in\{1,\ldots,a\}}\{sc(ia_m, s)\}$ to `setup_costs`. The list is then sorted in non-decreasing order and the sum of the first b elements is taken:

$$sc \geq \sum_{i=1}^{b} \texttt{setup\_costs}(i). \tag{9}$$

Altogether, we have the following lower bound on the setup costs

$$sc \geq \max\left(\sum_{r=1}^{a} b_r \cdot \min_{s\in\{1,\ldots,a\}}\{sc(s,r)\}, \sum_{i=1}^{b} \texttt{setup\_costs}(i)\right). \tag{10}$$

Note that it is impossible to obtain a lower bound on the setup costs by arranging the minimum number of batches per attribute (as calculated previously) in an order that minimizes the cumulative setup costs. Indeed, if the matrix of setup costs does not fulfill the triangle inequality, it can be advantageous to introduce additional batches if the sole objective is to reduce setup costs.

Number of tardy jobs. Regarding the number of tardy jobs, direct inference from the instance itself may be limited. However, we can obtain a lower bound on the number of tardy jobs by independently scheduling each job in a batch on its own on the first available machine and computing the completion time. Any job finishing after its latest end date is necessarily tardy in every solution.

4 Including lower bounds in solution methods

A recommended practice to build efficient exact models is to tightly restrict and bound the domain of variables (as suggested, for instance, by the MiniZinc guide on efficient modeling practices[3]). By employing tighter variable bounds, algorithmic efficiency can be significantly enhanced, facilitating faster convergence to optimal solutions or the identification of unfeasible regions. When solving the OSP with one of the exact methods, the lower bounds derived in Section 3 can be calculated in a preprocessing step and can then be provided to the model as part of the input data. The range of the variables corresponding to the individual objective components as well as the variable for the aggregated objective function can thus be bounded from below. Moreover, the aggregated objective value of the solution delivered by the construction heuristic can be used to bound the range of the objective function from above.

Problem-specific lower bounds can also have practical applications in metaheuristic algorithms, e.g., in SA. Lower bounds can be used to guide the search, e.g., as part of the termination criterion. This strategy allows for early interruption of the process, sparing computational resources while still achieving satisfactory solution quality

5 Experimental evaluation

In this experimental evaluation, we aim to analyze the quality of the theoretically derived lower bounds and their practical usefulness in helping to solve the OSP.

[3]see https://www.minizinc.org/doc-2.5.5/en/efficient.html

5.1 Benchmark instances

We consider the 80 benchmark instances by [9], which differ per number of jobs (10, 25, 50, or 100), number of machines (2 or 5), and number of attributes (2 or 5).

Moreover, we consider 40 new instances featuring a larger number of jobs (250 or 500) to reflect real-world scenarios better. This new set is generated using the specifications of the random instance generator provided by [8]. The instances can be retrieved from the public public GitHub repository https://github.com/iolab-uniud/osp-ls/.

For tuning purposes (i.e., when using SA), we generate 25 additional instances with similar characteristics as the initial benchmark set.

5.2 Experimental setup

We consider the following methods for the OSP:

- **Problem-specific lower bounds** (presented in 3): For the instances we consider, the bounds are calculated in 2.9 seconds on average (with a standard deviation of 6.9 s).
- **Construction heuristic** (proposed by [10], see 2.1): Since the solution is deterministically constructed, there is no need to execute the algorithm more than once. For the instance we consider, the solutions are retrieved in 0.2 seconds on average (with a standard deviation of 0.4 s).
- **Best performing exact methods** (proposed by [10], see 2.1): We refer to the methods as “cpopt” (interval variable model with representative jobs solved with `CP Optimizer`) and “mzn-gurobi” (MiniZinc-model with batch positions solved with `Gurobi`), as well as “cpopt-WS” and “mzn-gurobi-WS” for the variants with warm-start. Each method is run with a timeout of 1 hour per instance.
- **Local search approach with SA**(proposed by [11], see 2.1): The algorithm is tuned using automated parameter tuning with `irace` [14]. To account for the stochastic components of SA, we execute the algorithm 10 times per instance with a timeout of 6 minutes. Every 2 seconds we record the overall cost and the single objective components of the best solution encountered so far.

Details regarding the implementation, the tuned parameters of the SA and the hardware can be found in Section 6 of the appendix.

5.3 Lower bounds quality

Our objective is to assess the tightness of the calculated lower bounds. We examine the bound on the overall cost (obj) as well as the bounds on its three components individually (t, p, and sc). For the smaller benchmark instances with up to 100 jobs and the aggregated objection function, we refer to the best results per instance obtained by [10] with their proposed exact methods. For the larger benchmark instances with 250 or 500 jobs, we rerun the best-performing exact methods (“mzn-gurobi” and “mzn-gurobi-WS” as well as “cpopt” and “cpopt-WS”) and retrieve the best result per instance. Moreover, we run the exact models with the task of optimizing just one of the three components for the entire benchmark set. In our analysis of the lower bounds, we differentiate between those instances and objectives where an optimal solution cost is known and those where we do not know the optimum.

For those instances and objectives where the optimum solution is known, given an instance i, we compute the relative gap(i) between the calculated lower bound $b(i)$ and the optimal cost $s(i)$; specifically gap$(i) = 100 \cdot (s(i) - b(i))/s(i)$. Results show the general tendency that the larger the instances, the smaller the gap (see 2). Concerning the individual components, we observe that most room for improvement is left for the simple bounds for sc and t. Nonetheless, the gap for sc is less than 25% for more than half of the instances and the gap for t is less than 10% for 74% of the instances. For the cumulative processing times, the gap is less than 25% for 88% of instances and less than 10% for 61%. The results are promising, as they give reason to hope that the bounds are relatively tight for instances where the optimum is not known as well.

Whenever the optimal solution value is not known, we compare the problem-specific lower bounds with the lower bounds retrieved by `CP Optimizer` and `Gurobi` (specifically, "cpopt", "cpopt-WS", "mzn-gurobi", and "mzn-gurobi-WS") and retrieve the best, i.e., largest, lower bound found per instance. For each objective, we count how often the calculated lower bounds are better, worse, or equal to the best dual bounds found by the exact methods, see 1. The results show that both for the overall cost and its components, the calculated problem-specific lower bounds are better than those provided by any of the exact methods in the majority of the instances. The dominance of the problem-specific lower bounds is particularly clear for the larger instances with 100 jobs or more. Interestingly, this observation holds even for the objective components "setup costs" (problem-specific bounds are better or equally good in 2/3 of the instances) and "number of tardy jobs" (better or equally good results in 94 % of the instances) for which the calculated bounds are very simple.

Moreover, we investigated the gap between the calculated lower bounds and the upper bounds provided by the construction heuristic (see 2.1). For a total of 57 instances, this gap is less than 10% (see 6 for details).

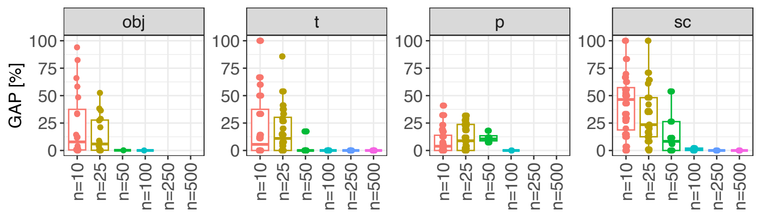


Fig. 2: Gap[%] between the known optimum and the calculated lower bounds.

Table 1: Comparison of the quality of calculated problem-specific ("calc.") and best solver lower bounds ("solv."). We consider only those instances for which no optimal solution is known. The label "calc." refers to the number of instances where the calculated bounds are better, "solv." to those where the solver bounds are better and "equ." to those where the bounds are equal.

	obj			t			p			sc		
	calc.	solv.	equ.	calc.	solv.	equ.	calc.	solv.	equ.	calc.	solv.	equ.
n	#	#	#	#	#	#	#	#	#	#	#	#
25	0	4	0	0	0	0	1	0	0	1	0	0
50	7	12	0	0	6	2	10	6	0	4	7	2
100	16	3	0	1	6	0	18	1	0	15	1	2
250	20	0	0	10	1	1	20	0	0	18	0	0
500	20	0	0	10	0	2	20	0	0	18	0	0
all	63	19	0	21	13	5	69	7	0	56	8	4

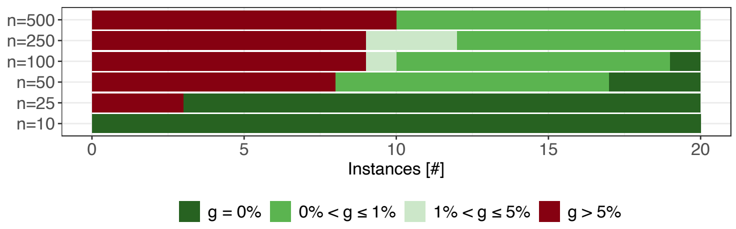


Fig. 3: Gap[%] between the best solution found and the best lower bounds.

5.4 Measuring solution quality

In this section, we use the lower bounds to assess the solution quality and benchmark the best-known solutions for the OSP with the best lower bounds. On the one hand, we consider the best solution found for each instance by the methods described in 2.1. On the other hand, we consider the best lower bound per instance among the calculated bounds and the ones retrieved by the exact methods. Then we calculate the relative gap between the best solution and the best lower bound per instance. Results are shown in 3. Almost all small instances with 25 or 50 jobs could be solved optimally. For larger instances, the solution methods find very good solutions (with a relative gap[%] $\leq$ 1%) for roughly half the instances. For most of the remaining instances, the gap is larger than 5%, showing that there is still room for improvement–both in terms of the solution quality and in terms of the lower bound quality.

5.5 Application of lower bounds

Exact methods We aim to understand whether using the calculated problem-specific lower bounds allows the exact methods to improve their results. As described in Section 4, we perform experiments where the objective function and its components are bounded from below by the calculated lower bounds. Moreover, we perform experiments additionally supplying the solvers with the upper bound on the objective obtained from the greedy construction heuristic.

2 presents results categorized by methods and types of bounds included; it displays: the number of instances for which the optimal solution, when known, was reached ("optimal"); the number of instances for which a feasible solution was found ("solved"), the number of instances for which the method could prove optimality ("proven opt"); the number of instances for which the best solution was found ("best"); the number of instances for which the best lower bound could be found ("best lower bound"); the average run time ("avg rt") and its standard deviation ("std rt") in seconds. Note that for the number of best solutions found and of best lower bounds found, the comparison is made among a single solution method, i.e., comparing results obtained when no non-trivial bounds are provided, when lower bound and when lower and upper bounds are provided. The statistics regarding runtime are calculated for the subset of instances for which the respective solution method could prove optimality when it was not provided with bounds (meaning that instances for which the time-out was reached are not included). The majority of solution methods, namely "mzn-gurobi", "cpopt" and "cpopt-WS", demonstrate greatly improved performance and solution quality when lower bounds are incorporated. For "mzn-gurobi-WS"[4], the contribution of the bounds is less clear: fewer instances are solved (optimally), but better solutions and better lower bounds can be found. The inclusion of upper bounds is not always advantageous for the exact methods, meaning that the solvers were not capable of finding a solution that was at least as good as the greedy solution within a time limit of 1 hour. For all analyzed solution methods, the presence of bounds facilitates the discovery of improved lower bounds by the commercial solvers, thus contributing to closing the optimality gap.

Table 2: Comparison of the results obtained with exact methods with and without the inclusion of bounds. Best results per solution method and performance parameter are highlighted in bold font. Numbers in brackets indicate the improvement obtained by supplying the respective solution methods with bounds.

solution	bounds incl.	optimal	solved	proven opt	best	best LB	avg rt	std rt
method	in model	#	#	#	#	#	(in s)	(in s)
mzn-gurobi	none	40	64	31	51	41	429.5	860.6
	LB	**41** (+1)	**78** (+14)	**36** (+5)	62 (+11)	**62** (+21)	**189.9**	387.6
	LB + UB	40 (+0)	73 (+9)	35 (+4)	**64** (+13)	55 (+14)	235.8	542.0
mzn-gurobi -WS	none	**41**	**89**	**34**	57	40	764.3	1217.9
	LB	40 (-1)	87 (-2)	**34** (+0)	**68** (+11)	65 (+25)	505.4	1069.8
	LB + UB	**41** (+0)	84 (-5)	**34** (+0)	65 (+8)	**70** (+30)	**493.9**	1083.2
cpopt	none	39	**114**	28	73	28	18.4	34.1
	LB	**40** (+1)	**114** (+0)	**33** (+5)	**79** (+6)	**118** (+90)	**17.8**	46.4
	LB + UB	39 (+0)	85 (-29)	**33** (+5)	57 (-16)	110 (+82)	17.9	43.1
cpopt-WS	none	38	**120**	28	70	28	17.6	30.0
	LB	**40** (+2)	**120** (+0)	**33** (+5)	81 (+11)	**118** (+90)	**15.9**	31.7
	LB + UB	**40** (+2)	**120** (+0)	**33** (+5)	**83** (+13)	117 (+89)	19.9	42.9

[4]The warm-start data provided to Gurobi only contains values for a subset of the decision variables. The solver thus needs to complete the partial solution and, for "mzn-gurobi-WS', fails to do so for many large instances.

3 offers a comprehensive comparison of overall best results. The inclusion of bounds enabled the methods to deliver three new optimality proofs and to find 23 better solutions. Additionally, the computational time reduces when bounds are utilized compared to when they are not.

Table 3: Overall comparison of the best results per instance achieved with exact methods without the inclusion of bounds and with the inclusion of bounds.

bounds included	optimal #	solved #	proven opt #	best #	best lower bound #	avg rt (in s)	std rt (in s)
no	41	120	38	76	42	486.8	1075.6
yes	41 (+0)	120 (+0)	41 (+3)	99 (+23)	116 (+74)	107.6	256.9

Local search Lower bounds provide a means to assess whether it is feasible to halt the search before reaching the termination criterion – in our case, the timeout. We aim to discern under which circumstances this is viable and how much time is necessary. Considering the overall cost, for 50 out of 120 instances, the gap[%] is lower than 1% (average time required 15.52 ± 39.85 s); for 60, the gap[%] is lower than 5% (average time required 3.86 ± 20.21 s), and for 67, it is lower than 10% (average time required 11.13 ± 34.92 s). This means that for roughly half of the benchmark instances, the search could be terminated early, delivering a solution of good quality. It is worth pointing out that this is merit also of a demonstrably good initial solution (see 6). 4 reports the distribution of minimum time required by SA to achieve such results.

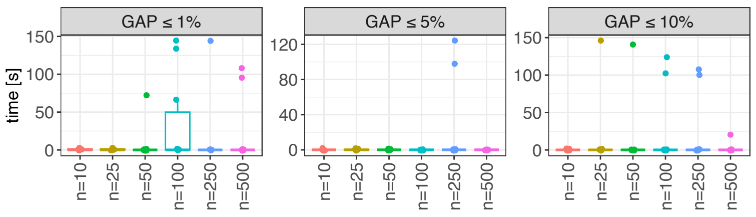


Fig. 4: Minimum time required by SA to reach a given gap[%] w.r.t. obj.

6 Conclusion

In this study, we introduced a procedure for calculating theoretical lower bounds for the OSP which can be calculated within a couple of seconds even for large instances. The experimental evaluation demonstrated their quality and practical utility when incorporated into exact methods or LS approaches. Our bounds can help to find better solutions, to deliver more optimality proofs, and to find high-quality solutions in a shorter time.

Notably, some of the bounds we developed are relatively simple, in particular those concerning job tardiness. This suggests that there is potential for further enhancements by refining these lower bounds with more sophisticated methods. Therefore, future extensions will focus on improving the presented bounds. Additionally, we aim to explore adaptive local search techniques, wherein neighborhood probabilities dynamically adjust based on the proximity to the lower bounds. Moreover, investigating alternative use cases, such as employing different weight sets on the objective function, may offer valuable insights.

Replicability The software toolbox can be retrieved from the public GitHub repository https://github.com/marielouiselackner/OvenSchedulingCLI, and the new benchmark instances are available at https://github.com/iolab-uniud/osp-ls/.

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, and SPECIES is gratefully acknowledged.

Appendix A – Example of an OSP Instance.

To better exemplify the problem, let us consider the following randomly created instance consisting of 10 jobs ($n = 10$), 2 machines ($k = 2$), and 2 attributes ($a = 2$). It presents the following characteristics:

m	M_1	M_2
c_m	18	20
ia_m	1	2
$[as, ae]$	[21,250]	[103,259]

$$st = \begin{pmatrix} 0 & 0 \\ 3 & 8 \end{pmatrix} \qquad sc = \begin{pmatrix} 6 & 8 \\ 10 & 10 \end{pmatrix}$$

j	1	2	3	4	5	6	7	8	9	10
$\mathcal{E}_j$	M_1 M_2	M_1 M_2	M_2	M_1	M_1 M_2	M_2	M_1 M_2	M_1	M_2	M_1 M_2
et_j	2	3	8	1	39	41	40	31	27	16
lt_j	16	20	43	24	55	64	56	89	58	27
$mint_j$	11	10	19	19	10	19	11	50	19	11
$maxt_j$	11	50	19	19	50	50	50	50	19	50
s_j	18	16	17	2	6	19	11	11	4	14
a_j	2	2	2	1	2	2	2	2	1	1

5 reports a possible solution to such an instance in the form of a Gantt Chart. The running time of the oven is $p = 158$, the number of tardy jobs is $t = 8$, and the setup costs amount to $sc = 72$. This solution is optimal when setting the weights in the objective function as $w_p = 4$, $w_t = 100$, and $w_{sc} = 1$.

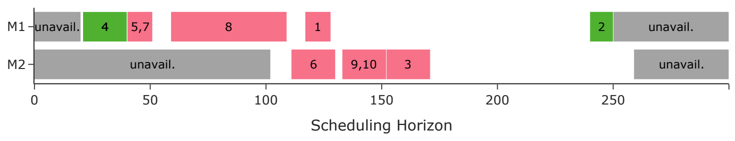


Fig. 5: Gantt chart of a solution of the OSP for an instance with 10 jobs. The label of each bar represents the jobs processed in the batch. Unavailabilities ("unavail.") are reported in gray. Batches with attribute 1 are colored in green, whereas those referring to attribute 2 are colored in magenta.

Appendix B – Formal statement and proof of the correctness of the GAC-bounds described in Section 3.1

In the following, we formulate the bounds described in the Section entitled *Alternative refinement of the bound for small jobs based on compatible job processing times* (starting on page 170) more formally and prove the correctness of the algorithm GAC+.

First, let us recall the compatibility requirement expressed in equation (5). Two jobs i and j with respective minimal and maximal processing times $mint_i, mint_j$ and $maxt_i, maxt_j$ may only be combined in a batch if the intervals of their processing times have a non-empty intersection:

$$[mint_i, maxt_i] \cap [mint_j, maxt_j] \neq \emptyset.$$

Now let us consider the following special case of the OSP:

> OSP*: Given a set of jobs $\mathcal{I}$ of unit size defined by their minimal and maximal processing times, i.e. $j = [mint_j, maxt_j]$ for all $j \in \mathcal{I}$, and a single machine with capacity $c \in \mathbb{N}$, how many batches do we need at least in order to process all jobs if jobs can only be processed in the same batch if the compatibility condition (5) is fulfilled?

Several variants of this problem have been studied in the literature, e.g. by Finke et al. [3]; the variant that we are interested in corresponds to the problem (P2) there. Solving the problem OSP* will allow us to obtain lower bounds for the OSP: Indeed, as in equation (1), we obtain lower bounds on the number of batches required and their processing times if we assume that jobs can be split into smaller jobs of unit size and that all jobs can be scheduled on the machine with largest machine capacity.

As stated in 3, equation (5) between jobs can be represented with the help of a *compatibility graph* $G = (V, E)$, where V is the set of all jobs $\mathcal{I}$ and $(i, j) \in E$ if and only if the jobs i and j have compatible processing times. In this graph, a batch forms a (not necessarily maximal) clique. The problem of solving an OSP instance with unit-sized jobs and a single machine with capacity c is thus equivalent to covering the nodes of the compatibility graph with the smallest number of cliques with size no larger than c.

A simple greedy algorithm to solve this problem is provided by [3] and referred to as the algorithm GAC (greedy algorithm with compatibility). By adapting the order in which jobs are processed by the GAC algorithm, we obtain an algorithm that minimizes both the number of batches and the cumulative batch processing time. We call this algorithm GAC+.

Algorithm GAC+. Consider the set of jobs $\mathcal{I}$ in non-increasing order $j_1, j_2, \dots, j_n$ of their minimal processing times $mint_j$, breaking ties arbitrarily. Construct one batch per iteration until all jobs have been placed into batches. In iteration i, open a new batch B_i and label it with the first job j^* that has not yet been placed in a batch. Starting with $j^* = [mint_{j^*}, maxt_{j^*}]$, place into B_i the first c not yet scheduled jobs j for which $mint_{j^*} \in [mint_j, maxt_j]$ (or all of them if there are fewer than c).

For a set $\mathcal{I}$ of unit size jobs and a maximum batch size $c \in \mathbb{N}$, we denote by $GACb(\mathcal{I}, c)$ the number of batches returned by the GAC+ algorithm above. Similarly, let $GACp(\mathcal{I}, c)$ denote the minimal processing time returned by the GAC+ algorithm for this instance.

Theorem 1. *For any given set of unit size jobs $\mathcal{I}$ and for any given constant $c \in \mathbb{N}$, Algorithm GAC+ solves the problem OSP*, i.e., $GACb(\mathcal{I}, c)$ is the minimum number of batches required under the condition that a batch may not contain more than c jobs. Moreover, the cumulative batch processing time $GACp(\mathcal{I}, c)$ is minimal.*

By slight abuse of notation, for a set $\mathcal{J}$ of jobs with arbitrary job sizes, let $GACb(\mathcal{J}, c)$ denote the number of batches returned by the GAC+ algorithm when replacing every job $j \in \mathcal{J}$ with s_j identical copies of unit size jobs. Similarly, let $GACp(\mathcal{J}, c)$ denote the minimal processing time returned by the GAC+ algorithm for this instance. With this notation, Theorem 1 yields the bounds reported in (7).

Proof (of Theorem 1). We follow the proof of Theorem 4 in [3], extending it to include the minimization of the cumulative batch processing time and adapting it to our variant of the algorithm. The proof is by induction over the number of jobs and the induction start with a single job is trivial.

Let us start with a simple observation about the minimum number of batches and the minimal batch processing time. For this, let $b(\mathcal{I}, c)$ denote the minimum number of batches required to schedule all jobs in $\mathcal{I}$ under the condition that a batch may not contain more than c jobs. Similarly, let $p(\mathcal{I}, c)$ denote the minimal cumulative batch processing time in any schedule of all jobs in $\mathcal{I}$. Then these two functions are monotonous in $\mathcal{I}$, i.e.:

$$\begin{aligned} &b(\mathcal{I}, c) \geq b(\mathcal{I} \setminus \{j\}, c) \\ \text{and } &p(\mathcal{I}, c) \geq p(\mathcal{I} \setminus \{j\}, c), \text{ for every } j \in \mathcal{I}. \end{aligned} \tag{11}$$

For the induction step, let $\mathcal{B} = (B_1, B_2, \ldots, B_b)$ be a sequence of batches constructed by the algorithm GAC+ for $\mathcal{I}$, B_1 being the first batch constructed by the algorithm and $p \in \mathbb{N}$ being the cumulative batch processing time of $\mathcal{B}$. Let the label of B_1 be the job $i = [mint_i, maxt_i]$, i.e., $mint_i$ is maximal among the minimal processing times and the processing time of B_1 is equal to $mint_i$. For the set of jobs $\mathcal{I} \setminus B_1$, the algorithm constructs the batch sequence $B_2, \ldots, B_b$ (see the definition of GAC+). By the induction hypothesis we know that $B_2, \ldots, B_b$ is optimal for $\mathcal{I} \setminus B_1$, i.e., $b(\mathcal{I} \setminus B_1, c) = |\mathcal{B}| - 1 = b - 1$ and $p(\mathcal{I} \setminus B_1, c) = p - mint_i$. It thus suffices to show that there exists a batch sequence of minimal length and with minimal batch processing time that contains the batch B_1.

Let O_1 be the batch containing i in an optimal sequence of batches $\mathcal{O}$ and let us choose $\mathcal{O}$ such that the size of the intersection $|O_1 \cap B_1|$ is maximal. We will prove that $O_1 = B_1$.

First note that $i \in O_1$ implies that $mint_i \in [mint_j, maxt_j]$ for all jobs $j \in O_1$: $mint_j \leq mint_i$ since $mint_i$ is maximal and $mint_i \leq maxt_j$ since every job $j \in O_1$ needs to be compatible with i. Thus the processing time of batch O_1 is equal to $mint_i$.

We now distinguish two cases: $|B_1| < c$ and $|B_1| = c$, where c is the maximum batch size. If $|B_1| < c$, the batch B_1 contains all neighbors of i in the compatibility graph G corresponding to the set of jobs $\mathcal{I}$. Since O_1 is a clique containing i, it follows that $O_1 \subseteq B_1$. Then by monotonicity (as stated in equation (11)), we have

$$\begin{aligned} |\mathcal{B}| - 1 = b(\mathcal{I} \setminus B_1, c) &\leq b(\mathcal{I} \setminus O_1, c) = |\mathcal{O}| - 1 \\ \text{and } p - mint_1 = p(\mathcal{I} \setminus B_1, c) &\leq p(\mathcal{I} \setminus O_1, c) = p(\mathcal{I}, c) - mint_i, \end{aligned}$$

which proves that $\mathcal{B}$ is optimal both in terms of the number of batches and in terms of the cumulative processing time.

For the case $|B_1| = c$, we assume towards a contradiction that there exists a job $j = [mint_j, maxt_j] \in B_1 \setminus O_1$. This implies that there must also exist a job $k = [mint_k, maxt_k] \in O_1 \setminus B_1$. (As before, $O_1 \subset B_1$ would imply that $\mathcal{B}$ is optimal. However, the job j could have been added to O_1 without having an impact on the number of batches or the batch processing time required by $\mathcal{O}$. This however is a contradiction to the choice of $\mathcal{O}$). From the definition of the algorithm, we know that B_1 consists of the first c jobs containing $mint_i$. Therefore, $j < k$ and $mint_j \geq mint_k$. Moreover, as noted earlier, we know that $mint_i \in k = [mint_k, maxt_k]$ and thus $[mint_j, mint_i] \subseteq k$.

We then define $O'_1 := (O_1 \setminus \{k\}) \cup \{j\}$ and redefine the batch $O \in \mathcal{O}$ that contains j as $O' := (O \setminus \{j\}) \cup \{k\}$. Both these batches fulfill the compatibility constraint for the processing times: O'_1 does because $mint_i$ is contained in j and in all jobs in O_1 and O' does because all jobs that are compatible with j are also compatible with k (If job s is compatible with j, this means that $s = [mint_s, maxt_s] \cap [mint_j, mint_i] \neq \emptyset$, since $mint_i$ is maximal among all minimal processing times. On the other hand, we already noted that $[mint_j, mint_i] \subseteq k$ and thus $s \cap k \neq \emptyset$, which means that s and k are compatible.) As for the processing times of the batches, both batches O_1 and O'_1 have the processing time $mint_i$ as they contain job i. For the batch O', we have replaced the job i with a job with smaller or equal processing time ($mint_i \geq mint_k$). Thus the processing time of batch O' is smaller or equal to the batch processing time of O. We have thus produced another optimal sequence of batches $\mathcal{O}' = \mathcal{O} \setminus \{O_1, O\} \cup \{O'_1, O'\}$. However, $|O'_1 \cap B_1| > |O_1 \cap B_1|$ which is in contradiction to the choice of $\mathcal{O}$. This finishes the proof. □

Appendix C – Detailed example for the calculation of lower bounds

We consider the example instance described in 6 to exemplify the calculation of the problem-specific lower bounds on the objective function as derived in Section 3.

The values of the lower bounds for the number of batches required and the cumulative batch processing times are summarized in Table 4 on page 183. We explain their calculation in what follows. The sets of large jobs are $J^l_1 = \emptyset$ and $J^l_2 = \{1, 2, 3, 6\}$, we thus need 4 batches for the large jobs of attribute 2 and none for attribute 1. The processing times for large batches are given by the minimal processing times of the large jobs and contribute $11 + 10 + 19 + 19 = 59$ to the cumulative batch processing time.

For the processing time of small jobs, we exemplify the calculation of the bound based on eligible machines for attribute 1 and the one of the bound based on compatible processing times for attribute 2. For attribute 1, we have three small jobs (4, 9, and 10) of which job 4 can only be processed on machine 1 and job 9 only on machine 2. Two different batches are thus required for these jobs. Since the cumulative remaining machine capacity ($2 \cdot \max\{c_m\} - (s_4 + s_9) = 40 - (2 + 4) = 34$) is sufficient to accommodate job 10 with $s_{10} = 14$, these two batches suffice. In this case, the runtime of the two batches is given by the minimal runtime of the two jobs 4 and 9, and is equal to 38 in total. As for attribute 2, the small jobs are 5, 7, and 8. Their respective intervals of possible processing times are $[10, 50]$, $[11, 50]$ and $[50, 50]$. To follow algorithm GAC+, we sort the list of jobs in decreasing order of their minimal processing times: $(8, 7, 5)$. A first batch with a processing time of 50 is created for job 8. The remaining capacity in this batch is $20 - 11 = 9$ (assuming that it is assigned to the batch with maximal capacity). We thus proceed in the list of jobs and add 9 of the 11 units of job 7 to this batch. For the remaining 2 units of job 7, a new batch with processing time 11 is created. We can add the entire job 5 to this batch. In total, two batches with a cumulative processing time of 61 are needed for the small jobs of attribute 2.

For the calculation of setup costs, equation (8) gives:

$$sc \geq b_1 \cdot \min_s\{sc(s, 1)\} + b_2 \cdot \min_s\{sc(s, 2)\}$$

$$= 2 \cdot \min(6, 10) + 6 \cdot \min(8, 10) = 60.$$

For equation (9), the list of minimal setup costs `setup_costs` contains $\min_s\{sc(1, s)\} = \min(6, 8)$ three times (twice for attribute 1 and once for the initial state of machine 1) and $\min_s\{sc(2, s)\} = \min(10, 10)$ seven times (six times for attribute 2 and once for the initial state of machine 2). We take the $b = 8$ smallest values from this list and thus have:

$$sc \geq \sum_{i=1}^{8} \texttt{setup\_costs}(i) = 3 \cdot 6 + 5 \cdot 10 = 68.$$

We take the maximum of these two values and obtain that $sc \geq 68$ for this instance.

Due to the given machine availability intervals for this instance, all jobs except jobs 5, 7, and 8 always finish late. Thus, the number of tardy jobs is ≥ 7 in any feasible solution.

The theoretical lower bound values are reported in Table 4.

Table 4: Lower bounds and optimal values for the number of batches, cumulative batch processing time, setup costs, and tardiness for the example instance with 10 jobs.

	number of batches			batch processing time			setup costs		tardiness
	(1)	b_r^E (3)	b_r^C (6)	large jobs	p_r^E	p_r^C (7)	(8)	(9)	
attribute 1	6	2	1	0	38	19	60	68	3 (jobs 4, 9, 10)
attribute 2		6	6	59	60	61			4 (jobs 1, 2, 3, 6)
lower bound	8			158			68		7
optimal values	8			158			72		8
gap (in %)	0			0			5.5		12.5

Using the weights and aggregating the lower bounds for three components of the objective function, we obtain that:

$$\text{obj} \geq \frac{4 \cdot 158/18 + 68/10 + 100 \cdot 7}{10 \cdot 105} \approx 0.7066.$$

Considering the optimal solution for this instance presented in 6, the gap between the calculated lower bounds and the optimal solution ($t = 8$, $p = 158$, and $sc = 72$) are thus 0% for the runtime, 5.5% for the setup costs, and 12.5% for the number of tardy jobs; the gap for the aggregated objective function is 11.7% (due to the high weight given to tardy jobs).

Appendix D – Details concerning the experimental setup

We consider the theoretical lower bounds as presented in 3. The code is implemented in `C#`. The experiments are executed on a machine featuring an Intel Core i7-1185G7 processor with 3.00GHz. Each run is executed on a single thread.

We consider the construction heuristic proposed by [10] (see 2.1). The solution method is implemented in `C++`. The code is compiled with `Clang++15`. All experiments are executed on a machine featuring 2x Intel Xeon Platinum 8368 2.4GHz 38C, 8x64GB RDIMM. Each run is executed on a single thread.

We consider the exact methods proposed by [10] (see 2.1). The “cpopt” is implemented with CPLEX Studio 22.11, whereas “mzn-gurobi” uses Minizinc 2.8.2 Gurobi 10.0.1. All experiments are executed on a machine featuring 2x Intel Xeon CPU E5-2650 v4 (12 cores @ 2.20GHz, no hyperthreading).

We consider the SA proposed by [11] (see 2.1). The SA is implemented using `EasyLocal++`, a `C++` framework for LS algorithms [2]. The code is compiled with `Clang++15`. All experiments are executed on a machine featuring 2x Intel Xeon Platinum 8368 2.4GHz 38C, 8x64GB RDIMM. Each algorithm is executed on a single thread. The algorithm is tuned using `irace` (v.3) [14]. We assign `irace` a total budget of $25,500$ experiments. Details on the parameter ranges and their final values are reported in 5.

Table 5: Parameter configurations for the SA algorithm.

Param.	Description	Range	Value
T_f	Final temperature.	0.001 – 0.01	0.004
α	Cooling rate.	0.985 – 0.995	0.988
ρ	Accepted move ratio.	0.05 – 0.7	0.309
p_{SCB}	Prob. of SCB move.	0 – 1	0.090
p_{IB}	Prob. of IB move.	0 – 1	0.293
p_{MJEB}	Prob. of MJEB move.	0 – 1	0.328
p_{MJNB}	Prob. of MJNB move.	0 – 1	0.289

Appendix E – Evaluation of the upper bounds provided by the construction heuristic

It is important to note that if the construction heuristic successfully schedules all jobs, as is the case for all our benchmark instances, the resulting solution is always feasible, making the obtained solution cost an upper bound on the optimal solution cost.

We compute the relative bound gap between the calculated lower bound and the cost of the solution generated by the greedy construction heuristic for each benchmark instance (considering the overall cost). The results are shown in Figure 6. The construction heuristic hardly ever finds optimal solutions (it does so for a single out of the 120 benchmark instances) and often the gap is very large between this upper bound and the calculated lower bound (the relative gap is nearly equal to 100% for a few instances). Surprisingly however, for 37 instances across all sizes, the relative bound gap is less than 1%, even for some of the large instances where no solver could provably find an optimal solution. Moreover, for a total of 57 instances, the gap is less than 10%. This suggests that within a short computation time (a maximum of 6 seconds, an average of 0.2 seconds), the construction heuristic together with the problem-specific lower bounds can provide good estimates of the optimal solution cost for a significant portion of the instance set, and rough estimates for nearly half of the instances.

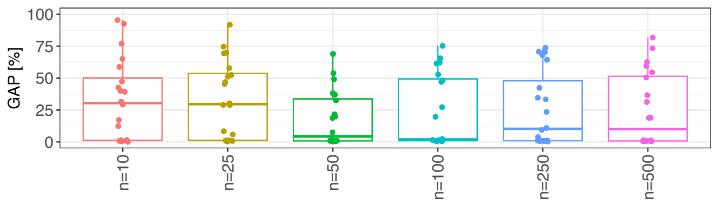


Fig. 6: Relative bound gap[%] between the upper bound found by the construction heuristic and the calculated lower bound per instance considering the overall cost.

References

1. Damodaran, P., Vélez-Gallego, M.C.: A simulated annealing algorithm to minimize makespan of parallel batch processing machines with unequal job ready times. Expert systems with Applications **39**(1), 1451–1458 (2012)
2. Di Gaspero, L., Schaerf, A.: EasyLocal++: An object-oriented framework for flexible design of local search algorithms. Software — Practice & Experience **33**(8), 733–765 (July 2003)
3. Finke, G., Jost, V., Queyranne, M., Sebő, A.: Batch processing with interval graph compatibilities between tasks. Discrete Applied Mathematics **156**(5), 556–568 (2008)
4. Fowler, J.W., Mönch, L.: A survey of scheduling with parallel batch (p-batch) processing. European Journal of Operational Research **298**(1), 1–24 (Apr 2022)
5. Hentenryck, P.V.: Constraint and integer programming in OPL. INFORMS Journal on Computing **14**(4), 345–372 (2002)
6. Kedad-Sidhoum, S., Solis, Y.R., Sourd, F.: Lower bounds for the earliness–tardiness scheduling problem on parallel machines with distinct due dates. European Journal of Operational Research **189**(3), 1305–1316 (2008)
7. Koh, S.G., Koo, P.H., Kim, D.C., Hur, W.S.: Scheduling a single batch processing machine with arbitrary job sizes and incompatible job families. International Journal of Production Economics **98**(1), 81–96 (2005)
8. Lackner, M.L., Mrkvicka, C., Musliu, N., Walkiewicz, D., Winter, F.: Minimizing Cumulative Batch Processing Time for an Industrial Oven Scheduling Problem. In: Michel, L.D. (ed.) 27th International Conference on Principles and Practice of Constraint Programming (CP 2021). Leibniz International Proceedings in Informatics (LIPIcs), vol. 210, pp. 37:1–37:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2021)
9. Lackner, M.L., Mrkvicka, C., Musliu, N., Walkiewicz, D., Winter, F.: Benchmark instances and models for the Oven Scheduling Problem [Data Set] (Dec 2022). https://doi.org/10.5281/zenodo.7456938
10. Lackner, M.L., Mrkvicka, C., Musliu, N., Walkiewicz, D., Winter, F.: Exact methods for the oven scheduling problem. Constraints **28**(2), 320–361 (2023)
11. Lackner, M.L., Musliu, N., Winter, F.: Solving an industrial oven scheduling problem with a simulated annealing approach. In: Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling. pp. 115–120 (2022)
12. Li, X., Chen, H., Du, B., Tan, Q.: Heuristics to schedule uniform parallel batch processing machines with dynamic job arrivals. International Journal of Computer Integrated Manufacturing **26**(5), 474–486 (2013)

13. Li, X., Li, Y., Huang, Y.: Heuristics and lower bound for minimizing maximum lateness on a batch processing machine with incompatible job families. Computers & Operations Research **106**, 91–101 (Jun 2019)
14. López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., Stützle, T.: The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives **3**, 43–58 (2016)
15. Mathirajan, M., Sivakumar, A.I.: A literature review, classification and simple meta-analysis on scheduling of batch processors in semiconductor. The International Journal of Advanced Manufacturing Technology **29**(9-10), 990–1001 (2006)
16. Tang, T.Y., Beck, J.C.: Cp and hybrid models for two-stage batching and scheduling. In: Hebrard, E., Musliu, N. (eds.) Integration of Constraint Programming, Artificial Intelligence, and Operations Research. pp. 431–446. Springer International Publishing, Cham (2020)
17. Wang, Q., Huang, N., Chen, Z., Chen, X., Cai, H., Wu, Y.: Environmental data and facts in the semiconductor manufacturing industry: An unexpected high water and energy consumption situation. Water Cycle **4**, 47–54 (2023)
18. Zhao, Z., Liu, S., Zhou, M., Guo, X., Qi, L.: Decomposition method for new single-machine scheduling problems from steel production systems. IEEE Transactions on Automation Science and Engineering **17**(3), 1376–1387 (2020)

Exact Methods for the Time Frame Rostering Problem in the Context of Tram Driver Scheduling

Lukas Frühwirth and Nysret Musliu

Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling, DBAI, TU Wien, Favoritenstraße 9, 1040 Vienna, Austria

Abstract. Creating more robust rosters that offer medium-term planning security for employees is a desired goal in the public transportation sector. To tackle this problem, we introduce a new approach in the context of tram driver scheduling called time frame rostering. In this approach, instead of directly assigning shifts to roster positions, time frames are first allocated to roster positions. These time frames are intervals wide enough to accommodate a variety of shifts. The shift assignment takes place only a few days before the actual workday. Thus, time frame rosters provide medium-term planning security, as tram drivers are only assigned shifts within their designated time frames. The goal of the time frame rostering problem is then to optimally assign time frames to a roster such that several constraints are met. In this paper, we formally define the time frame rostering problem and provide a solver-independent model of the problem. Furthermore, we compare two state-of-the-art solvers on real-world instances and demonstrate that optimal or almost optimal solutions can be found in a reasonable amount of time. Additionally, we verify these solutions by simulating absences and subsequent shift assignment.

Keywords: Tram Driver Scheduling, Crew Rostering, Public Transport Scheduling,

1 Introduction

Similar to other professions that operate in shifts, such as those in the medical field or industrial manufacturing, the shifts of tram drivers are assigned to a rotating schedule called a roster. Traditionally, this roster was created by assigning shifts to specific roster positions either by hand or by using workforce scheduling algorithms [8]. However, this approach proved inconvenient for tram drivers as well as rostering managers. A roster is typically scheduled weeks or months in advance, hence it undergoes several changes until the final shift assignment due to changes in shift plans, fluctuations in staff headcount, and various other factors. Consequently, the literature covers methods for constructing more robust rosters [6], [16], such as calculating the optimal amount of reserve shifts [18], [9], as well as re-rostering methods [17]. These methods, however, might still be inconvenient for tram drivers, as they potentially require a substantial number of reserve shifts or the repeated reallocation of shifts, which deteriorates the medium-term planning security for tram drivers.

As a consequence thereof, the idea emerged to introduce a time frame roster. In this approach, instead of directly assigning shifts to specific roster positions, first, time frames are assigned to roster positions. These time frames are intervals wide enough to accommodate a variety of shifts. The final shift assignment takes place only a few days before the actual workday. At this stage, shifts replace the drivers' time frames where the shifts' intervals fit within the time frames' intervals. Thus, the time frame roster provides medium-term planning security for tram drivers, as drivers will only be assigned shifts within their designated time frames.

However, to the best of our knowledge, the existing literature does not provide a formal definition or a formulated model of the time frame rostering problem that we consider in this paper. Furthermore, an efficient method for optimally assigning time frames to roster positions while considering specific criteria is currently unavailable. A main criterion is, for example, that even if drivers are absent, it must still be possible to assign shifts to drivers without violating their time frames.

The time frame rostering problem is a real-life combinatorial optimization problem with specific requirements tailored to the operation of tram networks. The design of the time frames requires a negotiation process and agreement between the employer and employees. Therefore, for the purpose of this paper, the time frames and their specifications (start time, end time, type) are regarded as predetermined.

The main contributions of this paper are:

- We provide a formal problem definition and formulate a solver-independent model of the time frame rostering problem.
- We publish real-world instances based on data provided by a public transportation company.
- We empirically evaluate different exact solving methods using these real-world instances and show that these are optimally solvable within 120 minutes.
- We demonstrate the feasibility of our solutions by simulating staff absences and subsequent shift assignment.

This paper is part of a master's thesis [5] that is currently under submission and is expected to be published by September 2024.

The article is structured as follows: In the next section, we provide an overview of the related work, followed by a high-level problem description in section 3. In section 4, we precisely define the time frame rostering problem, and formulate a solver-independent model. Subsequently, in section 5, we evaluate the model by solving real-world instances using the linear solver Gurobi [7] and the constraint solver OR-Tools [14]. Finally, we draw our conclusions in section 6.

2 Related Work

Tram driver scheduling can be considered a subset of crew scheduling, where each crew consists of just a single member, the tram driver. The time frame rostering problem is then a subset of the challenges encountered in crew planning or, generally, in workforce scheduling. To the best of our knowledge, the time frame rostering problem defined in this paper has not been addressed in the existing literature.

Crew planning for (public) transportation typically comprises two primary stages: the first stage involves crew scheduling, also known as shift design or shift/duty scheduling, where shifts are designed based on a predetermined timetable. The second stage, referred to as crew rostering, entails assigning these shifts to typically rotating or cyclical rosters [8].

These two stages are also integral components of many workforce scheduling approaches in general [11]. While the two stages are often treated as separate optimization problems [8], integrated approaches in crew planning with a single objective function exist [1], [12]. On a more general level, solving the general employee scheduling problem, as demonstrated by Kletzander and Musliu [10], is also done by integrating several stages.

Extensive literature explores how crew scheduling [8], crew rostering [8], and workforce scheduling problems in general [2,4,15], can be addressed using mathematical programming, constraint programming, answer-set programming, heuristics, metaheuristics, and combinations thereof. In shift scheduling, mathematical programming is the most popular approach [15]. Mathematical programming typically uses a set covering formulation, introduced by Dantzig in 1954 [3]. Our model of the time frame rostering problem is also based on such a formulation.

As mentioned in the introduction, a crew schedule, and particularly a crew roster, undergoes several re-rosterings due to factors such as timetable changes, construction sites, fluctuations in headcount, absences, and more. Thus, the literature also covers methods to deal with uncertainty by constructing more robust rosters [6], [16], such as introducing reserve duties [9], determining the optimal amount of reserve shifts [18], as well as re-rostering methods [17]. However, the methodology of introducing time frames and initially assigning these frames to rotating rosters based on the crew schedule (shifts), followed by assigning shifts within these time frames, has not been proposed to date. This time frame roster is more robust to changes than a typical shift roster, as the shift assignment occurs at a later stage where many factors leading to re-rostering are already known and accounted for in the assignment process.

3 Problem Description

This section provides a high-level problem description using a small sample roster. First, we outline a conventional rostering process that resembles a real-world implementation in a public transportation company. Second, we detail how our methodology diverges from this traditional approach, highlighting the differences and innovations we introduce.

Conventional Approach Typically, tram driver scheduling begins with a set of shifts containing all shifts for a week. Based on the size of this set, the demand for drivers and their days off is calculated. A roster containing only the days off is created. Each driver has either two specific consecutive days off or follows a rotational day off schedule, resulting in eight different day off types. Tram drivers are assigned to different roster weeks within their day off type and rotate through the roster weeks of their own type in ascending order. Upon reaching the last week of their day off type they continue with the first week of their type.

After the creation of the day off schedule, the shifts are allocated. In the exemplary shift roster shown in Table 1, each shift s represents a unique shift from the set that contains all shifts for a week. Each shift includes specific details about the work location (tram line), work hours (start time, breaks, end time), and whether it is a split shift (with a several-hours-long break) or not. Since the size of the day off schedule is determined by accounting for absences, among other things, there are considerably more roster positions than shifts. These empty roster positions are filled with reserve shifts, denoted as either r_e or r_l in Table 1. Reserve shifts provide drivers with rough time windows, typically distinguishing only between an early time window r_e (only shifts starting before noon are allowed) and a late time window r_l (only shifts starting after noon are allowed). Drivers with reserve shifts receive notice of their actual shift or if they are on stand-by only a few days before their scheduled workday.

day off type	Mo	Tu	We	Th	Fr	Sa	Su
o_0	off	s	s	s	s	s	off
o_0	...	s	s	s	s	s	...
o_1	off	off	s	s	s	s	s
o_1	...	...	r	r	r	r	r
o_2	s	off	off	s	s	s	s
o_2	s	...	...	s	s	s	s
o_3	r	r	off	off	s	s	s
o_3	s	s	..	...	r	r	r
o_4	s	s	s	off	off	s	s
o_4	s	s	s	...	...	s	s
o_5	s	s	s	s	off	off	s
o_5	s	s	s	s	...	...	s
o_6	s	s	s	s	s	off	off
o_6	r	r	r	r	r	...	...
o_7	off	off	s	s	s	s	s
o_7	s	off	off	r	r	r	r

Table 1: Exemplary Shift Roster

day off type	Mo	Tu	We	Th	Fr	Sa	Su
o_0	off	8	7	6	5	5	off
o_0	...	13	15	4	2	1	...
o_1	off	off	8	7	6	5	11
o_1	...	...	12	14	3	2	1
o_2	2	off	off	8	8	7	6
o_2	5	...	...	11	15	4	3
o_3	1	1	off	off	10	8	7
o_3	5	15	..	...	4	3	2
o_4	3	2	1	off	off	9	8
o_4	7	6	11	...	...	13	11
o_5	14	2	2	1	off	off	8
o_5	7	5	11	15	...	...	4
o_6	12	14	3	2	1	off	off
o_6	8	6	6	5	15	...	...
o_7	off	off	13	12	3	2	2
o_7	1	off	off	4	3	3	2

Table 2: Exemplary Time Frame Roster

New Approach – Time Frame Rostering The issue arising from the traditional approach is that drivers with reserve shifts do not know in advance when they will have to work. Moreover, absences of tram drivers and changes in the shift plans might require a reallocation of shifts. Our proposed method to prevent reserve shifts and reallocation is to introduce a time frame roster. In a time frame roster, rather than directly assigning shifts, we initially allocate time frames to roster positions (see Table 2).

In the time frame roster above, each time frame is indicated by a number between 1 and 15 since we consider 15 different time frames. Time frames are essentially intervals with predefined start and end times. The shift assignment is postponed until a few days before the actual workday, by which time many absences are already known to the roster managers. Shifts are assigned to time frames so that they fit within the intervals of the frames and are of the correct type. Some time frames allow for, or even require, split shifts, while others cannot accommodate split shifts.

The objective of the time frame rostering problem is to optimally assign time frames to roster positions so that shift assignment at a later stage remains possible. This requirement can be broken down into three major constraints:

Firstly, tram drivers may be absent with a certain probability, resulting in two scenarios. On the one hand, at the time of shift assignment, there might be more time frames (i.e., drivers) than shifts. In this case, it must be possible to assign all shifts to time frames; hence, it cannot be the case that shifts remain unassigned because they do not fit within the time frames. On the other hand, if there are fewer time frames than shifts, then all time frames must be assigned a shift, and it cannot be the case that there are time frames left in which none of the remaining shifts fits. The remaining shifts are then covered by drivers working overtime.

Secondly, to adhere to rest period regulations, two consecutive time frames cannot appear in the list of forbidden sequences.

Thirdly, there are restrictions on split shifts during a workweek. We provide a formal definition of the problem and its constraints in the next section.

4 Formal Problem Definition and Model

To formally define the time frame rostering problem, we first need to specify the given data. Secondly, we explain what constitutes a time frame roster, and the role it plays in tram driver scheduling. Thirdly, we will outline how to create a day off schedule. Finally, we will define the hard constraints, soft constraints, decision variables, and the objective function.

4.1 Provided Data

The provided data comprises four groups: shift-related, time frame-related, day off-related and constraint-related. This section specifies each of them.

1. Shift data:
   - There is a set of n shifts denoted by $S = \{s_1, ..., s_n\}$. Each shift has a start and end time denoted as an interval $[a_{s_i}, b_{s_i}],\ i \in \{1, ..., n\}$.
   - The weekday of a shift is represented by w_{s_i}, ranging from 0 to 6.
   - S_i represents the set of shifts for weekday i:
     $s \in S_i \Leftrightarrow w_s = i,\ i \in \{0, ..., 6\}$
   - Each shift has an assigned shift type t_{s_i}, with the value of 1 if the shift is a split shift and 0 otherwise.
2. Time frame data:
   - There is a set of m time frames indicated by $F = \{1, ..., m\}$, each with a start and end time forming an interval $[c_i, d_i],\ i \in \{1, ..., m\}$.
   - Each time frame has an assigned type t_i, with 0 allowing only shifts of type 0, 1 allowing only shifts of type 1, and 2 allowing shifts of any type.
3. Day off data:
   - There are 8 different day off types $o_0, ..., o_7$. Types 0 to 6 have two fixed consecutive days off, while type 7 follows a 16-week-long rotating day off schedule, which is provided by a public transport company.
4. Constraint data:
   - A list seq_{fb} of forbidden time frame sequences.
   - A list seq_{ud} of undesirable time frame sequences and their penalties.
   - A list $forb_{wt}$ of forbidden times frames for each workweek type wt. This workweek type is used to encode, on one hand, the proximity of a workday to the next day off, and on the other hand, whether a roster position is designated for early or late shifts.

4.2 Definition of a Time Frame Roster

A time frame roster R possesses the following properties:

- The size of the roster is determined by the sum of the sizes of each day off type. Thus, a roster R consists of $|R| = \sum_{i=0}^{7} |o_i|$ roster weeks.
- Each roster week $r_{o_i^j}$ has an assigned day off type o_i at week j and consists of 7 days.
- A roster position $r_{o_i^j,k}$ is then defined as a specific weekday k within the roster week $r_{o_i^j}$.
- Each roster position, excluding those designated as days off, will be assigned a time frame.
- Each roster position $r_{o_i^j,k}$ features a workweek type wt, ranging from -1 to 30.
- R_i represents a list of time frames assigned to roster R for weekday i.

Operating Principle of the Roster Each driver is associated with a specific day off type o_i and a unique roster week $r_{o_i^j}$. Tram drivers rotate through the roster weeks of their own day off type in ascending order. Upon reaching the last week of their day off type $o_i^{|o_i|}$, they continue with the first week of their day off type o_j^1. However, there might be more roster weeks than drivers, since the number of weeks assigned to each day off type has to be even. This is the case because drivers alternate between "early" and "late" weeks. During the early week, their shifts typically start before 10am, while in the late week, their shifts begin after 10am. Due to this alternating schedule, an even number of roster weeks is necessary for each day off type.

Time frame rosters are anonymous, i.e., the specific assignment of drivers to roster weeks is not given, putting the focus solely on constructing the roster itself. To comprehend the rotational principle of the roster, it is crucial to define what constitutes two consecutive positions in the roster.

Definition of Consecutive Roster Positions Given the roster positions

$$r_{o_i^k,m},\ r_{o_i^l,n} \in R$$

then $r_{o_i^k,m}$ is immediately followed by $r_{o_i^l,n}$ iff one of the following statement holds:

- Two consecutive days in the same roster week: $k = l,\ n = m + 1$.
- Sunday in one week (but not the last week of a day off type) followed by Monday in the next week: $l = k + 1,\ m = 6,\ n = 0$.
- Sunday in the last week of a day off type followed by Monday in the first week of the same day off type: $k = |o_i|,\ l = 1,\ m = 6,\ n = 0$.

4.3 Algorithm for Calculating the Minimum Number of Drivers Needed

We propose the *min_demand* algorithm to calculate the required number of drivers for a certain number of shifts given the drivers' absence probability. The algorithm starts with a lower bound (e.g., number of shifts) and increases the demand until the lower bound is covered with a probability of p_{suc}, i.e., until the binomial cumulative distribution function returns a probability greater than p_{suc}:

```
min_demand(p_abs, p_suc, lb):
  If(lb = 0)
    minDemand = 0
  Else
    minDemand = lb
    While(binom.cdf(minDemand−lb, minDemand, p_abs) <= p_suc
        )
      minDemand = minDemand + 1
  Return minDemand
```

This is the binomial cumulative distribution function used in the *min_demand* algorithm:

$$binom.cdf(k, n, p) = P(X \leq k) = \sum_{i=0}^{k} \binom{n}{i} p^i (1-p)^{n-i}$$

Given a number of n trials, the probability p of a trial being successful and a number of k successes, the binomial cumulative distribution function returns the probability that there are k or fewer successes.

The *min_demand* algorithm will be used in several constraints, so we want to clarify its meaning by providing an example. Assume that drivers are absent with a probability of 10% ($p_{abs} = 0.1$). Let's also assume that we have 10 shifts and aim to cover 9 of them ($cov_{fac} = 0.9$, $lb = 9$) with a probability of 99% ($p_{suc} = 0.99$). The question is then: How many drivers do we need to ensure that at least 9 drivers show up to work 99% of the time?

To determine this number, we use the function call: *min_demand*$(0.1, 0.99, 9)$. In the first iteration, the function *binom.cdf*(0,9,0.1) is called and returns the likelihood that from 9 drivers, 0 or fewer are absent given the absence probability of 10%. This value is 0.38742, which is smaller than the required 0.99, so the while loop continues. The loop stops with the call *binom.cdf*(4,13,0.1), where the binomial cumulative distribution function returns 0.99354. This means that in 99.354% of the days, no more than 4 out of 13 drivers are absent. Thus, we determine that the required number of drivers to cover the shifts is 13.

One might ask: Aren't there 10 shifts to be covered, not 9? That is correct, but we aim to cover only a certain percentage of shifts, as the remaining ones can be handled by drivers working overtime if necessary. If all shifts were covered with a probability of 99%, there would quite often be too many drivers on stand-by. The coverage factor provides the ability to regulate the extent to which you want to have more drivers working overtime (lower coverage, risk of being understaffed) or more drivers on stand-by (higher coverage, risk of being overstaffed).

4.4 Creating a Day Off Schedule

Before assigning time frames to roster positions, we first create a day off schedule to determine the size of the roster and the placement of the days off. We assume that drivers may be absent with a binomially distributed probability p_{abs}. Based on real-world observations, the demand for drivers dem_i for each weekday i is determined by calculating the minimum even number of drivers such that the number of drivers showing up is at least $(cov_{fac} \cdot 100)$% of the number of shifts $|S_i|$ with a probability of p_{suc}:

$$dem_i = \left\lceil \frac{min\_demand(p_{abs},\ p_{suc},\ \lceil |S_i| \cdot cov_{fac} \rceil)}{2} \right\rceil \cdot 2$$

The demand dem_i is then used to determine the day off schedule, for a detailed description, please refer to Appendix 6. Based on the day off schedule, we can proceed to construct and encode the empty roster R containing only the days off. Given this roster R, the goal now is to assign time frames to the empty positions in R such that the hard constraints are not violated and the objective function value is minimized. The subsequent three sections define the hard and soft constrains, as well as the objective function.

4.5 Hard Constraints

Given an assigned roster R and the probability p_{abs} that drivers are absent, it must be possible (with a probability of success of at least p_{suc}[2]) to assign every present driver a shift such that the shift fits within the interval provided by the assigned time frame and is of the correct type. There are two independent occasions to violate this abstract constraint: First, we only ensure with a probability of $P(A) = p_{suc}$ that there remain enough time frames f of type $t_f > 0$ to accommodate all split shifts and secondly, we guarantee with probability of $P(B) = p_{suc}$ that there remain enough time frames f of type $t_f \neq 1$ to accommodate the regular shifts, hence the overall probability of success is $P(A \cap B) = {p_{suc}}^2$. To check whether this abstract constraint holds, we would need to simulate the shift assignment to time frames which itself is a NP-hard problem [11]. Hence, we break down the abstract constraint into hard constraints $h_1 - h_3$ and soft constraint s_1. Additionally, we verify for each solution (i.e., for each time frame roster) by simulating absences and subsequent shift assignment whether constraints $h_1 - h_3$ and s_1 were successful in ensuring that the aforementioned constraint holds (for details see Section 5.2).

Minimum Coverage First, we define a coverage: Each weekday is split into time intervals $[\tau_j, \tau_{j+1})$ of length $\tau_{j+1} - \tau_j = 0.5$ (30 min), starting with $\tau_1 = 3$ (i.e., 3am on the current day) and ending with $\tau_{55} = 30$ (i.e., 6am on the next day). The shift coverage of a time interval $[\tau_j, \tau_{j+1})$ for weekday i is defined as the number of shifts $s \in S_i$ that start before τ_{j+1}, end after τ_j and are either split shifts or not:

$$\begin{aligned} min\_cov_{i,j} &= \sum_{s \in S_i,\ t_s = 0,\ a_s < \tau_{j+1},\ b_s > \tau_j} 1 \\ min\_cov\_split_{i,j} &= \sum_{s \in S_i,\ t_s = 1,\ a_s < \tau_{j+1},\ b_s > \tau_j} 1 \end{aligned}$$

The frame coverage is defined analogously:

$$\begin{aligned} fr\_cov_{i,j} &= \sum_{f \in R_i,\ t_f = 0,\ c_f < \tau_{j+1},\ d_f > \tau_j} 1 \\ fr\_cov\_split_{i,j} &= \sum_{f \in R_i,\ t_f > 0,\ c_f < \tau_{j+1},\ d_f > \tau_j} 1 \end{aligned}$$

The frame coverage must be greater or equal the shift coverage:

$$\begin{aligned} h_{1_a} &: \ fr\_cov_{i,j} \geq min\_cov_{i,j} && i \in \{0, ..., 6\},\ j \in \{1, ..., 54\} \\ h_{1_b} &: \ fr\_cov\_split_{i,j} \geq min\_cov\_split_{i,j} && i \in \{0, ..., 6\},\ j \in \{1, ..., 54\} \end{aligned}$$

Minimum Frame Set The hard constraint h_1 is not sufficient by itself to ensure that shifts fit into the available time frames. We need to guarantee that there are enough frames to accommodate the shifts, taken into account the probability of absences p_{abs}. A shift can lie in the interval of several time frames, hence we obtain a set of time frames

FS_s covering a shift s. We determine the minimum size required for each of these sets of time frames, similar to how we determine the necessary count of drivers and days off:

$$FS_s = \{f \mid f \in F,\ t_f \neq 1,\ s \in S,\ [a_s, b_s] \in [c_f, d_f]\}$$
$$FS_{split_s} = \{f \mid f \in F,\ t_f = 1,\ s \in S,\ [a_s, b_s] \in [c_f, d_f]\}$$

We also compute frame sets FS_j for artificial shifts of length 4 (assumed minimum shift length) to get every possible frame set. Subsequently, we count the shifts having the same frame set FS_j for weekday i:

$$count_{FS_{i,j}} = |\{s \mid s \in S_i,\ FS_j = FS_s\}|$$
$$count_{FS_{split_{i,j}}} = |\{s \mid s \in S_i,\ FS_{split_j} = FS_{split_s}\}|$$

The size of a frame set FS_j for weekday i is defined as the sum of the count of frame f appearing in roster column R_i over all frames $f \in FS_j$:

$$|FS_{i,j}| = \sum_{f \in FS_j} \sum_{f \in R_i} 1 \qquad |FS_{split_{i,j}}| = \sum_{f \in FS_{split_j}} \sum_{f \in R_i} 1$$

The minimum size of a frame set FS_j for weekday i is then the number of frames $|FS_{i,j}|$ necessary to cover at least $(cov_{fac} \cdot 100)\%$ of shifts possessing the frame set FS_j or subsets thereof with a probability of p_{suc}:

$$min_{FS_{i,j}} = min\_demand(p_{abs},\ p_{suc},\ \sum_{FS_k \subseteq FS_j} \lceil cov_{fac} \cdot count_{FS_{i,k}} \rceil)$$

The size of a frame set must be greater or equal the minimum size for this set:

$$h_{2_a} :\ |FS_{i,j}| \geq min_{FS_{i,j}} \qquad i \in \{0, ..., 6\},\ j \in \{1, ..., 30\}$$

Split shifts are treated differently since they contain a long break, increasing the overall shift duration. This makes it more important for drivers to known if their associated time frames allow or even require a split shift. Hence, there are special minimum frame set constraints for split shifts, for details please refer to Appendix 6.

Maximum Frame Set In addition to the hard constraints h_1 and h_2, we define maximum frame sets. Instead of counting shifts with the same frame set, we count how many shifts can be covered using any frame f in a frame set FS_j. This gives us the maximum number of shifts coverable by a frame from frame set FS_j:

$$max\_count_{FS_{i,j}} = |\{s \mid s \in S_i,\ \exists f \in FS_j :\ [a_s, b_s] \in [c_f, d_f]\}|$$

We once again employ the *min_demand* algorithm but this time in reverse. The maximum size of a frame set FS_j for weekday i is determined by the maximum number of frames $|FS_{i,j}|$ such that there is only a probability of $1 - p_{suc}$ for there to be more frames than shifts to cover:

$$max_{FS_{i,j}} = min\_demand(p_{abs},\ 1 - p_{suc},\ max\_count_{FS_{i,j}} + 1) - 1$$

The size of a frame set must be smaller or equal the maximum size for this set. However, this is restricted to frame sets of size less than 4, as larger sets render rosters infeasible. For frame sets of size greater than 3, we adjust the maximum to match the number of shifts $|S_i|$ for weekday i. In case the maximum is lower than the minimum, we decrease the minimum such that it equals the maximum.

$$h_{3_a} : |FS_{i,j}| \leq max_{FS_{i,j}} \qquad i \in \{0, ..., 6\},\ |FS_j| < 4$$
$$h_{3_b} : |FS_{i,j}| \leq |S_i| \qquad i \in \{0, ..., 6\},\ |FS_j| \geq 4$$

Forbidden Sequences Some sequences of time frames are forbidden, primarily due to rest time regulations. A consecutive time frame assignment (f_1, f_2) in roster R cannot appear in the list of forbidden sequences seq_{fb}:

$$h_4 : \forall (f_1, f_2) \in R : (f_1, f_2) \notin seq_{fb} \qquad f_1, f_2 \in F$$

Forbidden Frames Drivers follow an alternating scheme where in one week they are only assigned early shifts, and in the subsequent week, only late shifts. Split shifts can only be assigned during the first two days of the "early" week or the last day of the "late" week. A workweek type is utilized to encode this information for each roster positions. A time frame assignment f to a roster position of workweek type wt cannot appear in the list of forbidden time frames for this workweek type:

$$h_5 : \forall f \in R : f \notin forb_{wt} \qquad f \in F,\ f = r_{o_i^j,k},\ wt = wt_{r_{o_i^j,k}}$$

Our model includes additional hard constraints, which ensure that weekdays with similar shifts also have similar time frames, and that the solution contains a certain number of night and relief frames. These constraints are defined in Appendix 6.

Additionally, our problem includes several soft constraints. These constraints are defined in Appendix 6.

4.6 Decision Variable

The time frames represent the decision variables. For each roster position $r_{o_i^j,k}$ in roster R that is not assigned a day off, we assign a time frame $r_{o_i^j,k} := f \in F$.

4.7 Objective Function

The objective of this problem is to assign time frames to roster positions such that the costs associated with soft constraints s_1 to s_6 are minimized while ensuring that hard constraints h_1 to h_8 hold. To balance the costs, each soft constraint is multiplied by an adjustable weight before aggregation, resulting in the following function:

$$\begin{aligned} \min \quad & w_1 s_1 + w_2 s_2 + w_3 s_3 + w_4 s_4 + w_5 s_5 + w_6 s_6 \qquad w_1, w_2, w_3, w_4, w_5, w_6 \in \mathbb{N}^+ \\ \text{s.t.} \quad & h_1 - h_8 \text{ hold} \end{aligned}$$

4.8 Solver-independent Model

Based on the problem definition provided in this section, we create a solver-independent model of the time frame rostering problem by using MiniZinc [13]. A MiniZinc model allows us to use both linear and constraint solvers. Our model[1] includes all hard and soft constraints as well as the objective function outlined in the problem definition. To speed up the solving process, we also make use of redundant constraints and symmetry breaking.

Redundant Constraints We include redundant constraints for the forbidden sequence constraint. This is done by exploiting the time frames numbering. For some workweek types and time frames we can state, for example, that the following time frame must be smaller than or equal to the current one without checking the list of forbidden sequences.

Symmetry Breaking We can pre-assign time frames $f \in FS_j$ if there is only a single frame in the set ($|FS_j| = 1$). In such cases, there is no choice of time frames, and we know, due to the minimum frame set constraint, that a certain number of these time frames have to be in the roster. We pre-assign these time frames so that they are uniformly distributed across the roster and assigned to positions where they do not influence the overall solution quality.

5 Evaluation

We evaluate our proposed model using a diverse set of twelve real-world instances. These twelve distinct sets of shifts are extracted from real-world shift plans, that have been provided by a public transportation company. We also make this data available to the scientific community[2]. Instance properties, time frame properties and parameter settings can be found in Appendix 6. The evaluation process involves several steps: Firstly, we solve the instances using two different state-of-the-art solvers and compare the results. Secondly, we validate the solutions using a simulation model. Finally, we discuss the obtained results in detail.

5.1 Results

To evaluate our solver-independent model, we solve the instances using two conceptually different state-of-the-art solvers: linear solver Gurobi 11.0.0 [7] and constraint solver OR-Tools 9.8.3296 [14].

The test setup looks as follows: For each instance, we set the time limit to 120 minutes. The solvers are allowed to use up to 20 threads. The experiments are run on an Intel i5-13500 2.5GHz processor with 20 logical units and with 32GB of RAM available.

[1]https://github.com/lukasfruehwirth/time_frame_rostering/blob/main/frame_rostering_problem.mzn
[2]https://github.com/lukasfruehwirth/time_frame_rostering

We define the gap g between the objective value ov of a solution and the best known lower bound lb_{best} as:

$$g = (\frac{|lb_{best} - ov|}{ov}) \cdot 100$$

Table 3 shows the objective values, optimality gaps and runtimes for both solvers:

			Gurobi		OR-Tools		Gurobi	OR-Tools
Inst.	|S|	|R|	Obj. Value ov	Gap g	Obj. Value ov	Gap g	Runtime (mm:ss)	Runtime (mm:ss)
1	1,380	364	1,868,046	optimal	5,127,240	87.31%	16:15	TL
2	1,159	308	4,808,321	optimal	31,525,200	91.11%	10:04	TL
3	1,282	338	3,487,611	optimal	7,832,640	67.93%	39:49	TL
4	1,456	382	20,100,971	optimal	190,637,000	91.10%	21:35	TL
5	2,539	652	4,212,492	0.040%	113,196,000	99.39%	TL	TL
6	2,738	702	6,784,287	0.594%	303,801,000	99.83%	TL	TL
7	1,344	354	2,041,618	0.015%	4,138,020	74.92%	TL	TL
8	1,134	302	905,588	optimal	1,438,480	68.78%	17:29	TL
9	1,224	324	780,783	optimal	3,436,120	90.28%	29:46	TL
10	1,375	362	1,070,283	optimal	2,389,440	85.27%	42:38	TL
11	2,478	636	2,970,358	0.004%	10,427,500	94.47%	TL	TL
12	2,599	666	2,062,171	0.040%	160,549,000	99.80%	TL	TL

Table 3: Results – Comparison of Gurobi and OR-Tools

To improve the performance of both solvers, we tried to circumvent modeling hard constraints as soft constraints with high penalties (see soft constraint s_4). Instead of relying on $below(h_{2_d})$ and $below(h_{2_e})$, we pre-determine the maximum number of positions in roster R for weekday i that can accommodate frames of type $t_f \in \{1, 2\}$, according to the specified workweek types wt for each roster position. If the minimum demand for h_{2_d} and h_{2_e}, as formalized in Section 4.5, surpasses this maximum, we adjust the minimum to match the identified maximum. This approach allows us to define h_{2_d} and h_{2_e} as hard constraints in our model, while ensuring the model's satisfiability. The soft constraint s_4 is thus reduced to $s_4 = below(h_{2_f}) \cdot pen_{hard\_con}$. Nonetheless, this modification to the MiniZinc model did not really affect Gurobi's performance and only slightly improved OR-Tools' performance, which remained significantly below that of Gurobi. Consequently, we did not include the results in this paper. However, results for the modified model can be found in the master's thesis [5].

5.2 Simulation

To check whether our abstract constraint formulated in section 4.5 does indeed hold, we simulate absences. This is done by discarding some frames according to p_{abs}:

$$R_{i_{sim}} = [f \mid f \in R_i,\ rand \leq 1 - p_{abs}]$$

Where $rand$ is for each frame in R_i a newly sampled, random real number between 0 and 1.

Using this reduced list of frames, we simulate the shift assignment for weekday i by deploying a simulation model[3] modeled in MiniZinc. For a detailed model description, please refer to Appendix 6.

Since the simulation (shift assignment) itself is NP-hard [11] and rather time-consuming, we do not simulate the entire roster week at once but separately for each weekday. For each solution (i.e. time frame roster) and weekday i, we simulate 100 shift assignments. The number of failed shift assignments for weekday i is denoted as sim_{fail_i}. The rate of success is then defined as:

$$sim_{suc} = (1 - \frac{\sum_{i=0}^{6} sim_{fail_i}}{700}) \cdot 100$$

The objective is to achieve a success rate $sim_{suc} > p_{suc}{}^2$ (see Section 4.5). Table 4 presents the simulation results for all instances and solvers:

			Gurobi	OR-Tools
Inst.	|S|	|R|	Sim. Result sim_{suc}	Sim. Result sim_{suc}
1	1,380	364	99.14%	97.48%
2	1,159	308	98.43%	86.86%
3	1,282	338	98.57%	98.00%
4	1,456	382	96.14%	26.86%
5	2,539	652	99.29%	62.57%
6	2,738	702	99.71%	23.86%
7	1,344	354	99.00%	98.57%
8	1,134	302	98.71%	97.86%
9	1,224	324	99.14%	96.86%
10	1,375	362	99.00%	97.43%
11	2,478	636	99.71%	98.71%
12	2,599	666	98.71%	27.43%

Table 4: Simulation Results

5.3 Discussion

Table 3 clearly demonstrates that our model performs significantly better with the solver Gurobi compared to the OR-Tools solver. Gurobi consistently achieves superior results, quickly finding optimal solutions for 7 out of 12 instances and coming very close to optimality for the remaining ones. Notably, even for the larger instances 5, 6, 11, and 12, Gurobi produces solutions with a gap to the best known lower bound smaller than 0.6%. Conversely, the constraint solver OR-Tools fails to solve any instances within the time limit to optimality. The smallest gap reached by OR-Tools is 67.93%. Thus, all solutions

[3]https://github.com/lukasfruehwirth/time_frame_rostering/blob/main/shift_assignment_simulation.mzn

provided by OR-Tools are considerably far from the optimal solution. OR-Tools performs comparably well in finding a satisfiable solution and can compute approximately ten times more solutions than Gurobi within the designated time limit. However, these additional solutions provide only minor improvements over the first solution found. It appears that OR-Tools struggles with our model's objective function, which consists of several independent soft constraints.

Table 4 shows that for 11 out of 12 solutions generated by Gurobi, we achieve a success rate exceeding ${p_{suc}}^2 \cdot 100 = 98.01\%$. This suggests that hard constraints h_1 through h_3 and soft constraint s_1 effectively model the abstract constraint mentioned at the beginning of section 4.5. The only instance where the success rate falls below 98.01% is instance 4. If we take a closer look at the solution of this instance, we can see that the softened hard constraints h_{2_d} and h_{2_f} have to be violated in order to not violate hard constraint h_5 (forbidden frames), resulting in an objective value significantly surpassing 1,000,000, as Table 3 reveals. To improve the success rate for these instances, practitioners may need to consider relaxing some of the hard constraints h_4 to h_8 and, in particular, h_5. Since we have softened some hard constraints, it becomes crucial to obtain solutions close to the optimum; otherwise, the success rate of the simulation falls below acceptable levels. This observation is supported by examining the simulation results computed for solutions provided by the OR-Tools solver. Table 4 shows that for 10 out of 12 solutions generated by OR-Tools, the success rate is below ${p_{suc}}^2 \cdot 100 = 98.01\%$.

6 Conclusion

In this study, we introduce the time frame rostering problem, a novel challenge arising from the desire for enhanced medium-term planning security and, generally, more robust rosters in tram driver scheduling. We translate the abstract problem description into a formal problem definition and provide a solver-independent model using MiniZinc. This translation involves processing the given data to derive lower and upper bounds for some of the hard constraints of the model. Additionally, we verify whether the abstract constraint mentioned in Section 4.5 is indeed fulfilled by simulating tram drivers' absences and subsequent shift assignments.

The results reveal that the linear solver Gurobi is able to solve 7 out of 12 real-world instances to optimality in less than an hour. For the remaining instances, Gurobi generates solutions very close to the optimum within two hours. Furthermore, the simulation shows that solutions based on our model are indeed feasible and deployable in practice, as for almost all instances and simulation runs, the assignment of shifts to time frames is successfully completed. Some instances had a slightly lower success rate in the simulation because we allowed certain hard constraints to be violated to satisfy others. This is a trade-off that practitioners should consider when using our model. In summary, it can be stated that we have successfully developed a model for the time frame rostering problem, that can be used to solve real-world instances to (nearly) optimal levels within a reasonable amount of time.

Future work could test if other constraint solvers outperform OR-Tools and explore approaches to improve the performance of constraint solvers. Moreover, further investigation into optimizing time frame rosters may involve penalizing the interval width of the time frames to further enhanced medium-term planning security of tram drivers.

Acknowledgements 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.

Appendix A - Day Off Schedule

Every driver works 5 days per week, but the sum $dem = \sum_{i=0}^{6} dem_i$ is not necessarily a multiple of 5. Hence, we need to slightly adjust the demand:

$$frac = \frac{\sum_{i=0}^{6} d_i}{5} - \left\lfloor \frac{\sum_{i=0}^{6} d_i}{5} \right\rfloor$$

If $frac = 0.2$ then dem_5 += 2, dem_6 += 2,
else if $frac = 0.4$ then dem_6 −= 2,
else if $frac = 0.8$ then dem_6 += 2,
else if $frac = 0.8$ then dem_5 −= 2, dem_6 −= 2

We determine the number of roster positions of roster R by summing up the demands and dividing by 5:

$$|R| = \frac{\sum_{i=0}^{6} dem_i}{5}$$

The day off demand do_i for weekday i is then the size of the roster minus the driver demand dem_i:

$$do_i = |R| - dem_i, \quad i \in \{0, ..., 6\}$$

The days off have to be two consecutive days. Furthermore, the size of a day off type must be even and we limit the size of type o_i to $|o_i| \leq 2|o_{i+1}|$ and $2|o_i| \geq |o_{i+1}|$ for $i \in \{0, ..., 5\}$. In order to find the optimal day off schedule, we formulate the following model in MiniZinc[4] [13]:

Variables:

$$|o_i| \text{ ... size of day off type } i \in \{0, ..., 6\}$$

$$x_i = \begin{cases} |o_i| + |o_{i+1}|, & \text{if } i < 6 \\ |o_0| + |o_6|, & \text{if } i = 6 \end{cases}$$

Objective function:

$$\begin{aligned} \min \ & |do_i - x_i| & i \in \{0, ..., 6\} \\ \text{s.t.} \ & \sum_{i=0}^{6} |o_i| = |R| & \\ & |o_i| \mod 2 = 0 & i \in \{0, ..., 6\} \\ & |o_i| \leq 2|o_{i+1}| & i \in \{0, ..., 5\} \\ & 2|o_i| \geq |o_{i+1}| & i \in \{0, ..., 5\} \end{aligned}$$

[4]https://github.com/lukasfruehwirth/time_frame_rostering/blob/main/day_off_type_size.mzn

After finding the optimal size for each day off type o_0 to o_6, day off type o_7 is introduced, which has a 16-week-long rotating day off schedule. The structure of this schedule is provided by a public transportation company. Based on real-world observations, the size of day off type o_7 is calculated by using a fixed share of 30%:

$$|o_7| = \left\lfloor \frac{|R| \cdot 0.3}{16} \right\rfloor \cdot 16$$

Day off types 0-6 are reduced accordingly:

$$|o_i| = \begin{cases} |o_i| - \frac{|o_7|}{8}, & \text{if } i < 6 \\ |o_i| - \frac{|o_7|}{4}, & \text{if } i = 6 \end{cases}$$

Appendix B- Minimum Frame Sets for Split Shifts

The number of time frames in the roster allowing only split shifts to be assigned equals 80% of the total split shifts (see $h2_b, h2_c$). The utilization of 80% aims to accommodate small changes in the shift plan without making the rosters unsatisfiable. The minimum size of a split frame set FS_{split_j} is defined similarly to a regular frame set, except that it must cover 100% of split shifts (see $h2_d, h2_e$). There are frames that can accommodate regular as well as split shifts, hence we also calculate the minimum demand for all early week time frames including the once designated for split shifts (see $h2_f$). Furthermore, time frames of type greater 0 are only allowed if the set of shifts contains split shifts (see $h2_g$).

$$h2_b : |FS_{split_{i,j}}| = |\{s \mid s \in S_i,\ t_s = 1\}| \cdot 0.8$$
$$i \in \{0, ..., 6\},\ FS_{split_j} = \{f \mid f \in F,\ t_f = 1\}$$
$$h2_c : |FS_{split_{i,j}}| \geq count_{FS_{split_{i,j}}} \cdot 0.8$$
$$i \in \{0, ..., 6\},\ |FS_{split_j}| = 1$$
$$h2_d : |FS_{split_{i,j}}| \geq min\_demand(p_{abs},\ p_{suc},\ |\{s \mid s \in S_i,\ t_s = 1\}|)$$
$$i \in \{0, ..., 6\},\ FS_{split_j} = \{f \mid f \in F,\ t_f > 0\}$$
$$h2_e : |FS_{split_{i,j}}| \geq min\_demand(p_{abs},\ p_{suc},\ count_{FS_{split_{i,j}}})$$
$$i \in \{0, ..., 6\},\ FS_{split_j} \in \{\{12, 13\}, \{14, 15\}\}$$
$$h2_f : |FS_{i,j}| + |FS_{split_{i,h}}| \geq$$
$$min\_demand(p_{abs},\ p_{suc},\ |\{s \mid s \in S_i,\ t_s = 1\}| + \sum_{FS_k \subseteq FS_j} \lceil cov_{fac} \cdot count_{FS_{i,k}} \rceil)$$
$$i \in \{0, ..., 6\},\ FS_j = \{1, 2, 3, 4, 11, 14, 15\}$$
$$h2_g : |\{s \mid s \in S_i,\ t_s = 1\}| = 0 \implies |\{f \mid f \in R_i,\ t_f > 0\}| = 0$$
$$i \in \{0, ..., 6\}$$

Appendix C - Additional Hard Constraints

Maximum Frame Deviation Weekdays with similar shifts should also have similar time frames. To ensure this, we define a maximum deviation of a time frame count between two weekdays. The following algorithm uses the minimum coverage definition (see h_1):

```
max_deviation(min_cov, dev_allowed):
  For(i in {0,...,5})
    x = $\sum_{j=0}^{29} min\_cov_{i,j}$
    y = $\sum_{j=0}^{29} min\_cov_{i+1,j}$
  If(x < y)
    dev = y / x
  Else
    dev = x / y
  maxDev[i] = dev_allowed + ceil(500(dev − 1))
Return maxDev
```

For weekdays 0 to 5 the absolute difference in the count of frame f between two consecutive days $(i, i+1)$ must be smaller or equal $maxDev[i]$ (see h_{6_a}). For weekdays 0 to 4 the absolute difference in the count of frame f between any of these days must be smaller or equal $maxDev[i]$ (see $h_{6_{b-d}}$).

$$h_{6_a}: \mid \sum_{f\in R_i} 1 - \sum_{f\in R_{i+1}} 1 \mid \leq maxDev[i] \qquad i\in\{0,...,5\},\ f\in F$$

$$h_{6_b}: \mid \sum_{f\in R_i} 1 - \sum_{f\in R_{i+2}} 1 \mid \leq maxDev[i] \qquad i\in\{0,1,2\},\ f\in F$$

$$h_{6_c}: \mid \sum_{f\in R_i} 1 - \sum_{f\in R_{i+3}} 1 \mid \leq maxDev[i] \qquad i\in\{0,1\},\ f\in F$$

$$h_{6_d}: \mid \sum_{f\in R_i} 1 - \sum_{f\in R_{i+4}} 1 \mid \leq maxDev[i] \qquad i=0,\ f\in F$$

Night Frames Time frames with an end time after 2am are called night frames and are only allowed if there are also shifts ending after 2am. Furthermore, for frame sets consisting of only night frames, the maximum frame set size is set to the minimum frame set size:

$$h_{7_a}: |\{s \mid s\in S_i,\ b_s > 26\}| = 0 \implies |\{f \mid f\in R_i,\ d_f > 26\}| = 0$$

$$h_{7_b}: max_{FS_{i,j}} := min_{FS_{i,j}} \quad i\in\{0,...,6\},\ FS_j = \{f \mid f\in F,\ d_f > 26\}$$

Relief Frames Two of the time frames $FS_r = \{11, 15\}$ are considered to be relief frames, since they can accommodate early and late shifts. These time frames are necessary to balance potential imbalances between early time frames $FS_e = \{1,2,3,4,11,14,15\}$ and late time frames $FS_l = \{5,6,7,8,9,11,15\}$ and thus, appear in both sets $FS_r = FS_e \cap FS_l$. We determine the minimum size of frame set FS_r by calculating the maximum difference between early and late frames given that drivers are absent with probability p_{abs}:

$$lb_i = \arg\min_x(\sqrt{1 - p_{suc}} \leq binom.cdf(x,\ min(|FS_{i,e}|, |FS_{i,l}|),\ 1 - p_{abs}) - 1$$

$$ub_i = \arg\min_x(1 - \sqrt{1 - p_{suc}} \leq binom.cdf(x,\ max(|FS_{i,e}|, |FS_{i,l}|),\ 1 - p_{abs})$$

$$h_8 :\ min_{FS_{i,r}} = ub_i - lb_i \quad i \in \{0, ..., 6\}$$

Appendix D - Soft Constraints

Deviation from Desired Coverage In addition to the hard constraints $h_1 - h_3$ we introduce a soft constraint that penalizes the deviation from the desired coverage. The aim is to satisfy the abstract constraint mentioned at the beginning of section 4.5. We calculate the desired coverage by again using the *min_demand* algorithm:

$$cov\_des_{i,j} = min\_demand(p_{abs},\ p_{suc},\ min\_cov_{i,j})$$
$$cov\_split\_des_{i,j} = min\_demand(p_{abs},\ p_{suc},\ min\_cov\_split_{i,j})$$

To calculate the deviation penalty, we square the difference between frame coverage and desired coverage if the frame coverage is greater, and take the difference to the power of 4 otherwise. The difference between split coverage and desired split coverage is simply squared:

$$cov\_pen_{i,j} = \begin{cases} (cov\_des_{i,j} - fr\_cov_{i,j})^4 & \text{if } cov\_des_{i,j} \leq fr\_cov_{i,j} \\ (cov\_des_{i,j} - fr\_cov_{i,j})^2 & \text{if } cov\_des_{i,j} > fr\_cov_{i,j} \end{cases}$$
$$cov\_split\_pen_{i,j} = (cov\_split\_des_{i,j} - fr\_cov\_split_{i,j})^2$$

$$s_1 = \sum_{i=0}^{6} \sum_{j=1}^{55} cov\_pen_{i,j} + cov\_split\_pen_{i,j}$$

Undesirable Sequences Some sequences of time frames are undesirable, primarily due to rest time regulations. A consecutive time frame assignment (f_1, f_2) in roster R appearing in the list of undesirable sequences seq_{ud} incurs a penalty $p(f_1, f_2)$ (penalty function p is given):

$$s_2 = \sum_{(f_1,f_2)\in R,\ (f_1,f_2)\in seq_{ud}} p(f_1, f_2)$$

Below Minimum Frame The hard constraints $h_{2_d} - h_{2_f}$ sometimes result in unsatisfiability. To prevent this outcome, we transform them into soft constraints and introduce a high penalty if the frame set counts fall below the minima. The function $below(h)$ returns the extent to which a hard constraint h is undershot:

$$s_4 = (below(h_{2_d}) + below(h_{2_e}) + below(h_{2_f})) \cdot pen_{hard\_con}$$

Frame Deviation Weekdays with similar shifts should also have similar time frames. This is in particular the case for weekdays 0 to 4, since they usually have very similar sets of shifts. We penalize the frame deviation between these weekdays:

$$s_3 = \sum_{i=1}^{4} \sum_{f=1}^{15} \mid \sum_{f\in R_i} 1 - \sum_{f\in R_{i+1}} 1 \mid$$

Same Frame Next Day To provide drivers with some variety in the sequence of time frames, we introduce a penalty for consecutive assignments of the same time frame (f, f) in roster R:

$$s_5 = \sum_{(f,f) \in R,\ f \in F} 1$$

Same Frames Next Week To achieve a more even distribution of time frames within a day off type, we introduce a penalty for assigning the same time frames to two consecutive "early" or "late" weeks. The penalty s_6 is similar to s_5, we simply count all occurrences.

Appendix E - Instance and Parameter Settings

Table 5 illustrates the main properties of the twelve instances, where the column $|Sp_i|$ stands for the number of split shifts in weekday i:

Inst.	$\|S_0\|$ - $\|S_3\|$	$\|S_4\|$	$\|S_5\|$	$\|S_6\|$	$\|Sp_0\|$ - $\|Sp_4\|$	$\|Sp_5\|$	$\|Sp_6\|$
1	223	223	142	123	29	4	0
2	185	185	123	111	27	6	0
3	204	204	139	123	23	9	0
4	233	233	153	138	37	7	0
5	408	408	265	234	56	10	0
6	437	437	292	261	60	16	0
7	217	219	132	125	22	0	0
8	183	183	110	109	21	3	0
9	194	194	134	120	15	0	0
10	218	218	148	137	23	0	0
11	400	402	242	234	43	3	0
12	412	412	282	257	38	0	0

Table 5: Instance Properties

For all instances, we use the following parameter settings and time frames:

Parameter	Setting
cov_{fac}	0.90
p_{abs}	0.25
p_{suc}	0.99
$dev\_allowed$	5
pen_{hard_con}	1,000,000
w_1	1
w_2	10
w_3	10,000
w_4	1
w_5	100
w_6	100

Table 6: Parameter Settings

Time Frame	Interval Length (hh:mm)	Type
1	10:30	0
2	11:00	0
3	12:30	0
4	12:30	0
5	11:00	0
6	12:00	0
7	12:30	0
8	11:00	0
9	7:30	0
10	15:00	0
11	16:00	0
12	16:30	1
13	17:00	1
14	12:30 / 16:30	2
15	16:00 / 17:00	2

Table 7: Time Frame Properties

Time frames 14 and 15 can accommodate regular and split shifts and are, therefore, a combination of other time frames. This is the reason why there are two different interval lengths given for these time frames in Table 7. If time frame 14 is assigned a regular shift, then its properties are equal to those of time frame 4. If time frame 14 is assigned a split shift, then its properties are equal to those of time frame 12. Similarly, if time frame 15 is assigned a regular shift, then its properties are equal to those of time frame 11. Conversely, if time frame 15 is assigned a split shift, then its properties are equal to those of time frame 13.

Appendix F - Simulation Model

Variables:

$$S_i \text{ ... list of shifts for weekday } i$$
$$R_{i_{sim}} \text{ ... list of time frames for weekday } i$$
$$X_i \text{ ... list of length } |R_{i_{sim}}| \text{ with domain 0 to } |S_i|$$

Constraints:

$$c_1 : \text{ All elements of } X_i \text{ except 0 must be different}$$
$$c_2 : \forall s[j] \in S_i : t_{s[j]} = 1 \implies j \in X_i$$
$$c_3 : |R_{i_{sim}}| \leq |S_i| \implies 0 \notin X_i$$
$$c_4 : |R_{i_{sim}}| > |S_i| \implies \sum_{x \in X_i,\ x=0} 1 = |R_{i_{sim}}| - |S_i|$$
$$c_5 : \forall j \in [0, ..., |R_{i_{sim}}|] : ((X[j] > 0) \implies t_{s[X[j]]} = 0) \implies t_{R_{i_{sim}}[j]} \neq 1 \land [a_{s[X[j]]}, b_{s[X[j]]}] \in [c_{R_{i_{sim}}[j]}, d_{R_{i_{sim}}[j]}]$$
$$c_6 : \forall j \in [0, ..., |R_{i_{sim}}|] : ((X[j] > 0) \implies t_{s[X[j]]} = 1) \implies t_{R_{i_{sim}}[j]} > 0 \land [a_{s[X[j]]}, b_{s[X[j]]}] \in [c_{R_{i_{sim}}[j]}, d_{R_{i_{sim}}[j]}]$$

We cannot assign the same shift to different time frames (c_1). Split shifts must be assigned (c_2). If the number of remaining time frames is less or equal the number of shifts, all of these time frames have to be assigned a shift (c_3). If the number of remaining time frames is greater than the number of shifts, then all shifts have to be assigned to time frames (c_4). The number of time frames without a shift assigned is then $|R_{i_{sim}}| - |S_i|$. If a shift of type 0 is assigned to a time frame, then this time frame cannot be of type 1 and the shift's interval must lie within the time frame's interval (c_5). If a shift of type 1 is assigned to a time frame, then this time frame cannot be of type 0 and the shift's interval must lie within the time frame's interval (c_6).

References

1. Borndörfer, R., Schulz, C., Seidl, S., Weider, S.: Integration of duty scheduling and rostering to increase driver satisfaction. Public Transport **9**, 177–191 (2017)
2. Burke, E.K., Causmaecker, P.D., Berghe, G.V., Landeghem, H.V.: The state of the art of nurse rostering. J. Sched. **7**(6), 441–499 (2004)
3. Dantzig, G.B.: A comment on edie's "traffic delays at toll booths". Journal of the Operations Research Society of America **2**(3), 339–341 (1954)
4. Ernst, A.T., Jiang, H., Krishnamoorthy, M., Sier, D.: Staff scheduling and rostering: A review of applications, methods and models. European journal of operational research **153**(1), 3–27 (2004)
5. Frühwirth, L.: Exact Methods for the Time Frame Rostering Problem in the Context of Tram Driver Rostering. Master's thesis, Technische Universität Wien (2024)

6. Ge, L., Voß, S., Xie, L.: Robustness and disturbances in public transport. Public Transport **14**(1), 191–261 (2022)
7. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2023), https://www.gurobi.com
8. Heil, J., Hoffmann, K., Buscher, U.: Railway crew scheduling: Models, methods and applications. European journal of operational research **283**(2), 405–425 (2020)
9. Ingels, J., Maenhout, B.: The impact of reserve duties on the robustness of a personnel shift roster: An empirical investigation. Computers & Operations Research **61**, 153–169 (2015). https://doi.org/https://doi.org/10.1016/j.cor.2015.03.010, https://www.sciencedirect.com/science/article/pii/S0305054815000684
10. Kletzander, L., Musliu, N.: Solving the general employee scheduling problem. Computers & Operations Research **113**, 104794 (2020). https://doi.org/https://doi.org/10.1016/j.cor.2019.104794, https://www.sciencedirect.com/science/article/pii/S0305054819302369
11. Lau, H.C.: On the complexity of manpower shift scheduling. Computers & Operations Research **23**(1), 93–102 (1996)
12. Lin, D.Y., Tsai, M.R.: Integrated crew scheduling and roster problem for trainmasters of passenger railway transportation. IEEE Access **7**, 27362–27375 (2019). https://doi.org/10.1109/ACCESS.2019.2900028
13. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: Minizinc: Towards a standard cp modelling language. In: International Conference on Principles and Practice of Constraint Programming. pp. 529–543. Springer (2007)
14. Perron, L., Didier, F.: Cp-sat v9.8 (2023), https://developers.google.com/optimization/cp/cp_solver
15. Van den Bergh, J., Beliën, J., De Bruecker, P., Demeulemeester, E., De Boeck, L.: Personnel scheduling: A literature review. European Journal of Operational Research **226**(3), 367–385 (2013). https://doi.org/https://doi.org/10.1016/j.ejor.2012.11.029, https://www.sciencedirect.com/science/article/pii/S0377221712008776
16. Wickert, T.I., Smet, P., Vanden Berghe, G.: Quantifying and enforcing robustness in staff rostering. Journal of Scheduling **24**(3), 347–366 (2021)
17. Wickert, T.I.: Personnel rostering: models and algorithms for scheduling, rescheduling and ensuring robustness. Doctoral thesis, Universidade Federl Do Rio Grande Do Sul and KU Leuven (2019)
18. Xie, L., Suhl, L.: A stochastic model for rota scheduling in public bus transport. In: Proceedings of 2nd Stochastic Modelling Techniques and Data Analysis International Conference. pp. 785–792 (2012)

KHE24: Towards a Practical Solver for Nurse Rostering

Jeffrey H. Kingston

The University of Sydney, Australia
jeff@it.usyd.edu.au
http://jeffreykingston.id.au

Abstract. Nurse rostering—assigning nurses to the shifts of a hospital ward—is one of the most studied of all timetabling problems. This paper describes ongoing work on the KHE24 nurse rostering solver. KHE24 is an improved version of the KHE18 solver published previously. It uses a time sweep algorithm to find an initial solution, which it then repairs using ejection chains and optimal reassignment by dynamic programming. Results are presented for several well-known data sets. They show that KHE24 is making progress towards success in practice, taking running time and breadth of application into account as well as solution cost.

Keywords: Nurse rostering, Time sweep, Ejection chains, XESTT

1 Introduction

Nurse rostering—assigning nurses to the shifts of a hospital ward—is one of the most studied of all timetabling problems. It is an NP-complete problem, and exact algorithms are out of reach in general, although many smaller instances have been solved to optimality using integer programming [4,36].

Many inexact methods have been tried. Recent work covers a wide range; it includes integer programming [15,29,32,36,40], weighted maxSAT [10,11], simulated annealing [9,38], hyper-heuristics [2,16,33], variable neighbourhood search [39], and constraint programming [34]. For older work, see [41].

The solver presented here, KHE24, is the 2024 version of the main solver built by the author on his KHE solver platform [20]. It is an improved version of the KHE18 solver described in [22]. It runs in polynomial time and aims to find competitive but not optimal solutions quickly, across a wide range of instances. It finds an initial solution using a time sweep algorithm (Section 3.1). This timetables the first day of the cycle, then the second, and so on. It then tries two repair methods: ejection chains (Section 3.3) and optimal reassignment using dynamic programming (Section 3.4).

The author's intention is to test KHE24 on four well-known data sets, to demonstrate its ability to solve a wide range of practical instances. At present, however, it has been tested on only two data sets and half of a third (Section 4). Although KHE24 has not found any new best solutions, on its own terms (that is, considering running time and breadth of application as well as cost) it promises to be successful. The work is ongoing.

2 Nurse rostering and its XESTT formulation

Nurse rostering is the problem of assigning shifts to the nurses of a hospital ward. Hospitals operate 24 hours a day, so there are usually at least three types of shifts: morning, afternoon, and night. Each shift demands a certain number of nurses, often with specific skills. There may be some flexibility in how many nurses to assign, and the number typically changes from day to day.

Perhaps the most characteristic feature of the problem is the large array of requirements that each nurse's timetable must satisfy. In addition to workload limits, there are rules such as 'a nurse must have a day off after a sequence of night shifts', 'a nurse may work at most four days in a row', and so on. These requirements are different at different institutions.

Instead of the usual formulas, this paper's formal definition of the nurse rostering problem is supplied by the XESTT nurse rostering data format [23]. XESTT is an XML format which is capable of representing the instances found in all the well-known data sets. It is based on the XHSTT high school timetabling format [30,31]; the name 'XESTT' was chosen to be reminiscent of 'XHSTT', with 'employee scheduling' replacing 'high school'. Full details of XESTT appear online [17] and will not be repeated here. Instead, this section offers an overview, and explains the importance of XESTT to the present work.

An XESTT instance consists of the *cycle* (the sequence of *times* that events may be assigned); a set of *resources* (entities that attend events); a set of *events* (meetings); and a set of *constraints*, specifying conditions that solutions should satisfy, and penalties to impose when they don't.

Each event contains a *starting time*, which may either be preassigned a time or left open for a solver to assign; a *duration*, which is a fixed positive integer giving the number of consecutive times, starting at the starting time, that the event is running; an optional *workload*, which is a fixed non-negative integer representing the workload of the event in arbitrary units, for example in minutes; and any number of *event resources*, each specifying one resource which attends the event for the full duration, which may either be preassigned a resource or left open for a solver to assign.

In nurse rostering instances, each resource represents one nurse, and each event represents one shift. Each event has duration 1; its actual duration in minutes can be expressed as a workload, if needed. Its starting time is preassigned to a time unique to the shift. For example, if on each day there is a morning, afternoon, and night shift, then each day will contain three times, one for each shift. Within an event, each event resource represents a demand for one nurse.

A referee has commented that the time aspect of this mapping from nurse rostering to XESTT seems forced. And indeed it is fair to ask why times are needed, if they are in one-to-one correspondence with shifts. It would in fact be possible to omit times, but here are several reasons why that would not necessarily be an improvement. Timetabling in general is about assigning times and resources to events, and in nurse rostering it is a fact that the events have preassigned times. If times were omitted, the properties that times naturally have (such as belonging to a day, and chronological order) would have to be passed on to the shifts. Also, the use of times allows constraints from other sub-disciplines of timetabling (unavailable times, for example) to be applied without change to nurse rostering.

Arbitrary sets of times, resources, and events may be defined, called *time groups*, *resource groups* and *event groups*. Each resource has one *resource type*, saying what type of resource it is. In nurse rostering there is just one type, `Nurse`.

XESTT offers 18 constraint types, but 9 are not used in nurse rostering, mainly because all the events have preassigned times. Of the 9 types that are used, 3 are *cover constraints*, specifying the number of resources that should attend each event, and the skills they need. The remaining 6, called *resource constraints* here, constrain the timetables of individual resources, specifying unavailable times, workload limits, unwanted patterns (for example, a day shift immediately following a night shift), limits on the number of consecutive busy days, and so on. Considerable flexibility is available, owing to the use of arbitrary time groups in constraints. For example, by defining one time group holding the times of the first weekend, another holding the times of the second weekend, and so on, one can construct a constraint which places limits on the number of busy weekends, despite weekends not being built-in to XESTT.

Each constraint contains a Boolean *required* flag indicating whether it is *hard* or *soft*, and an integer *weight*. When a constraint is violated, the degree of violation is multiplied by the weight to give a cost. Algorithms aim to minimize firstly the total cost of hard constraints (the *hard cost*), and secondly the total cost of soft constraints (the *soft cost*). In nurse rostering, solutions with non-zero hard cost are usually considered to be *infeasible*, that is, useless.

XESTT is important here for two reasons. First, it makes it easy to test KHE24 on a wide range of instances, because all these instances have been converted from their original formats to XESTT.

Second, XESTT uses just 9 types of constraints to represent the constraints found in other models. It can do this because its constraints are very flexible, as instanced above by the example of limiting busy weekends. Algorithms like the ejection chain algorithm in this paper, which handle each constraint type explicitly, have only 9 types to handle. Without XESTT or something like it, the number of constraint types would be much larger, and such an approach would hardly be feasible.

3 The KHE24 solver

The KHE24 solver presented here is built on the author's KHE solver platform and is available from the KHE web site [20]. It is an improved version of the KHE18 nurse rostering solver [22], which itself was descended from the KHE14 high school timetabling solver [19]. Complete details appear online, in the KHE documentation [20]; this section covers the main points.

In the KHE platform, instances and solutions are distinct objects. Instances are immutable after creation; solutions change as solving proceeds. A solution consists of a set of *meets*, each representing one event. Within each meet is a variable holding the meet's starting time, plus a set of *tasks*, one for each event resource of the meet's event. Each task is a variable to which one resource may be assigned; it represents an indivisible piece of work for one resource.

We will use ‘task’ in the following in preference to ‘shift’, because ‘shift’ is ambiguous: it can mean one shift for one nurse (‘Smith and Jones swapped shifts’), or one shift type (‘he prefers the night shift’), or one shift on one day (‘the shift passed quietly’).

KHE24 is a general timetabling solver: it assigns both times and resources. Nurse rostering events have preassigned times, so KHE24’s time assignment part merely converts the time preassignments in the events of the instance into time assignments in the meets of the solution. This takes almost no time.

After time assignment comes resource assignment, which assigns resources to tasks. KHE24 first carries out a *construction phase* which builds an initial assignment, then continues with a *repair phase* which tries several kinds of repairs that improve that assignment. The repair phase is run twice, to enable repairs of one kind to open the way to repairs of other kinds.

KHE24 has many parameters (called *options*), settable from the command line, determining such things as how to divide up the available running time, the maximum number of days to include when swapping the timetables of two resources, and so on. However, in this paper, the default values of all these options are used. There is no tuning of parameters for different instances, or even different data sets. The only options passed to the calls to KHE24 reported on here are those that set its time limit, the number of solutions to make for each instance, and the number of solves to run in parallel; and those options (except the time limit) are the same for all runs. This is an important aspect of our claim that KHE24 could become a *practical* solver.

Although we usually speak of finding one solution using one run of KHE24, the intention is that in practice several independent runs will occur in parallel, with the final reported solution being the best of those found by the different runs. Our results (Section 4) present the best of 24 runs, for example. We stress that multiple solves are not done just for testing; they are an integral part of the use of KHE24 in practice, included to take advantage of the multiple cores usually found in contemporary desktop computers.

3.1 Construction using time sweep

Because of the high density of constraints in nurse rostering, it may be easier to avoid introducing a problem than to remove it later. So it makes sense for the construction phase to try hard to produce a very good solution [27].

Many nurse rostering constraints concern what happens over consecutive days within nurses’ timetables. This suggests that the initial solution should be constructed day by day—the tasks of the first day assigned first, then the tasks of the second day, and so on. As each day is assigned, these kinds of constraints can often be satisfied. KHE24 uses this *time sweep* method.

The only other use of time sweep known to the author is [27], which finds initial solutions one week at a time in chronological order. It cites an aircrew scheduling paper [35] as its inspiration. The time sweep idea seems to be well known, however. A referee mentioned a similar algorithm from vehicle routing, which sweeps through drop-off points in order of their angle from the base.

To carry out the assignments for one day, KHE24 uses weighted bipartite matching. Each task s running on that day is a demand node, each resource r is a supply node, and an edge joins s to r when assigning r to s is possible. The edge is weighted by the change (often negative) in solution cost that the assignment would produce. A matching of minimum total cost is found and used to determine the assignments of resources to tasks on that day. Further details may be found in the earlier paper [22].

3.2 Task grouping

The KHE platform allows tasks to be *grouped*. Assigning a resource to one element of a group assigns it to all. Grouping was included to support high school timetabling: the lessons of one course should be assigned a common teacher. It has also proved useful in nurse rostering.

For example, in instance `COI-GPost` (Section 4.1), a nurse who takes a night task on a Friday should also take night tasks on the next two nights. This is because constraints penalize Friday night tasks before free weekends, incomplete weekends (working on Saturday or Sunday but not both), and day tasks after night tasks. Then the Monday and Tuesday night tasks can be grouped, as can the Wednesday and Thursday night tasks, owing to limits (minimum 2 and maximum 3) on the number of consecutive night tasks.

KHE24 has a grouping phase which precedes time sweep. It runs quickly but is general enough to be widely useful. For example, it groups the tasks of solutions of instance `COI-GPost` following the line of argument just given. The full details are rather intricate. They appear in the earlier paper [22].

A grouping will occasionally be inadvisable for some unexpected reason. So KHE24 applies the groupings during the construction and first repair phases, but then removes them, so that if something else is needed there is a chance to find it during the second repair phase. The author has observed a few cases where omitting to remove groupings led to infeasible solutions.

3.3 Repair using ejection chains

After constructing an initial solution using time sweep, KHE24 repairs it using two methods. The most effective of these is *ejection chain repair*. Although KHE18 also used ejection chain repair, the details were not settled and the published description was quite sketchy. KHE24's version is much more settled, so this section presents it in detail.

A *defect* in a solution is one violation of a constraint. For example, if nurse `N2` should work at most two weekends but in fact works three, that is a defect.

A *repair* is a change to a solution which removes a defect. For example, unassigning one of nurse `N2`'s busy weekends repairs the defect just described.

An ejection chain is like a path in a graph where each node represents one defect and each edge represents one repair. Starting from some defect, the first repair removes that defect but introduces one new defect. The second repair removes that defect but introduces another new defect, and so on. If some repair removes a defect without introducing a new defect, then the chain ends successfully: the solution has been improved. Or if the repair of one defect introduces two or more new defects, then the chain ends unsuccessfully: the repair has to be undone, with no improvement. (More precisely, a chain ends successfully whenever the current solution cost is less than it was at the start of the chain; new defects are acceptable, if their cost is low. A chain may be extended whenever a repair introduces a new defect whose removal would make the current solution cost less than it did at the start of the chain.)

There are usually several ways to repair a defect. For example, nurse N2's defect can be repaired by unassigning any one of the three busy weekends. So finding a successful chain involves a search tree: if the first repair does not begin a successful chain, then the second is tried, and so on recursively.

The main loop of the ejection chain repair algorithm visits each defect of the current solution and attempts to remove it by searching a tree of ejection chains, stopping at the first successful chain, if any. It cycles around the defects until a complete cycle of attempts has failed, at which point it terminates.

There are several ways to limit the method to polynomial time. The usual one, which KHE24 uses, is to refuse to visit the same part of the solution twice while repairing a given defect [18,19]. There is also a time limit (Section 3.5).

Each type of constraint gives rise to one type of defect (or two: maximum and minimum limit violations are repaired differently). For each defect, a set of repairs suited to its type is tried, making the chains *polymorphic*.

At the lowest level, just one basic operation can change a solution: the *task move*, which changes the assignment of task s from resource r_1 to resource r_2, where $r_1 \neq r_2$. Here r_1 may be NULL, in which case the task move is also a *task assignment*; or r_2 may be NULL, making it also a *task unassignment*. One repair is (can only be) a set of these basic operations.

Several authors (see below) use a swap of the timetables of two resources over several consecutive days as their main repair operation. KHE24 also does this; the maximum number of days is 16. It also tries assignments to a resource r_2 over several days, and unassignments of a resource r_1 over several days. In these cases, the maximum number of days is much smaller, just 4. All these operations are sets of task moves.

Unlike metaheuristics, which choose repairs at random, KHE24's ejection chain algorithm is quite focused. It finds all *small repairs* (minimal repairs that correct the current defect, typically affecting just one or two days), and then for each small repair it tries a set of alternative *large repairs*, defined by *expanding* each small repair: making the same change over a larger set of adjacent days including the original ones. Simple checks ensure that the same large repair is almost never tried twice when repairing a given defect.

It is not hard to find suitable small repairs, as was done above for nurse N2's busy weekends. If a resource r is overloaded during some set of times (one day, weekend, or whatever), all small repairs are tried that either unassign r during those times or swap its timetable on those times' days with another resource which is free then (or in some cases less busy). If r is underloaded during some set of times, all small repairs are tried that either assign r to some unassigned task during those times or swap its timetable on those times' days with some other resource which is busier then. Expanded versions of these two kinds of repairs are used to repair all kinds of resource constraint defects. For example, if r has too many consecutive night shifts, that is treated as r being overloaded during the times of the two endpoints of the over-long sequence.

Cover constraint defects are handled similarly. For example, a task needing assignment is handled by finding all small repairs that assign a suitable resource to the task, then expanding that assignment over adjacent days.

It is common for several tasks to be equivalent, in the sense that the effect of assigning any given resource to any one of them is the same. For example, if the Thursday night shift demands one senior nurse and three ordinary nurses, there will be one task requiring a senior nurse which is not equivalent to any other task, and three equivalent tasks demanding ordinary nurses. If the demand is for two or three ordinary nurses, then the three tasks are still equivalent but there is a preference to assign resources to the first two rather than to the third. In such cases the ejection chain solver will try one repair per equivalence class, not one repair per task, saving time.

The author knows of three other papers that use ejection chains in nurse rostering. The first two are fairly old and use data sets that are not in use today, making their results hard to evaluate. One [12] includes a repair which swaps the timetables of two resources over one week. The other [28] uses chains of task moves in a tabu search framework. The chains are rather different from those used here: each is generated at random in a way that preserves coverage, but without checking other costs until the whole chain is generated. Reference [28] cites [1] as a source for these chains—a very early use. The third previous use of ejection chains [3,4,7] is much more recent. Its basic repair swaps the timetables of two resources over a variable number of consecutive days.

3.4 Repair using optimal reassignment

There is a dynamic programming algorithm, well known to those who use column generation to solve nurse rostering problems, for finding an optimal assignment of tasks to a single nurse, given fixed timetables for the other nurses. The author has recently generalized this algorithm to find an optimal reassignment of an arbitrary subset of the nurses on an arbitrary subset of the days of the cycle, leaving the other nurses and the other days fixed [25].

The algorithm described in [25] was deficient in one respect: it did not support the XESTT limit workload constraint, which limits total workload measured in arbitrary units, for example minutes. This deficiency is now fixed.

Analysis shows that the running time of this algorithm is polynomial in the number of days it reassigns, but exponential in the number of resources it reassigns [25]. The author's practical experience bears this out: it costs very little to reassign a few more days, but reassigning just one more resource can increase the running time dramatically. Disappointingly, despite the author's best attempts at optimization, he has been able to reliably reassign up to only three resources, although for a practically unlimited number of days.

Using this algorithm, the author has implemented a VLSN search which chooses 3 resources and 28 consecutive days at random, optimally reassigns those resources over those days, and then repeats using new random choices until time runs out or 50 consecutive attempts produce no improvement. It is nowhere near as effective as ejection chains, but it operates on quite different principles and occasionally makes a contribution.

3.5 Making good use of available running time

Running time is a major issue for the larger instances, so it must be used well. Each of KHE24's phases yields diminishing returns as its running time increases, so it would be a mistake to spend all of the available time on one phase and have nothing left for the others.

KHE24 limits each day during time sweep (including repairs associated with that day) to 3 seconds. Two-thirds of the remaining time is then given to the first repair phase, and one-third to the second. Within these phases, the two repair methods (ejection chains and optimal reassignment) get equal time. If ejection chains finish early, optimal reassignment gets the surplus.

The ejection chain algorithm of [7] imposes a time limit on each search for a chain that repairs one defect. KHE24 imposes a limit of 120 on the number of recursive calls when repairing one defect, which has a similar effect.

Actual running time need not match the time limit exactly. On the one hand, a phase will often end of its own accord well before its time limit. On the other, the time limits are *soft*: instead of being interrupted, each phase consults wall clock time periodically and decides for itself whether to stop. In practice, KHE24 never significantly overruns its time limit, as the actual running times reported in the results tables of Section 4 will show.

3.6 Randomization

Randomization is not stressed in algorithmic solvers like it is in, for example, metaheuristic ones. Still, including some randomization allows the algorithm to be run multiple times with different seeds to obtain different results. Doing this and keeping the best solution is a simple way to utilize multiple cores and trade off solution quality and running time.

For example, ejection chain repair randomizes by trying alternative repairs starting at a random point in the sequence of possible repairs. Similar methods are used in the other phases. These methods are not deep, but they seem to work, judging from the spread of solution costs they produce.

4 Results

This section presents the results of running KHE24 on several well-known sets of instances, after conversion to XESTT by the NRConv program [21,23], with comparisons with previous results. The converted instances and solutions are available, in the form of XESTT archive files, at [21].

Our aim here is not to find new best solutions, but rather to achieve success in practice, which means to find good solutions to a wide variety of real-world instances quickly. This idea has been formalized by the author [26] as follows:

> *A solver is* successful in practice *if, on every instance that is likely to be encountered in practice, it finds a solution whose cost is within 10% of the best known when run for 5 minutes, and within 5% of the best known when run for 60 minutes.*

A justification appeared in [26], but since the idea is somewhat novel, yet crucial to this paper, we'll discuss it now in some detail.

The basic idea, then, is that a solver satisfying these conditions can be used by practitioners with confidence that it will work well on their instances. Running for 5 minutes will produce a solution suitable for testing a possible scenario; running for 60 minutes will produce a solution for actual use which is not easy to distinguish from the best known solution. (If all constraints have the same weight, being within 5% means that for every 20 defects in the best known solution, there are 20 or 21 defects in the solver's solution.)

This definition of success in practice is not realistic in one case: when the best known solution has no defects, or just a few. In that case, in reality a solution would be considered successful in practice if it contained just a few more defects, whatever the percentage is. For example, the best known solution to instance `COI-GPost` (Section 4.1) has cost 5. KHE24's solution has cost 8, which in reality would be judged to be successful in practice, even though it is numerically 60% worse. Rather than complicating the rule to take account of such cases, we'll simply point them out as they arise.

Our task, then, is to evaluate the KHE24 solver against this standard of success in practice. As a proxy for 'every instance that is likely to be encountered in practice', we intend to test the solver on four widely used data sets, although in this paper we only test on two data sets and part of a third. And for 'best known solutions' we will use sets of solutions from other authors which are either the best, or among the best known.

KHE24 (as described in Section 3) is run 24 times on each instance, with a different random seed for each run, and the best of the 24 solutions is kept. We call this version of the solver KHE24x24. We stress that running multiple solves in parallel and keeping the best solution is an integral part of our solving strategy, not something we are doing merely to gain insight into the variability of the solver (although that is an interesting question worthy of investigation).

Twelve threads are used, running on the author's 12-core Intel i7-12700 processor. (The Intel web site gives several clock frequencies, from 1.6GHz to 4.9GHz, chosen depending on various factors including temperature, so it's hard to be definite about processor speed.) Each individual solve is given a time limit which is half the overall time limit (half of either 5 minutes or 60 minutes). For each instance, the reported running time is the wall clock time in seconds from the start of the first solve to the end of the last one. It does not include file read and write times; they are negligible.

Each result table was generated by the author's HSEval program from an XESTT archive file and included with no hand editing. A blank entry indicates that there is no solution for the instance of its row in the solution group of its column. If there are multiple solutions, one with minimum running time among all solutions with minimum cost is shown. The minimum solution costs in each row appear in bold. Any solutions with non-zero hard cost are shown as 'inf.' (infeasible). No costs are reported on trust; all are calculated from the solutions in the archive file, and hence verified, by HSEval.

Adjacent to each cost c, in the 'Rel.' column, is a relative cost: the value of c divided by the best cost in the row, when the best cost is non-zero. When KHE24x24 is run for 5 minutes, a relative cost of up to 1.10 represents success in practice; when it is run for 60 minutes, 1.05 represents success in practice.

Running times are shown (in seconds) where present in the archive. HSEval cannot verify them. KHE24x24 running times are as defined above.

Care is needed with the average costs at the foot of each table, since costs are sensitive to the scale of the constraint weights. Constraints that measure workload in minutes may produce costs in the thousands.

4.1 The Curtois original instances

The Curtois original instances are the instances available online at [8] under the heading 'Original instances.' Their XESTT versions appear in XESTT archive file `COI.xml` at [21], along with the solutions posted with the instances. Nearly all of these solutions are optimal, according to Table 3 of [4].

Table 1 compares KHE24x24's solutions with the solutions from [8]. It is important to analyse what is happening, and not simply take these results at face value. Several cases of a phenomenon described earlier, arising when the best known solution has very low cost, occur here. For example, KHE24 is successful in practice on the Post and Ikegami instances, even though the relative costs are numerically high. Also, in every solution of `COI-WHPP` with cost below 1000, some nurses must take night shifts only, and the rest must take non-night shifts only—hardly a real-world scenario. (The author has addressed this issue with `COI-WHPP` in work carried out after this table was produced.)

In summary, KHE24 is successful in practice, or nearly so, on most of the Curtois original instances, the main exceptions being the larger ones, such as `COI-ERMGH`, `COI-CHILD`, `COI-ERRVH`, and `COI-MER`. The large differences in cost between the 5-minute and 60-minute runs on these instances suggest that their large size is overwhelming the solver, although further analysis is needed. The Curtois original instances are rarely used for testing these days, which is a pity, considering how interesting and varied they are.

4.2 The First International Nurse Rostering Competition

The First International Nurse Rostering Competition [14] has published many instances, still available from the competition web site [13]. XESTT versions are available in files `INRC1-Long-And-Medium.xml` and `INRC1-Sprint.xml` [21], along with the GOAL research group's virtually optimal solutions [37].

Tables 2 and 3 show KHE24's results. Running times are not given for the GOAL solutions because the GOAL solution files do not contain any; but the GOAL web site has a table of running times. About 10 instances, from the Long and Medium sets, have running times of about 4 hours. Many others have running times under one minute, often well under.

Another source of virtually optimal solutions to these instances is the branch and price algorithm whose results are reported in Table 5 of [4]. The author has not tried to obtain these solutions. Their reported running times are better than the GOAL ones, never exceeding about 10 minutes.

KHE24 is generally successful in practice on these instances, although once again there are some exceptions, including `INRC1-LH04`, `INRC1-LH05`, `INRC1-MH01`, and

Table 1: Results of running KHE24x24 on the Curtois original instances. Column Misc shows the best solutions from XESTT archive file `COI.xml`, taken from Curtois' web site [8]. The other columns show the results for KHE24x24, running for 5 minutes and 60 minutes. Here and elsewhere, running times are in seconds. This table is derived from XESTT archive file `KHE24-2024-03-07-COI.xml`, available at [21].

Instances (27)	**Misc**			**KHE24x24-05**			**KHE24x24-60**		
	Cost	**Rel.**	**Time**	**Cost**	**Rel.**	**Time**	**Cost**	**Rel.**	**Time**
COI-Ozkarahan	**0**	1.00	-	**0**	1.00	0.1	**0**	1.00	0.1
COI-Musa	**175**	1.00	-	**175**	1.00	15.9	**175**	1.00	16.9
COI-Millar-2.1	**0**	1.00	1.0	**0**	1.00	6.8	**0**	1.00	7.3
COI-Millar-2.1.1	**0**	1.00	-	**0**	1.00	0.0	**0**	1.00	0.0
COI-LLR	**301**	1.00	10.0	**301**	1.00	46.5	**301**	1.00	50.8
COI-Azaiez	**0**	1.00	600.0	**0**	1.00	300.1	**0**	1.00	1800.0
COI-GPost	**5**	1.00	-	8	1.60	143.5	8	1.60	139.4
COI-GPost-B	**3**	1.00	-	5	1.67	157.5	5	1.67	1800.0
COI-QMC-1	**13**	1.00	-	16	1.23	70.6	16	1.23	62.5
COI-QMC-2	**29**	1.00	-	30	1.03	44.1	30	1.03	43.5
COI-WHPP	**5**	1.00	-	2001	400.20	300.1	2001	400.20	3600.1
COI-BCV-3.46.2	**894**	1.00	17840.0	896	1.00	300.1	**894**	1.00	3600.1
COI-BCV-4.13.1	**10**	1.00	-	**10**	1.00	300.1	**10**	1.00	3600.1
COI-SINTEF	**0**	1.00	-	**0**	1.00	6.9	**0**	1.00	7.4
COI-ORTEC01	**270**	1.00	105.0	300	1.11	174.3	295	1.09	211.4
COI-ORTEC02	**270**	1.00	-	295	1.09	154.3	295	1.09	158.8
COI-ERMGH	**779**	1.00	124.0	1481	1.90	301.7	882	1.13	3600.4
COI-CHILD	**149**	1.00	-	2824	18.95	324.8	261	1.75	3600.3
COI-ERRVH	**2001**	1.00	-	6739	3.37	307.7	2219	1.11	3829.4
COI-HED01	**136**	1.00	-	138	1.01	254.3	**136**	1.00	1824.7
COI-Valouxis-1	**20**	1.00	-	100	5.00	300.0	100	5.00	866.1
COI-Ikegami-2.1	**0**	1.00	13.0	**0**	1.00	150.0	**0**	1.00	1800.1
COI-Ikegami-3.1	**2**	1.00	21600.0	10	5.00	300.1	9	4.50	3600.3
COI-Ikegami-3.1.1	**3**	1.00	2820.0	14	4.67	300.2	15	5.00	3600.5
COI-Ikegami-3.1.2	**3**	1.00	2820.0	15	5.00	300.1	10	3.33	3600.4
COI-BCDT-Sep	**100**	1.00	-	230	2.30	300.1	210	2.10	3600.3
COI-MER	**7081**	1.00	36002.7	56048	7.92	406.4	13840	1.95	3616.5
Average	**454**	1.00		2653	17.52	195.0	804	16.47	1801.4

Table 2: Results for the Long and Medium instances of the First International Timetabling Competition. The GOAL column shows the solutions from the GOAL research group web site [37], which the GOAL group has shown to be virtually optimal. The other columns show the results for KHE24x24, running for 5 minutes and 60 minutes. This table is derived from the instances and solutions stored in XESTT archive file `KHE24-2024-03-07-INRC1-Long-And-Medium.xml`, available at [21].

Instances (30)	**GOAL**		**KHE24x24-05**			**KHE24x24-60**		
	Cost	**Rel.**	**Cost**	**Rel.**	**Time**	**Cost**	**Rel.**	**Time**
INRC1-L01	**197**	1.00	199	1.01	222.4	199	1.01	405.5
INRC1-L02	**219**	1.00	227	1.04	219.9	225	1.03	775.3
INRC1-L03	**240**	1.00	**240**	1.00	226.0	**240**	1.00	480.4
INRC1-L04	**303**	1.00	305	1.01	219.1	304	1.00	740.4
INRC1-L05	**284**	1.00	285	1.00	218.4	285	1.00	686.9
INRC1-LH01	**346**	1.00	367	1.06	237.4	354	1.02	2471.2
INRC1-LH02	**89**	1.00	95	1.07	249.3	93	1.04	2192.8
INRC1-LH03	**38**	1.00	44	1.16	235.8	42	1.11	1779.7
INRC1-LH04	**22**	1.00	33	1.50	229.8	30	1.36	1741.4
INRC1-LH05	**41**	1.00	52	1.27	224.4	47	1.15	1957.4
INRC1-LL01	**235**	1.00	252	1.07	235.9	246	1.05	2451.3
INRC1-LL02	**229**	1.00	250	1.09	239.2	238	1.04	2455.2
INRC1-LL03	**220**	1.00	262	1.19	248.3	244	1.11	2445.9
INRC1-LL04	**222**	1.00	256	1.15	235.3	237	1.07	2443.1
INRC1-LL05	**83**	1.00	87	1.05	245.0	84	1.01	1908.4
INRC1-M01	**240**	1.00	243	1.01	179.2	243	1.01	186.1
INRC1-M02	**240**	1.00	244	1.02	182.2	244	1.02	214.9
INRC1-M03	**236**	1.00	239	1.01	180.2	239	1.01	197.4
INRC1-M04	**237**	1.00	240	1.01	182.3	240	1.01	198.8
INRC1-M05	**303**	1.00	308	1.02	209.3	308	1.02	285.8
INRC1-MH01	**111**	1.00	140	1.26	219.2	132	1.19	1471.3
INRC1-MH02	**221**	1.00	237	1.07	226.9	231	1.05	1040.9
INRC1-MH03	**34**	1.00	41	1.21	208.9	41	1.21	301.0
INRC1-MH04	**78**	1.00	85	1.09	218.8	85	1.09	458.2
INRC1-MH05	**119**	1.00	130	1.09	222.7	132	1.11	1102.9
INRC1-ML01	**157**	1.00	170	1.08	234.0	163	1.04	1079.7
INRC1-ML02	**18**	1.00	26	1.44	139.6	26	1.44	241.2
INRC1-ML03	**29**	1.00	35	1.21	125.7	35	1.21	141.5
INRC1-ML04	**35**	1.00	41	1.17	183.5	41	1.17	330.8
INRC1-ML05	**107**	1.00	114	1.07	217.6	114	1.07	1309.5
Average	**164**	1.00	175	1.11	213.9	171	1.09	1116.5

Table 3: Results for the Sprint instances of the First International Timetabling Competition. Details as for Table 2. This table is derived from XESTT archive file `KHE24-2024-03-07-INRC1-Sprint.xml`, available at [21].

Instances (30)	**GOAL**		**KHE24x24-05**			**KHE24x24-60**		
	Cost	**Rel.**	**Cost**	**Rel.**	**Time**	**Cost**	**Rel.**	**Time**
INRC1-S01	**56**	1.00	**56**	1.00	42.0	**56**	1.00	48.1
INRC1-S02	**58**	1.00	**58**	1.00	38.5	**58**	1.00	38.1
INRC1-S03	**51**	1.00	**51**	1.00	50.2	**51**	1.00	50.4
INRC1-S04	**59**	1.00	**59**	1.00	44.1	**59**	1.00	44.9
INRC1-S05	**58**	1.00	**58**	1.00	49.7	**58**	1.00	50.4
INRC1-S06	**54**	1.00	**54**	1.00	52.0	**54**	1.00	50.5
INRC1-S07	**56**	1.00	**56**	1.00	46.9	**56**	1.00	45.3
INRC1-S08	**56**	1.00	**56**	1.00	40.4	**56**	1.00	40.4
INRC1-S09	**55**	1.00	**55**	1.00	59.1	**55**	1.00	61.5
INRC1-S10	**52**	1.00	**52**	1.00	46.8	**52**	1.00	46.7
INRC1-SH01	**32**	1.00	34	1.06	31.6	34	1.06	31.4
INRC1-SH02	**32**	1.00	**32**	1.00	30.0	**32**	1.00	29.8
INRC1-SH03	**62**	1.00	**62**	1.00	42.7	**62**	1.00	42.3
INRC1-SH04	**66**	1.00	67	1.02	58.6	67	1.02	58.1
INRC1-SH05	**59**	1.00	**59**	1.00	54.2	**59**	1.00	54.2
INRC1-SH06	**130**	1.00	134	1.03	59.8	134	1.03	58.7
INRC1-SH07	**153**	1.00	**153**	1.00	53.0	**153**	1.00	52.6
INRC1-SH08	**204**	1.00	206	1.01	124.2	206	1.01	127.4
INRC1-SH09	**338**	1.00	**338**	1.00	149.7	**338**	1.00	223.6
INRC1-SH10	**306**	1.00	**306**	1.00	72.3	**306**	1.00	70.0
INRC1-SL01	**37**	1.00	38	1.03	50.5	38	1.03	50.2
INRC1-SL02	**42**	1.00	43	1.02	37.4	43	1.02	36.8
INRC1-SL03	**48**	1.00	49	1.02	51.5	49	1.02	53.9
INRC1-SL04	**73**	1.00	**73**	1.00	139.8	**73**	1.00	231.6
INRC1-SL05	**44**	1.00	45	1.02	48.9	45	1.02	46.0
INRC1-SL06	**42**	1.00	**42**	1.00	20.2	**42**	1.00	20.0
INRC1-SL07	**42**	1.00	43	1.02	35.8	43	1.02	34.4
INRC1-SL08	**17**	1.00	**17**	1.00	19.4	**17**	1.00	19.8
INRC1-SL09	**17**	1.00	**17**	1.00	19.0	**17**	1.00	18.8
INRC1-SL10	**43**	1.00	**43**	1.00	31.0	**43**	1.00	32.5
Average	**78**	1.00	79	1.01	53.3	79	1.01	58.9

`INRC1-ML02`. Analysis of these cases is needed. On the Sprint instances, running for 60 minutes is never better than running for 5.

4.3 The Second International Nurse Rostering Competition

The Second International Nurse Rostering Competition [5,6] was notable for requiring its instances to be solved week by week, as occurs in the real world. However, here we focus on a set of conventional 4-week and 8-week instances from the competition that have been tackled by several authors [27,38]. Their converted versions appear in files `INRC2-4.xml` and `INRC2-8.xml` at [21], along with some solutions produced by the authors of [27].

Table 4: Results for the Second International Timetabling Competition 4-week instances. The LOR17 column shows the solutions obtained from the authors of [27]. Table derived from archive file `KHE24-2024-03-07-INRC2-4.xml`, available at [21].

Instances (20)	**LOR17**		**KHE24x24-05**			**KHE24x24-60**		
	Cost	**Rel.**	**Cost**	**Rel.**	**Time**	**Cost**	**Rel.**	**Time**
INRC2-4-030-1-6291	**1695**	1.00	2040	1.20	300.1	1865	1.10	3600.1
INRC2-4-030-1-6753	**1890**	1.00	2230	1.18	300.1	2005	1.06	3600.2
INRC2-4-035-0-1718	**1425**	1.00	1835	1.29	300.1	1590	1.12	3600.2
INRC2-4-035-2-8875	**1155**	1.00	1535	1.33	300.1	1360	1.18	3600.1
INRC2-4-040-0-2061	**1685**	1.00	2125	1.26	300.1	1875	1.11	3600.1
INRC2-4-040-2-6106	**1890**	1.00	2360	1.25	300.1	2020	1.07	3600.1
INRC2-4-050-0-0487	**1505**	1.00	2015	1.34	300.1	1745	1.16	3600.1
INRC2-4-050-0-7272	**1500**	1.00	1975	1.32	300.1	1690	1.13	3600.1
INRC2-4-060-1-6115	**2505**	1.00	3335	1.33	300.2	2840	1.13	3600.2
INRC2-4-060-1-9638	**2750**	1.00	3785	1.38	300.2	3165	1.15	3600.2
INRC2-4-070-0-3651	**2435**	1.00	3225	1.32	300.1	2850	1.17	3600.1
INRC2-4-070-0-4967	**2175**	1.00	3035	1.40	300.1	2565	1.18	3600.1
INRC2-4-080-2-4333	**3340**	1.00	4015	1.20	300.1	3775	1.13	3600.3
INRC2-4-080-2-6048	**3260**	1.00	4230	1.30	300.1	3750	1.15	3600.2
INRC2-4-100-0-1108	**1245**	1.00	2100	1.69	300.2	1670	1.34	3600.2
INRC2-4-100-2-0646	**1950**	1.00	2700	1.38	300.2	2275	1.17	3600.2
INRC2-4-110-0-1428	**2440**	1.00	3245	1.33	300.2	2755	1.13	3600.2
INRC2-4-110-0-1935	**2560**	1.00	3550	1.39	300.2	3015	1.18	3600.2
INRC2-4-120-1-4626	**2170**	1.00	3060	1.41	300.2	2600	1.20	3600.3
INRC2-4-120-1-5698	**2220**	1.00	3140	1.41	300.2	2650	1.19	3600.2
Average	**2090**	1.00	2777	1.34	300.1	2403	1.15	3600.2

The results for the 4-week instances appear in Table 4. There is no running time information in the solution files obtained from other authors. Reference [27] gives a formula that was used to determine the running time limit; it is about 20 minutes for the largest 4-week instances. Reference [38] used a running time limit of 2750 seconds (about 45 minutes).

KHE24's results are disappointing, relative to the best known solutions and to the author's own testing. For example, the cost of the 5-minute solution to instance INRC2-4-030-1-6291 reported here is 2040 (Table 4), but in other 5-minute tests the author has obtained solutions whose cost is as low as 1785. This problem will probably be easy to fix, but the paper submission deadline intervened before the the author had time to fix it.

Sadly, the publication deadline has prevented the author from tackling the 8-week instances with KHE24x24. A few preliminary tests have revealed that KHE24x24's solutions are not currently competitive. The author's experience with other data sets suggests that detailed analysis will reveal concrete areas in which the algorithm is performing poorly, whose improvement will lead to more competitive results. However this is speculation at present.

4.4 The 2014 Curtois and Qu instances

The instances here are the 24 instances published in 2014 by Curtois and Qu [7,8]. Converted versions appear in file `CQ14.xml`, which also holds four sets of solutions received from Curtois via private correspondence.

Again, the publication deadline has prevented the author from presenting results for this data set. He does not have even preliminary results to report.

There are more recent results for these instances [8,10,11]. Reference [8] has slightly better results than the ones in file `CQ14.xml`, with lower bounds. The bounds show that many of the results are optimal, although the last five instances, which are much larger than the others, still have significant optimality gaps. We regard those last five instances as not relevant to the concerns of this paper, since they have cycles of 26 and 52 weeks, much longer than is encountered in practice.

5 Conclusion

This paper has presented KHE24, a nurse rostering solver which aims to find very good but not optimal solutions quickly across a wide range of instances. Polynomial-time methods are used: time sweep for the initial assignment, and ejection chains and optimal reassignment using dynamic programming for repair. No parameter tuning is needed.

The results so far give hope that the solver will eventually be successful in practice (that is, taking cost, running time, and breadth of application into account). At the time of writing, however, many tests are missing, and for some of those the author's preliminary results are uncompetitive. There are some worrying tendencies: relative solution cost seems to deteriorate as instance size increases, and the solver does not always make effective use of the longer (60-minute) running time. The author is actively continuing this work.

References

1. J. L. Arthur and A. Ravindran A multiple objective nurse scheduling model, AIIE Transactions 13(1), pages 55–60 (1981).

2. Shahriar Asta and Ender Özcan, A tensor-based approach to nurse rostering, In Ender Özcan, Edmund Burke, and Barry McCollum (eds.), PATAT 2014 (Tenth international conference on the Practice and Theory of Automated Timetabling, York, UK, August 2014), 442–445 (2014)
3. Edmund K. Burke, Tim Curtois, Rong Qu, and Greet Vanden Berghe, A time predefined variable depth search for nurse rostering, INFORMS Journal on Computing, 25(3), 411–419 (2013), accessed via `http://eprints.nottingham.ac.uk/28283/1/JOC12vds.pdf`
4. Edmund K. Burke and Tim Curtois, New approaches to nurse rostering benchmark instances, European Journal of Operational Research 237, 71–81 (2014)
5. Sara Ceschia, Nguyen Thi Thanh Dang, Patrick De Causmaecker, Stephaan Haspeslagh, and Andrea Schaerf, Second international nurse rostering competition (INRC-II), problem description and rules. oRR abs/1501.04177 (2015). URL http://arxiv.org/abs/1501.04177
6. Sara Ceschia, Nguyen Thi Thanh Dang, Patrick De Causmaecker, Stephaan Haspeslagh, and Andrea Schaerf, Second international nurse rostering competition (INRC-II) web site, URL http://mobiz.vives.be/inrc2/.
7. Tim Curtois and Rong Qu, Computational results on new staff scheduling benchmark instances URL http://www.cs.nott.ac.uk/ psztc/NRP/ (2014)
8. Tim Curtois, Employee Shift Scheduling Benchmark Data Sets, `http://www.schedulingbenchmarks.org/` (2019)
9. Nguyen Thi Thanh Dang, Sara Ceschia, Andrea Schaerf, Patrick De Causmaecker, and Stefaan Haspeslagh, Solving the multi-stage nurse rostering problem, PATAT 2016 (Eleventh international conference on the Practice and Theory of Automated Timetabling, Udine, Italy, August 2016), 473–475 (2016)
10. Emir Demirovíc, Nysret Musliu, and Felix Winter, Modeling and solving staff scheduling with partial weighted maxSAT, PATAT 2016 (Eleventh international conference on the Practice and Theory of Automated Timetabling, Udine, Italy, August 2016), 109–125 (2016)
11. Emir Demirovíc, Nysret Musliu, Felix Winter, and Peter J. Stuckey, Solution-based phase saving and MaxSAT for employee scheduling: a computational study, PATAT 2018 (Twelfth international conference on the Practice and Theory of Automated Timetabling, Vienna, August 2018), 453–457 (2018)
12. Kathryn A. Dowsland, Nurse scheduling with tabu search and strategic oscillation, European Journal of Operational Research 106, 393–407 (1998)
13. Stefaan Haspeslagh, Patrick De Causmaecker, Martin Stølevik, and Andrea Schaerf, First international nurse rostering competition website, URL: http://www.kuleuven-kortrijk.be/nrpcompetition (2010)
14. Stefaan Haspeslagh, Patrick De Causmaecker, Martin Stølevik, and Andrea Schaerf, The first international nurse rostering competition 2010, Annals of Operations Research, 218, 221–236 (2014)
15. Han Hoogeveen and Tim van Weelden, Personalized nurse rostering through linear programming, In Ender Özcan, Edmund Burke, and Barry McCollum (eds.), PATAT 2014 (Tenth international conference on the Practice and Theory of Automated Timetabling, York, UK, August 2014), 476–478 (2014)
16. Ahmed Kheiri, Ender Özcan, Rhyd Lewis, and Jonathan Thompson, A sequence-based selection hyper-heuristic: a case study in nurse rostering, PATAT 2016 (Eleventh international conference on the Practice and Theory of Automated Timetabling, Udine, Italy, August 2016), 503–505 (2016)
17. Jeffrey H. Kingston, The HSEval High School Timetable Evaluator, URL `http://jeffreykingston.id.au/cgi-bin/hseval.cgi` (2010)
18. Jeffrey H. Kingston, Repairing high school timetables with polymorphic ejection chains, Annals of Operations Research, DOI 10.1007/s10479-013-1504-3

19. Jeffrey H. Kingston, KHE14: An algorithm for high school timetabling, In Ender Özcan, Edmund Burke, and Barry McCollum (eds.), PATAT 2014 (Tenth international conference on the Practice and Theory of Automated Timetabling, York, UK, August 2014), 269–291
20. Jeffrey H. Kingston, KHE web site, `http://jeffreykingston.id.au/khe` (2014).
21. Jeffrey H. Kingston, XESTT web site, `http://jeffreykingston.id.au/xestt` (2017)
22. Jeffrey H. Kingston, KHE18: a solver for nurse rostering, PATAT 2018 (Twelfth international conference on the Practice and Theory of Automated Timetabling, Vienna, August 2018)
23. Jeffrey H. Kingston, Gerhard Post, and Greet Vanden Berghe, A unified nurse rostering model based on XHSTT, PATAT 2018 (Twelfth international conference on the Practice and Theory of Automated Timetabling, Vienna, August 2018)
24. Jeffrey H. Kingston, Modelling history in nurse rostering, PATAT 2018 (Twelfth international conference on the Practice and Theory of Automated Timetabling, Vienna, August 2018). Also Annals of Operations Research, DOI 10.1007/s10479-019-03288-x
25. Jeffrey H. Kingston, Improving the dynamic programming algorithm for nurse rostering, PATAT 2022 (Thirteenth International Conference on Practice and Theory of Automated Timetabling, Leuven, Belgium, August 2022).
26. Jeffrey H. Kingston, Timetabling research: a progress report, PATAT 2022 (International Conference on the Practice and Theory of Automated Timetabling, Leuven, Belgium, August 2022).
27. A. Legrain, J. Omer, and S. Rosat, A rotation-based branch-and-price approach for the nurse scheduling problem (2017). Working paper, available at `https://hal.archives-ouvertes.fr/hal-01545421`.
28. M. J. Louw, I. Nieuwoudt, and J. H. Van Vuuren, Finding good nursing duty schedules: a case study. Technical report, Department of Applied Mathematics, Stellenbosch University, South Africa (2005). Received from Tim Curtois and held by this author.
29. Antonın Novák, Roman Václavık, Premysl Sucha, and Zdenek Hanzálek, Nurse rostering problem: tighter upper bound for pricing problem in branch and price approach, MISTA 2015 (7th Multidisciplinary International Conference on Scheduling: Theory and Applications), Prague, August 2015, 759–763
30. Gerhard Post, XHSTT web site, `https://www.utwente.nl/en/eemcs/dmmp/hstt/` (2011)
31. Samad Ahmadi, Sophia Daskalaki, Jeffrey H. Kingston, Jari Kyngäs, Cimmo Nurmi, Gerhard Post, David Ranson, and Henri Ruizenaar, An XML format for benchmarks in high school timetabling, Annals of Operations Research, 194, 385–397 (2012)
32. Florian Mischek and Nysret Musliu, Integer programming and heuristic approaches for a multi-stage nurse rostering problem, PATAT 2016 (Eleventh international conference on the Practice and Theory of Automated Timetabling, Udine, Italy, August 2016), 245–262 (2016)
33. Christopher Rae and Nelishia Pillay, Investigation into an evolutionary algorithm hyper-heuristic for the nurse rostering problem, In Ender Özcan, Edmund Burke, and Barry McCollum (eds.), PATAT 2014 (Tenth international conference on the Practice and Theory of Automated Timetabling, York, UK, August 2014), 527–532 (2014)
34. Erfan Rahimian, Kerem Akartunali, and John Levine, A Hybrid Constraint Integer Programming Approach to Solve Nurse Scheduling Problems, MISTA 2015 (7th Multidisciplinary International Conference on Scheduling: Theory and Applications), Prague, August 2015, 429–442
35. M. Saddoune, G. Desaulniers, and F. Soumis, Aircrew pairings with possible repetitions of the same flight number, Computers and Operations Research **40**, 3 (2013), 805–814.
36. Haroldo G. Santos, Túlio A. M. Toffolo, Rafael A. M. Gomes, and Sabir Ribas, Integer programming techniques for the nurse rostering problem, Annals of Operations Research 239, 225–251 (2016)

37. Haroldo G. Santos, Túlio A. M. Toffolo, Rafael A. M. Gomes, and Sabir Ribas, `http://www.goal.ufop.br/nrp/`
38. Sara Ceschia and Andrea Schaerf, Solving the INRC-II nurse rostering problem by simulated annealing based on large neighborhoods, PATAT 2018 (Twelfth international conference on the Practice and Theory of Automated Timetabling, Vienna, August 2018), 331–338
39. Pieter Smet and Greet Vanden Berghe, Variable neighbourhood search for rich personnel rostering problems, MISTA 2015 (7th Multidisciplinary International Conference on Scheduling: Theory and Applications), Prague, August 2015, 928–930
40. Pieter Smet, Constraint reformulation for nurse rostering problems, PATAT 2018 (Twelfth international conference on the Practice and Theory of Automated Timetabling, Vienna, August 2018), 69–80
41. J. Van den Bergh, J. Belien, P. De Bruecker, E. Demeulemeester and L. De Boeck, Personnel scheduling: a literature review, European Journal of Operational Research, 226(3), 367–385, (2013).

Part III

Extended Abstracts

Efficient constraint evaluation for the nurse rostering problem

Robin Tourlamain, Pieter Smet, and Greet Vanden Berghe

KU Leuven, Gebroeders de Smetstraat 1, 9000 Gent, Belgium
robin.tourlamain@kuleuven.be

Keywords: Nurse rostering, Constraint evaluation, Delta-evaluation

1 Introduction

Local search algorithms, which are often utilized to generate solutions for the nurse rostering problem, rely heavily on the value the evaluation function produces at each iteration. The speed of the evaluation process directly correlates with the number of iterations such algorithms can perform and, consequently, the quality of the solutions that are ultimately obtained. While the evaluation can be performed in a number of different ways, almost no publications report how they do so. Moreover, the significance of the evaluation approach extends beyond performance, with the reproducibility of algorithms and their results negatively affected when this design choice goes unreported.

Burke et al. [1] introduced a generalized constraint evaluation model for nurse rostering. It is one of the few counterexamples to the lack of reporting on evaluation functions. The model focuses on a flexible and user-friendly way to handle the evaluation of rosters. It re-evaluates the entire roster of at least one nurse after each update. We believe we can further increase the evaluation performance.

We introduce an efficient approach to constraint evaluation in local search algorithms, and leverage our results to highlight the importance of disclosing evaluation techniques. By accelerating the evaluation process, we aim to achieve better results with existing algorithms in time-constrained scenarios. We will demonstrate how it is worth treating one's evaluation approach as a design decision, given the tangible impact it has on the effectiveness of an algorithm.

2 Problem context

To evaluate nurse rostering solutions appropriately and efficiently, it is necessary to consider both the (soft) constraint type and the nature of the modification. The constraint types we take into consideration are those described by Bilgin et al. [2], which accurately model many real-world restrictions:

- Counters: constrain the number of occurrences of a specified set of assignments.
- Series: constrain the number of consecutive occurrences of a specified set of assignments.
- Successive series: constrain the consecutive occurrences of disjoint series.

Note that we treat these constraints as soft constraints and that the degree of their violation determines the significance of the penalty. The evaluation function is a weighted sum of all penalties.

The modifications we consider are single assignment modifications: changing, adding, or removing a single shift on a day in a nurse's roster. Compound moves such as swaps can be created by simply linking together multiple modifications.

It is possible to assess the impact of one's evaluation function on instances from various existing datasets. The instances and constraints from Bilgin et al. [2], often referred to as the KaHo instances, and those from the international nurse rostering competition [3] all exclusively contain constraints that can be expressed as one of the three aforementioned general types and will therefore be used for the benchmarking process.

3 Evaluation methods

The most straightforward approach to constraint evaluation is to re-evaluate the entire solution after each update. This method is easy to implement and requires no extra logic besides the evaluation itself. However, upon closer inspection, the method requires many unnecessary computations. The evaluation can therefore easily be improved by recalculating only the penalties in the roster of the affected nurse. Figure 1 provides a side by side comparison of these two approaches.

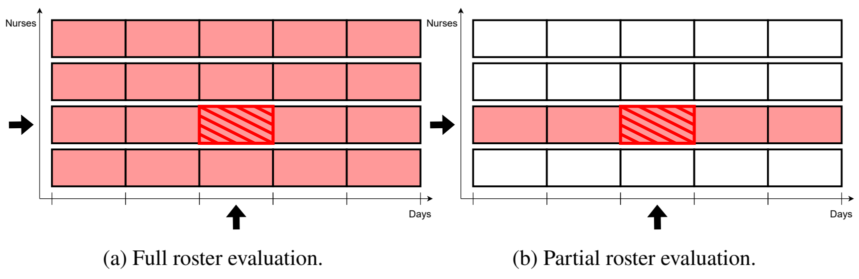


Fig. 1: Evaluation comparison

While evaluating only a single nurse's roster is a step in the right direction, modifications seldom impact all constraints associated with a nurse's roster. In pursuit of greater efficiency, we consequently propose a delta-evaluation method that only iterates over both the constraints and the days that are affected by the modification. The method has the potential to dramatically accelerate iterative improvement algorithms for nurse rostering and other timetabling problems. Figure 2 provides a visual example of our proposed delta-evaluation method. The method will be detailed at the conference.

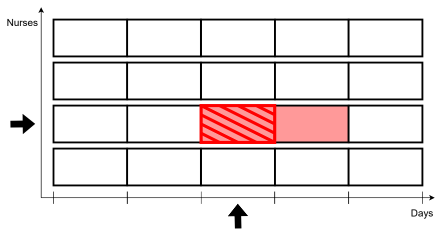


Fig. 2: Delta evaluation.

4 Preliminary results

The number of iterations per time unit impacts the number of iterations one can perform and, ultimately, the final solution quality attained. For the following benchmarking excercise, a single iteration consists of applying a random feasible modification and recalculating the roster penalty. There is no overarching search algorithm present in these experiments because we are simply counting the number of completed random iterations. We compare our delta-evaluation with the speed of evaluating the roster of a single nurse. By way of example, Table 1 documents the average evaluation speedup over ten 1-second runs per instance from the KaHo dataset.

Table 1: Evaluation speedup per instance.

Instance	**# Days**	**# Nurses**	**Speedup**
Hospital1-Emergency-Absence	28	27	x4.35
Hospital1-Emergency-Normal	28	27	x4.40
Hospital1-Emergency-Overload	28	27	x4.40
Hospital1-Geriatrics-Absence	28	21	x3.73
Hospital1-Geriatrics-Normal	28	21	x4.00
Hospital1-Geriatrics-Overload	28	21	x3.99
Hospital1-Meal Preparation-Absence	29	32	x3.34
Hospital1-Meal Preparation-Normal	29	32	x3.25
Hospital1-Meal Preparation-Overload	29	32	x3.57
Hospital1-Psychiatry-Absence	31	19	x5.89
Hospital1-Psychiatry-Normal	31	19	x5.91
Hospital1-Psychiatry-Overload	31	19	x6.19
Hospital1-Reception-Absence	42	19	x7.32
Hospital1-Reception-Normal	42	19	x7.37
Hospital1-Reception-Overload	42	19	x7.56
Hospital2-Palliative Care-Absence	91	27	x13.65
Hospital2-Palliative Care-Normal	91	27	x13.90
Hospital2-Palliative Care-Overload	91	27	x13.34

Evaluating a single nurse's entire roster leaves room for improvement, as the delta-evaluation outperforms it on every instance in the dataset. Indeed, our worst case performance would equal that of the partial roster evaluation. As the time horizon of an instance lengthens, the delta-evaluation eliminates the need to evaluate a greater proportion of days and therefore results in more significant speedups. The new delta-evaluation method can be applied within any iterative algorithm for roster optimization.

References

1. Burke, E. et al. (2003). Fitness Evaluation for Nurse Scheduling Problems. Proceedings of the IEEE Conference on Evolutionary Computation, CEC. 2. 10.1109/CEC.2001.934319.
2. Bilgin, B. et al. Local search neighbourhoods for dealing with a novel nurse rostering model. Annals of Operations Research **194**, 33–57 (2012). https://doi.org/10.1007/s10479-010-0804-0
3. Haspeslagh, S. et al. (2010). First international nurse rostering competition 2010. Annals of Operations Research. 218. 10.1007/s10479-012-1062-0.

Matheuristic for Approximating a Frontier for a Many-objective University Timetabling Problem

Matthew Davison[1], Ahmed Kheiri[2], and Konstantinos G. Zografos[3]

[1] STOR-i Centre for Doctoral Training, Lancaster University, Lancaster, UK
`m.davison2@lancaster.ac.uk`
[2] Department of Management Science, Lancaster University, Lancaster, UK
`a.kheiri@lancaster.ac.uk`
[3] Department of Management Science, Centre for Transport and Logistics (CENTRAL), Lancaster University, Lancaster, UK
`k.zografos@lancaster.ac.uk`

Keywords: University Timetabling, Hybrid Teaching, Mixed Integer Programming, Matheuristic

1 Introduction

The University Course Timetabling Problem (UCTTP) involves assigning *events* (such as lectures and seminars) to *times* and *locations* along with assigning *staff* and *students* to these events. One approach to formulating this mathematically is to model the problem as a Mixed Integer Linear Program (MILP) [1]. Given the resulting complexity of the problem for real-life instances, heuristic approaches have been proposed to solve the problem [2].

One matheuristic used is known as fix-and-optimise [4] where neighbourhoods are allowed to improve while the rest of the solution is unchanged. Typically, this has been used as part of a single-objective approach. However, the UCTTP can be modelled as a multi-objective problem. Trade-offs in conflicting objectives can be found using an ϵ-constraint approach but this is hard when working with many objectives.

In a decision support context, it is desirable to quickly find these trade-offs by generating an approximation(s) of the Pareto frontier(s). Generating high-quality frontiers allows decision-makers to understand the trade-offs so the most desirable timetable for implementation can be selected.

We propose a method that leverages the gradual improvement of fix-and-optimise to search the objective space but still allows for elements of ϵ-constraint approaches to be added and removed when needed. This results in a matheuristic approach suitable for any many-objective problem.

2 Problem description

The details of the specific UCTTP we are solving here are described in [3]. The key aspect of the UCTTP in this paper is incorporating hybrid teaching, where classes can happen in-person, online or in a hybrid mode. In this model, students can express a preference for a certain mode of study. For this reason, we focus on the following three objectives:

z_1: Maximise total module attendance.
z_2: Minimise total number of deviations from mode preferences.
z_3: Minimise total number of student scheduling issues.

3 Method description

The method requires at least one feasible solution as input. We understand that this itself is not a trivial task for any UCTTP however many techniques have been proposed [2]. We will assume that the (approximate) nadir point is known. The flowchart in Figure 1 outlines the general structure of the method.

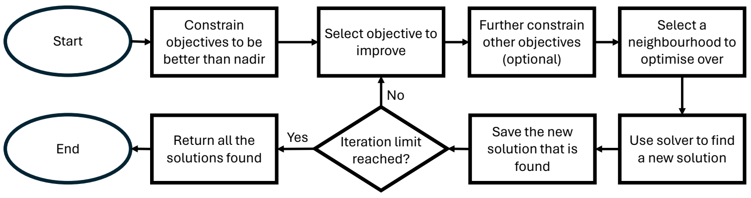

Start
Constrain objectives to be better than nadir
Select objective to improve
Further constrain other objectives (optional)
Select a neighbourhood to optimise over
Use solver to find a new solution
Save the new solution that is found
Iteration limit reached?
No
Yes
Return all the solutions found
End

Fig. 1: Flowchart providing an overview of the method.

The pseudocode presented in Algorithm 1 describes a simple variant of this method. There are two comments on this pseudocode. These indicate the key places where more sophisticated selection methods could be used instead of random selection. The combined use of neighbourhoods and unconstrained objectives allows for small changes in objectives like an ϵ-constraint approach.

4 Method demonstration

The instance for this demonstration is a modification of the instance *mary-fal18* from the fourth International Timetabling Competition [5]. The significant changes are that we only consider 400 students and reduce physical room capacity fourfold (for details see [3]). The initial solutions are four lexicographic solutions (see Table 1) and using the objective values for these solutions we can obtain the exact nadir point. The method used for the demonstration is described in Algorithm 1. We have $|S| = 4$ and use the parameters $I = 50$, $R = 5$ and $N = 99$. This produces a new pool of solutions S^{new}. Any solution in S^{new} dominated by another solution in this set is removed. The results were found with a machine with Ubuntu 22.04.1 LTS using an Intel(R) Xeon(R) Gold 6248R CPU running at 3.00GHz and 16GB of RAM. The method was implemented in Python 3.10.12 using Gurobi 10.0.2.

This run of the method yields 68 non-dominated alternative solutions. It can be shown that some of the non-dominated solutions were found after visiting what would turn out

Algorithm 1: Pseudocode for simple variant of method

```
Input: Set of initial solutions $S$, # iterations $I$, # repeats $R$, neighbourhood size $N$.
Output: New set of solutions $S^{new}$.
$S^{new} \leftarrow \{\}$;
Constrain all objectives to be better than the nadir point;
for $r = 0$ to $R$ do
    for $s$ in $S$ do
        $s^{current} \leftarrow s$;
        for $i = 0$ to $I$ do
            Randomly select objective $z$ to optimise ;      // Selecting objective
            Randomly select $N$% of students to fix;     // Selecting neighborhood
            Fix selected students according to $s^{current}$;
            Optimise $z$ to find $s^{new}$;
            $S^{new} \leftarrow S^{new} \cup \{s^{new}\}$;
            $s^{current} \leftarrow s^{new}$;
        end
    end
end
return $S^{new}$;
```

Table 1: All lexicographic orderings of objectives and their corresponding objective values. Some orderings attained the same objective values.

Ordering(s)	z_1	z_2	z_3
(z_1,z_2,z_3), (z_1,z_3,z_2)	1,599	102	53
(z_2,z_1,z_3)	1,506	0	26
(z_2,z_3,z_1), (z_3,z_2,z_1)	1,486	0	0
(z_3,z_2,z_1)	1,554	68	0

to be a dominated solution. Figure 2 shows that the alternative solutions stay close to the starting solutions and that some starting solutions produced more alternative solutions than others. Only certain sections of the efficient frontier have been approximated but this may be improved using more sophisticated objective and neighbourhood selection procedures.

5 Future work

There are several possible research directions to take this work in:

- Develop better objective and neighbourhood selection procedures so that more of the frontier can be approximated.
- Compare the generated frontiers with frontiers found using an exact method to assess the quality of the method's output.
- Validate the quality of this method by using a large test set with a wide range of instances.

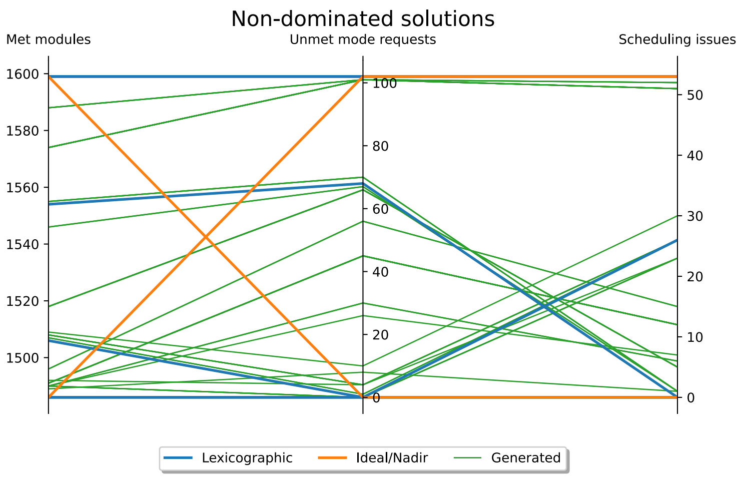


Fig. 2: Plot of non-dominated solutions and their corresponding objective values.

- Experiment with real-life instances rather than using instances that have been reduced in size.
- Apply the method to a variant of the UCTTP with more than three objectives.

Acknowledgements We gratefully acknowledge the support of the EPSRC funded EP/S022252/1 STOR-i Centre for Doctoral Training.

References

1. Aziz, N.L.A., Aizam, N.A.H.: A brief review on the features of university course timetabling problem. AIP Conference Proceedings **2016**(1) (2018). https://doi.org/10.1063/1.5055403, https://aip.scitation.org/doi/abs/10.1063/1.5055403
2. Babaei, H., Karimpour, J., Hadidi, A.: A survey of approaches for university course timetabling problem. Computers & Industrial Engineering **86**, 43–59 (2015). https://doi.org/https://doi.org/10.1016/j.cie.2014.11.010, https://www.sciencedirect.com/science/article/pii/S0360835214003714
3. Davison, M., Kheiri, A., Zografos, K.: Modelling and solving the university course timetabling problem with hybrid teaching considerations (Under Submission, 2024)
4. Mikkelsen, R.Ø., Holm, D.S.: A parallelized matheuristic for the International Timetabling Competition 2019. Journal of Scheduling **25**(4), 429–452 (Aug 2022). https://doi.org/10.1007/s10951-022-00728-8, https://doi.org/10.1007/s10951-022-00728-8
5. Müller, T., Rudová, H., Müllerová, Z.: Real-world university course timetabling at the international timetabling competition 2019. Journal of Scheduling (Apr 2024). https://doi.org/10.1007/s10951-023-00801-w, https://doi.org/10.1007/s10951-023-00801-w

‘I know it when I see it’ – Developing Quality Schedules Considering Subjective or Unspecified Criteria

Douglas Moody

Computer Systems Technology Department,
New York City College of Technology,
Brooklyn NY,11201, USA
dmoody@citytech.cuny.edu

Abstract. Most timetabling problems have a given objective function to measure the quality of a solution. However, users may have a “I know it when I see it” recognition of a quality schedule, without specifying the complete basis for their judgment. In this situation, the objective function cannot be exclusively used as a solution quality measurement. This work presents an AI based approach to aid in categorizing the solution’s quality when the users have not explicitly defined all factors used in their criteria.

Keywords: Timetabling, Artificial intelligence, Image classification

1 Introduction

This work examines when the user community is not able to precisely articulate their preferences, factors and criteria for evaluating the quality of a timetable solution. This may cause the true user’s quality measurement to vary from the objective function. Without the complete knowledge of the user community’s criteria, the scheduler is at a loss to improve the quality of the schedule. The user may employ a “I know it when I see it” approach [6] - where the user, without articulating specific criteria, judges the solution’s quality only after viewing.

This work focuses on a common timetabling problem - the Travelling Tournament Problem (TTP) defined in [1], to illustrate this possible scenario. This problem has been well studied and has a very precise and calculable objective function. We add new criteria to the objective function, though this criterion will be unknown to the timetabling algorithm. With use of AI, this work will still be able to recognize quality solutions considering the unknown criteria. The solution of the TTP has been addressed by a wide variety of integer programming, constraint programming, tiling and search techniques [7][4][3][2]. Our focus is not on improving the optimization of the distance objective function, but rather the incorporation of additional evaluation criteria. Others like Tuffaha et al. [11] have introduced known and measurable criteria to the TTP problem, such as season duration. This work looks to incorporate unknown and subjective criteria.

2 Challenge

Our scenario begins with the user community providing the teams and distances to enable the creation of a schedule according to the hard constraints of the TTP problem.

A scheduler uses software to create the schedule with a minimal total distance according to its algorithm. However, upon showing the schedule to the user community, the schedule receives a poor rating. The scheduler is not sure why, so additional schedules are generated for user review. These additional schedules have slightly higher distance values, but the perceived quality of solution by the user community varies, though no specific criticisms and feedback are provided to the scheduler.

For an example in a real-world scenario, let us assume that the user community would favor each team to have at least 3 "*long road trips*". We define a long road as a trip where the distance travelled is at least 80% of the longest trip taken by any team within the league. The reasoning for this user preference could be as follows:

- For a professional league, all teams should have to bear the burden of a long road trip (perhaps changing time zones) a few times to share the pain of the travel across the league.
- For a youth sports league, the road trips might represent exciting opportunities for hotel stays for the players. Each team should have a few fun weekends.

Hence a high-quality schedule is one that has relatively low total travel distance, but also provides that most teams have at least 3 long road trips. Our scenario is assuming the user community has never explicitly stated or even realizes this preference. This new criterion is known for our analyses and simulation in our project but is not known or used by algorithm evaluating or generating candidate schedules. Our approach consists of 3 major steps:

1) Convert the solution of the TTP algorithm to an image – Rendering the Schedule
2) Train the AI model to recognize good and poor schedules.
3) Generation and classification of candidate schedules.

We have chosen to work on the NFL32 instance of the TTP. This instance has sufficient teams to enable a schedule image to be rendered. The information is provided in a github repository of TTP instances and solutions on the Robin X website [12]. The algorithm selected is the NFL32Sol_ModifiedCircle solution [9]. The next sections describe each step.

3 Rendering the TTP Schedule as an Image

The output of a typical TTP solution is a file indicating the home team, away team and the slot (or week) of their game. The RobinX web site provides for the formats of TTP problems. This solution output must be converted to an image, where each pixel carries information about the solution. Since the TTP is a distance-oriented problem, we construct our grid will use colors to show the distance travelled each week. Our color will be a shade of a grayscale. Each color RGB pixel color will be one value providing example pixel colors of (1,1,1), (2,2,2), (50,50,50), (200,200,200), etc. The exception is those pixels close white (over 200) are colored yellow (255,255,0) to enable distinction with the white background. Pixel values are calculated based on the distance to the next game. The dark blocks indicate light or no travel for the team, while the brighter shades

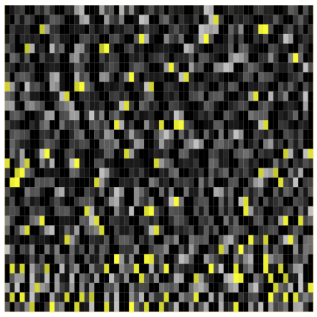


Fig. 1: A schedule image of the base solution for the project.

of gray and yellow (bright white in grayscale) indicate longer trips to the game location. The row order rendering of the TTP (each row is a particular team) is based upon the number of long trips within a season for the team. A sample TTP screen image output based on this coloration is shown in Fig. 1.

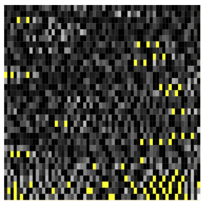
4 Training the AI Image Classification Model

We use the public and free image classification training platform provided by Google's Teachable [10]. The platform allows the AI user to create image via a web cam and mark their classification. We use a webcam to project the rendering of two sets of schedules – each referring to a good schedule or a poor schedule.

Our schedule generation process starts the NFL32Sol_ModifiedCirclec solution provided in the RobinX web site. This solution is based on relaxation algorithms and well-known techniques for creating tournament matchup combinations. [13][5]. It is a high-quality solution according to the TTP objective function. We then performed several perturbations on this solution. A perturbation was done by switching of the physical home locations of two teams. For example, switching between New York and Boston would require a team to travel its distance to Boston, though the schedule states the away team is New York. We utilized the pathway2code.com platform [8] for the calculation of the new schedules and the rendering of the schedule image. The project produced a series of 100 schedules for classification training purposes. An example input to the training model is shown in Fig. 2.

5 Image Classification and Results

After training, the project generated 2 sets of 25 schedule images for classification. The first set had only 2 perturbations each from the best schedule in RobinX, while the second set had 30 perturbations. Schedules from both sets were given to the AI tool for classification. The AI tool returned the probability that the schedule was a good or poor schedule.



Total Distance: 1178493
Distance Variance from Base: 0%
Category Classification: POOR
Actual embedded road trips:
Teams with < 3 Long Road Trips: 22
Teams with 3- 7 Long Road Trips: 7
Teams with > 8 Long Road Trips: 3
(not provided to training model)

Fig. 2: Base Solution Schedule Image and Long Road Trip Counts

The first set of 2 perturbations contained generated schedules that varied 10% from the best schedule. Nearly all the schedules involved more than half the teams with fewer than 3 road trips. The classifier was able to categorize these schedules as "poor" with a greater than 90% probability. The second set of generated schedules with 30 perturbations were on average only 10-25% higher in distance, however the perturbation led to more teams having longer road trips. The classification effort again was successful over 90% in labelling these schedules as good. Some schedules had nearly identical distances in the two groups, but a different distribution of long road trips and were classified correctly. Also, artificial colorings of schedules with extremely high or low coloring regions were classified successfully based on road trip distribution.

6 Summary

The work shows that a solution rendered as an image can used by a classifier to categorize the quality of the schedule after a training phase. The AI model is unaware of the actual criteria. This classifier will recognize patterns in the schedule based on pixel colorings, enabling the classifier to categorize the quality of the solution without criteria knowledge These criteria are not always available in practice, can be subjective, and vary among users in the community. For users who only know a high-quality schedule when they see it, we show quality schedules can still be generated.

References

1. Easton, K., Nemhauser, G., & Trick, M. (2001). The traveling tournament problem: description and benchmarks.
2. Easton, K., Nemhauser, G., & Trick, M. (2002, August). Solving the travelling tournament problem: A combined integer programming and constraint programming approach. In International Conference on the Practice and Theory of Automated Timetabling (pp. 100-109). Springer, Berlin, Heidelberg.
3. Fujiwara, N., Imahori, S., Matsui, T., & Miyashiro, R. (2006, August). Constructive algorithms for the constantdistance traveling tournament problem. In International Conference on the Practice and Theory of Automated Timetabling (pp. 135-146). Springer, Berlin, Heidelberg.

4. Goerigk, M., Hoshino, R., Kawarabayashi, K, & Westphal, S. (2014). Solving the Traveling Tournament Problemby Packing Three-Vertex Paths. Proceedings of the AAAI Conference on Artificial Intelligence, 28(1).
5. Hoshino, R. and Kawarabayashi, K. Generating approximate solutions to the traveling tournament problem using a linear distance relaxation J. Artificial Intelligence. Res., 2012, 45, 257-286.
6. Jacobellis v. Ohio, 378 U.S. 184 (1964)
7. Kim,J.,Han,J. & Jeong,S. ,Solving the traveling tournament problem based on the simulated annealing and Tabusearch algorithm, Journal of Engineering and Applied Sciences,13(21):9204-9212.
8. Pathway2Code Homepage, https://www.pathway2code.com, last accessed 2024/1/21.
9. RobinX web site https://github.com/Robin-X/RobinX/blob/master/Repository/TravelOptimization/Instances/NFL32.xml.
10. Teachable Machine Homepage, https://teachablemachine.withgoogle.com/v1/, last accessed 2024/3/12.
11. Tuffaha, T., Cavdarogu, B. & Atan T, (2021), Timetabling round robin tournaments with the consideration of rest durations – PATAT 2022 – Volume II 2021 p. 389-397.
12. Van Bulck, D., Goosens, D. Sc Schönberger, J., & Guajardo, M. (2020). Robinx: A three-field classification and unified data format for round-robin sportstimetabling. European Journal of Operational Research, 280 (2), 568–580.
13. Westphal, Stephan and Noparlik, Karl ,Miyashiro R., Matsui T. & Imahori, S. An approximation algorithm for the traveling tournament problem Ann. Oper. Res., 2012, 194, 317-324.

Multi-period waste collection with preferred pickup days

Csaba Kebelei[1] [0009-0007-1132-638X] and Balázs Dávid[2,3] [0000-0003-4414-4797]

[1] Eötvös Loránd University
kebeleicsaba@student.elte.hu
[2] InnoRenew CoE, Izola, Slovenia
balazs.david@innorenew.eu
[3] University of Primorska, FAMNIT, Koper, Slovenia
balazs.david@famnit.upr.si

Abstract. This paper studies the multi-period vehicle routing problem with preferred pickup days in waste collection scenarios. The planning horizon includes a number of consecutive periods, with customers having preferred pickup days in each period. Deviation from these days is allowed, but comes at a penalty representing customer dissatisfaction. Additionally, a maximum duration between consecutive visits is imposed to manage waste accumulation.
We propose an adaptive large neighborhood search heuristic to solve this problem. The method assigns service days to customers and optimizes vehicle routes for each day of the planning horizon. The heuristic is validated on a real-world dataset, and different penalty scenarios are compared for the deviations from preferred days.

Keywords: Multi-period vehicle routing, Waste collection, Preferred days, Large neighborhood search .

1 Introduction

Regular periodical visits to customers are important in the case of many problems arising in real life, and waste collection is no exception. Accumulated waste has to be collected from customers and transported to drop-off points in given periods, preferably on days when it is the most suitable for the customers. However, deviating from these preferences can come with cost benefits in exchange for customer dissatisfaction. While constructing optimal routes for vehicles servicing customers belongs to the class of vehicle routing problems (VRP), the introduction of periodic decisions leads to a more general version of this problem.

Periodic variations of the VRP exist in the literature, most of which fall into the periodic vehicle routing (PVRP) problem class. PVRP is the generalization of the classical VRP, where routes are constructed over a time horizon T, and customers have a service frequency, pre-defining their possible visit patterns over the days of this period [3]. The goal of the PVRP is to assign customers to a visit pattern and prepare optimal vehicle routes for the days of the period based on these patterns. The PVRP with service choice [7] introduces extra complexity to this decision by making determination of the visit frequency part of the optimization as well. Multi-periodicity in vehicle routing is introduced usually combined with inventory routing decisions, where customers have to be visited in multiple periods (which are the individual days) [2,8]. Dynamic variations of the PVRP also exist, focusing on due dates of the deliveries and introducing penalties for lateness or cancelled visits [1,10].

This paper deals with the problem of multi-period vehicle routing with preferred pickup days that can arise in various real-world waste collection scenarios. The planning horizon of the problem contains P consecutive periods, each period consisting of D days. For every customer, a period pickup frequency is given to denote the periods in which they have to be visited, as well as a preferred day on which they are expecting the visit in a period. Visits to a customer are also allowed outside of their preferred days, however, these come with a penalty of deviating from the original schedule. Moreover, as the customers are accumulating waste over time, a maximum duration between two consecutive visits to the same customer also has to be introduced. The optimization questions in this case are twofold: first, a decision has to be made on choosing the visit days for each customer, then optimal vehicle routes have to be constructed for each day of the planning horizon. While the concept of preferred visiting days have been examined in the past [6], the deciding on the visiting days of each customer in each period over a longer time horizon with limits between two visits has not been studied to our knowledge. This paper will present our progress in an adaptive large neighborhood heuristic for the solution of the above problem. This method will consider both assigning the days of visit for customers in each period, as well as optimizing vehicle routes on each day based on the customers to be visited. The heuristic will be validated on inputs based on both real-world data and adapted benchmark datasets from the literature, and compared to the scenarios where no deviation is allowed from the preferred days of customers.

2 Problem definition

The problem introduced in Section 1 can be formalized as follows. Let us consider a planning horizon T, with length (in days) $|T|$, and period P with length $|P|$. T can be divided into $|T|/|P|$ periods. A specific i-th period of T is denoted by P_i, with starting day $1 + (i-1)|P|$ and ending day $i|P|$. The customers of the problem are given by set C. Each customer has to be serviced exactly once in each period. Every customer $c_i \in C$ has a preferred collection day p_i, where $1 \leq p_i \leq |P|$. Servicing a customer is allowed outside of their preferred day: the maximum allowed deviation (in days) is defined by Δ. Moreover, two consecutive visits to the same customer have to happen in at most M days. While the set of visit patterns could be defined for each customers over the entire planning horizon with the help of δ and M, this would yield a large amount of possibilities even for horizons with 3 or 4 periods.

If the service days are known for the customers, the route planning for each day can be defined as a capacitated vehicle routing problem with deliveries (to waste disposal sites) and route length constraints.

3 Solution method

An adaptive large neighborhood search algorithm (ALNS) was developed for the solution of the above problem. This algorithm closely follows the outline given by Ropke and Pisinger in [9].

In order to create an initial solution, we considered every customer to be serviced on their preferred day in each period. We solved the arising daily vehicle routing problems with two different methods.

The MILP model of Buhrkal et al. [4] was modified to accommodate all constraints of our problem. While this provides optimal vehicle routes with regards to the preferred days of the customers, the occasionally long running times for larger days made it an inefficient method to use in the ALNS. However, the results provided by this model were utilized for the evaluation of the heuristic.

A greedy heuristic was also developed for quick initial solution construction. The heuristic utilizes a 'best-fit' approach. Customers are ordered in ascending distance form the depot, and the current customer is scheduled to the vehicle with the least cost. Vehicle capacities and route lengths are managed by sending the vehicles to a disposal site if they reach a certain threshold of capacity/route length.

This initial solution was modified in each iteration using randomly selected destroy and repair methods.

The objective of the algorithm is to minimize the combined travel duration of vehicles and the deviation penalty from the preferred days. Each day of deviation by a customer service from its preferred day uniformly contributes a δ penalty to the costs.

4 Results

Input instances were generated based on real-world data from a waste collection company. Several instance sets of varying sizes were generated based on this data by randomly selecting a given number of customers. Each input instance considered a 28-day planning horizon, with four 7-day periods in the horizon. Datasets with 50, 100, and 150 customers per period have been created, resulting in 200, 400, and 600 customer visits over the horizon respectively. Ten different inputs were created in each instance set,

The original preferred days were used for each customer, and M (the maximum number of days between two consecutive visits) was set to 8. No parameter tuning has been performed yet on the ALNS: the original parameters from [9] have been used instead. The number of iterations was set to 25 000, which was the only terminating condition for the algorithm. Three scenarios were considered for the δ penalties for deviation: 0%, 5%, 10% or 15% of the average daily costs given by the optimal solution of the MILP model when servicing every customer on their preferred days.

A summary on our preliminary results can be seen in Table 1.

The three rows of the table give the average results of the 10 inputs for each instance set. The columns show the scenarios with different values of deviation penalty. The results in the table give the ratio of the ALNS solution costs compared to the optimal solution with preferred days. The expectation for the algorithm would be that if it can't find a better solution then the preferred day one, then it should find the optimum with

Table 1: Aggregated test results for all instance sets.

Instance set	δ =0%	δ =5%	δ =10%	δ =15%
200 visits	0.72	0.86	0.97	1.03
400 visits	0.76	1.03	1.12	1.09
600 visits	0.74	1.13	1.23	1.14

preferred days (which would provide the 1.0 ratio). The starting costs of the greedy initial solution were more than double that of the preferred day optimum.

It can be seen from the results that without any additional penalty for deviation, the solutions found by the ALNS were about 75% of the costs of the preferred day optimum for every instance set. The 200 visit instances also met the original expectation of finding a solution close to the 1.0 ratio if the penalties turn out to be too high. However, with an increase in the number of visits, this ratio became harder to reach for the ALNS. Our preliminary tests showed that the number of iterations is too low for these instances sizes, as the algorithm still found better results if left running. Moreover, we have not experimented with setting the parameters of the ALNS, which might also improve the quality of the results.

5 Conclusions and future work

This paper presented our preliminary results for solving a multi-period vehicle routing with preferred days for waste collection. An ALNS algorithm was developed for the problem, and initial test results were shown on real-world data. While the results are promising, there is still room for improvement. The ALNS itself can be improved both by finding the appropriate parameters for this problem class, and also by implementing additional destroy and repair algorithms. The scope of testing will also be increased by using benchmark datasets from the literature [5], but a method should be developed for transforming the visit patterns of these input to preferred days.

Acknowledgements The research was supported by the BioLOG project: the authors are grateful for the support of National Center of Science (NCN) through grant DEC-2020/39/I/HS4/03533, the Slovenian Research and Innovation Agency (ARIS) through grant N1-0223 and the Austrian Science Fund (FWF) through grant I 5443-N. The authors also gratefully acknowledge the Slovenian Research and Innovation Agency (ARIS) and the Ministry of Economy, Tourism and Sport (METS) for the grant V4-2351 from the CRP 2023 programme. The authors are grateful for the support of the Slovenian Research and Innovation Agency (ARIS) through grant J1-50000.

References

1. Angelelli, E., Bianchessi, N., Mansini, R., Speranza, M.G.: Short term strategies for a dynamic multi-period routing problem. Transportation Research Part C: Emerging Technologies **17**, 106–119 (2009). https://doi.org/10.1016/j.trc.2008.02.001

2. Archetti, C., Jabali, O., Speranza, M.G.: Multi-period vehicle routing problem with due dates. Computers and Operations Research **61**, 122–134 (9 2015). https://doi.org/10.1016/j.cor.2015.03.014
3. Beltrami, E.J., Bodin, L.D.: Networks and vehicle routing for municipal waste collection. Networks **4**, 65–94 (1 1974). https://doi.org/10.1002/net.3230040106
4. Buhrkal, K., Larsen, A., Ropke, S.: The waste collection vehicle routing problem with time windows in a city logistics context. Procedia - Social and Behavioral Sciences **39** (2012). https://doi.org/10.1016/j.sbspro.2012.03.105
5. Cordeau, J.F., Gendreau, M., Laporte, G.: A tabu search heuristic for periodic and multi-depot vehicle routing problems. Networks **30** (1997). https://doi.org/10.1002/(SICI)1097-0037(199709)30:2<105::AID-NET5>3.0.CO;2-G
6. Estrada-Moreno, A., Savelsbergh, M., Juan, A.A., Panadero, J.: Biased-randomized iterated local search for a multiperiod vehicle routing problem with price discounts for delivery flexibility. International Transactions in Operational Research **26** (2019). https://doi.org/10.1111/itor.12625
7. Francis, P., Smilowitz, K., Tzur, M.: The period vehicle routing problem with service choice. Source: Transportation Science **40**, 439–454 (2006). https://doi.org/10.1287/trsc.l050.0140
8. Mancini, S.: A real-life multi depot multi period vehicle routing problem with a heterogeneous fleet: Formulation and adaptive large neighborhood search based matheuristic. Transportation Research Part C: Emerging Technologies **70**, 100–112 (9 2016). https://doi.org/10.1016/j.trc.2015.06.016
9. Ropke, S., Pisinger, D.: An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation Science **40** (2006). https://doi.org/10.1287/trsc.1050.0135
10. Wen, M., Cordeau, J.F., Laporte, G., Larsen, J.: The dynamic multi-period vehicle routing problem. Computers and Operations Research **37**, 1615–1623 (9 2010). https://doi.org/10.1016/j.cor.2009.12.002

Flexible shift scheduling of healthcare workers using branch and price

David Ajit Kirpekar-Sauer[1] [0009−0001−3216−2157] and Jens O. Brunner[1] [0000−0002−2700−4795]

Technical University of Denmark, Anker Engelundsvej, 2800 Kongens Lyngby, Denmark

Abstract. We consider a strategic shift scheduling problem, consisting of assigning health care workers in a hospital to shifts for handling a given number of tasks spread out over a fixed time horizon. We formulate and present a branch-and-price algorithm with a novel network flow formulation for the subproblem.

Keywords: Branch-and-price, shift scheduling, shortest path

1 Branch-and-price

Most shift scheduling and rostering problems in literature are solved using decomposition algorithms, Mixed Integer Programs (MIPs), e.g. Brunner et. al. [3] or heuristics, e.g Van Huele and Vanhoucke [1], as presented by Van den Bergh et. al. [2]. We will look into an extension of decomposition of a MIP model.

We will implement a branch-and-price algorithm with the master problem assigning schedules to workers and the subproblem generating columns (shift schedules) to add to the master problem. The master problem will minimize the total cost of the chosen shift schedules and assign outside workers to handle understaffing. The novelty in our approach lies in the specific formulation of the network flow model for schedule generation. This is fairly well-known in literature, e.g. Akbarzadeh and Maenhout (2021) [5] use such an approach for scheduling medical students and . Instead of formulating the subproblem as a MIP, we will make use of a novel network formulation. We define a network first introduced [4] By traversing the network from the starting period to the end, a path, corresponding to a valid shift schedule, will be found. As such, the subproblem can be solved, and a maximum reduced cost schedule can be found by solving the shortest path problem for the graph. We will implement Dijkstra's algorithm with a labelling and readout step to find the shortest path through the network.

As branching strategies we wish to test two separate approaches. The first is approach is to branch on the most fractional masterproblem variable $\lambda_j = \lambda_j^*$, by setting upper and lower bounds in the left and right branch respectively. This is easily included by fixing the value in each iteration of the master problem. However the in the left branch to ensure that the schedule is not regenerated, we solve a k-shortest path problem and compare the generated schedules with all schedules in the current masterproblem. This is equivalent to introducing schedule elimination constraints in an IP version of the subproblem [6].

The second strategy is to branch on the most fractional set of working periods, by restricting the usage of all schedules using these working periods, a strategy used in

[4]. This can be implemented by a restriction in the masterproblem and by restricting this combination of working periods in the network, e.g. via removing weights and generating a set of shortest paths. This way all the generated schedules will adhere to the imposed restrictions. At each node of the branching tree, the lower bound is found via solving the LP relaxation of the MP to optimality. The upper bound is found via a simple rounding heuristic that fixes a single employee to a specific schedule and covers the remaining demand via external staff. This is used as the computational cost is low. A more precise upper bound is found by solving the MP as an IP problem, but due to the higher computational cost, this is only done at certain intervals.

Any under coverage resulting from these schedules will be handled by adding external staff in the MP.

We will look into the possibility of scheduling flexible and in-flexible workers to achieve a greater flexibility in the generated schedules. The fraction of flexible workers will be evaluated with varying proportions to ensure a feasible application of the method. The extended branch-and-price method will be applied and tested on a number of test instances generated from real-world data.

References

1. Van Huele C, Vanhoucke M (2015): Decomposition-based heuristics for the integrated physician rostering and surgery scheduling problem, Health Systems 4(3), 159-175.
2. Van den Bergh, J., Beliën, J., De Bruecker, P., Demeulemeester, E., De Boeck, L. (2013). Personnel scheduling: A literature review. European Journal of Operational Research, 226(1), 367-385.
3. Brunner J. O., Bard JF, Kolisch R (2010): Midterm scheduling of physicians with flexible shifts using branch and price, IIE Transactions 43(2), 84-109.
4. Brunner, J.O., 2015. Flexible personnel scheduling using branch and price. 8th International Congress on Industrial and Applied Mathematics, August 11, 2015, Beijing, China.
5. Akbarzadeh B., Maenhout B. (2021), An exact branch-and-price approach for the medical student scheduling problem, Computers & Operations Research, Volume 129, 105-209,
6. Brunner, J. O.; Bard, J. F.; Köhler, J. M. (2013), Bounded flexibility in days-on and days-off scheduling, Naval Research Logistics, Volume 6(8), pp. 678-701

A Hybrid Approach for the Artificial Teeth Scheduling Problem

Felix Winter and Nysret Musliu

Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling, DBAI, TU Wien, Favoritenstraße 9, 1040 Vienna, Austria
`{felix.winter,nysret.musliu}@tuwien.ac.at`

Keywords: Artificial Teeth Scheduling Problem, Hybrid Approach, Metaheuristics, Constraint Programming

1 Introduction

Modern-day manufacturing of artificial teeth relies on a highly automated production process that utilizes complex machine environments. Therefore, a large number of teeth products are manufactured daily to fulfill orders from customers all around the world. Due to these large-scale requirements, efficient automated production scheduling methods are required to minimize costs and consider all the constraints arising in the complex machine environment.

Previously, we introduced the artificial teeth scheduling problem (ATSP) originating from the industry in [4]. In addition to a formal specification of the problem, we provided an exact constraint-modeling approach to solve the problem. Further, we proposed a simulated annealing approach to tackle large-scale instances from the industry. An experimental evaluation on real-life instances showed that the exact approach can find optimal cost results for some small instances. However, the heuristic method was required to provide solutions for large instances within a reasonable time.

In [5], we further proposed a set of low-level heuristic operators that can be utilized with hyper-heuristic approaches. Experiments showed that hyper-heuristics could efficiently solve realistic instances and can improve results over the previous heuristic results in many cases. The existing heuristic approaches can produce high-quality solutions for practical instances. However, optimal solutions are still unknown for all large real-life instances.

This work proposes a novel hybrid approach for the ATSP that combines exact constraint-modeling techniques with a heuristic approach. In particular, the proposed technique decomposes the problem into two phases. In the first phase, optimized job patterns are determined for a given instance using an exact approach based on constraint-modeling. Afterwards, the optimized patterns are used to build an initial job sequence as a starting point for the second phase, where a heuristic further optimizes the solution.

Solving machine scheduling problems in two phases has been successfully applied in the past (e.g., [6,3]). However, existing decomposition methods cannot directly be applied to solve the ATSP, as they assume that jobs are part of the input.

2 A 2-Phase Approach for Artificial Teeth Scheduling

The ATSP can be viewed as a complex single-machine scheduling problem originating from industrial teeth manufacturing. However, in contrast to traditional machine scheduling problems, the jobs that need to be scheduled are not given as input to the problem. Instead, problem instances specify demands for various teeth products, and a part of the decision-making is to group demanded teeth products into jobs. Several complex constraints impose restrictions on how products can be grouped. Thus, in practical instances, it is often required to overproduce certain products to fulfill job capacity constraints.

The aim of the ATSP is to find schedules that minimize three objective criteria: Makespan, Waste, and Tardiness. While the first two objectives are minimized by finding efficient jobs that have a short duration and keep overproduction as low as possible, job tardiness is mainly influenced by the sequence of the jobs. For the publicly available real-life instances, all three objectives are weighted uniformly. A comprehensive problem specification of the ATSP can be found in [4].

The existing exact- and heuristic approaches solve the problem in a single phase, i.e., the problem is modeled to simultaneously consider decisions on creating the jobs and the scheduling aspect. In this work, we propose to decompose the problem in two phases. In the first phase, the method aims to create a set of jobs for all given product demands to minimize total job duration. Afterwards, the approach focuses on job scheduling in phase 2. Note that the 2-phase solution approach is incomplete, even if we reach optimal results in both phases, as the first phase entirely neglects the tardiness objective. However, as finding high-quality solutions for large-scale instances is challenging, the decomposition approach can potentially find improved upper bounds compared to existing techniques.

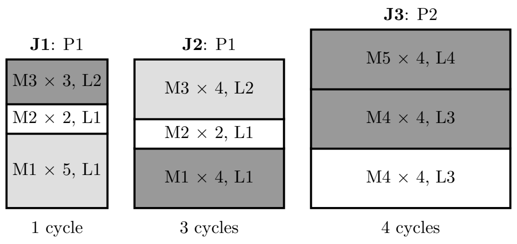
Figure 1 illustrates solutions to phases 1 and 2. In Figure 1a we can see three jobs (J1-J3), each including various different product configurations (Production molds M1-M5, different colors, production lines L1-L4, and production programs P1-P2). Each job in the example also uses a different number of production cycles which directly determine the job's length. The second phase schedules the jobs that were created in phase 1. In the example in Figure 1b, the jobs have been scheduled one after the other, although a different sequence would have been possible. All three jobs have precise start and ending time points ($t1 - t6$), and the arrows between the jobs visualize the setup time length between jobs.

3 Solution Method & Preliminary Experimental Results

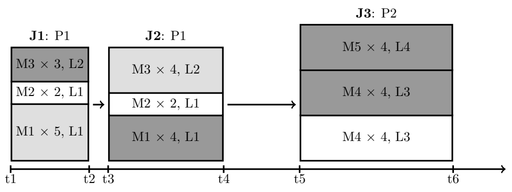
For the first phase, we have implemented a constraint model using the high-level modeling language MiniZinc [1], which can be used with constraint programming and mixed integer programming solvers as an exact approach. For the first set of preliminary experiments, we used the solver CP-SAT [2] in version 9.8.

To solve the second phase, we use a variant of the local search-based simulated annealing approach from [4], that only activates the job swap neighborhood. Thus, the heuristic focuses on optimizing the job sequence without modifying the jobs.

We conducted experiments with the proposed approach using the benchmark instances from [5]. Instances 1-6 include small scenarios, whereas instances 7-20 consist



(a) An example solution for the first phase consists of three jobs with different durations (i.e., production cycles).



(b) An example solution of the second phase consists of the three jobs created in phase 1 (See Figure 1a).

Fig. 1: Visual representation of example solutions for the 2-phase approach.

of large-scale real-life instances. The experimental environment was similar to the one used in [5] using a time limit of 1 hour. Note that we dedicated the most time to the exact solver and left the heuristic only the last 10 seconds of the time limit to find an optimized job sequence. We chose these time restrictions, as finding efficient jobs in phase 1 is particularly challenging. The exact technique could find optimal solutions only for a few small instances in this phase.

Table 1 gives an overview of the cost results achieved by existing methods and the proposed approach. Columns 2 (LB) & 3 (Exact) display the best lower bounds and upper bounds of the cost results achieved with exact techniques for all instances in [4]. Column 4 (SA) shows the best upper bounds achieved with the simulated annealing approach from [4], whereas Column 5 (HH) shows upper bounds achieved by the hyper-heuristic approach from [5]. Finally, Column 6 (Hybrid) displays the results of the hybrid approach proposed in this work (i.e., the final objective cost results after phase 2). Best cost results are formatted in boldface. A - indicates no solution was achieved within the time limit.

The results show that the proposed 2-phase hybrid approach cannot produce competitive results compared to existing techniques on small instances. In these cases, existing methods can likely reach solutions not explored by the decomposition approach. However, the novel approach produces improved results for most real-life instances, finding new upper bounds in 12 cases. These results indicate that the proposed hybrid approach can be a promising technique, especially for large-scale instances that lead to a vast search space for existing methods.

Instance	LB	Exact	SA	HH	Hybrid
Instance 1	2.08	**2.53**	**2.53**	**2.53**	2.54
Instance 2	1.25	1.96	1.96	**1.94**	2.04
Instance 3	2.23	**2.23**	**2.23**	**2.23**	2.24
Instance 4	2.54	**2.54**	**2.54**	**2.54**	**2.54**
Instance 5	1.63	**2.1**	2.13	2.12	2.20
Instance 6	3	**3**	**3**	**3**	**3**
Instance 7	0.5	-	2.95	2.85	**2.72**
Instance 8	0.15	-	2.38	2.47	**1.77**
Instance 9	0.59	-	2.99	2.85	**2.43**
Instance 10	0.53	-	2.67	2.66	**2.02**
Instance 11	0.34	-	2.76	2.78	**2.43**
Instance 12	1.02	-	2.97	2.91	**2.53**
Instance 13	0.6	-	2.97	2.85	**2.57**
Instance 14	0.46	-	2.99	2.84	**2.65**
Instance 15	0.56	-	2.99	2.81	**2.06**
Instance 16	0.37	-	2.98	**2.67**	2.76
Instance 17	0.2	-	2.94	**2.79**	2.89
Instance 18	0.39	-	2.98	2.72	**2.22**
Instance 19	0.18	-	2.94	2.7	**2.14**
Instance 20	0.18	-	2.98	2.78	**2.37**

Table 1: Cost results achieved by existing methods and the proposed approach.

Acknowledgements The financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development and the Christian Doppler Research Association is gratefully acknowledged.

References

1. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: Minizinc: Towards a standard CP modelling language. In: CP. Lecture Notes in Computer Science, vol. 4741, pp. 529–543. Springer (2007)
2. Perron, L., Didier, F.: Cp-sat, https://developers.google.com/optimization/cp/cp_solver/
3. Tang, T.Y., Beck, J.C.: CP and Hybrid Models for Two-Stage Batching and Scheduling. In: Integration of Constraint Programming, Artificial Intelligence, and Operations Research. pp. 431–446. Lecture Notes in Computer Science (2020)
4. Winter, F., Mrkvicka, C., Musliu, N., Preininger, J.: Automated Production Scheduling for Artificial Teeth Manufacturing. Proceedings of the International Conference on Automated Planning and Scheduling **31**, 500–508 (May 2021)
5. Winter, F., Musliu, N.: An investigation of hyper-heuristic approaches for teeth scheduling. In: MIC. Lecture Notes in Computer Science, vol. 13838, pp. 274–289. Springer (2022)
6. Zhao, Z., Liu, S., Zhou, M., Guo, X., Qi, L.: Decomposition Method for New Single-Machine Scheduling Problems From Steel Production Systems. IEEE Transactions on Automation Science and Engineering **17**(3), 1376–1387 (Jul 2020)

A Decision Support System Prototype for Automated Bus Driver Scheduling

Nikolaus Frohner[1], Esther Mugdan[1], Lucas Kletzander[1], and Nysret Musliu[1]

Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling, TU Wien, Vienna, Austria
{firstname.lastname}@tuwien.ac.at

Keywords: Bus driver scheduling, Multi objective optimization, Multi criteria decision making, Decision support systems

The bus driver scheduling problem (BDSP, [11]) deals in its essence with assigning bus tours' legs to drivers. Such an assignment induces a shift for each driver, consisting of categorized pieces of work like active driving time, manipulation time, passive ride time to change between tours, breaks and shift splits (see Figure 1). Feasible assignments must adhere to labor law and regulations imposed by collective agreements, such as a maximum drive time and a sufficient number of breaks depending on the duration and layout of a shift. For assignments to be implemented in practice, they must be economically viable for the bus operator. At the same time, employees need to be happy with their shifts to avoid absenteeism and unnecessary staff fluctuation.

In an ongoing project, we develop and study different features of a decision support system (DSS) prototype for automated BDS together with the personnel planning consultancy company XIMES. The concrete problem variant at hand was introduced by Kletzander et al. [4] featuring complex break constraints and seven different objectives to capture the main qualities of an assignment: the paid time t^{paid} (corresponding to the actual costs), the wasteful time paid up to the minimum shift length of six hours if they are shorter t^{mpaid}, the overall shift span including unpaid breaks t^{shift}, the number of employees n^{emp}, the number of shift splits n^{splits}, where we have a long unpaid break, the number of tour changes employees have to perform n^{change}, and the passive ride time t^{ride}.

So far, this BDSP variant has been studied with domain experts and a well-defined single weighted-sum objective with manually tuned weights in an expensive trial-and-error phase. The current state-of-the-art approaches are based on Branch and Price by Kletzander et al. [6] and a large neighbourhood search (LNS) by Mazzoli et al. [8]. Instead, we focus on finding assignments using a DSS without having to explicitly state any prior preferences by weights but by setting goals for different objectives. Furthermore, the DSS should help visualize a diverse set of solutions, facilitate learning about dependencies between objectives, and suggest concrete hints on how to adjust overly optimistic goals. To this end, we adopt and extend three different decision support methods:

– *Automated Weight Tuning (AWT)*: A recent automated approach introduced by Böðvarsdóttir et al. [2] and extended by Kletzander et al. [5] which performs intertwined

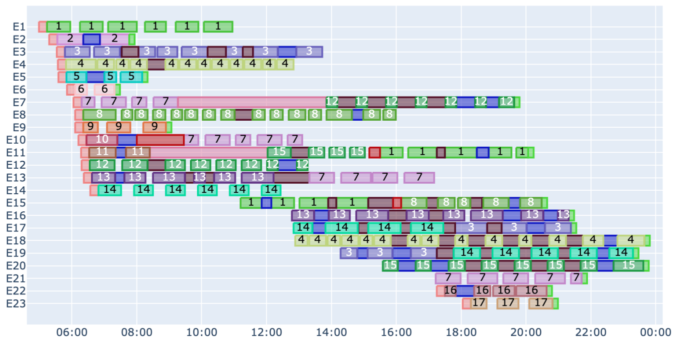


Fig. 1: Example solution to BDS instance with 23 employees and 17 tours. Active driving times are tour-numbered blocks, brown/blue blocks are unpaid/paid breaks, red blocks are passive ride time for a tour change, and long pink blocks are unpaid shift splits.

violation-dependent weight updates and optimization runs until an acceptable solution is found or too-conflicting goals are identified and reported back to the decision maker (DM). Acceptability is defined by a feasible solution that meets current thresholds on objectives, which are updated interactively until the DM is satisfied.

– *Reference eXplainable Interactive Multiobjective Optimization (R-XIMO)*: A reference point based method by Misitano et al. [10]. First, the Pareto front is approximated to quickly retrieve solutions by a scalarization function, taking a reference point as input. SHAP values [7] are then used to quantify the contribution of different input dimensions to the output and are converted into a hint on how to update the reference point to improve a selected objective.
– *R-XIMO with Shapley regression values*: Mischek and Musliu [9] extend the R-XIMO approach by directly using Shapley regression values instead of SHAP values using a black box predictor which permits missing inputs. Their experiments use weights as a preference structure to retrieve a solution from the Pareto front. Then, Shapley values identify a rival of a desired target that the DM wishes to improve. This leads to improvements more frequently for a test laboratory scheduling problem than updating only the target's weight.

First results. AWT consists of an exploratory phase taking time t^{exp} of intertwined shorter optimization runs, in our case using simulated annealing (SA) and weight updates depending on the current violations, either hard or soft. A final longer SA run of duration t is performed using the weights of the best-found solution during exploration. Table 1 shows the result of eight AWT runs over 50 BDS instances from [4]. BDS-1 is the problem variant without thresholds on the objectives starting from hard and soft constraint weights all-equal-1 $\mathbf{1}$ and $k = 3$ exploration threads. The acceptance rate, how often an acceptable solution was found, is denoted by r^{acp}.

The values of the objectives are normalized instance-wise using gross leg times L (including bus idle times), either by the total sum or in units of 8 h blocks. In BDS-1, we

Table 1: Interactive AWT for BDS instance-average results with acceptance rate r^{acp} using k threads and uniform ($\mathbf{1}$) or learned ($\boldsymbol{\mu}_{h,s}$) initial weights $\boldsymbol{w}_0$ within n^{it} iterations and (exploration) runtimes (t^{exp}) t in minutes. Further solution metrics minimum paid, work time, span, number of employees, shift split frequency, tour changes, and ride time.

pid	*thresholds*	k	$\boldsymbol{w}_0$	$r^{\text{acp}}[\%]$	$\overline{n^{\text{it}}}$	$\overline{t^{\text{exp}}}[']$	$\bar{t}[']$	$\overline{t_{\text{L}}^{\text{mpaid}}}$	$\overline{t_{\text{L}}^{\text{work}}}$	$\overline{t_{\text{L}}^{\text{paid}}}$	$\overline{t_{\text{L}}^{\text{shift}}}$	$\overline{n_{\text{L}_{8\text{h}}}^{\text{emp}}}$	$\overline{T_{\text{L}_{8\text{h}}}^{\text{splits}}}$	$\overline{n_{\text{L}_{8\text{h}}}^{\text{change}}}$	$\overline{t_{\text{L}_{8\text{h}}}^{\text{ride}}}$
BDS-1		3	$\mathbf{1}$	100	3.1	3.4	13.5	0.44	0.97	1.41	**1.02**	1.68	**940.3**	**0.29**	**0.56**
BDS-2	$t_{h^{\text{emp}}}$	3	$\mathbf{1}$	98	8.4	9.2	19.2	0.16	0.96	1.13	1.13	1.34	10.4	2.69	12.07
BDS-3	$+t_{s^{\text{change}}}$	3	$\mathbf{1}$	86	9.2	10.0	20.0	0.23	0.98	1.20	1.11	1.38	18.2	1.03	11.07
			$\boldsymbol{\mu}_h$	100	3.8	4.2	14.2	0.21	0.96	1.17	1.08	1.38	35.9	0.90	7.13
BDS-4	$+t_{s^{\text{mpaid}}}$	3	$\boldsymbol{\mu}_h$	84	9.5	10.2	20.3	**0.02**	1.11	1.13	1.47	**1.27**	2.9	1.32	52.77
BDS-5	$+t_{s^{\text{span}}}, t_{s^{\text{splits}}}$	1	$\boldsymbol{\mu}_h$	90	7.3	7.3	17.4	0.17	**0.96**	**1.13**	1.09	1.32	23.3	1.05	9.88
		3	$\boldsymbol{\mu}_h$	98	4.4	4.7	14.8	0.17	0.97	1.14	1.09	1.33	27.9	0.99	10.85
			$\boldsymbol{\mu}_{h,s}$	98	**2.4**	**2.6**	**12.7**	0.16	0.97	1.13	1.09	1.33	30.2	1.05	10.76

see that 1.68 employees are used on average to serve a gross 8 h block, with 0.29 tour changes per 8 h block and very rare shifts splits (every $T^{\text{splits}} = 940$th block on average). There are way too many employees with too short shifts. Therefore, for BDS-2, the DM sets a threshold of 1.35 on the number of employees per 8 h block. This leads to a desired massive reduction in costs per gross leg time (from 1.41 down to 1.13) at the inconvenience of quite frequent shift splits (every 10th block) and tour changes (2.7 per block). Hence, the DM introduces another threshold in BDS-3 to reduce the tour changes, which mildly increases costs and number of employees. Acceptable solutions are found slightly less frequently, in 86% of cases. This process is continued by adding two more thresholds until the DM is satisfied. The initial hard and soft weights are either all-equal-1 or taking (hard or hard/soft) centroid weights $\boldsymbol{\mu}$ derived from AWT runs on separate training data set to speed up the search of newly occurring online instances. The impact of using parallel weight updates ($k = 1$ vs $k = 3$) and such learned centroid weights is best seen for BDS-5, where the mean number of iterations goes down from

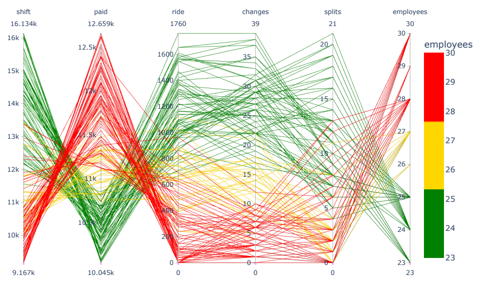


Fig. 2: Parallel coordinate plot of non-dominated solutions for a BDS instance.

7.3 to 2.4, also increasing the acceptance rate to almost 100%, to achieve the goals set by the DM.

The other DSS features require an initial approximation of the Pareto front for a given instance. A parallel coordinates plot [1] of non-dominated solutions created by a first mutation-only evolutionary algorithm with NSGA-II selection rule shows first promising results in Figure 2. The tradeoff between paid time/number of employees and the shift ergonomy aspects are visible. Current work deals with a comparison with Pareto Simulated Annealing (PSA) [3] and the implementation of other DSS approaches [10,9] with Shapley value based guidance to update preference structures like reference points and weights.

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.

References

1. Bagajewicz, M., Cabrera, E.: Pareto optimal solutions visualization techniques for multiobjective design and upgrade of instrumentation networks. Industrial & engineering chemistry research **42**(21), 5195–5203 (2003)
2. Böðvarsdóttir, E.B., Smet, P., Berghe, G.V.: Behind-the-scenes weight tuning for applied nurse rostering. Operations Research for Health Care **26**, 100265 (2020)
3. Czyzżak, P., Jaszkiewicz, A.: Pareto simulated annealing—a metaheuristic technique for multiple-objective combinatorial optimization. Journal of multi-criteria decision analysis **7**(1), 34–47 (1998)
4. Kletzander, L., Musliu, N.: Solving large real-life bus driver scheduling problems with complex break constraints. In: Proceedings of the International Conference on Automated Planning and Scheduling. vol. 30, pp. 421–429 (2020)
5. Kletzander, L., Musliu, N.: Dynamic weight setting for personnel scheduling with many objectives. Proceedings of the International Conference on Automated Planning and Scheduling **33**(1), 509–517 (2023)
6. Kletzander, L., Musliu, N., Hentenryck, P.V.: Branch and price for bus driver scheduling with complex break constraints. In: Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Virtual Event, February 2-9, 2021. pp. 11853–11861. AAAI Press (2021)
7. Lundberg, S.M., Lee, S.I.: A unified approach to interpreting model predictions. Advances in neural information processing systems **30** (2017)
8. Mazzoli, T.M., Kletzander, L., Van Hentenryck, P., Musliu, N.: Investigating large neighbourhood search for bus driver scheduling. In: Proceedings of the Internat. Conference on Automated Planning and Scheduling. vol. 34, pp. 360–368 (2024)
9. Mischek, F., Musliu, N.: Preference explanation and decision support for multi-objective real-world test laboratory scheduling. In: Proceedings of the International Conference on Automated Planning and Scheduling. vol. 34, pp. 378–386 (2024)
10. Misitano, G., Afsar, B., Lárraga, G., Miettinen, K.: Towards explainable interactive multiobjective optimization: R-XIMO. Autonomous Agents and Multi-Agent Systems **36**(2), 43 (2022)
11. Wren, A., Rousseau, J.M.: Bus driver scheduling—an overview. In: Computer-Aided Transit Scheduling: Proceedings of the Sixth International Workshop on Computer-Aided Scheduling of Public Transport. pp. 173–187. Springer (1995)

Ambulance routing for inter-hospital patient transfers in Sri Lanka

Sudheeraka Wickramarachchi, Kazuki Hasegawa, and Wei Wu

Graduate School of Integrated Science and Technology, Shizuoka University, Japan
sudheeraka.22@shizuoka.ac.jp

Keywords: Inter-hospital ambulance routing problem, Minimum risk time, Machine learning, Vehicle routing problem with time windows.

1 Introduction

The ambulance routing problem (ARP) involves finding optimal routes for ambulances to reach emergency sites, taking into account variables like traffic congestion, geographical limitations, and urgency of patient needs [2]. Essentially, the effectiveness of emergency medical transport system (EMTS) relies on the efficiency in defining and solving the ARP. Most existing studies investigated the scenario of reaching an emergency site and transporting a patient to an appropriate hospital. However, to the best of our knowledge, there are no studies that focus on designing and analyzing inter-hospital ARPs. In this study, we define an unconventional ARP in the optimization of inter-hospital transport, taking into account the particularities of the EMTS in Sri Lanka. This ARP incorporates a range of characteristics such as assignment, scheduling, and routing. To obtain good solutions for this new ARP, we propose a mathematical programming model and three two-phase heuristic approaches. Two of the heuristic approaches use machine learning techniques in the first phase.

2 Inter-hospital ambulance routing problem in Sri Lanka

The public health service of Sri Lanka is one of the major services provided free of charge to the public. Based on to the facilities that can be provided, the hospitals in Sri Lanka are categorized into 6 levels: National Hospitals (Level 1), Teaching Hospitals (Level 2), Provincial General Hospitals (Level 3), Base Hospitals (Level 4), Divisional Hospitals (Level 5), and Primary Medical Care Units (Level 6). In ascending order of levels (Level 1 to 6), space facilities, personnel with expertise availability, intensive care and surgery facilities are limited. According to the recommendations of medical experts, some patients should be transferred to other hospitals that are better facilitated, depending on the current hospital and their conditions.

In this study, we consider Colombo district, which has the highest population density of Sri Lanka. Transferring a patient may be an ‘immediately incurred transfer’ (IIT) or a ‘scheduled transfer’ (ST). An IIT can be clarified as a random occurrence such as an accident patient. In each type of transferring scenario, medical experts recommend not to exceed the minimum risk time (MRT). In this case study, we focus on the ST case, and identify the MRT as a highly significant factor, which can be expressed in terms of a due time for each patient. We follow the ST case to conduct our analysis, and try to obtain a good mathematical solution for the routing system.

3 Problem formulation

In this section, we first describe the ARP arising in Sri Lanka, and then propose a mixed integer programming (MIP) model which can well address the objective of minimizing the maximum tardiness that can occur in a patient transfer system.

In our ARP, we are given a set of patients $V_s = \{1, 2, \ldots, m\}$, a set of available beds $V_d = \{m+1, m+2, \ldots, m+n\}$, and a set of ambulances $K = \{1, 2, \ldots, k_{\max}\}$. The ambulance depot is denoted by 0. Each bed (demand) is associated with its level l_j (i.e., the bed is in a level-l_j hospital). Each patient (supply) i is required to be transferred to a bed (in a hospital) with a level of l_i or less, before the MRT d_i of patient i. A penalty weight w_i occurs, if the transfer request of patient i is ignored (not scheduled). The maximum limit of patient ignorance for the system is denoted by W. The value c_{ij} indicates the traveling time when transferring patient i to bed j. In this study, we consider all ambulances are homogeneous in traveling time. The inter-hospital ambulance routing problem (IH-ARP) aims to generate an ambulance routing solution so as to minimize the maximum tardiness among transferred patients.

From the problem input, we consider to generate a directed graph $G = (V, E)$ with $V = \{0\} \cup V_s \cup V_d$ and $E = E_{sd} \cup E_{ds} \cup \{(0, j) \mid j \in V_s\} \cup \{(i, 0) \mid i \in V_d\} \cup \{(0, 0)\}$, where $E_{sd} = \{(i, j) \mid i \in V_s, j \in V_d, l_j \leq l_i\}$ and $E_{ds} = \{(i, j) \mid i \in V_d, j \in V_s\}$. We use c_{sum} to denote the value $c_{\text{sum}} = \sum_{(i,j)\in E} c_{ij}$. Next, we introduce the variables used in the proposed model. The variable x_{ijk} indicates whether the edge (i, j) is visited by the ambulance k. We use p_i to denote the departure time of patient i, and p_j to denote the arrival time at bed j, and $T_{\max}$ to denote the maximum tardiness. By using these variables, the IH-ARP can be modeled as,

$$\min \quad T_{\max} \tag{1}$$

$$\text{s.t.} \quad \sum_{i\in V} x_{jik} = \sum_{i\in V} x_{ijk} \qquad \forall j \in V, \forall k \in K \tag{2}$$

$$\sum_{k\in K} \sum_{j\in V_d} x_{ijk} \leq 1 \qquad \forall i \in V_s \tag{3}$$

$$\sum_{k\in K} \sum_{i\in V_s} x_{ijk} \leq 1 \qquad \forall j \in V_d \tag{4}$$

$$\sum_{i\in V_d \cup \{0\}} x_{0ik} = 1 \qquad \forall k \in K \tag{5}$$

$$p_0 = 0 \tag{6}$$

$$\sum_{i\in V_s} w_i \left(1 - \sum_{k\in K} \sum_{j\in V_d} x_{ijk}\right) \leq W \tag{7}$$

$$p_j \geq p_i + c_{ij} - (1 - x_{ijk}) c_{\text{sum}} \qquad \forall (i, j) \in E_{ds} \cup E_{sd}, \forall k \in K \tag{8}$$

$$T_{\max} \geq \sum_{j\in D}\sum_{k\in K} c_{ij}x_{ijk} - d_i - c_{\text{sum}}\left(1 - \sum_{k\in K}\sum_{j\in V_{\text{d}}} x_{ijk}\right) \qquad \forall i \in V_{\text{s}} \tag{9}$$

$$p_i \geq 0 \qquad \forall i \in V \tag{10}$$

$$T_{\max} \geq 0 \tag{11}$$

$$x_{ijk} \in \{0,1\} \qquad \forall (i,j) \in E, \forall k \in K. \tag{12}$$

Constraints (2) confirm the flow conservation. Constraints (3) ensure that each patient can be transferred to at most one bed. Constraints (4) show similar relations for each bed. Constraint (5) states that each ambulance departs from the depot. The starting time is defined by constraints (6). The maximum limit of patient ignorance is denoted by the constraints (7). Constraints (8) to (9) define linkage between arrival time of each visited node and status of the MRT value.

4 Heuristic approaches

We design three heuristic approaches, all of which consist of two phases.

Approach based on local search (αLS): In the first phase, we assign patients to convenient beds depending on an assignment cost $c'_{ij} = c_{ij} + \alpha d_i$ that combines the traveling time c_{ij} and MRT d_i of each patient i, where α is an algorithm parameter determined by trial and error. In the resulting assignment problem, we may ignore some patients because of the limited number of beds. This supply-demand unbalance can be resolved by introducing dummy nodes.

We consider the output (patient-bed assignment) of the first phase as task nodes to construct a graph for the second phase. Each task node v_{ij} represents a transfer of patient i to bed j. We consider a directed complete graph consisting of all task nodes and a special node v_{00} representing the depot. The problem in the second phase can be identified as a vehicle routing problem with time windows (VRP-TW) depending on the MRTs of patients.

For solving this VRP-TW, we first construct an initial solution by utilizing a nearest neighbor method with time window restrictions. Thereafter, to obtain an improved solution, we propose a local search (LS) algorithm using generalized OR-opt operations extended from the classical OR-opt operations [1].

Approach based on ML incorporated with MIP (ML-MIP): In the second approach, we use ML methods for patient selection. After the first phase, we can determine which patients must to be transferred (without being ignored) in the system. Then, the route construction is performed using a simplified version of the MIP with a reduced feasible region of the original model (1)–(11).

Approach combining machine learning and local search (ML-LS): This approach improve the first phase of the αLS approach. In the first phase, we utilize ML to obtain multi-class (OneVsRest) predictions, that is, probability β_{ih} of assigning patient

i to hospital h (including a dummy hospital to denote the ignorance). Then, we set $c''_{ij} = 1 - \beta_{ih}$ if bed j is in hospital h, and use this modified cost c''_{ij} for the assignment problem in the first phase.

5 Computational results

We performed computational experiments on 5 instances from real-world data for the proposed model (MIP) in Section 3, and three approaches (αLS, ML-MIP, ML-LS) in Section 4. When machine learning is involved, we employed logistic regression (LR), gradient boosting (GB), random forest (RF), artificial neural network (ANN). Preliminary experiments showed that GB and LR outperformed the others in ML-MIP and ML-LS approaches, respectively. Thus, we report the results with their best machine learning models. The time limit for each instance was set to 300 seconds. The parameter α used in αLS is set to $\alpha = 5$.

Table 1 shows the running times in seconds (time) and objective function values (obj), where notation '—' indicates that no feasible solution was obtained in the time limit. From Table 1, we observe that MIP and ML-MIP showed similar performance in terms of both running time and objective function value. The αLS and ML-LS approaches obtained good solutions for large-scale instances within a small time compared to MIP and ML-MIP. Compared to ML-LS, for some small-scale instances, αLS failed to obtain an optimal solution within the time limit due to the poor assignment obtained in the first phase.

Table 1: Comparison of four approaches.

instance			MIP		ML-MIP		αLS		ML-LS	
m	n	$k_{\max}$	time	obj	time	obj	time	obj	time	obj
25	17	4	26.1	0	25.4	0	0.3	2	0.7	0
25	22	5	300.0	33	300.0	43	0.4	0	0.5	0
45	42	9	37.6	0	13.1	0	0.9	9	0.8	0
80	77	16	—	—	—	—	2.0	0	1.9	5
96	94	22	—	—	—	—	2.7	0	2.8	0

References

1. G Babin, S.D., Laporte, G.: Improvements to the or-opt heuristic for the symmetric travelling salesman problem. Journal of the Operational Research Society **58**(3), 402–407 (2007)
2. Tlili, T., Harzi, M., Krichen, S.: Swarm-based approach for solving the ambulance routing problem. Procedia Computer Science **112**, 350–357 (2017)

A Dantzig-Wolfe Reformulation for Automated Aircraft Arrival Scheduling in TMAs

Roghayeh Hajizadeh[1] [0000−0002−7402−6292], Tatiana Polishchuk[2] [0000−0002−9869−5992], Elina Rönnberg[1] [0000−0002−2081−2888], and Christiane Schmidt[2] [0000−0003−2548−5756]

[1] Department of Mathematics (MAI), Linköping University, Sweden
[2] Department of Science and Technology (ITN), Linköping University, Sweden
`firstname.lastname@liu.se`

Keywords: Dantzig-Wolfe Decomposition, Aircraft Arrival Routes, Automated Aircraft Separation

1 Introduction

Air traffic volumes have increased for decades, and though there was a significant drop because of the Covid-19 pandemic, IATA [4] projects that the demand for air travel will double by 2040, with an average annual growth rate of 4.3%. These high air traffic volumes result in an elevated environmental impact and significantly increased complexity for air traffic controllers (ATCOs). Both play a particular large role in Terminal Maneuvering Areas (TMAs)—the airspace around one or several aerodromes—where all air traffic merges and which is, hence, particularly impacted by both congestion and noise. To be able to handle the ever increasing volumes, it is crucial to alleviate the environmental impact and the ATCO workload by providing improved arrival and departure procedures, which still enable a high runway throughput.

An approach to lower the environmental impact are so-called continuous descent operations (CDOs), optimal engine-idle descents, which can reduce fuel burn, gaseous emissions, noise and fuel costs [3]. CDOs are optimal for the specific aircraft capabilities. Thus, different aircraft have different optimal trajectories. These do not fit together with the strategical standard terminal arrival routes (STARs) and they do decrease vertical and temporal predictability—a situation to which ATCOs answer with increasing separation buffers, which negatively effects throughput, or with issueing instructions that alter the optimal trajectories, which negatively impacts the environmental benefits. To be able to apply CDOs, tools that supply automated separation to ATCOs are needed.

For example, Choi et al. [2] presented a genetic-algorithm approach to compute aircraft arrival routes and the arrival sequence: they first developed distinct route topologies and then evaluated those with the heuristic-based scheduler.

In a series of papers, a group of authors [1,8,6,5,7] presented a MIP model to design optimal aircraft arrival routes with fully automated scheduling of CDOs with guaranteed aircraft separation and an operational concept that allows the usage of these routes. All aircraft fly according to their optimal neutral CDO speed profiles, where an aircraft's arrival to the TMA entry point can be adapted within a time window. In the model, the correct speed profile is picked by the length of the arrival route from entry point to runway. Moreover, the progress of aircraft along routes within the given grid graph are tracked. While this framework showed the feasiblity of the approach, it suffers from long runtimes: generating the arrival trees for a one-hour scenario took between 1.58h for low-traffic cases to 40.9h for high-traffic cases (with light aircraft added to the flow). This is of course not feasible for the real-world application, where new arrival routes should be computed regularly (ca. every 30 minutes, within the time frame that an aircraft spends in the TMA), and the framework certainly could not handle adding on more features, in particular, the influence of wind (the speed profile does not only depend on the length of the descent, but also on the wind direction in relation to the aircraft's trajectory). Hence, in this paper, we provide a Dantzig-Wolfe refomulation of a simplified model of MIP model in [7] (with fixed entry times and one separation time independent of wake categories) and show that this can yield significantly decreased runtimes—a promising approach for the full model and to handle even more practical aspects like wind in the future.

For the given location of TMA entry points and the runway for an airport, and a set of aircraft planned with fixed arrival time to their entry point, we aim to compute dynamic arrival trees for which

1. No more than two routes merge at a point (merge points require ATCO attention, we aim for the lowest possible complexity).
2. Merge points are separated by a minimum distance (otherwise many merge points could be located within an arbitrarily small area).
3. Routes do not make sharp turns (infeasible by aircraft dynamics).
4. Obstacles, like no-fly zones, are avoided.
5. All aircraft are temporally separated along the arrival route.
6. All aircraft follow CDO speed profiles (dependent on the arrival-route length).

2 Model

We discretize the TMA by creating a square grid with an edge length equal to the lower bound on separation and snapping the locations of both entry points and the runway to the grid. This leads to a bi-directed graph $G = (V, E)$ with nodes N and edges E where each grid node is connected to its 8 neighbors and for any two neighboring nodes i and j, both edges (i, j) and (j, i) exist in E. Let l_{ij} denote the length of edge $(i, j) \in E$, $\mathcal{P}$ the set of entry points, r the runway, $\mathcal{A}_b$ the set of all aircraft arriving at entry point $b \in \mathcal{P}$, $\mathcal{A} = \bigcup_{b\in\mathcal{P}} \mathcal{A}_b$ the set of all aircraft, and $|\mathcal{A}_b|$ the number of aircraft entering entry point b. In addition, $\bar{t}_a$ denotes the planned arrival time of aircraft a to its entry point and $T = \{0, \dots, \overline{T}\}$ is the considered time interval.

As part of making a Dantzig-Wolfe refomulation, all possible paths from each entry point to the runway are generated beforehand. This is done with an upper bound on the length of feasible paths and with respect to forbidden sharp turns and obstacles avoidance,. Let Π_b denote the set of paths from entry point b to the runway and $\mathbf{\Pi} = \bigcup_{b\in\mathcal{P}} \Pi_b$ be the set of all paths. For any $\pi \in \mathbf{\Pi}$, we define θ_π as the set of edges that path π passes through.

We introduce binary variables ρ_π for each path $\pi \in \mathbf{\Pi}$, indicating whether path π is used in the arrival tree and x_{ij} indicating whether the edge (i, j) participates in the arrival tree, our Dantzig-Wolfe-reformulation-based model is:

$$\min \ \beta \sum_{(i,j)\in E} l_{ij}x_{ij} + (1-\beta)\sum_{b\in\mathcal{P}}\sum_{\pi\in\Pi_b}\sum_{(i,j)\in\theta_\pi} |\mathcal{A}_b| l_{ij}\rho_\pi$$

$$\sum_{j:(j,i)\in E} x_{ji} \le 2 \qquad \forall i \in V \setminus \{\mathcal{P} \cup r\} \qquad (1)$$

$$\sum_{j:(i,j)\in E} x_{ij} \le 1 \qquad \forall i \in V \setminus \{\mathcal{P} \cup r\} \qquad (2)$$

$$\sum_{\pi\in\Pi_b} \rho_\pi = 1 \qquad \forall b \in \mathcal{P} \qquad (3)$$

$$\sum_{b\in\mathcal{P}}\sum_{a\in\mathcal{A}_b}\sum_{\pi\in\Upsilon_{ait}} \rho_\pi \le 1 \qquad \forall i \in V, \forall t \in \{0, \dots, \overline{T}-\sigma\} \qquad (4)$$

$$\sum_{\pi\in\mathbf{\Pi}:\ (i,j)\in\theta_\pi} \rho_\pi \le Qx_{ij} \qquad \forall (i,j) \in E \qquad (5)$$

$$\rho_\pi \le x_{ij} \qquad \forall \pi \in \mathbf{\Pi}, \forall (i,j) \in \theta_\pi \qquad (6)$$

$$x_{i,i+1+n} + x_{i+1+n,i} + x_{i+n,i+1} + x_{i+1,i+n} \le 1$$
$$\forall i \in V' \setminus \{\mathcal{P} \cup r\} : i+1+n, i+n, i+1 \notin \{\mathcal{P} \cup r\} \qquad (7)$$

$$x_{i,i+1+n} + x_{i+n,i+1} + x_{i+1,i+n} \le 1 \qquad \forall i \in \mathcal{P} \cap V' \qquad (8)$$

$$x_{i,i+1+n} + x_{i+1+n,i} + x_{i+1,i+n} \le 1 \qquad \forall i : i+1 \in \mathcal{P} \qquad (9)$$

$$x_{i,i+1+n,} + x_{i+n+1,i} + x_{i+n,i+1} \le 1 \qquad \forall i : i+n \in \mathcal{P} \qquad (10)$$

$$x_{i+1+n,i} + x_{i+n,i+1} + x_{i+1,i+n} \le 1 \qquad \forall i : i+n+1 \in \mathcal{P} \qquad (11)$$

The objective function is a convex combination of the length of the paths and the tree weight. Constraints (1) and (2) ensure that all the nodes except the entry points and the runway have an indegree of maximum 2 and an outdegree of maximum 1. Constraint (3) states that exactly one path from each entry point should be used. A minimum separation of σ time units between all aircraft at all nodes is given in Constraint (4) where Υ_{ait} is the set of all paths (starting from the corresponding entry point and passing node i) on which aircraft a with entry time $\bar{t}_a$ occupies node i between time t and $t + \sigma - 1$. Constraint (5) and/or Constraint (6) connect the variables where Q is a large number (the number of entry points in our case). Because we do not solely aim for shortest paths, we need to prohibit crossing. We could either take the crossings into account when generating the routes or we can prohibit them by Constraints (7)-(11) where $V' = V \setminus \{\text{last grid row}\} \setminus \{\text{last grid column}\}$.

3 Results and Conclusions

We used data for Stockholm-Arlanda-airport TMA with a 15×11 grid (which ensures a separation of 6NM) and solved our model using the Gurobi optimization solver with gurobipy as interface to Python on a MacBook Pro, M1 2020. We have made preliminary experiments where the running time of our reformulated model for a medium-traffic case (20 aircraft in one hour) with fixed arrival time for all aircraft including the generation of the paths was less than one minute.

Comparing our preliminary results to those given in [1,8,6], confirms that our model significantly outperformed in terms of computational efficiency. This gives the possibility for improved runtimes even for the case with flexible entry times and separation based on wake-turbulence categories ([5,7]). Moreover, this indicates that our framework may be capable of handling real-world operations and generating new arrival routes regularly on a personal computer.

Acknowledgements Supported by grant 2022-03178 (OPT@ATM: Optimization Methods for Large Air Traffic Management Problems) from the Swedish Research Council (Vetenskapsrådet). The computational experiments were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS), partially funded by the Swedish Research Council through grant agreement no. 2022-06725.

References

1. Andersson, T., Polishchuk, T., Polishchuk, V., Schmidt, C.: Automatic Design of Aircraft Arrival Routes with Limited Turning Angle. In: Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS) (2016)
2. Choi, S., Robinson, J.E., Mulfinger, D.G., Capozzi, B.J.: Design of an optimal route structure using heuristics-based stochastic schedulers. In: IEEE/AIAA 29th Digital Avionics Systems Conference (DASC) (Oct 2010)
3. Eurocontrol: Continuous climb and descent operations. https://www.eurocontrol.int/articles/continuous-climb-and-descent-operations (2019), accessed December 15, 2017
4. IATA: Global outlook for air transport: Highly resilient, less robust (2023)
5. Polishchuk, T., Polishchuk, V., Schmidt, C., Sáez, R., Prats, X., Hardell, H., Smetanová, L.: How to Achieve CDOs for All Aircraft: Automated Separation in TMAs - Enabling Flexible Entry Times and Accounting for Wake Turbulence Categories. In: 10th SESAR Innovation Days (SIDs) (2020)
6. Sáez, R., Prats, X., Polishchuk, T., Polishchuk, V., Schmidt, C.: Automation for Separation with CDOs: Dynamic Aircraft Arrival Routes. AIAA J. Air Transp. **28**(4), 144–154 (2020). https://doi.org/10.2514/1.D0176
7. Sáez, R., Polishchuk, T., Schmidt, C., Hardell, H., Smetanová, L., Polishchuk, V., Prats, X.: Automated sequencing and merging with dynamic aircraft arrival routes and speed management for continuous descent operations. Transportation Research Part C: Emerging Technologies **132**, 103402 (2021)
8. Sáez, R., Prats, X., Polishchuk, T., Polishchuk, V., Schmidt, C.: Automation for separation with CDO:s dynamic aircraft arrival routes. In: Thirteenth USA/Europe Air Traffic Management Research and Development Seminar (ATM2019) (2019)

Minimal and fair waiting times for single-day sports tournaments with multiple fields

Lisa Garcia Tercero[1,2] [0009-0005-9058-0508], Dries Goossens[1,3] [0000-0003-0224-3412], and David Van Bulck[1,3] [0000-0002-1222-4541]

[1] Faculty of Economics and Business Administration, Ghent University, Tweekerkenstraat 2, 9000 Ghent, Belgium
{lisa.garciatercero,dries.goossens,david.vanbulck}@ugent.be
[2] KU Leuven, Department of Computer Science, CODeS, Gebroeders De Smetstraat 1, 9000 Ghent, Belgium
[3] FlandersMake@UGent - core lab CVAMO

Keywords: Sports timetabling, Integer programming, Fairness

1 Introduction

In this extended abstract, we consider a sports tournament where all participants gather at one location for a day and a limited number of fields is available. We focus on a single round-robin tournament (i.e. all teams have to play each other exactly once), where each team needs a resting time of at least one time slot between every two games they play. This setting occurs regularly in practice; our work is particularly inspired by an amateur badminton tournament called "PK WVBF" in West-Flanders, Belgium. Sometimes having more resting time is perceived as desirable, yet in amateur tournament environments teams prefer to play their games in quick succession so they can return home without delay. Therefore, we generate timetables minimizing waiting times, defined for each team as the total number of time slots they have to be present in addition to their games and resting times. We focus on creating timetables that are both efficient (by minimizing the total waiting time) and fair (by minimizing the maximum waiting time).

Knust [2] considers a tournament which is similar except that it spans multiple days and only one field is available. Moreover, the tournament is divided into different blocks (or playing days) such that each team plays twice per block. Knust proposes an exact polynomial-time algorithm that simultaneously optimizes the total and maximum waiting times. We will generalize Knust's algorithm into a heuristic to generate timetables for a single-day tournament with multiple fields. Note that the single-day constraint complicates the timetabling efforts, since the blocks created by Knust's algorithm cannot simply be scheduled one after the other due to the required resting times.

For reviews on generating timetables for sports tournaments, we refer to the surveys [1], [3], and [4].

2 Problem formulation

In the IP model we consider the number of time slots and fields as a given. First, given a set of teams T, we assume the number of fields equals $\lceil(|T|-1)/4\rceil$. Considering only $\lceil(|T|-1)/2\rceil$ games can be played simultaneously and a team can only play one game per two time slots, increasing the number of fields would mainly increase the number of empty slots without offering much advantage in terms of our objectives. With this number of fields, preliminary experiments show that timetables can be created of length $2|T|$. Furthermore, we assume a day is always long enough to schedule all games. We assign a time slot to each game, such that (i) the number of fields is always respected, (ii) the required resting times are respected, and (iii) the total waiting time and the maximum waiting time are minimized.

Parameters

T	Set of teams
G	Set of games, constituting a single round-robin tournament
$G_t \subset G$	Set of games that team $t \in T$ has to play
S	Set of time slots, $|S| = 2|T|$
f	Number of fields, $f = \lceil(|T|-1)/4\rceil$

Decision variables

x_{gs}	1 if game $g \in G$ is scheduled at time slot $s \in S$, 0 otherwise.
w_t	waiting time of team $t \in T$

The model is formulated as follows, using a lexicographic bi-objective function:

$$\min \sum_{t \in T} w_t \tag{1}$$

$$\min \left(\max_{t \in T} w_t \right) \tag{2}$$

subject to

$$\sum_{s \in S} x_{gs} = 1 \qquad \forall g \in G \tag{3}$$

$$\sum_{g \in G} x_{gs} \le f \qquad \forall s \in S \tag{4}$$

$$\sum_{g \in G_t} x_{gs} + x_{g(s+1)} \le 1 \qquad \forall t \in T, s = 1, ..., |S|-1 \tag{5}$$

$$\sum_{s \in S} \left((s+1)x_{gs} - s x_{g's} \right) \le w_t + (2|T|-1) \qquad \forall t \in T, \forall g, g' \in G_t \tag{6}$$

$$x_{gs} \in \{0, 1\} \qquad \forall g \in G, s \in S \tag{7}$$

$$w_t \ge 0 \qquad \forall t \in T \tag{8}$$

The first objective minimizes the total waiting time, whereas the second minimizes the maximum waiting time across all teams. Constraints (3) and (4) ensure that every game is played exactly once and that games are restricted to the number of available fields.

Furthermore, Constraints (5) guarantee that the required resting times are respected. Next, the waiting time of each team is defined by Constraints (6); the games and resting times of each team require $2|T|-1$ time slots. Finally, Constraints (7) and (8) are variable domain constraints. Note that the integrality of w_t follows from Constraints (6).

3 Preliminary computational results

In this section, we show how to generalize Knust's algorithm [2] into a heuristic for a single-day tournament with multiple fields. The strategy is to generate blocks such that each team plays two games per block, and schedule them consecutively on a single day. In order to do this, we assume $|T|$ to be odd as this enables us to follow the structure of the timetables generated by Knust. The blocks are constructed by generating the games in the same order as the single-field variant and scheduling them at the earliest feasible time slot, resulting in a total waiting time of 2 per block. Intuitively, this follows from the fact that some games cannot be scheduled on their "ideal" time slot due to all fields already being occupied. We ensure resting times are respected by scheduling an empty slot between the blocks, after which games are moved forward individually. Details will be discussed during the talk.

Since the number of fields equals $\lceil (|T|-1)/4 \rceil$ and $|T|$ games are played per block, we need at least 4 slots per block; the heuristic creates blocks of length 5. Together with the empty slots after each block and the fact that $(|T|-1)/2$ blocks have to be planned, we have an upper bound on the number of time slots of $3(|T|-1)$.

We compare the running time of the heuristic to the IP model, solved with Gurobi version 10.0.03, in Table 1.

Table 1: Running times of the IP model and the heuristic for odd numbers of teams.

|T|	IP model	Heuristic
11	2m46s	0s
13	1m12s	12ms
15	8m38s	1ms
17	>30m	1ms
...	...	...
501		2.8s

Despite the IP model being too slow to create optimal schedules for a large number of teams, we suspect that for each odd $|T|$ a schedule exists such that:

$$\sum_{t \in T} w_t = 2|T| - 4 \tag{9}$$

$$\max_{t \in T} w_t = 2 \tag{10}$$

This conjecture is confirmed for the cases where $|T| \leq 15$. Assuming these results hold for every odd $|T|$, we compare them against our heuristic in Figure 1.

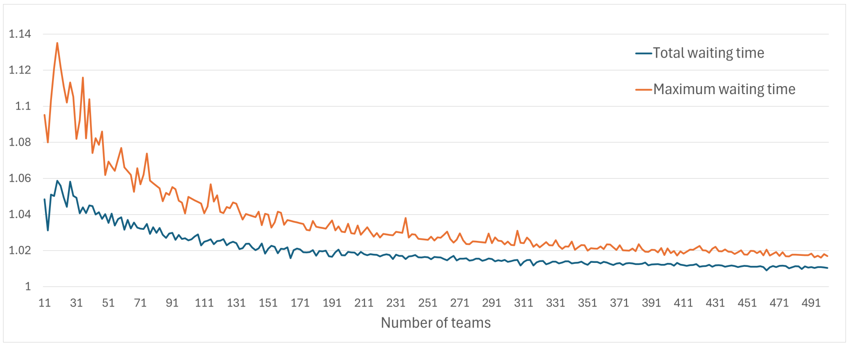


Fig. 1: Ratio of the results obtained by the heuristic to the conjectured optimal results, for odd numbers of teams $|T|$.

This extended abstract represents an initial step towards the general problem of a single-day round-robin tournament with any given number of available fields and any number of teams participating. For instances where the number of teams is even, we can easily add a dummy team. However, the impact of this on waiting times requires investigation. Furthermore, we plan to examine how the structure of the schedules should be adapted when fewer fields are available, while keeping the waiting times within limits.

References

1. Kendall, G., Knust, S., Ribeiro, C., Urrutia, S.: Scheduling in sports: An annotated bibliography. Computers & Operations Research **37**(1), 1–19 (2010)
2. Knust, S.: Scheduling sports tournaments on a single court minimizing waiting times. Operations Research Letters **36**(4), 471–476 (2008)
3. Ribeiro, C.: Sports scheduling: Problems and applications. International Transactions in Operational Research **19**(1-2), 201–226 (2012)
4. Van Bulck, D., Goossens, D., Schönberger, J., Guajardo, M.: RobinX: A three-field classification and unified data format for round-robin sports timetabling. European Journal of Operational Research **280**(2), 568–580 (2020)

Cohort-Based Timetabling with Integer Linear Programming

Richard Hoshino and Jameson Albers

Northeastern University, Vancouver BC, Canada
Khoury College of Computer Sciences
{r.hoshino, albers.j}@northeastern.edu

Abstract. At some educational institutions, students are divided into cohorts, where they complete the same set of courses with everybody else in that cohort. In this paper, we describe an Integer Linear Programming (ILP) solution to the School Timetabling Problem (STP), for schools that are cohort-based. Our Master Timetable is generated from the input data, a user-friendly Excel document that lists all of the course/cohort/teacher/classroom constraints.

Our model, which is coded in Python and solved using the MPSolver from Google OR-Tools, generated the 2023-2024 Master Timetable for three schools in Canada: an elementary school, a middle school, and a post-secondary institute. All three timetables satisfied 100% of the school's hard constraints, and were computed in less than 60 seconds.

Keywords: Integer Programming, Linear Programming, Timetabling

1 Introduction

Timetabling is defined as "the act of scheduling something to happen or do something at a particular time" [2]. This simple definition conceals the challenge and complexity of timetabling.

For educational institutions, the Master Timetable dictates to every single teacher and student where they need to be at each hour of the school day. Given its importance, some school administrators spend weeks or months constructing the annual Master Timetable, often using post-it notes or wall magnets to assign a teacher, classroom, and timeslot to each section of a course.

When timetables are constructed by hand, the process is inefficient and the product is sub-optimal. This is why researchers have investigated the School Timetabling Problem (STP) for sixty years [6], creating timetables for schools all over the world [11].

In the most basic version of the STP, the objective is to assign courses to teachers, timeslots, and classrooms, subject to the following constraints: a teacher cannot teach two courses in the same timeslot, no classroom can be used by two courses simultaneously, and each teacher has a set of unavailable teaching timeslots. This basic version of the STP is NP-complete [4].

Although scholars have conducted research on educational timetabling since the 1960s [1], it took until the mid-1990s to standardize educational timetabling problems, along with their corresponding benchmark data sets [3].

To advance the field of educational timetabling, a group of researchers have run the International Timetabling Competition, with the 2011 edition focused on high school timetabling [12] and the 2019 edition focused on university timetabling [10]. Both competitions were modeled on real-world data sets.

The Post-Enrollment Course Timetabling Problem (PECTP) was introduced to incorporate student course preferences into the STP [9]. The PECTP involves student-related hard constraints, such as ensuring that no student is enrolled in multiple sections of the same course, and the objective function is to maximize the number of occurrences where students are enrolled in their desired courses.

The lead author has created 40 Master Timetables for various Canadian high schools over the past five years, using his published algorithms to solve large real-life PECTP instances for schools: using graph coloring [7] and large neighborhood search [8]. While each Master Timetable was custom-built to meet the school's specific requirements, several of these timetabling projects were extremely similar in that the schools were *cohort-based*, where students were divided into fixed groups and took the same set of courses.

For cohort-based timetables, the PECTP reverts back to the STP, since student preferences do not exist. Thus, the optimal timetable can be generated by solving an Integer Linear Program (ILP) where the objective function considers teacher preferences for the timeslots they would like to teach their courses.

We now present our solution to solving virtually any cohort-based STP. Our automated timetabling algorithm requires a single input file: an Excel document that contains the teacher preferences as well as all of the constraints involving courses, cohorts, teachers, and rooms. We will explain how we worked with school administrators at a Kindergarten to Grade 5 elementary school, a Grade 6 to Grade 8 middle school, and a design academy for post-secondary students, to generate each institution's provably-optimal 2023-2024 Master Timetable.

Despite the different contexts of all three of these educational institutions, we used the *exact same Python program* to create all three timetables; the only difference was that each school had its own input Excel file. At the end of our Python program, we call MPSolver, the Mathematical Programming solver from Google OR-Tools [5] that solves Mixed Integer Programs (MIPs).

2 Mathematical Model

Each course c has one or more lessons (or meetings) in a week. Thus, we define our main binary decision variable as $X_{m,c,d,p}$, which equals 1 if and only if meeting m of course c is scheduled on day d in period p. Otherwise, $X_{m,c,d,p} = 0$.

We can view each (d, p) pair as a *timeslot*, and each (m, c) pair as a single *event* that is attended by one or more student cohorts, is taught by one or more teachers, and is offered in one or more rooms. Our ILP will generate the Master Timetable by assigning exactly one timeslot to each event.

Let D be the set of days, P be the set of periods, and C be the set of courses. For each course c, if there are $L(c)$ lessons (i.e., events) that must be scheduled during the $|D|$-day timetable, then we have the following constraint.

$$\sum_{d \in D} \sum_{p \in P} X_{m,c,d,p} = 1 \qquad \forall\, c \in C, m \in [1, L(c)] \tag{1}$$

Each course $c \in C$ is unique, where the teacher(s) and room(s) for course c are pre-assigned by the school.

For example, one of our school clients has 6A-Science, 6B-Science, 6C-Science in its set of courses C since each of the three Grade 6 cohorts has its own Science course that meets three times each week. Additionally, this school has all of its Grade 6 students taking Physical Education at the same time. This single course, 6-PhysEd, is offered to all three cohorts (6A, 6B, 6C), is co-taught by three teachers, and takes place in two rooms (Gym1, Gym2).

By specifying the cohorts/teachers/rooms for each event, once our ILP solver determines all four-tuples (m, c, d, p) for which $X_{m,c,d,p} = 1$, we can rapidly generate the various "cross-sections" of our Master Timetable to determine the timetable from the perspective of each course, each cohort, each room, and each teacher.

For each cohort, each teacher, and each room, we can determine E, the set of events (m, c) involving that entity, and ensure that there are no scheduling conflicts. Thus, we have our next set of hard constraints.

$$\sum_{(m,c) \in E(h)} X_{m,c,d,p} \leq 1 \ \forall\, d \in D, p \in P, h \in \text{Cohorts} \tag{2}$$

$$\sum_{(m,c) \in E(t)} X_{m,c,d,p} \leq 1 \ \forall\, d \in D, p \in P, t \in \text{Teachers} \tag{3}$$

$$\sum_{(m,c) \in E(r)} X_{m,c,d,p} \leq 1 \ \forall\, d \in D, p \in P, r \in \text{Rooms} \tag{4}$$

Finally, we define $PR_{m,c,d,p}$ to be the *preference* of having meeting m of course c scheduled on day d and period p. This preference coefficient may be influenced by a teacher's desire to teach on certain days and periods, or pedagogical reasons of having certain courses assigned to particular timeslots. Thus, the **objective function** of our ILP is

$$\sum_{m \in M} \sum_{c \in C} \sum_{d \in D} \sum_{p \in P} PR_{m,c,d,p} X_{m,c,d,p}.$$

There are two major families of constraints in our model, and we now explain each one in detail.

2.1 Family I: Restrictions on Sets of Events

Let E be a set of events and let T be a set of timeslots. Then we can connect E and T via the following linear constraint.

$$\sum_{(m,c)\in E} \sum_{(d,p)\in T} X_{m,c,d,p} \quad \{=,\leq,\geq\} \quad n \tag{5}$$

For each constraint, we choose the appropriate sign from $\{=,\leq,\geq\}$, and set n to be a specific non-negative integer. Let us provide several examples on the versatility of this family of constraints.

(i) "Teacher X is unavailable to teach on Tuesdays": E is the set of events taught by Teacher X, T is the set of timeslots with $d = 2$, our sign is =, and $n = 0$.
(ii) "The majority of the three 6A-French lessons must occur before lunch": E is the set of 6A-French events, T is the set of timeslots that occur before lunch, our sign is $\geq$, and $n = 2$.
(iii) "There is at most one Grade 8 Art class scheduled in Period 4": E is the set of events whose course is Grade 8 Art, T is the set of timeslots with $p = 4$, our sign is $\leq$, and $n = 1$.
(iv) "Ensure that Grade 7s do not have Physical Education more than once on any day": E is the set of events whose course is Grade 7 Physical Education, T is the set of timeslots with $d = k$, our sign is $\leq$, and $n = 1$. We repeat this constraint for each $k \in [1, |D|]$.
(v) "Ensure that Grade 7s do not have Physical Education on three consecutive days": E is the set of events whose course is Grade 7 Physical Education, T is the set of timeslots with $d \in [k, k+1, k+2]$, our sign is $\leq$, and $n = 2$. We repeat this constraint for each $k \in [1, |D| - 2]$.

This framework enables us to model constraints that relate almost *any* set of events to *any* set of timeslots, including the five examples provided above. We can place constraints on teacher and room availability, guarantee that certain events are scheduled (or not scheduled) in certain timeslots, spread out the multi-lesson courses taken by each cohort, and ensure that each teacher has a reasonable schedule each day without too many consecutive lessons or large gaps of non-teaching periods.

2.2 Family II: Relationships between Sets of Events

Let $E_i = (m_i, c_i)$ for each $i \geq 1$, and let $E_1, E_2, \ldots, E_v$ be a set of v events. Using a linear equation or linear inequality, we can model five additional timetabling constraints that relate these v events.

(i) All v events must occur in the same timeslot.

$$X_{m_i,c_i,d,p} = X_{m_{i+1},c_{i+1},d,p} \qquad \forall d \in D, p \in P, i \in [1, v-1] \tag{6}$$

(ii) All v events must occur on the same day.

$$\sum_{p\in P} X_{m_i,c_i,d,p} = \sum_{p\in P} X_{m_{i+1},c_{i+1},d,p} \tag{7}$$

$$\forall d \in D, i \in [1, v-1]$$

(iii) The v events must occur on v different days.

$$\sum_{i \in [1,v]} \sum_{p \in P} X_{m_i,c_i,d,p} \leq 1 \quad \forall d \in D \tag{8}$$

(iv) The v events must occur on v (different) consecutive days, with E_i occurring before E_j for all $i < j$.

$$\sum_{p \in P} X_{m_i,c_i,d_1,p} + \sum_{p \in P} X_{m_{i+1},c_{i+1},d_2,p} \leq 1 \tag{9}$$
$$\forall i \in [1, v-1] \text{ and } d_1, d_2 \in D \text{ with } d_2 - d_1 \neq 1.$$

A similar set of inequalities also allows us to ensure that the v events must occur in v consecutive periods of the same day, with E_i occurring before E_j for all $i < j$.

(v) There is a minimum gap of g days between events E_i and E_{i+1}, for all $1 \leq i \leq v-1$.

$$\sum_{p \in P} X_{m_i,c_i,d_1,p} + \sum_{p \in P} X_{m_{i+1},c_{i+1},d_2,p} \leq 1 \tag{10}$$
$$\forall i \in [1, v-1], \text{ and } d_1, d_2 \in D \text{ with } |d_2 - d_1| < g. \tag{11}$$

To get a maximum gap of g days, we simply replace $<$ with $>$ in the above inequality.

This versatile and flexible framework enables us to model constraints that relate almost any set of events to each other. For example, we can ensure that certain courses are not scheduled on the same day, that two cohorts have their French courses at the exact same time each week, and that a part-time teacher's work times are limited to two consecutive days in the timetable.

Real-life School Timetabling Problems (STPs) can be modeled effectively using the two families of constraints provided in this section. In fact, for all three of our cohort-based schools, we were able to model 100% of their constraints using these two families of constraints, and generate each school's Master Timetable in less than 60 seconds. We now explain how we accomplished these results.

3 Solving Three Different STPs

Victoria is the capital city of the Canadian province of British Columbia, and is the home of two leading independent co-educational K-12 preparatory schools named St. Michaels University School (SMUS) and Glenlyon Norfolk School (GNS). Victoria is also the location of Pacific Design Academy (PDA), an innovative post-secondary institute that offers eight full-time diploma programs including Fashion Design, Interior Design, and Graphic Media Design.

We were hired to create the Master Timetable for the SMUS Junior School (Kindergarten to Grade 5), the GNS Middle School (Grade 6 to Grade 8), and the entire PDA academic timetable with its eight different diploma programs.

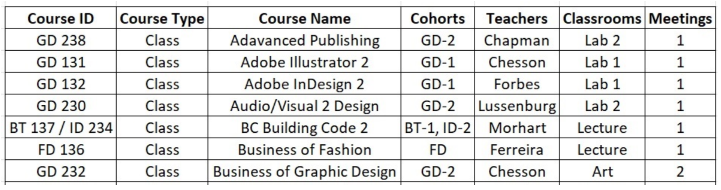
We worked closely with the administrators at the three institutions to create an Excel document that encoded all of the school's constraints and requirements, as well as teacher preferences, and would serve as the input file to our Python program. As mentioned earlier, we used the same Python program for all three timetables, which called Google's MPsolver to solve our Integer Linear Program. Each school's input Excel file consists of the following five worksheets.

1. Timetable Structure
2. Timetable Content
3. Event Set Constraints
4. Event Relationship Constraints
5. Teacher Preferences

The **Timetable Structure** worksheet contains $|P|$ rows and $|D|$ columns, indicating the names of each day (1-Monday, 2-Tuesday, . . .) and each period (Period 1, Period 2, . . .). Each of the $|D||P|$ timeslots is labeled d-p, for each day and period. For example, 2-4 is Tuesday Period 4.

Each of our three schools had a different number of timeslots, with SMUS having 5 days and 7 periods, GNS having 10 days and 5 periods, and PDA having 5 days and 3 periods.

The **Timetable Content** worksheet provides the complete set of events, containing the ID of each unique course and the number of total meetings for that course. For each course $c \in C$, this worksheet also lists the name of the course, and all of the affected cohorts, teachers, and classrooms. Figure 1 provides an excerpt of this worksheet for PDA.

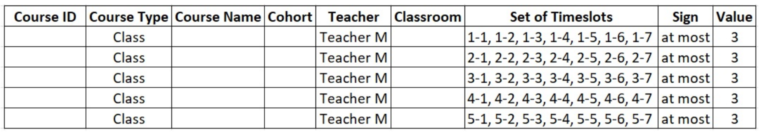


Course ID	Course Type	Course Name	Cohorts	Teachers	Classrooms	Meetings
GD 238	Class	Adavanced Publishing	GD-2	Chapman	Lab 2	1
GD 131	Class	Adobe Illustrator 2	GD-1	Chesson	Lab 1	1
GD 132	Class	Adobe InDesign 2	GD-1	Forbes	Lab 1	1
GD 230	Class	Audio/Visual 2 Design	GD-2	Lussenburg	Lab 2	1
BT 137 / ID 234	Class	BC Building Code 2	BT-1, ID-2	Morhart	Lecture	1
FD 136	Class	Business of Fashion	FD	Ferreira	Lecture	1
GD 232	Class	Business of Graphic Design	GD-2	Chesson	Art	2

Fig. 1: Excel Worksheet Listing the Set of Events.

While most rows in this Excel worksheet have "Class" as their Course Type, we can also include "Day Off" and "Prep Time" to denote events such as teachers needing a day off, or one or more teachers requiring a common period to plan together. Since no teacher can be scheduled for two events in the same timeslot, our automated timetabling program guarantees off-days for certain teachers as well as ensuring that a set of teachers can have overlapping non-teaching timeslots in which to prepare for future lessons.

As a concrete illustration, SMUS required one part-time teacher (Teacher M) to have exactly one non-teaching day (i.e., no teaching for all seven periods on any one of the five days) in addition to at most three periods of teaching on each of the remaining four days. The administrator at SMUS informed us that Teacher M's non-teaching day could be any day between Monday and Friday, but that a non-teaching day was required for this teacher.



Course ID	Course Type	Course Name	Cohort	Teacher	Classroom	Set of Timeslots	Sign	Value
	Class			Teacher M		1-1, 1-2, 1-3, 1-4, 1-5, 1-6, 1-7	at most	3
	Class			Teacher M		2-1, 2-2, 2-3, 2-4, 2-5, 2-6, 2-7	at most	3
	Class			Teacher M		3-1, 3-2, 3-3, 3-4, 3-5, 3-6, 3-7	at most	3
	Class			Teacher M		4-1, 4-2, 4-3, 4-4, 4-5, 4-6, 4-7	at most	3
	Class			Teacher M		5-1, 5-2, 5-3, 5-4, 5-5, 5-6, 5-7	at most	3

Fig. 2: Excel Worksheet of the Set of Event Constraints.

For each row of the worksheet provided in Figure 2, our Python program generates E, the set of events that are consistent with the leftmost six columns. In Figure 2, the set E refers to the set of events taught by Teacher M that have Course Type = "Class". In other words, by labeling her off-day with a different Course Type, we do not violate the constraint that she can be assigned at most three events (i.e., classes) on each day between Monday and Friday.

We then created a 7-meeting course named NoTeacherM, with Course Type set to "Day Off". In the **Event Relationships** worksheet, we added a row to indicate that these 7 events (i.e., meeting i for Teacher M, for each $1 \leq i \leq 7$) must occur in 7 consecutive periods of the same day. This worksheet contains all the constraint options provided in the Family II subsection of our model.

Finally, the **Teacher Preferences** worksheet indicates information on when teachers would prefer to teach their classes during all the timeslots they are available. At PDA, each preference coefficient $PR_{m,c,d,p}$ was marked as 2 points whenever the teacher of course c *wanted* to teach on day d period p, and was marked as 1 point whenever that teacher *could* teach in that timeslot. (For GNS and SMUS, each PR coefficient was 1 as teachers could not indicate preferences.)

This five-worksheet Excel file serves as the input to our Python program. We now provide the key statistics for each of our schools, listing the number of cohorts, the total number of events (m, c) in the timetable, the number of rows in our Event Constraints worksheet, the number of rows in our Event Relationships worksheet, and the average total running time of our Python program on this input file over ten iterations. This information is provided in Table 1.

All calculations were made on a stand-alone laptop, specifically a 8GB Lenovo running Windows 10 with a 2.1 Ghz processor.

From the table above, we see that it took less than one minute to generate the 2023-2024 Master Timetables for these three educational institutions. All three institutions accepted and implemented our timetable, and have hired us to build their Master Timetable again in 2024-2025.

School Name	PDA	GNS	SMUS
Days in Timetable	5	10	5
Periods in Day	3	5	7
Total Timeslots	15	50	35
Cohorts at the School	8	9	12
Total Events Scheduled	91	348	402
Event Constraint Rows	22	74	134
Event Relationship Rows	16	20	38
Running Time (in seconds)	0.54	13.48	45.51

Table 1: Statistics for our Automated Timetabling Program.

Acknowledgments The authors thank the administrators of the three educational institutions cited in this paper. Specifically we thank Jamie Kemp and Isabel Yu (Pacific Design Academy), Denise Lamarche, Becky Anderson, and Richard Brambley (St. Michaels University School), and Samantha Goddard and Erin Dallin (Glenlyon Norfolk School).

References

1. Appleby, J., Blake, D., Newman, E.: Techniques for producing school timetables on a computer and their application to other scheduling problems. The Computer Journal **3**(4), 237–245 (1961)
2. Collins Dictionary: Definition of timetabling. https://www.collinsdictionary.com/dictionary/english/timetabling (2024), accessed: 2024-03-07
3. Cooper, T.B., Kingston, J.H.: The complexity of timetable construction problems. In: Burke, E., Ross, P. (eds.) Practice and Theory of Automated Timetabling. pp. 281–295. Springer Berlin Heidelberg, Berlin, Heidelberg (1996)
4. Even, A.S.S., Itai, A., Shamir, A.: On the complexity of timetable and multicommodity flow problems. SIAM Journal on Computing **5(4)**, 691–703 (1976)
5. Google OR-Tools: Fast and portable software for combinatorial optimization. https://developers.google.com/optimization (2024), accessed: 2024-03-07
6. Gotlieb, C.: The construction of class-teacher timetables. IFIP congress, Amsterdam **62**, 73–77 (1963)
7. Hoshino, R., Fabris, I.: Optimizing student course preferences in school timetabling. Proceedings of the 17th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR 2020) pp. 283–299 (2020)
8. Hoshino, R., Fabris, I.: Partitioning students into cohorts during COVID-19. Proceedings of the 18th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR 2021) pp. 89–105 (2021)
9. Lewis, R., Paechter, R., McCollum, B.: Post enrolment based course timetabling: a description of the problem model used for track two of the second international timetabling competition. Cardiff Working Papers in Accounting and Finance **A2007-3** (2007)
10. Müller, T., Rudová, H., Müllerová, Z.: Real-world university course timetabling at the international timetabling competition 2019. Journal of Scheduling pp. 1–21 (2024)
11. Pillay, N.: A survey of school timetabling research. Annals of Operations Research **218(1)**, 261–293 (2014)

12. Post, G., Di Gaspero, L., Kingston, J., Mccollum, B., Schaerf, A.: The third international timetabling competition. Annals of Operations Research **239** (2013)

Metaheuristic optimization of Danish High-School Timetables

Victor Alexander Brandsen[1] [0009-0008-4823-1496] and Thomas Stidsen[2] [0000-0001-6905-5454]

[1] Technical University of Denmark, Kgs. Lyngby 2800, Denmark s184014@student.dtu.dk
[2] Technical University of Denmark, Kgs. Lyngby 2800, Denmark thst@dtu.dk

Abstract. Creating schedules for high schools are a massive undertaking, with schools having hundreds of students, teachers, rooms, and courses, all with different requirements. Here we briefly explain a methodology to create schedules of 10, 20, and 40 weeks for Danish high schools, including event chains.

Keywords: Scheduling, Metaheuristics, Mathematical Programming.

1 The Danish High School Timetabling Problem

The High School Timetabling Problem (HSTP) is a well-known problem, and even though it is very dependent on the country of the high schools in question, general versions have been formulated [1]. A common part of HSTP is however to have a cyclic schedule, where a fixed timetable is used for the entire school year. Here we will briefly describe the Danish High School Timetabling Problem (DHSTP). The model described is now used by (practically) all Danish high schools and has some unique features. The current model of DHSTP is used for cyclic repetition of the timetable, but here we experiment with using metaheuristics for solving the DHSTP for not one week, but up to 40 weeks.

The DHSTP model utilize both hard and soft constraints, which are shown below in table 1. Metaheuristics and Mixed Integer Programming (MIP) have previously been applied to the DHSTP ([2,3,4]), showing the viability of both methods in scheduling over 1 and 2 weeks. Real-life instances are often too complex for scheduling 10 or more weeks, whereby planning for two weeks and repeating the schedule over the longer horizon is a viable solution. This however means that single instances of resources not being available during the longer horizon cannot be considered, potentially leading to sub-optimal solutions. If, however, creating schedules of 10 weeks or more were viable in practice, more information about the school year can be taken into account since the optimization processes then can re-assign schedule elements as needed.

1.1 Event chains

An important difference between the HSTP and DHSTP is the addition of event chains. Event chains describe a requirement for a series of events to be placed at certain timeslot offsets relative to a shared origin, e.g.:

Table 1: Hard constraints in the DHSTP

	Hard constraints
Placement	Events requires a timeslot
Event locks	Events requiring a specific timeslot/room
Resource conflict	Limits on usage of resources in a timeslot
Availability	Events or resources only being available in certain timeslots
Event chains	Events required to take place in relation to other events
Work limit	Teachers not having more than a set amount of events per day
Days off	Teachers having at least a set amount of days off
Days off stability	Days off each week for a teacher not differing by more than 1
Day conflict	Events in the same course not being assigned to the same day

– Events in the chain have to be assigned the same timeslot.
– Events in the chain must be assigned contiguous timeslots, in order.
– One event in the chain must occur exactly 2 timeslots after another event.

Moving where this shared origin lies within the schedule therefore changes the correct position of the event chain. These are of great practical use, e.g. when planning educational days with a specific theme, but large chains can cause problems for schedulers due to the significant space required in the schedule to place these correctly.

1.2 Objective Function

A MIP-model for the DHSTP has previously been presented [2,4], including the hard constraints described in Table 1. This model is largely applied here, with one significant change: Event chains are changed from a hard constraint to multiple heavily penalized soft constraints.

To do this, the following goals for placing event chains have been identified:

1. Events in event chains are assigned correct timeslots relative to each other
2. Events in event chains are assigned a timeslot

Regarding goal 1: A definition is required for when an event in an event chain is assigned correctly. As seen in Fig. 1, there can be multiple interpretations of the same schedule, potentially leading to different amounts of correctly placed events. This can be alleviated by deciding on a specific event in the event chain (the "root-event" of the chain), which, if placed, the other events must be placed relative to. In Fig. 1, Case 1 corresponds to having the event in timeslot 5 as the root-event, and Case 2 corresponds to having the event in timeslot 1 as the root event. Since the only way to assess if events are correctly placed is if the root-event is placed, a penalty is added on events in event chains without the root-event assigned. Additionally, a penalty is added to incorrectly placed events in the chain when the root-event is placed. These penalties in combination achieve goal 1 and partly goal 2.

Regarding goal 2: This is achieved by penalizing unassigned events in event chains, only when the root-event of the chain is placed, since it is deemed undesirable to have the event chains only partly scheduled.

The objective of the model is therefore to minimize a weighted sum of:

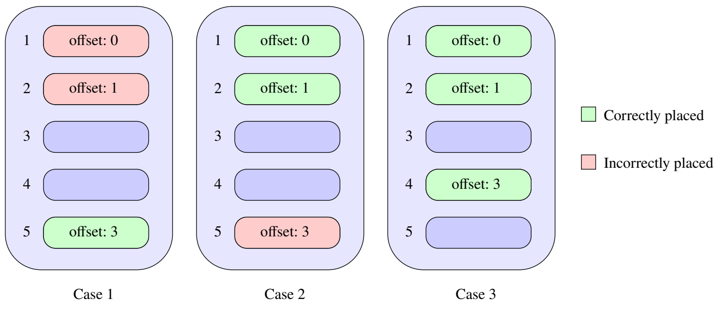


Fig. 1: Two cases scheduling three events in an event chain, on a single day with five timeslots. The required relative offsets for the three events is noted.
Case 1: Event with offset 3 is considered as being correctly placed.
Case 2: Event with offset 0 is considered as being correctly placed.
Case 3: All events are correctly placed.

1. The number of events without a timeslot assigned,
2. The number of events without a room assigned,
3. The index of the timeslots assigned to each event,
4. The number of teachers scheduled at timeslots they wish not to be scheduled on,
5. A desirability-score of the rooms assigned to events,
6. The number of events on neighboring days for courses,
7. The number of idle timeslots for teachers and students,
8. The number of rooms used by each course in excess of one,
9. The number of work days for teachers,
10. The number of days where students have zero events scheduled,
11. The number of days where teachers have exactly one event scheduled,
12. The maximum difference in the number of scheduled events in a week for courses,
13. The number of events in event chains without the root-event placed,
14. The number of incorrectly placed events in event chains with the root-event placed, and
15. The number of unassigned events in event chains with the root-event placed.

2 Solution Methods

A dataset of 20 instances of schedules for 1 and 2 weeks from [2], with between 300 and 2500 events and between 140 and 660 classes, are used. As mentioned, it is of interest to create schedules of 10 or more weeks, but due to the results from the MIP-approach in [2] on 1 and 2 weeks, it is deemed infeasible to find exact solutions to the DHSTP for longer planning horizons, and a metaheuristics-approach is therefore used. Adaptive Large Neighborhood Search (ALNS) and Tabu Search (TS) algorithms are proposed for generating solutions for schedules of 1, 2, 10, 20, and 40 weeks. For schedules of 10 or more weeks, three approaches are furthermore explored:

- A cold-start approach, where a greedy heuristic is applied to empty schedules of 10, 20, and 40 weeks, which are then used as starting points for the metaheuristics.
- A warm-start approach, where schedules for 1 or 2 weeks are created with the cold-start approach, which then are duplicated to create initial schedules of 10, 20, and 40 weeks. Illegally placed events are removed and metaheuristics are again applied.
- A two-stage approach, where timeslots are assigned first and rooms assigned afterward, as outlined in [2]. This approach can be combined with the cold-start or warm-start approaches.

3 A brief conclusion

At the presentation, a summary of the model used and conclusions drawn from the results from using the above methods will be presented.

References

1. Kristiansen, S., Sørensen, M., Stidsen, T.R.: Integer programming for the generalized high school timetabling problem. Journal of Scheduling **18**, 377–392 (2015)
2. Schjørring, N.D.: Mathematical Optimisation of High-School Timetabling. Master's thesis, Technical University of Denmark (2023)
3. Sørensen, M., Stidsen, T.R.: High School Timetabling: Modeling and solving a large number of cases in Denmark. Proceedings of the Ninth International Conference on the Practice and Theory of Automated Timetabling (PATAT) pp. 359–364 (2012)
4. Sørensen, M., Stidsen, T.R.: Integer programming and adaptive large neighborhood search for real-world instances of high school timetabling. Annals of Operations Research, PATAT (2012)

The University Examination Timetabling Problem with Uncertain Timeslot Capacity: A Two-stage Stochastic Programming Approach

Sara Ceschia$^{1\,[0000-0003-1191-1929]}$, Daniele Manerba$^{2\,[0000-0002-3502-5289]}$, Andrea Schaerf$^{1\,[0000-0001-6965-0536]}$, Eugenia Zanazzo^{1}, and Roberto Zanotti$^{3\,[0000-0002-3073-4895]}$

[1] Polytechnic Dept. of Engineering and Architecture, Università di Udine, Italy
`{sara.ceschia,andrea.schaerf,eugenia.zanazzo}@uniud.it`

[2] Dept. of Information Engineering, Università degli Studi di Brescia, Italy
`daniele.manerba@unibs.it`

[3] Dept. of Clinical and Experimental Sciences, Università degli Studi di Brescia, Italy
`roberto.zanotti@unibs.it`

Abstract. In this work, we extend the original Uncapacitated Examination Timetabling problem by introducing capacity constraints that limit the number of exams schedulable per timeslot and, to take into account possible unexpected disruptive events, by considering such a capacity as a random variable. We propose a two-stage Stochastic Programming approach for this stochastic variant in which recourse actions allow rescheduling exams in successive timeslots or moving students to *spot-market* rooms. Then, we conduct an in-depth analysis of the impact of uncertainty on solutions using a deterministic equivalent Mixed-Integer Linear Programming formulation. Additionally, we plan to develop a Progressive Hedging algorithm, leveraging the efficiency of a specialized optimizer [4], to address the computational challenges posed by the stochastic nature of the problem even for small-medium size instances. Preliminary results are promising, underscoring the significance of accounting for stochasticity in the problem formulation.

Keywords: Examination timetabling, Uncertain timeslot capacity, Two-stage Stochastic Programming with recourse

1 Introduction

In the context of university organization, the Examination Timetabling problem (ETT) aims at assigning exams to timeslots ensuring that *i*) each exam is scheduled exactly once during the examination period, *ii*) two conflicting exams are not scheduled in the same timeslot, and *iii*) the total penalty associated with the created timetable is minimized [6]. Among many existing formulations, the most classic, known as Uncapacitated ETT (UETT), was introduced in [5] and specifically penalizes exams with students in common scheduled within a distance less than or equal to 5 timeslots.

In the UETT, it is assumed that the number of exams scheduled in each timeslot is unbounded. However, in practical applications, physical constraints (number of available rooms and their capacity) must be taken into consideration. In addition, after the exam calendar is released, uncertain events may occur and reduce the day-by-day availability of resources (rooms, teachers, timeslots), thus making the original schedule infeasible. To address this issue, we investigate a Stochastic Capacitated ETT under uncertain timeslot capacity (S-CETT), in which the number of exams schedulable in each timeslot is modeled using a random variable. In particular, we formulate the S-CETT as a two-stage Stochastic Programming (SP) problem and study the impact of uncertainty on timetables given the implementation of reasonable recourse actions. Finally, we plan to implement an efficient algorithm approach hybridizing an SP decomposition-based matheuristic and a tailored state-of-the-art heuristic method.

To our knowledge, little attention has been paid to uncertainty in the context of ETT problems (see [2,3]). Instead, the problem of finding a robust timetable for the Curriculum-Based University Course Timetabling problem subject to different types of disruptions has been addressed in [1,7,10]. Such a problem is usually modeled as a minimum perturbation problem with a bi-criteria objective function, where the first objective is related to the quality of the solution and the other is about the robustness of the timetable.

2 ILP formulation for the S-CETT

Let us consider a set E of exams, to be scheduled during an examination period at the end of the semester, and a set S of students. Each student is enrolled in a non-empty subset of exams. The examination period is divided into T ordered timeslots, each having a scheduling capacity of B exams. Let n_e be the number of students enrolled in exam $e \in E$. Given two exams $e, e' \in E$, let $n_{e,e'}$ be the number of students enrolled in both. Two exams $e, e' \in E$ are called *conflicting* if they have at least one student enrolled in both, i.e., if $n_{e,e'} > 0$. Let us define the set C of conflicts, including all the exam pairs $[e, e']$ with $e, e' \in E$ for which $n_{e,e'} > 0$. *Conflicting* exams cannot take place during the same timeslot. Moreover, to foster the creation of timetables that are more sustainable for the students, a penalty is assigned for each couple of conflicting exams scheduled up to a *distance* of 5 timeslots. More precisely, given two exams $e, e' \in E$ scheduled at distance i of time-slots, with $1 \leq i \leq 5$, the relative penalty is $2^{(5-i)} n_{e,e'}$. Finally, let $\tilde{B}_t$ be a stochastic variable representing the loss of capacity of scheduled exams in timeslot $t = 1, \ldots, T$, with $0 \leq \tilde{B}_t \leq B$.

Let us define a binary variable $y_{e,t}$ determining the assignment of exam $e \in E$ to time-slot $t = 1, \ldots, T$, and binary variable $u^i_{e,e'}$ which takes value 1 if the conflicting exams pair $[e, e'] \in C$ is scheduled $i = 1, \ldots, 5$ time-slots apart, and 0 otherwise. Then, our S-CETT can be formulated as follows:

$$\text{(S-CETT)} \quad \min \quad \frac{1}{|S|} \sum_{i=1}^{5} \sum_{[e,e'] \in C} 2^{(5-i)} n_{e,e'} u^i_{e,e'} \tag{1}$$

subject to

$$\sum_{t=1}^{T} y_{e,t} = 1 \qquad e \in E \tag{2}$$

$$y_{e,t} + y_{e',t} \leq 1 \qquad [e, e'] \in C, t = 1, \ldots, T \tag{3}$$

$$y_{e,t} + y_{e',t+i} \leq 1 + u^i_{e,e'} \qquad [e, e'] \in C, i = 1, \ldots, 5, t = 1, \ldots, T - i \tag{4}$$

$$\sum_{e \in E} y_{e,t} \leq B - \tilde{B}_t, \qquad t = 1, \ldots, T. \tag{5}$$

The objective function (1) minimizes the overall penalty by summing up individual penalties for each couple of conflicting exams. Constraints (2) ensure that each exam is scheduled exactly once. Constraints (3) ensure that two conflicting exams can not be scheduled in the same timeslot. Constraints (4) ensure that if two conflicting exams are scheduled i timeslots apart (i.e., both the y variables in the inequality take value 1), then the relative u variable must take value 1 as well. Finally, constraints (5) ensure that the number of exams scheduled in a timeslot is limited by the stochastic capacity $B - \tilde{B}_t$.

3 SP framework and solution approach

We tackle the problem using a two-stage SP paradigm, in which first-stage variables concern the pre-scheduling of exams to timeslots. In contrast, the second-stage recourse actions include the possibility of *i*) rescheduling the exams in a different timeslot after the pre-scheduled one and *ii*) moving an exam to a *spot-market* room in the same timeslot. Note that it is possible to relocate any number of exams to the *spot-market* room. This way the model always guarantees a feasible solution. However, in addition to the basic penalties due to exam incompatibilities, extra penalties, proportional to the number of students affected by rescheduled exams or moved to the spot-market room, must be considered in the expected value.

To practically address the problem via state-of-the-art MIP solvers, we create a deterministic equivalent formulation by approximating the behavior of the random variables involved through a finite (but sufficiently large) number of future scenarios, each occurring with a given probability. This allows us to validate our model by assessing standard SP indicators, such as the *Value of the Stochastic Solution* (VSS) and the *Expected Value of the Perfect Information* (EVPI). VSS represents the penalty saving given by using our SP approach instead of a deterministic model, while EVPI represents how much we would be willing to pay for not having uncertain data. Figure 1 presents boxplots on the percentage values of these two indicators obtained on a set of 20 small instances, each with $|E| = 10$, $T = 7$, $B = 2$, and 20 scenarios. VSS and EVPI both have an average value of around 30%, indicating that there is a significant gain in accounting for stochasticity but also a notable gap from the case in which the values of all random variables are known beforehand. The VSS, ranging from 10 to 50%, particularly proves the importance of a more robust provisional schedule and more flexible recourse decisions.

In the same figure, we also show an additional indicator, named *Stochastic Loss* (SL), that measures the percentage difference between the first-stage penalty in our stochastic variant and the objective function value of the deterministic problem. This indicator shows that, on average, the solutions of our variant include a 65% higher penalty when it comes to the provisional schedule of the exams to be more conservative and handle the uncertainty of future events more effectively.

Apart from the above validation, the use of an exact technique can be computationally too expensive against real-size instances with a representative number of scenarios.

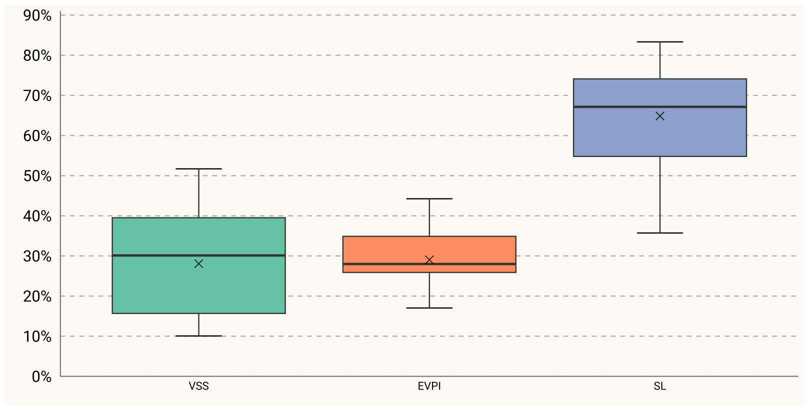


Fig. 1: Percentage values of VSS, EVPI, and SL over all the benchmark instances.

Hence, we plan to develop a heuristic convergence framework based on a Progressive Hedging (PH) algorithm ([8],[9]), which decomposes the problem per scenario and forces a *consensus* solution among the scenarios via an Augmented Lagrangian Relaxation approach. To solve the deterministic mono-scenario subproblems, iteratively created during the PH, we will use the Simulated Annealing-based algorithm developed in [4] for the UETT conveniently adapted to the capacitated version. Extensive computational results of the PH framework will be presented at the conference.

References

1. Akkan, C., Gülcü, A., Kuş, Z.: Minimum penalty perturbation heuristics for curriculum-based timetables subject to multiple disruptions. Computers & Operations Research **132**, 105306 (2021)
2. Bassimir, B., Wanka, R.: Probabilistic curriculum-based examination timetabling. In: Proc 12th International Conference on the Practice and Theory of Automated Timetabling (PATAT). pp. 273–285 (2018)
3. Bassimir, B., Wanka, R.: Robustness approaches for the examination timetabling problem under data uncertainty. In: Proc. 9th Multidisciplinary International Scheduling Conference: Theory & Applications (MISTA) (2019)
4. Bellio, R., Ceschia, S., Di Gaspero, L., Schaerf, A.: Two-stage multi-neighborhood simulated annealing for uncapacitated examination timetabling. Computers & Operations Research **132**, 105300 (2021)
5. Carter, M.W., Laporte, G., Lee, S.Y.: Examination timetabling: Algorithmic strategies and applications. Journal of the Operational Research Society **47**, 373–383 (1996)
6. Ceschia, S., Di Gaspero, L., Schaerf, A.: Educational timetabling: Problems, benchmarks, and state-of-the-art results. European Journal of Operational Research **308**(1), 1–18 (2023)
7. Lindahl, M., Stidsen, T., Sørensen, M.: Quality recovering of university timetables. European Journal of Operational Research **276**(2), 422–435 (2019)
8. Løkketangen, A., Woodruff, D.L.: Progressive hedging and tabu search applied to mixed integer (0,1) multistage stochastic programming. Journal of Heuristics **2**(2), 111–128 (1996)

9. Manerba, D., Perboli, G.: New solution approaches for the capacitated supplier selection problem with total quantity discount and activation costs under demand uncertainty. Computers & Operations Research **101**, 29–42 (2019)
10. Phillips, A.E., Walker, C.G., Ehrgott, M., Ryan, D.M.: Integer programming for minimal perturbation problems in university course timetabling. Annals of Operations Research **252**, 283–304 (2017)

Predicting employee absenteeism to generate robust rosters

Pieter Smet[1], Martina Doneda[2,3], Giuliana Carello[2], Ettore Lanzarone[4], and Greet Vanden Berghe[1]

[1] KU Leuven, Department of Computer Science, CODeS, Gent, Belgium
[2] Politecnico di Milano, Department of Electronics, Information and Bioengineering, Milan, Italy
[3] National Research Council, Institute for Applied Mathematics and Information Technologies, Milan, Italy
[4] University of Bergamo, Department of Management, Information and Production Engineering, Dalmine (BG), Italy

Abstract. Employee absences often lead to disruptions in rosters, necessitating last-minute changes to employee schedules. A common strategy to minimize the adverse effects of these changes is to assign employees to on-call duties, thereby increasing the robustness in the rosters. This study explores the effectiveness of a data-driven robust rostering approach, using predictions of employee absences to schedule an appropriate number of on-call duties. Numerical experiments demonstrate how the accuracy of absence predictions significantly impacts the robustness of the resulting rosters. We introduce a methodology to assess the conditions under which a data-driven robust rostering approach can outperform simple, non-data-driven rostering strategies.

Keywords: Personnel rostering, Employee absenteeism, Robustness, Proactive rostering, Machine learning.

1 Introduction

Employee absenteeism is the term used to describe when an employee is not present at work during their scheduled hours. Intertwined factors such as health issues, difficulties in achieving work-life balance and instances of workplace harassment can all contribute to employee absenteeism. Whatever the root cause, absenteeism typically has several negative effects on organizations: reduced productivity, additional costs from overtime or from hiring and training replacement employees, low team morale and job disengagement [2].

There have been several studies on how to make personnel rosters more robust with respect to disruptions caused by employee absenteeism. The most common approach is to include *buffers* in the roster that manage unexpected absences through the use of surplus resources. Capacity buffers involve assigning more employees than required to a shift [3]. Meanwhile, reserve shift buffers are created by assigning a subset of employees to special on-call duties which can be converted into working shifts to cover for absences [4].

Approaches employing buffers typically have one or more parameters to set buffer size, thereby affecting the degree of robustness of the generated roster. These parameters are usually set by a human expert or based on results from extensive empirical studies. In our work, we investigate how a Machine Learning (ML) model for predicting employee absenteeism can help determine a suitable number of reserve shifts. More specifically, we analyze under which conditions an ML-informed robust rostering approach can outperform non-data-driven approaches that schedule a fixed number of reserve shifts on each day.

2 Problem definition

The considered personnel rostering problem is based on a general problem definition [1]. The goal is to find an assignment of shifts to employees subject to various personal and organizational constraints. Table 1 provides an overview of the problem's hard and soft constraints. The roster of the preceding scheduling period is taken into account to correctly evaluate the constraints at the beginning of the current scheduling period. Robustness is ensured by including a number of reserve shifts in the roster on each day of the scheduling period. The objective function is a weighted sum of the scheduled employee wage costs (regular, overtime and on-call), the wages of interim personnel needed to cover any understaffing and a penalty term for assigning fewer reserve shifts than required.

Hard constraints
At most one shift assignment per day per employee
Skill requirements
Forbidden shift succession (e.g. no early after late shift)
Minimum number of days worked per employee
Maximum number of consecutive working days
Maximum number of consecutive night shift assignments
Shift and day off requests
Soft constraints
Minimum staffing requirements for each day, shift and skill
Maximum number of days worked per employee
Number of reserve shifts in the roster

Table 1: Hard and soft constraints in the problem.

3 ML-informed robust rostering

The problem described in Section 2 is modeled as an integer programming problem and solved using Gurobi 10.0.3. The number of reserve shifts required on each day is determined by an ML model. In contrast to other studies, we do not actually train an ML model. Instead, we propose a way of *simulating* the predictions a model would make at a given prediction performance level, i.e., given the ground truth and a performance characterization of the ML model, our methodology derives what predictions the model would make. These predictions are used to determine the number of reserve shifts that must be included in the roster. Predicting whether or not an employee will be absent on a given day is a binary classification problem. We use the True Positive Rate α and True Negative Rate β to characterize the prediction performance of the ML model. Figure 1 shows a confusion matrix, used to compare ground truth and model predictions for binary classification problems. Given a confusion matrix, we can compute $\alpha = TP/(TP+FN)$ and $\beta = TN/(TN + FP)$.

		Ground truth	
		Positive	**Negative**
Predicted	**Positive**	True positive (TP)	False positive (FP)
	Negative	False negative (FN)	True negative (TN)

Fig. 1: Confusion matrix.

The probability that an employee is absent on a given day, derived from historical data, is denoted by ρ. Note that we do not consider employee-specific absence probabilities, but instead use the average over all employees. The correct prediction of an absence by the ML model depends, with probability α, on whether the realization will be correctly classified as a True Positive. With probability $1 - \alpha$, a true absence will result in a False Negative. Similarly, if the employee is not absent, this will be considered a potential False Positive with probability $1 - \beta$. Any potential False Positive will become a proper False Positive with probability ρ, so that the overall number of absences will be reasonable even when β is very small. Each time the prediction results in a True Positive or a False Positive, the number of reserve shifts required is increased by one. In case of True Negatives or False Negatives, the number of reserve shifts is unaffected.

The robustness of the generated roster is measured by the expected re-rostering cost. This cost is computed by running several simulations in which employees become absent and the roster is repaired using an exact re-rostering method. The re-rostering costs obtained for different values of α and β are recorded and compared against those obtained by a non-data-driven baseline approach. More specifically, we compare against an approach from the literature that has no knowledge about what will happen and instead schedules a fixed number of reserve shifts on each day [4]. The results of numerical experiments enable us to identify for which levels of sensitivity and specificity better solutions are generated. Detailed results will be presented at the conference.

Acknowledgements This research was supported by KU Leuven C24E/23/012 "Human-centred decision support based on new theory for personnel rostering". Editorial consultation provided by Luke Connolly (KU Leuven).

References

1. Ceschia, S., Dang, N., De Causmaecker, P., Haspeslagh, S., Schaerf, A.: The second International Nurse Rostering Competition. Annals of Operations Research **274**(1-2), 171–186 (2019)
2. Hudson, C.K., Shen, W.: Understaffing: An under-researched phenomenon. Organizational Psychology Review **5**(3), 244–263 (2015)
3. Ingels, J., Maenhout, B.: Optimised buffer allocation to construct stable personnel shift rosters. Omega **82**, 102–117 (2019)
4. Wickert, T.I., Smet, P., Vanden Berghe, G.: Quantifying and enforcing robustness in staff rostering. Journal of Scheduling **24**(3), 347–366 (2021)

Incorporating Nurse Preferences in the Nurse Scheduling Problem

Eva van Rooijen, Shayekh Hassan[0000−0002−6196−0318], and Qing Chuan Ye[0000−0002−8249−9890]

ORTEC B.V., Houtsingel 5 2719 EA Zoetermeer, The Netherlands eva.vanrooijen@ortec.com

Abstract. As the current healthcare labour market is volatile, due to employees having bad experiences with irregular shifts and unconventional working hours, it is important to make an effort to retain existing and attract new healthcare employees. This research explores the effect of scheduling decisions on job satisfaction of nurses in Dutch hospitals. We examine if nurse satisfaction can be improved using mathematical optimization, and at what cost. Incorporating results from interviews and a survey, this research presents a formulation of the nurse scheduling problem including both capacity coverage and nurse satisfaction in the problem's objective. The problem is solved using an exact (MIP) and a heuristic (VDS) approach. Using benchmark instances for the nurse scheduling problem, results show that nurse satisfaction can be improved without decreasing the capacity coverage.

Keywords: Nurse scheduling problem, Schedule satisfaction, Nurse job satisfaction, Mathematical programming, Variable depth search

1 Introduction

A recent study in The Netherlands reports an expected shortage of 140,000 healthcare employees by 2031 [4]. Two main reasons for this shortage are an increased demand for healthcare and a shortage on the healthcare labour market. The irregular shifts and unconventional working hours make nurses quit their profession and discourage others to apply. In order to keep nurses healthy and prevent burn-outs, their personal scheduling preferences should be incorporated in the scheduling process [3,6]. However, in practice, nurse preferences are complex and difficult to quantify in a single score per nurse per schedule. In this research, we combine the results of interviews and a survey to redesign the objective in the nurse scheduling problem. This particular combination of research methods is novel, as previous research has either focused on the quantitative solution methods, or used a qualitative approach to study nurse job (schedule) satisfaction, which mainly originates from human resources or social sciences fields.

2 Methodology

To gain an understanding of the preferences of nurses, we use a mixed-methods approach. This approach combines quantitative and qualitative methods to answer research questions on complex issues in the social sciences [5]. First, interviews were held at the

Martini Hospital (Groningen, The Netherlands) to gain an understanding of the most relevant preferences nurses have regarding the scheduling process. Based on the results, we cluster nurse preferences in five categories: incidental requests for (not) working a particular day or shift; preferences regarding the length of a consecutive series of shifts (consecutiveness); the shift types nurses are assigned to work; the scheduling of weekend shifts; and the scheduling of night shifts. Second, we designed a survey with closed questions on preferences for these five clusters. The survey concludes with a question asking participants to divide a total of 50 points across these five clusters to ask about the relative importance for each of these clusters. Results of the survey show that nurses find the adherence to their requests and their consecutiveness preferences most important for their schedule satisfaction. Also, consecutiveness preferences are correlated with the number of contract hours nurses are assigned to work. Part-time nurses typically prefer to work between 2 and 3 consecutive days on average whereas full-time nurses prefer to work a consecutive series of minimum 3 and maximum 4 shifts. Finally, we use the results of the interviews and surveys as input for our mathematical formulation. As there are multiple objectives, for both the planner and nurses, we make use of a weighted sum approach, as this is easy to interpret by the users and the weights can easily be adjusted to their preferences.

3 Mathematical formulation

Most nurses selected the requests and consecutiveness as their top two priorities. The consecutiveness penalty for a nurse ($i \in N$) is calculated based on the difference between the actual consecutiveness of the assigned blocks of shifts and the simulated preferences of the same nurse. Similarly, the request penalty for a nurse ($i \in N$) is calculated by counting the number of times the schedule fails to meet indicated preferences. An individual's satisfaction score will therefore be a weighted sum (with $0 \leq \alpha_i \leq 1$) of these two indicators of satisfaction:

$$P_i = \alpha_i \cdot \text{consecutivenessPenalty}_i + (1 - \alpha_i) \cdot \text{requestPenalty}_i \quad \forall\, i \in N \tag{1}$$

The individual satisfaction scores are aggregated using the worst-off score and the sum of all scores. These can be balanced using $\gamma_1, \gamma_2 \in \{0, 1\}$, respectively, depending on the scheduling policy. By setting $\gamma_1 > 0$, additional penalty is added for not dividing the total sum of satisfaction evenly across the employees which can be regarded as unfair. The objective function combines the satisfaction and coverage scores using the parameter $0 \leq \beta \leq 1$:

$$\min \quad \beta(\gamma_1 \max_{i \in N} P_i + \gamma_2 \sum_{i \in N} P_i) + (1 - \beta)(\sum_{d \in D} \sum_{t \in T} y_{dt} v^{min} + \sum_{d \in D} \sum_{t \in T} z_{dt} v^{max}) \tag{2}$$

with y_{dt} the total number of unassigned shifts, z_{dt} the total number of over-assigned shifts for day d and shift type t; v^{min} the penalty per unassigned shift, and v^{max} the penalty per over-assigned shift.

4 Computational results

Results are obtained using data from the nurse scheduling benchmark instances [2]. These instances contain data on the available employees, shift types, cover requirements and nurse requests. We will use *small instances* 1, 2 and 3, which have a scheduling period of two weeks and up to 20 employees, and *large instances* 11 and 12, which have a scheduling period of four weeks and up to 60 employees. These instances are solved for both the objective function without satisfaction (only coverage penalty, $\beta = 0$), and with the satisfaction scores $\beta = 0.5$, $\gamma_1 = 1$, and $\gamma_2 = 1$. Since we aim to optimize for nurse satisfaction using the consecutiveness preferences, we simulate these preferences based on the obtained probability distributions per contract type through our survey. The problem is modelled as a mixed integer programming model (MIP) and is solved using IBM ILOG CPLEX 22.1.0, on an Intel Core i7 2.8 GHz processor and 16GB RAM, and using a heuristic based on a Variable Depth Search (VDS) based on Burke et al. (2013) [1].

Table 1 shows the results obtained using the MIP with a maximum runtime of 1 hour, whereas Table 2 shows the results obtained using VDS with a runtime of 1 hour, using a fixed set of simulated preferences for the nurses. These results show that nurse satisfaction can be improved without decreasing the coverage by including satisfaction in the objective function of the nurse scheduling problem. To investigate the effect of the simulated preferences, we also run the instances with newly generated preferences for the nurses in every run. The MIP is run 100 times for small instances, and 5 times for large instances, due to the higher runtime. In all simulation runs, the results still hold, where the coverage stays the same and the nurse satisfaction is improved.

Table 1: MIP results with $\beta = 0$ and $\beta = 0.5$

	$\beta = 0$					$\beta = 0.5$				
instance	time (s)	gap	coverage	max P_i	$\sum P_i$	time (s)	gap	coverage	max P_i	$\sum P_i$
1	0.203	0	600	2	5.807	1.078	0	600	1	2.548
2	7.781	0	800	5	12.232	26.36	0	800	1	3.785
3	32.10	0	1000	4	21.371	490	0	1000	1.77	8.67
11	9.36	0	3423	4	60.432	3600	0.01	3423	4	33.163
12	3600	0.000	4001	9	82.062	3600	0.018	4000	9	70.872

5 Conclusion

The two most important indicators of nurse schedule satisfaction are the adherence to requests made by nurses to (not) work specific shifts and the consecutiveness of assigned shifts in the schedule. When nurses are assigned too many or too few consecutive shifts per block, their schedule satisfaction decreases as they cannot balance their workload with enough rest. However, the minimum and maximum number of preferred consecutive shifts differs per nurse because of personal differences. Additionally, the importance of

Table 2: VDS results with $\beta = 0$ and $\beta = 0.5$, runtime of 1 hour

	$\beta = 0$			$\beta = 0.5$		
instance	coverage	max P_i	$\sum P_i$	coverage	max P_i	$\sum P_i$
1	600	2	5.807	600	1	2.548
2	800	5	12.232	800	1	3.785
3	1000	4	21.371	1000	1.77	8.67
11	3827	5	27.36	3827	5	27.36
12	4900	6.025	38.386	4900	6.025	38.386

requests versus the consecutiveness of shifts differs per person. Therefore, including these personal preferences in the objective function of a nurse scheduling problem requires input from the nurses. The effect of including nurse satisfaction in the objective function shows that the satisfaction of the nurses can be improved without decreasing the coverage. Therefore, it does not cost anything in terms of coverage to improve the satisfaction of the nurse.

References

1. Burke, E. K., Curtois, T., Qu, R., & Vanden Berghe, G. (2012). A time predefined variable depth search for nurse rostering. *INFORMS Journal on Computing* 25(3), pp 411-419.
2. Curtois, T., & Qu, R. (2014). Computational results on new staff scheduling benchmark instances. *Technical report ASAP Research Group.* http://www.schedulingbenchmarks.org/papers/computational_results_on_new_staff_scheduling_benchmark_instances.pdf
3. Maqbali, M. A. (2015). Factors that influence nurses' job satisfaction: a literature review. *Nursing Management* 22 (2), pp 30-37.
4. Ministerie van Algemene Zaken (Mar. 2022): Kamerbrief over nieuwe prognose verwachte personeelstekort. https://www.rijksoverheid.nl/documenten/kamerstukken/2022/01/20/kamerbrief-over-nieuwe-prognose-verwachte-personeelstekort
5. Timans, R., Wouters, P. & Heilbron, J. (2019). Mixed methods research: what it is and what it could be. *Theory and Society* 48, pp 193–216.
6. Van den Bergh, J., Beliën, J., De Bruecker, P., Demeulemeester, E. & De Boeck, L. (2013). Personnel scheduling: A literature review. *European Journal of Operational Research* 226 (3), pp. 367–385.

The invigilator assignment problem

Hannah Verplancke[1,2][0000−0001−9104−3169], David Van Bulck[1,2][0000−0002−1222−4541], Veronique Limère[1,2][0000−0001−6858−6309], and Nico André Schmid[1,3][0000−0002−2432−5913]

[1] Ghent University, Ghent, Belgium
[2] Core lab CVAMO, FlandersMake@UGent, Belgium
hannah.verplancke@ugent.be
[3] IESEG School of Management, Univ. Lille, CNRS, UMR 9221 - LEM - Lille Economie Management, F-59000 Lille, France

Keywords: Invigilator assignment, Examination scheduling.

1 Introduction

The invigilator assignment problem (IAP) aims at providing an invigilator schedule, appointing sufficient invigilators to all exams of a predetermined examination schedule. These invigilators perform the required logistic and administrative tasks when students take exams, such as distributing copies, preventing students from cheating, or answering questions. Even though the need for research on the IAP had already been raised in the nineties (see, e.g., [2]), only recently the problem gained more importance in literature (e.g., [5] and [6]).
As stated above, the IAP uses a predetermined examination schedule, the outcome of the widely studied examination timetabling (ETT) problem, as an input. A first variant of the ETT, known as uncapacitated ETT, assigns a set of examinations to a fixed number of periods while avoiding students having to take two exams at the same time [3]. A second variant, the capacitated ETT, also considers the assignment of exams to examination rooms with individual sizes, thereby limiting the number of exams that can be taking place concurrently [7]. The assignment of exams to timeslots and rooms is managed on the university level, while the assignment of invigilators is situated on the faculty level. Moreover, the ETT problem is typically solved long before the actual exams take place, so students can optimally plan, whereas the IAP can be solved until a few days before the exams, to capture the invigilator availabilities as accurate as possible. Therefore, a sequential solution approach has been chosen to address those problems. For an extensive overview of educational timetabling problems, including the ETT problem, we refer to [4].
Previous research on the IAP mostly focused on providing formulations and heuristic solutions for a variety of case-specific problems. A general formulation seems to be missing, as well as insights on the complexity of the problem at hand. In this research, we want to summarize common elements identified through the different previous papers and provide insights on the complexity of the proposed base model and some interesting variants.

2 Problem description

We identify the following base problem, which forms the foundation of almost any IAP studied in literature. In the IAP, we are presented with a set of examination periods P, invigilators I, and exams E. Each exam $e \in E$ is preassigned to a period $p \in P$ and requires d_e invigilators. Assigning invigilator i to exam e results in cost $c_{i,e}$. This cost represents the preference of the invigilator, where a more preferred time slot induces a lower cost and a less preferred time slot induces a higher cost. An understaffing cost σ is incurred for each invigilator allocated less than needed. Notice that this cost will be much higher than any of the preference costs $c_{i,e}$. The IAP is to assign invigilators to exams in a way that minimizes total costs related to invigilator assignments and potential understaffing. Additionally, the solution must satisfy two hard constraints: C1 and C2.

C1 Each invigilator is limited to supervising at most one exam in any given period, ensuring there are no scheduling conflicts.
C2 The total number of exams assigned to invigilator i should be between $\underline{w}_i$ and $\overline{w}_i$.

These constraints may result in incomplete assignments for some exams, as captured in the model by the understaffing costs. In practice, these remaining invigilators are staffed manually after personal correspondence with invigilators who have less than $\overline{w}_i$ assignments and verifying their availability. If desired, this decision could also be included in the model by removing the understaffing cost and including a preference cost equal to the understaffing cost for those time slots outside the initially provided invigilators' preferences. In practice, this also requires manual verification of the invigilators' availability. A final alternative could be to treat C2 as a soft constraint, but according to our experience this could result in very unbalanced solutions.

Theorem 1. *The IAP can be solved in polynomial time.*

Proof. We show that IAP can be modelled as a special case of the Minimum Cost Network Flow Problem (MCNFP), which can be solved in polynomial time [1].
MCNFP
Input. A directed graph $G(V, A)$ with a net supply b_i for each vertex $i \in V$, and a capacity $u_{i,j}$ and cost $c_{i,j}$ for each arc $(i, j) \in A$.
Output. A minimum cost flow respecting the net supply at each vertex and the capacity at each arc.

To construct the graph G, we start by creating a node for each invigilator $i \in I$ with a net supply of $\overline{w}_i$ and for each invigilator also $|P|$ invigilator-period nodes with zero net supply. Each invigilator node is connected to its corresponding period nodes with arcs of zero cost and unit capacity.

Next, for each exam e scheduled in period p, an exam node is created with net supply $-d_e$, linked to all invigilator-period nodes associated with period p with unit capacity arcs costing $c_{i,e}$.

Finally, we add two dummy nodes. A first dummy node, x, has net supply equal to $s = \sum_{e \in E} d_e$ and is connected to each of the exam nodes using an arc with capacity d_e and cost σ, indicating potential understaffing for exams. Secondly, to balance the network, we include a dummy exam node y with a net supply of $-t = \sum_{i \in I} \overline{w}_i$ and link it with invigilator nodes using a zero-cost arc with capacity $\overline{w}_i - \underline{w}_i$. Finally, we add a zero-cost arc from the dummy-invigilator node to the dummy exam node with capacity v. □

In order to solve the associated MCNFP instance, we use a standard linear programming formulation, which is known to be totally unimodular (i.e., it is sufficient to consider the LP-relaxation to obtain integral solutions) [1]. This way, we obtain optimal solutions within seconds, even for instances involving hundreds of exams. Although, this base problem recurs in most of the studied literature, different objectives and constraints are needed to tackle context-specific problems, making it hard to combine all of them in a single model or definition.
In this talk, we show how the addition of certain constraints or objectives turns the problem NP-hard, e.g., including fairness costs to limit the difference in satisfaction of different invigilators. We contribute to the literature by discussing the complexity of the base model and some interesting variants.

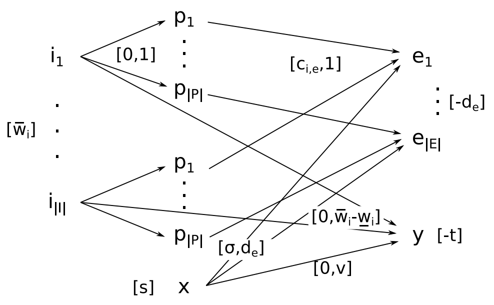


Fig. 1: Network G used to transform IAP into an instance of MCNFP. Numbers between brackets denote the net supply at each nonde, and the cost and capacity at arcs.

3 Case study

A variant of the model above was used to schedule invigilators atthe faculty of Economics and Business Administration at Ghent University. The tests were conducted on an instance with 195 exams and a total demand of 1140 invigilators, spread over 67 time periods and 321 invigilators. Prior to the implementation of our model, invigilators had to fill in a shared file to immediately register for specific exams and rooms in a First Come, First Serve (FCFS) manner, oftentimes resulting in unfulfilled demand. To assess the benefit of an optimization-based solution, we compare it to various simulation runs using the FCFS principle that varied the arrival sequence of invigilators for a predefined set of available time slots per invigilator. Invigilators would be available for 5, 10, or 20 time slots and register for 3-4 exams as long as there were exams during such time slots that still required invigilators. This approach is compared to the optimization-based approach where the invigilators' preferences are used to find an optimal solution. Table 1 compares the optimal solution to the simulation runs by reporting the percentage of demand that is covered by available invigilators (Coverage %) and the average preference values per assignment (Avg. pref.) with lower preference values indicating more preferable assignments. One can observe the high variability in performance depending on the random sequence created for the FCFS approach, indicating a low robustness of this approach. The optimal approach significantly reduces the number of missing invigilators compared to any result obtained by the FCFS principle. In some cases, this increases average preference values, but this effect is nullified when the number of preferences per invigilator is sufficiently large. All experiments were solved optimally on a laptop with 8GB RAM and an Intel m5-6Y57 processor with 1.1 GHz in less than 3 seconds.

During the pandemic period we implemented an adapted version of this base model, minimizing invigilator contacts instead of preference values. A contact would occur for each pair of supervisors assigned to the same examination room at the same time. To solve this computationally intractable problem, we developed a fix-and-optimize heuristic which will be presented during this talk along with results from the real-life case study at the faculty of Economics and Business Administration of Ghent University, Belgium.

Table 1: FCFS results over 10,000 simulation runs compared to optimal solution.

Approach	Pref./Inv.	Coverage %	Avg. pref.
Optimal	5	75.70	2.59
FCFS	5	[73.07-75.18]	[1.8-1.95]
Optimal	10	96.74	3.85
FCFS	10	[86.49-90.70]	[3.02-3.34]
Optimal	20	100	4.22
FCFS	20	[92.89-97.71]	[4.15-4.86]

References

1. Ahuja, R., Magnanti, T., Orlin, J.: Network flows: theory, algorithms, and applications. Prentice-Hall (1993)
2. Burke, E., Elliman, D., Ford, P., Weare, R.: Examination timetabling in British universities: A survey. In: Burke, E., Ross, P. (eds.) Practice and Theory of Automated Timetabling. pp. 76–90. Springer Berlin Heidelberg, Berlin, Heidelberg (1996)
3. Carter, M.W.: OR Practice – a survey of practical applications of examination timetabling algorithms. Oper. Res. **34**, 193–202 (1986)
4. Ceschia, S., Di Gaspero, L., Schaerf, A.: Educational timetabling: Problems, benchmarks, and state-of-the-art results. Eur. J. Oper. Res. **308**, 1–18 (2022)
5. Çimen, M., Belbağ, S., Soysal, M., Sel, Ç.: Invigilators assignment in practical examination timetabling problems. International Journal of Industrial Engineering **29**, 351–371 (2022)
6. Kahar, M.N.M., Kendall, G.: Universiti Malaysia Pahang examination timetabling problem: scheduling invigilators. Journal of the Operational Research Society **65**(2), 214–226 (2014)

7. McCollum, B., McMullan, P., Burke, E.K., Parkes, E.J., Qu, R.: The second international timetabling competition: Examination timetabling track. Tech. rep., Queen's University, Belfast (UK) (2007)

A Hierarchical Framework for Planning & Control in the Judicial System

I.M.W. (Ieke) Schrader[1] [0009−0004−1052−9327], E.W. (Erwin) Hans[1] [0000−0002−6618−4661], and J.M.J. (Marco) Schutten[1] [0000−0001−5924−223X]

University of Twente, the Netherlands, i.m.w.schrader@utwente.nl

Abstract. The delivery of judicial services by courts involves a labour-intensive service supply chain attributed to a diversity of services within and between the law sectors, the scale of the courts, and the relationships with external organisations. Court operations are planned with the goal of maintaining accessibility for litigants in terms of timeliness and transparency while maintaining a balanced work environment for court staff. Despite current efforts, courts experience low clearance rates and high waiting times.

Given the societal importance of the judicial system and the vast number of operational challenges, it is surprising that so little research has been done within the OR/OM community. To classify the court's planning and control activities, we propose a hierarchical framework. Its aim is to facilitate a common language for research and practice. Concepts from widely used frameworks in domains such as healthcare, manufacturing, and project planning are studied as inspiration.

We use the framework to position the existing literature. In a case study of the Dutch judiciary, we analyse the various planning activities and position them into the framework. This facilitates the development of a research agenda and innovation strategy for the Dutch legal system.

Keywords: Hierarchical decision making, Framework, Resource capacity planning, Judicial System

Performance impact of constraint variants on a MILP formulated hearing scheduling problem

I.M.W. (Ieke) Schrader[1] [0009–0004–1052–9327], E.W. (Erwin) Hans[1] [0000–0002–6618–4661], and J.M.J. (Marco) Schutten[1] [0000–0001–5924–223X]

University of Twente, the Netherlands, i.m.w.schrader@utwente.nl

Keywords: Block Scheduling, Mixed Integer Linear Programming, Skills planning, Judicial System

Keywords: Block Scheduling, Mixed Integer Linear Programming, Skills planning, Judicial System

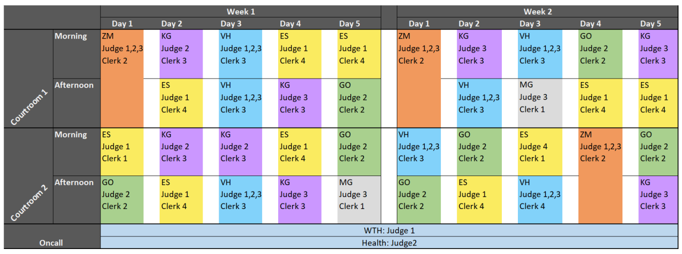
1 Introduction

The courts within the judicial system involve a labour-intensive service supply chain, where planners are faced with the challenge of maintaining accessibility for litigants while balancing the workload of court staff (clerks and judges).

Over the past 50 years, to the best of our knowledge, only 42 articles have been published that adopt operation research and management to court operations. One of the earlier papers,[3], researched the impact of organisational changes on delay for felony defendants through simulation. [2] proposed a two-step ILP model that calculates the optimal number of judges per district and distributes judges over districts by maximising the disposal of cases. [4] developed a strategic model to define court districts and the locations of courts within those districts. [1] proposed a model to schedule sessions and allocate judges to them by minimising the number of violations per judge. This paper is one of the few that shows similarities with our problem.

The central problem of this research is the combined decision of scheduling hearing blocks over time and the allocation of resources to these hearing blocks. When made correctly, the service and case mix the court aims to cover are met, aligning demand and supply. Currently, the court of law experiences difficulties in reaching the agreed case mix. Solving the central problem results in a hearing block schedule, which reserves sufficient capacity for court case groups. Given such a block schedule, cases can be assigned to these blocks in the subsequent operational Case Booking Problem (CBP). In this research, we only focus on the Hearing Scheduling Problem (HSP), part of the tactical level of planning problems in this service supply chain.

Figure 1 shows an example of a non-cyclic hearing block schedule. As can be found in the example, hearing blocks for specific case groups are scheduled on day-parts. These blocks require particular resources in terms of courtrooms and skilled staff members. Some blocks need three judges and one clerk; others need only one judge and one clerk, later referred to as multi-judge or single-judge blocks. Moreover, staff members are allocated to on-call duties required for urgent cases entering the court system.



		Week 1					Week 2				
		Day 1	Day 2	Day 3	Day 4	Day 5	Day 1	Day 2	Day 3	Day 4	Day 5
Courtroom 1	Morning	ZM Judge 1,2,3 Clerk 2	KG Judge 2 Clerk 3	VH Judge 1,2,3 Clerk 3	ES Judge 1 Clerk 4	ES Judge 1 Clerk 4	ZM Judge 1,2,3 Clerk 2	KG Judge 3 Clerk 3	VH Judge 1,2,3 Clerk 3	GO Judge 2 Clerk 2	KG Judge 3 Clerk 3
	Afternoon		ES Judge 1 Clerk 4	VH Judge 1,2,3 Clerk 3	KG Judge 3 Clerk 3	GO Judge 2 Clerk 2		VH Judge 1,2,3 Clerk 3	MG Judge 3 Clerk 1	ES Judge 1 Clerk 4	ES Judge 1 Clerk 4
Courtroom 2	Morning	ES Judge 1 Clerk 1	KG Judge 2 Clerk 3	KG Judge 2 Clerk 3	ES Judge 1 Clerk 4	GO Judge 2 Clerk 2	VH Judge 1,2,3 Clerk 3	GO Judge 2 Clerk 2	ES Judge 4 Clerk 1	ZM Judge 1,2,3 Clerk 2	GO Judge 2 Clerk 2
	Afternoon	GO Judge 2 Clerk 2	ES Judge 1 Clerk 4	VH Judge 1,2,3 Clerk 3	KG Judge 3 Clerk 3	MG Judge 3 Clerk 1	GO Judge 2 Clerk 2	ES Judge 1 Clerk 4	VH Judge 1,2,3 Clerk 4		KG Judge 3 Clerk 3
Oncall		WTH: Judge 1									
		Health: Judge2									

Fig. 1: Example of a hearing block schedule as used in the Court of Law

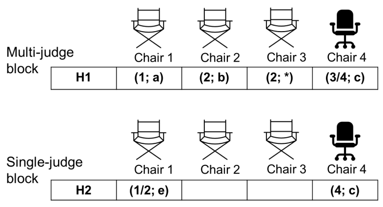


Fig. 2: Skill set (experience level; expertise area) required for chairs during a multi-judge or single-judge blocks.

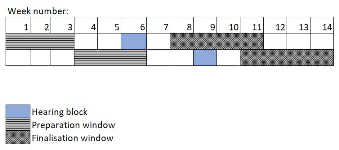


Fig. 3: Execution of desk time activities in time windows.

The capacity and skill set of the staff members determines the maximum number of scheduled hearing blocks. So, when generating the schedule, the assigned staff member must be available and have a suitable skill set. Each staff member has one experience level and could have multiple expertise areas. Figure 2 visualises that each chair part of a hearing block requires a specific combination of experience level and expertise area.

Highly interesting to this problem is the consumption of capacity by desk activities that prepare and finalise cases handled on the hearing blocks. When scheduling a hearing block, the chosen staff members must have sufficient capacity for preparation and finalisation in a specific time window. Figure 3 shows that a block scheduled in week 6 must be prepared in weeks 1 to 3 and finished in weeks 8 to 10. Depending on the case group, preparation and finalisation take between 4 and 10 hours. Desk time activities for different blocks can be parallel executed and staff members decide when this is done during the allowed window.

In the remainder of this extended abstract, we introduce two MILP variants for the desk time activities and we explain our experimental design, which analyses the computational performance of different formulations for the desk time assignment. We conclude with the variant that outperforms the other and further research steps.

2 Model formulation

In this section, we provide important parts of our MILP formulation for the HSP. We have a set of hearing block types $\mathcal{H}$, a set of chairs per type h $\mathcal{I}_h$, a set of courtrooms $\mathcal{R}$, a set of day parts $\mathcal{D}$, and a set of staff members $\mathcal{S}$. In line with the proposed definition for the HSP, the following decisions are made:

$$X_{h,r,d} = \begin{cases} 1, & \text{if hearing block of type } h \text{ is assigned to courtroom } r \text{ on day part } d. \\ 0, & \text{otherwise.} \end{cases}$$

$$Y_{s,h,i,r,d} = \begin{cases} 1, & \text{if staff member } s \text{ is allocated to slot } i \text{ in hearing block of type } h \\ & \text{scheduled in room } r \text{ on day part } d. \\ 0, & \text{otherwise.} \end{cases}$$

The primary objective of the HSP is to maximise the number of hearing blocks over the schedule horizon:

$$\max \sum_{h\in\mathcal{H}} \sum_{d\in\mathcal{D}} \sum_{r\in\mathcal{R}} X_{h,r,d}$$

The decisions are made under the condition that each staff member is assigned at most once per hearing block, courtroom, and day part combinations. Moreover, a hearing block can be allocated once per combination of a day part and courtroom, and only if courtroom r is available on day part d. As Section 1 explains, assigning a staff member with the appropriate skill set for a slot part of the hearing block is key. It is also important that, at most, one eligible staff member is assigned to a chair. At last, when the multi-judge block is required, all slots must be filled with the required staff members. We elaborate on the formulations of these restrictions in the full paper.

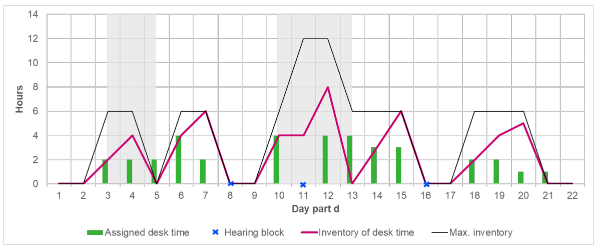


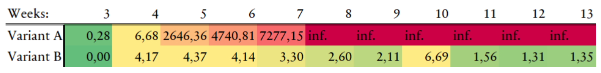
Fig. 4: Concept behind variant B for desk time activities

An interesting condition is the capacity consumption by desk activities associated with an assigned hearing block. We decide to formulate this in two different ways. Variant A assigns desk time in hours to a day part d for each scheduled hearing block without exceeding a staff member's capacity. Variant B is inspired by inventory balance equations and aggregates desk time into an inventory that is consumed when a desk activity needs to be finished. Figure 4 illustrates this concept. For a block scheduled on day part 8, inventory for preparation is increased within the allowed window from day part 3 till 5 and thereafter consumed on day part 5.

3 Solution method and experimental design

Our experiments focus on the computational performance of different formulations for desk time assignments, using generated instances based on real-life data. We generated 10 instances per instance class. Each class has a different schedule period, which increases from 3 to 15 weeks. A maximum of 15 weeks is chosen, because the court of law uses quarterly schedules. Between individual instances within classes, the staff member availability per day part is uniformly distributed between 2 and 4 hours. All the other parameters remain similar between individual instances and instance classes.

The MILP is implemented in AIMMS and solved by GUROBI 11.0.1 with a Lenovo Thinkpad with an Intel(R) Core(TM) i7-6700HQ CPU @ 2.600GHz 2.6 GHz and 16GB RAM CORE i7.



Weeks:	3	4	5	6	7	8	9	10	11	12	13
Variant A	0,28	6,68	2646,36	4740,81	7277,15	inf.	inf.	inf.	inf.	inf.	inf.
Variant B	0,00	4,17	4,37	4,14	3,30	2,60	2,11	6,69	1,56	1,31	1,35

Fig. 5: Integrality GAP after 300 sec.

Figure 5 compares the average integrality GAP after solving for 300 seconds between variant A and B. GUROBI cannot find a solution for variant A for instances larger than 7 weeks. Variant B outperforms variant A, since GUROBI finds a solution with a smaller integrality GAP for all instance classes. When comparing the variants, the difference in integrality gap can be declared by the strong increase in number of constraints and variables as shown in Figure 6.

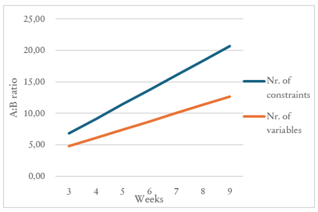


Fig. 6: Increase of ratio A:B

4 Conclusion and Further steps

Our contribution is two-fold: first, we introduced a MILP formulation of the underexposed HSP, and second, we analysed its computational performance under various constraint formulations. Our results show an improved computational performance when modelling desk time assignment as an inventory balance equation.

However, improvement of the modelling approach is still possible since the generated schedule shows an imbalanced spread over time in the hearing blocks scheduled per type. Therefore, our current steps focus on generating a schedule in which blocks are spread over the horizon. This is done by extending the objective with a secondary objective function incorporating a spread measurement. In our presentation, we will provide the MILP formulation more extensively and discuss the preliminary results of our next steps.

References

1. Brooks, J.P.: The court of appeals of virginia uses integer programming and cloud computing to schedule sessions **42**(6), 544–553. https://doi.org/10.1287/inte.1110.0598, https://www.scopus.com/inward/record.uri?eid=2-s2.0-84873652142&doi=10.1287%2finte.1110.0598&partnerID=40&md5=1b4362e857d4eb8287dc0f80c85ac695
2. Gupta, M., Bolia, N.B.: Redistribution of judicial resources for improved performance . https://doi.org/10.1007/s10479-023-05389-0, https://doi.org/10.1007/s10479-023-05389-0
3. Taylor, J.G., Navarro, J.A.: Simulation of a court system for the processing of criminal cases **10**(5), 235–240. https://doi.org/10.1177/003754976801000505
4. Teixeira, J.C., Bigotte, J.F., Repolho, H.M., Antunes, A.P.: Location of courts of justice: The making of the new judiciary map of portugal **272**(2), 608–620. https://doi.org/10.1016/j.ejor.2018.06.029, https://linkinghub.elsevier.com/retrieve/pii/S0377221718305642

Sports timetabling: towards generic algorithms and performance insights

David Van Bulck[1,2][0000−0002−1222−4541] and Dries Goossens[1,2][0000−0003−0224−3412]

[1] Ghent University, Ghent, Belgium
[2] Core lab CVAMO, FlandersMake@UGent, Belgium
{david.vanbulck,dries.goossens}@ugent.be

Keywords: Sports scheduling, Algorithm benchmarking, Instance Space Analysis, ITC2021.

1 Introduction

For decades, sports timetabling has been a case-study driven field, with researchers developing tailor-made algorithms. The value of these case studies notwithstanding, the lack of a framework for algorithm benchmarking made it difficult to compare timetable requirements and algorithm performance across different studies. With the recent introduction of RobinX [4] and the International Timetabling Competition 2021 (ITC2021, [2]), a unified file format and a set of common benchmark instances is now available. Even though ITC2021 has paved the way for more generic algorithms it also indicates that, depending on the specifics of the sports tournament, some types of algorithms may be more suitable than others. This extended abstract is based on Van Bulck et al. [3] and demonstrates how to use techniques from instance space analysis to predict which algorithm is most suited for a given sports timetabling application. Our results are based on large computational experiments involving about 50 years of CPU time on more than 500 newly generated problem instances.

2 The International Timetabling Competition 2021

The task in ITC2021 is to construct a compact timetable for a double round-robin tournament, meaning that each team faces every opponent twice, both at home and away, with exactly one game per round. Constraints are categorized as hard or soft: hard ones are fundamental and cannot be broken, while soft ones are preferences. The objective is to respect as many soft constraints as possible, while adhering to all hard constraints. Nine constraint types are considered, grouped into four classes (see also [4]).

Capacity constraints regulate when teams play at home or away. We consider four types: CA1, CA2, CA3, and CA4.

Break constraints regulate the number of breaks (i.e., consecutive home or consecutive away games) in the timetable. We consider two types: BR1 and BR2.

Game constraints enforce or forbid specific assignments of games to rounds. We consider only one type: GA1.

Fairness and separation constraints are always soft and only two types are considered. FA1 limits the maximal difference in home games played by any two teams at any point in the season, whereas SE1 requests a minimal number of rounds between games with the same opponents.

In addition, some problem instances require a 'phased' timetable, meaning that each team plays against every other team once before any rematch takes place. ITC2021 offers a set of 45 problem instances, which are all expressed in the standard file format of RobinX. The number of teams varies from 16 to 20. Instances and best solutions are available from the competition website at itc2021.ugent.be.

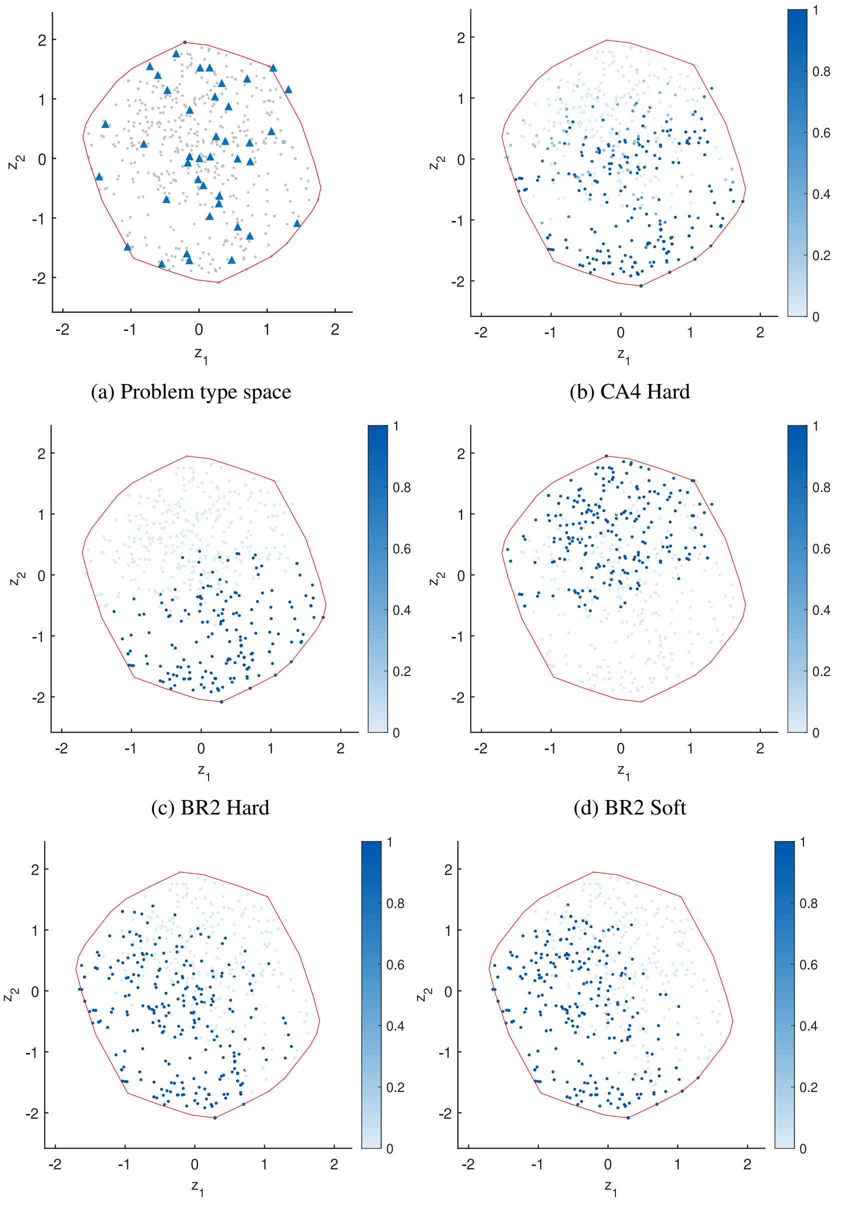
3 A Problem Type Analysis for ITC2021

Since the set of 45 problem instances from ITC2021 is rather limited to apply machine learning tools, we used the generator from [2] resulting in a diverse set of 518 additional instances. This was done in such a way that the instances are well scattered in the so-called two-dimensional (2D) problem type space (see Figure 1a). The axes of this 2D space correspond to linear combinations of the number and type of constraints (i.e., the problem type) present in each instance, combined in such a way that the performance of the algorithms linearly varies over the 2D space.

In order to get more insights into the composition of the problem type space, Figure 1 also shows the distribution of some prominent constraint types over the space. Besides, Figure 2 visualizes the regions where some of the algorithms we consider are expected to perform well. Goal and UoS are matheuristics (fix-and-optimize and variable neighborhood descent), FBHS shares similarities with the well-known first-break-then-schedule decomposition method, and Udine is a simulated annealing metaheuristic. For more details, see Van Bulck et al. [3].

Comparing the distribution of the problem type characteristics (cf., Figure 1) with the algorithm footprints (cf., Figure 2), we derive the following insights. First, the footprint of Udine shows that the algorithm not only finds a feasible solution for the majority of the problem instances, but also that the solutions found are of high quality. However, near the top of the instance space where there are many BR2 soft constraints, FBHS seems more promising. Finally, for instances near the middle-left to bottom-right diagonal, Goal and UoS are a suitable choice. As none of the problem type characteristics dominate in this region, these instances could be considered as 'average instances'. On the other hand, all algorithms struggle to find feasible solutions near the bottom of the space where the 'hardest instances' from our dataset are located. This part of the problem type space corresponds to problems that are phased or have SE1 soft constraints, in combination with many BR2 and CA4 hard constraints. On the other hand, near the middle of the space, and especially near the middle left, there are several instances for which multiple or even all algorithms find a good solution. Based on Figure 1, this area is characterized by the lack of BR2 hard and soft constraints.

Finally, we note that we can also use the coordinates of the problem instances in the 2D problem-type space as input to a machine learning model to predict which algorithm is expected to perform best. The results of these predictions, made with the



(a) Problem type space

(b) CA4 Hard

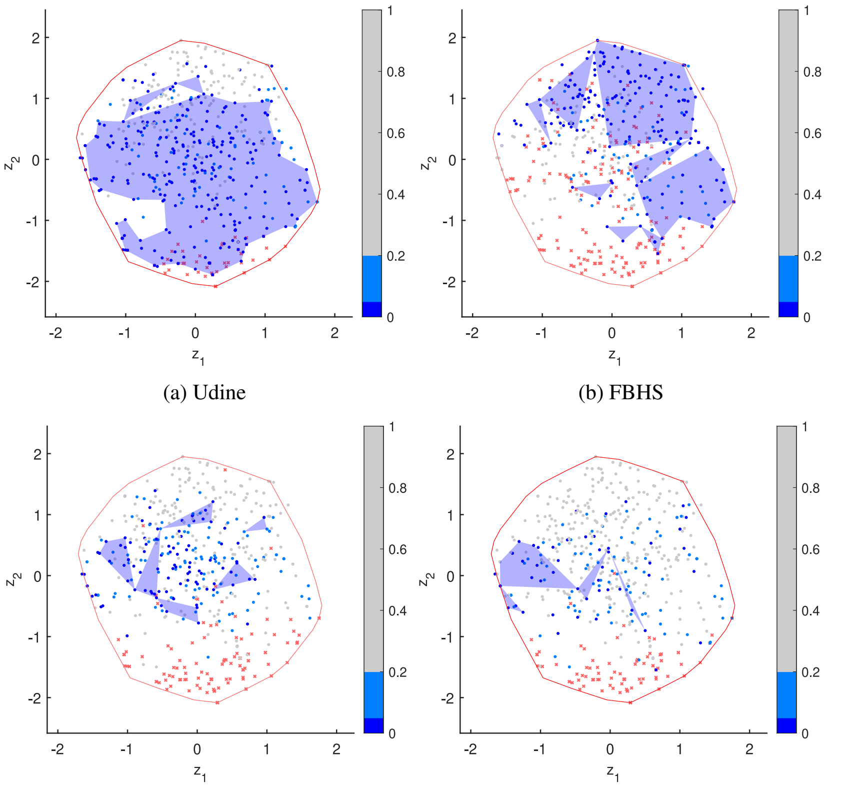
(c) BR2 Hard

(d) BR2 Soft

(e) SE1 Soft

(f) Phased

Fig. 1: Distribution of the number of constraints for some prominent constraint types over the so-called problem type space constructed with the ISA toolkit [10]. In (a), blue triangles denote the 45 original ITC2021 instances, whereas grey dots denote the 518 additional instances.



(a) Udine (b) FBHS

(c) Goal (d) UoS

Fig. 2: Algorithm footprints illustrating regions where the algorithm is expected to excel (blue regions, as identified by the ISA-toolkit [10]). Colours denote the relative gap compared to the best solution found by any of the algorithms, with red x-marks indicating instances for which no feasible solution was found.

ISA toolkit [10], and the quality of the predictions in terms of the average relative gap with regard to the best solution found by any of the algorithms can be found in Figure 3. Using the recommended algorithm for each instance, we are able to reduce the average relative gap for the Udine solver (the single-best algorithm) from 12.8% to just 4.93% (measured over a set of unseen test instances; always predicting the best algorithm results in a gap of 0%). This lets us conclude that we are able to effectively predict which algorithm a practitioner should use, when given the type of constraints typically present in the sports competition under consideration.

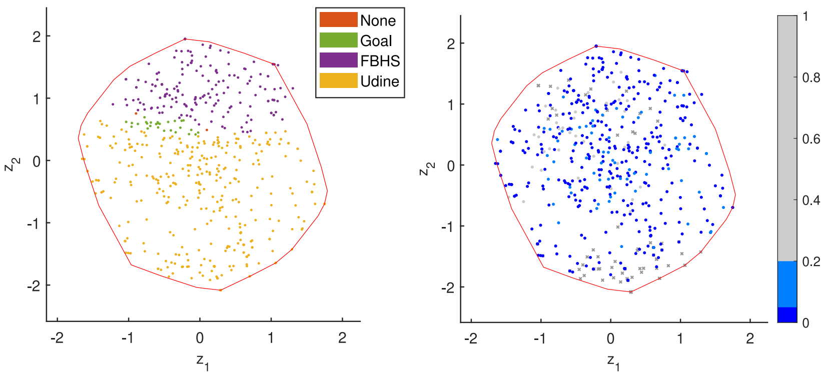


Fig. 3: Algorithm recommendations by the ISA toolkit [1] and performance thereof using as features the 2D coordinates in the problem type space. UoS was never predicted to be best.

References

1. Smith-Miles, K., Muñoz, M.A.: Instance space analysis for algorithm testing: Methodology and software tools. ACM Comput. Surv. **55** (2023)
2. Van Bulck, D., Goossens, D.: The international timetabling competition on sports timetabling (ITC2021). Eur. J. Oper. Res. **308**, 1249–1267 (2023)
3. Van Bulck, D., Goossens, D., Clarner, J., Dimitsas, A., Fonseca, G.H.G., Lamas-Fernandez, C., Lester, M.M., Pedersen, J., Phillips, A.E., Rosati, R.M.: Which algorithm to select in sports timetabling? Eur. J. Oper. Res. (2024), https://doi.org/10.1016/j.ejor.2024.06.005, in press
4. Van Bulck, D., Goossens, D., Schönberger, J., Guajardo, M.: RobinX: A three-field classification and unified data format for round-robin sports timetabling. Eur. J. Oper. Res. **280**, 568 – 580 (2020)

Optimized automated university timetabling with Covid-19 social distancing restrictions

Efstratios Rappos and Pier Donini

Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud (HEIG-VD), University of Applied Sciences of Western Switzerland (HES-SO), Yverdon-les-Bains, Switzerland
{efstratios.rappos,pier.donini}@heig-vd.ch

Abstract. This extended abstract article describes the impact of the COVID-19 health restrictions in the teaching activities at the HEIG-VD university in Switzerland and the way that teaching activities could be maintained by using a custom timetabling algorithm incorporating the necessary social distancing health measures. The modified algorithm, using a mixed-integer program model, produced a new timetable and at the same time automatically selected the optimal mode of delivery for each lecture: remotely by video conference or physically in the classroom. The optimization model ensured that physical contact among students was minimized, while at the same time guaranteed that the courses or laboratories requiring physical presence were still able to take place.

Keywords: Educational timetabling, mixed integer programming.

1 Introduction

University timetabling is a well-studied optimization problem in the academic literature [3,12,6,5,9] and a large volume of research has looked at every aspect of academic timetable production. The COVID-19 health crisis added new challenges in timetable construction, one of which is presented here.

The onset of the COVID-19 crisis had an impact on all aspects of life where social interaction is required, including education. In particular, during the spring of 2020, schools and universities in Switzerland, including ours, were obliged to restrict the use of classrooms and lectures were delivered online using videoconferencing. The only exceptions were a limited number of classes requiring specialized equipment or laboratories as well as the final exams, which were allowed to take place in the classrooms with protective measures.

The planning for the following academic year 2020–21 was subject to a number of COVID-19 social distancing restrictions, which are summarized as follows: (i) any lectures requiring specialized equipment or laboratories must take place physically at the university buildings, (ii) the remaining lectures may be given either online or at the university, and (iii) students can only be physically present at the university for a maximum of three days a week. The idea was to allow students at the university for a limited number of days, whilst optimizing the mode of delivery of each course. To implement this, each lecture was assigned a weight, ranging from 3 for a lecture which must be given in the classroom, 2 for a lecture ideally given in a classroom and 1 for lectures best suited for online lecturing. Some courses could have lectures of two types, for example, for Physics the practical experiments (one lecture a week) required physical presence, whereas the theoretical lectures (twice a week) could be given online.

The requirements created a challenging timetabling problem. We had to place the lectures which must take place in the classroom during a maximum of three days a week for each student (noting that some courses are shared between diplomas or orientations, so these three days can be different for each student) while filling in the gaps in the timetable using other courses which can be delivered in either mode. Conversely, all courses planned during the two days (at a minimum) during which a student is not at the university must take place online.

Note that the mode of teaching for each course must be the same for all students who follow it, so it is not possible for the same lecture to be "in the classroom" for some students and "online" for others. However, we assigned rooms to online lectures also: these rooms were used by students as a space to connect to an online lecture if they had to be at the premises for other lectures, and also in case the health measures were abandoned, enabling a return to classrooms.

At the same time the student population had to be assigned by the algorithm to the different parallel classes, since most courses are given multiple times (up to eight times) in order to ensure that classes contain no more than 25 to 30 students. Students are assigned to parallel classes each semester at the time of creation of the timetable. Furthermore, the usual quality constraints had to be maintained: avoid having gaps in the student timetables, distribute lecture of the same course over a number of days and so on.

If we consider the recent literature, it is clear that we were not the only university facing such challenges. Various COVID-19 timetabling challenges are studied in [1,8,13,4,2,7] although in many cases the size of the timetable considered is much smaller that the problem described here.

2 Mathematical modeling

The timetable for undergraduate engineering degrees at the HEIG-VD university is produced with an in-house software which models the timetabling problem as a mixed-integer program, similar to [10]. In particular this model calculates three elements, modeled as binary 0-1 decision variables: the start time of each class, the rooms allocated for each class and the assignment of classes to each student.

In order to model the COVID-19 requirements we introduced a fourth element to be calculated by the optimization model: a variable D_l, shared by all students, denoting whether the class takes place in the classroom or via videoconferencing. For every lecture l to be placed on the timetable:

$$D_l = \begin{cases} 1 & \text{if the lecture } l \text{ takes place via videoconference (online),} \\ 0 & \text{if the lecture } l \text{ takes place in a university classroom.} \end{cases}$$

With regards to the additional constraints imposed on the student timetable, an additional variable per student and per day was added, denoting if a student is entirely online that day (all his courses on that particular day are "via videoconferencing") and therefore not present in any university building, or not (at least one course he has to follow takes place in a classroom). This is modeled as follows: for every student s and day j we define:

$$V_{sj} = \begin{cases} 1 & \text{if all the lectures followed by student } s \text{ during the day } j \text{ are online,} \\ 0 & \text{if at least one lecture followed by } s \text{ takes place at the university.} \end{cases}$$

The addition of these two types of decision variables allowed us to formulate all the necessary COVID-19 timetabling constraints and define the additional terms of the objective function. It is possible for a student to have lectures given in a classroom and online on the same day. In those cases the room assigned to the online lecture could be used by students (and sometimes lecturers) as a space to connect to the lecture online if desired.

The implementation was done by adding the additional variables and constraints to the existing timetabling mixed-integer programming model used by the university, which already includes over 40 types of timetabling constraints and solved with a two-stage algorithm very similar to [11]. The software is programmed in C++ using a commercial version of Cplex as the mixed-integer programming solver and typically takes several days to solve due to the sequential addition of the constraints and the need to remove some default constraints for some courses when infeasibility occurs.

The addition of the COVID-19 constraints did not have a big impact on the overall complexity or time needed to produce a solution of acceptable quality, however the final timetable produced placed lectures at very different locations to our standard timetable without any COVID-19 constraints. The characteristics of the timetabling problem are shown in Table 1. The number of courses denotes the total number of times the same subject is given (parallel courses), each student class is assigned to only one course for each subject it follows. The number of lectures is the number of events to be scheduled each week.

Table 1: Characteristics of the timetabling problem.

Number of students	962 (405 classes)
Number of different subjects	239
Number of different courses	377
Number of lectures (lessons) per week	709
Number of teaching staff	200
Number of rooms	87
Number of time slots	10 per day × 5 days
Duration of each lecture	2, 3 or 4 slots
Average size of each lecture	18.6 students

The results of this approach were very positive. Thanks to our ability to develop this mixed-integer timetabling model during the summer of 2020, we were able to construct a timetable for 2020–21 which reduced the number of students present at the university by 40% from an average of 840.6 students to 501.4 students present per day (Table 2). At the same time, we were able to place at least one lecture per week of every subject during the three days the students were present, a positive outcome, ensuring that contact between students and teachers was maintained, and activities such as tests could take place in the classroom during the semester.

Table 2: Number of students present and courses taking place in university buildings

Usual timetable	*Mon*	*Tue*	*Wed*	*Thu*	*Fri*	
Students physically present:	893	894	798	874	744	**840.6 (average)**
Lectures held in classrooms:	158	157	144	152	98	**709 (total)**
Modified Covid timetable	*Mon*	*Tue*	*Wed*	*Thu*	*Fri*	
Students physically present:	636	513	465	661	232	**501.4 (average)**
Lectures held in classrooms:	107	88	80	111	38	**424 (total)**

3 Conclusion

This paper presented the work carried out during the height of the COVID-19 crisis at the HEIG-VD university in Switzerland to guarantee the best teaching outcomes for the undergraduate engineering courses. Using a modification of the actual mixed-integer programming timetabling model we were able to incorporate the necessary social distancing health restrictions. This produced a timetable for the 2020–21 academic year which took into account the requirement to limit the number of days the students are present at the university buildings, while at the same time guaranteeing that courses requiring specialized laboratory equipment were able to take place in person.

References

1. Al-Tarawneh, H., Al-Kaabneh, K., Alhroob, A., Migdady, H., Alhadid, I.: A hybrid heuristic algorithm for solving COVID-19's social distancing at universities campus. Computer Systems Science & Engineering **41**(3) (2022)
2. Barnhart, C., Bertsimas, D., Delarue, A., Yan, J.: Course scheduling under sudden scarcity: Applications to pandemic planning. Manufacturing and Service Operations Management **24**, 727–745 (2021). https://doi.org/10.1287/msom.2021.0996
3. Burke, E.K., Petrovic, S.: Recent research directions in automated timetabling. European Journal of Operational Research **140**(2), 266–280 (2002)
4. Cardonha, C., Bergman, D., Day, R.: Maximizing student opportunities for in-person classes under pandemic capacity reductions. Decision Support Systems **154**, 113697 (2022). https://doi.org/10.1016/j.dss.2021.113697
5. Ceschia, S., Di Gaspero, L., Schaerf, A.: Educational timetabling: Problems, benchmarks, and state-of-the-art results. European Journal of Operational Research **308**(1), 1–18 (2023). https://doi.org/10.1016/j.ejor.2022.07.011
6. Chen, M.C., Sze, S.N., Goh, S.L., Sabar, N.R., Kendall, G.: A survey of university course timetabling problem: Perspectives, trends and opportunities. IEEE Access **9**, 106515–106529 (2021). https://doi.org/10.1109/ACCESS.2021.3100613

7. Davison, M., Kheiri, A., Zografos, K.: Optimising scheduling of hybrid learning using mixed integer programming. In: De Causmaecker, P., Özcan, E., Vanden Berghe, G. (eds.) Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling - PATAT 2022. pp. 279–286 (2022)
8. Hoshino, R., Fabris, I.: Partitioning students into cohorts during COVID-19. In: Stuckey, P.J. (ed.) Integration of Constraint Programming, Artificial Intelligence, and Operations Research. pp. 89–105. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-78230-6
9. Müller, T., Rudová, H., Müllerová, Z.: Real-world university course timetabling at the international timetabling competition 2019. Journal of Scheduling (2024). https://doi.org/10.1007/s10951-023-00801-w
10. Rappos, E., Thiémard, E., Robert, S., Hêche, J.F.: International timetabling competition 2019: A mixed integer programming approach for solving university timetabling problems. In: De Causmaecker, P., Özcan, E., Vanden Berghe, G. (eds.) Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling - PATAT 2021. pp. 357–360 (2021)
11. Rappos, E., Thiémard, E., Robert, S., Hêche, J.F.: A mixed-integer programming approach for solving university course timetabling problems. Journal of Scheduling **25**, 391–404 (2022)
12. Rudová, H., Müller, T., Murray, K.: Complex university course timetabling. Journal of Scheduling **14**, 187–207 (2011). https://doi.org/10.1007/s10951-010-0171-3
13. Ulucan, A., Atici, K.B., Sarac, S.B.: A university-wide orientation course timetabling model and its modification for pandemic period. OPSEARCH **60**, 1575–1602 (2023). https://doi.org/10.1007/s12597-023-00675-8

An efficient algorithm for the truck driver scheduling problem

Niels De Walsche[0009−0003−0643−4352], Greet Vanden Berghe[0000−0002−0275−5568], and Pieter Smet[0000−0002−3955−7725]

KU Leuven, Ghent Campus, Gebroeders De Smetstraat 1, 9000 Ghent, Belgium
{niels.dewalsche,greet.vanden.berghe,pieter.smet}@kuleuven.be

Abstract. In a world where the wellbeing of drivers is, quite rightly, receiving more attention, algorithms that can efficiently create schedules that comply with the regulations are becoming increasingly important. This extended abstract investigates how to schedule long-distance truck drivers in accordance with the European Union's Hours-of-Service regulations. We introduce a vehicle routing algorithm that takes into account both the working hours of drivers and time windows of customers. The heuristic algorithm outperforms the current state-of-the-art approach for a public data set.

Keywords: Truck Driver Scheduling Problem, Hours-of-Service regulations, Vehicle scheduling

1 Introduction

In recent years, there has been an increased focus on employee wellbeing. This is particularly relevant for truck drivers, who are often on the road for long periods of time. The European Union has imposed regulations to ensure the safety and wellbeing of such drivers, namely (EC) No. 561/2006. These regulations are also important for drivers' employers, who risk significant penalties if their drivers are not compliant. Goel and Vidal [3] highlighted the fact that adopting hours-of-service regulations can more than double the total travel duration. It is therefore crucial for companies to carefully optimize the routes of their drivers.

The truck driver scheduling problem (TDSP), introduced by Goel [2], aims at creating schedules that comply with the aforementioned European Union regulations. Ensuring compliant schedules represents a crucial component of the Vehicle Routing and Truck Driver Scheduling Problem (VRTDSP), where the goal is to find routes for the trucks and schedules for the drivers that minimize both the total distance traveled and the number of drivers.

2 Problem description

A solution of the TDSP, consists of a sequence of distinct activities, that describe what a driver is doing at certain points in time: service, driving, break and rest. Feasible schedules must comply with a number of constraints:

- The driver must carry out a service activity at each customer location, the execution of which is restricted by a time window.
- The time it takes to travel between two customers is equal to the total driving time, which may be interrupted by breaks or rests.
- Compliance with the European Union regulations, namely:
  - A driver must take a break of at least 45 minutes if they have driven for 4.5 hours since the end of their last break or rest period.
  - A driver must take a rest of at least 11 hours if they have driven for 9 hours since the end of their last rest period.
  - A driver must rest for at least 11 hours within 24 hours of the end of their last rest period.

3 Proposed algorithm

Our truck driver scheduling algorithm is based on the multilabel method introduced by Goel [2]. Compared to the original method, we have made several modifications, such as (1) an improved labeling strategy that finds a higher number of feasible schedules and (2) an effective pruning strategy that reduces the number of schedules that have to be checked for feasibility. The vehicle routing component of the problem is solved by the SISRs algorithm introduced by Christiaens and Vanden Berghe [1]. SISRs suggests routes for the drivers and our algorithm determines whether an EU-compliant schedule can be generated for these routes.

4 Preliminary results

The performance of the proposed algorithm is evaluated using the VRTDSP instances introduced by Goel [2]. These instances are based on the Vehicle Routing Problem with Time Windows instances of Solomon [4] and modified for the VRTDSP. The instances are divided into three geographical layouts: random (R), clustered (C) and random-clustered (RC). Each geographical layout consists of two types of instances: type 1 and type 2 (e.g. R1 & R2). Type 2 instances have a higher vehicle capacity and longer time windows compared to type 1 instances.

We compare our results against the Hybrid Genetic Search with Adaptive Diversity Control (HGSADC) algorithm proposed by Goel and Vidal [3], which is currently the best performing heuristic algorithm for the VRTDSP. Given that our algorithm does not consider split breaks, split rests or extended driving times, the results are compared against the "No split" version of HGSADC. Similar to Goel and Vidal [3], we solve the problem in accordance with a hierarchical objective, where minimizing the number of drivers is prioritized over minimizing the total distance traveled. Note that the number of drivers determines the fleet size. Also note that the results for HGSADC are directly taken from the original paper and thus there is a difference in the hardware used for the experiments.

The results in Table 1 are presented in terms of the average and fewest number of drivers in addition to the average and shortest total distance traveled. Column $T_{Avg.}$ details the average execution time of our algorithm in seconds, averaged over five runs per instance. The algorithm ran for 50000 iterations, with 10% of the iterations in the fleet minimization phase. A time limit of 1 hour was set, again with a maximum of 10% of the time spent in the fleet minimization phase.

The algorithm was implemented in Rust (1.78.0). All our experiments were conducted on a computer with an AMD Ryzen 7 5800X processor at 3.8 GHz with 8 cores, 32GB of RAM and Windows 10 operating system.

Table 1: Results for the Goel [2] instances

	Our algorithm					HGSADC [3]				
	Avg. Fleet	Avg. Dist.	Best Fleet	Best Dist.	$T_{Avg.}$	Avg. Fleet	Avg. Dist.	Best Fleet	Best Dist.	$T_{Avg.}$
R1	98.60	11 860.77	**98.00**	**11 810.67**	7.09	98.80	11 769.13	98.00	11 835.89	–
R2	55.00	10 807.90	**54.00**	**10 694.42**	27.90	62.60	10 294.36	62.00	10 279.25	–
C1	90.00	7618.23	**90.00**	**7609.33**	3.23	90.40	7630.25	90.00	7628.73	–
C2	35.00	5486.92	**35.00**	**5440.08**	15.56	40.00	5754.04	40.00	5753.30	–
RC1	72.00	9101.35	72.00	8918.92	5.84	72.00	8915.07	**72.00**	**8892.74**	–
RC2	47.00	9204.53	**45.00**	**9220.92**	12.78	50.00	8960.99	50.00	8917.25	–
All	397.60	54079.70	**394.00**	**53694.33**	12.40	413.80	53323.84	412.00	53307.16	3240

The first observation we can make from from Table 1 is that our algorithm performs well when considering the primary objective: minimizing the number of drivers required, especially for the type 2 instances. Moreover, despite the reduction concerning the number of drivers, the total distance traveled remains close to the values produced by the HGSADC algorithm. Finally, it is also worth observing the significant variation concerning average execution time between type 1 and 2 instances.

By the time of our presentation at PATAT, we will present results regarding the completeness of our algorithm. Put another way: what percentage of feasible schedules does the algorithm determine to be feasible. We also aim to gain further insight into the results by investigating the difference in difficulty between the type 1 and 2 instances.

This research was supported by KU Leuven IOF (C2E/23/012: Human-centred decision support based on a new theory for personnel rostering).

References

1. Christiaens, J., Vanden Berghe, G.: Slack induction by string removals for vehicle routing problems. Transportation Science **54**(2), 417–433 (2020)
2. Goel, A.: Vehicle scheduling and routing with drivers' working hours. Transportation Science **43**(1), 17–26 (2009)
3. Goel, A., Vidal, T.: Hours of service regulations in road freight transport: An optimization-based international assessment. Transportation science **48**(3), 391–412 (2014)
4. Solomon, M.M.: Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations research **35**(2), 254–265 (1987)

Combining Aircraft Maintenance Routing with a Distribution Objective

Lucas Kletzander[1] [0000-0002-2100-7733], Ida Gjergji[2] [0000-0002-2970-1535], and Nysret Musliu[1] [0000-0002-3992-8637]

[1] Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling, DBAI, TU Wien
[2] TU Wien, Vienna, Austria
{lucas.kletzander,ida.gjergji,nysret.musliu}@tuwien.ac.at

Keywords: Aircraft Maintenance Routing, Workload Distribution, Constraint Optimization

Scheduling airline operations is a very challenging and demanding task situated in a competitive market with high operational cost and strong regulations [3]. The scheduling process is typically divided into multiple steps due to the overall complexity, which are the flight scheduling problem (FSP), the fleet assignment problem (FAP), the aircraft maintenance routing problem (AMRP), and finally the crew scheduling problem (CSP).

This line of work deals with the AMRP, where the flight legs and the type of aircraft to fly these legs are already fixed, but the specific aircraft (usually identified by its tail number) still needs to be assigned. Since aircraft need a range of different maintenance operations that depend both on the sequence of flights and their location, the assignment of flight legs and maintenance operations is typically combined in the Aircraft Maintenance Routing Problem. There are some recent reviews dealing with this problem [8,9], and a large body of work exists, going back to early work with reduced problem formulations [4]. Capacity constraints were rarely considered, or only with upper limits [3]. Solution methods include heuristic techniques [5], network-based MIP formulations [6,3], and branch-and-price techniques [7]. Heuristic approaches often outperform exact approaches on instances of realistic scale [1,2].

However, as already highlighted by Eltoukhy et al. [3] regarding future research directions, the AMRP rarely includes more considerations on how the maintenance workers are utilised. However, this can have very negative effects on the workforce, leading to both times with very high peak workloads, creating stress and fatigue, while other times are underutilised, leading to excess costs. Therefore, one major contribution of this work is to propose a maintenance distribution objective that aims to evenly distribute maintenance work or can also be easily adapted to other given maintenance target distributions.

Problem Definition We work on a version of the problem where the routing aspect is in the background since there is only one major hub that does the maintenance for all aircraft. Therefore, outgoing and consecutive incoming flight legs are combined, and each resulting (longer) flight leg starts and ends at the same hub. However, we want to extend our problem variant with maintenance distribution to multi-hub problems in the future.

A set of n flight legs $T = \{t_1, \dots, t_n\}$ is given, each leg i is associated with a start time s_i, an end time e_i, and flight time f_i.

Note that there are different ways to specify aircraft maintenance tasks, often in the notion of checks of type A, B, C, and D. This version of the problem uses a different notion, but the general ideas are independent of the specific requirements. Each aircraft needs the following three types of maintenance:

- Routine: The aircraft can fly for at most 47 hours after the end of the previous routine maintenance, then it needs routine maintenance taking 2.5 hours before taking off again.
- Weekly: The aircraft can fly for at most 156 hours after the end of the previous weekly maintenance, then it needs weekly maintenance taking 7 hours before taking off again. Weekly includes routine 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 950 hours of cumulative flight time since the last maintenance of the same type, the aircraft needs major maintenance taking 14 hours before taking off again. Each type of major maintenance includes routine and weekly maintenance. Further, the types `MH1` and `MH2` require the hangar, meaning that only 1 aircraft can perform any of these two types of maintenance at once.

A set of m aircraft $A = \{a_1, \dots, a_m\}$ is given, each aircraft j is associated with history regarding the last flight leg and last maintenance tasks.

A feasible solution assigns all flight legs to available aircraft, and the required types of maintenance to specific time intervals, such that:

- The flight history is respected.
- No overlapping legs and tasks are assigned to any aircraft.
- No maintenance intervals are violated.
- At most 1 aircraft is assigned `MH1` or `MH2` at any point in time.

For optimisation, the total number of aircraft in any type of maintenance m_k is calculated for each minute k in the planning period, and $\sum_k m_k^2$ is minimised. This optimises the distribution of maintenance tasks (since peaks are penalised more), and also the total amount of maintenance (leading to maximisation of the available intervals between maintenance tasks). In addition, for each major maintenance the difference to the maximum flight time is added to prevent the maintenance from being done too soon.

Instance Generator One of the issues in the area of aircraft scheduling is that typically instance data is not publicly available, since airlines do not want to reveal details about their operation. However, this creates a difficult situation for scientific comparison, as this makes a thorough comparison of different methods very difficult, if not impossible.

Therefore, while we can also not share real-life data, we developed a flexible instance generator for these kinds of problems with several features to recreate characteristics of real-world instances:

- Instance size: Configurable number of aircraft, time horizon, granularity of flight times, and length of flight legs.

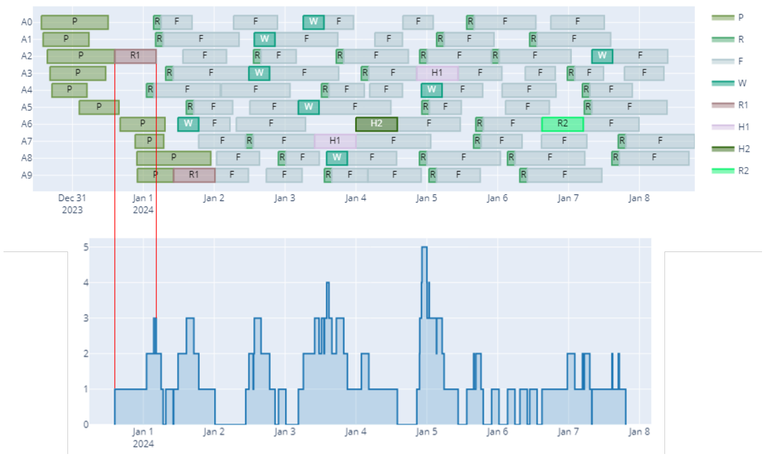


Fig. 1: Solution visualization: Aircraft on top, combined maintenance below

- Density: The density of the instances can be configured in several ways. A density of 1 represents a schedule where there is no idle time of aircraft besides flight legs and maintenance, leading to instances where it is difficult to find a feasible solution. Lower density leads to instances with more idle time where it might be easier to find feasible solutions, therefore, more emphasis can be put on the ideal distribution of maintenance. There are also options for up- or down-sloping density during the planning horizon.
- Peak behaviour: In real life, there are demand peaks during the day, e.g., for long distance flights many of them might start around 10 to 11 am. Peak intensity can therefore be chosen using parameters.

Instances are generated around a feasible, but most likely not optimal solution. We will further extend the generator to allow the generation of instances with different maintenance requirements. The current state of the generator as well as the new set of benchmark instances are available online[3].

Solution Methods We provide a model in the constraint modeling language MiniZinc for this problem. The decision variable $assignment_i$ shows the aircraft to which each leg i is assigned to. As there is at most one of each type of major maintenance per scheduling horizon, the major maintenance tasks are captured with a decision variable that denotes the start time per type for $j \in A$. Given that routine and weekly maintenance tasks can happen multiple times, the start times for these tasks are presented with optional decision variables. Additionally, we use the disjunctive constraint to ensure that no overlapping flights are assigned to any aircraft or tasks to the hangar at any given point in time.

[3]https://cdlab-artis.dbai.tuwien.ac.at/papers/amr-md/

We have also been working on a solution method using Simulated Annealing, where moves either reassign flight legs to different aircraft, or change the location of a maintenance task within an aircraft.

Preliminary Results We created a new set of benchmark instances ranging from 10 aircraft for one week up to 50 aircraft for a month with various density and peak options. First results indicate MiniZinc together with the solver OR-Tools can solve small instances in short time, and good results can be obtained with Simulated Annealing for larger instances. Compared to pure maintenance assignments (as late as possible), peaks can be reduced significantly while keeping the same quality of aircraft schedules, showing the usefulness of our extension.

Acknowledgements 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 partially 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/).

References

1. Başdere, M., Bilge, Ü.: Operational aircraft maintenance routing problem with remaining time consideration. European Journal of Operational Research **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. Computers & Industrial Engineering **137**, 106045 (2019)
3. Eltoukhy, A.E., Chan, F.T., Chung, S.H.: Airline schedule planning: a review and future directions. Industrial Management & Data Systems **117**(6), 1201–1243 (2017)
4. Feo, T.A., Bard, J.F.: Flight scheduling and maintenance base planning. Management Science **35**(12), 1415–1432 (1989)
5. Gopalan, R., Talluri, K.T.: The aircraft maintenance routing problem. Operations research **46**(2), 260–271 (1998)
6. Haouari, M., Shao, S., Sherali, H.D.: A lifted compact formulation for the daily aircraft maintenance routing problem. Transportation Science **47**(4), 508–525 (2013)
7. Sarac, A., Batta, R., Rump, C.M.: A branch-and-price approach for operational aircraft maintenance routing. European Journal of Operational Research **175**(3), 1850–1869 (2006)
8. Temucin, T., Tuzkaya, G., Vayvay, O.: Aircraft maintenance routing problem–a literature survey. Promet-Traffic&Transportation **33**(4), 491–503 (2021)
9. Xu, Y., Wandelt, S., Sun, X.: Airline scheduling optimization: Literature review and a discussion of modeling methodologies. Intelligent Transportation Infrastructure p. liad026 (2023)

A Finite Automata Reduction Procedure to Improve MIP regular Formulations of Personnel Scheduling Problems

Guillaume Ghienne[1], Odile Bellenguez[1], Guillaume Massonnet[1], and María I. Restrepo[1]

IMT Atlantique, L2SN, UMR CNRS 6004, 4 rue Alfred Kastler, 44300 Nantes, France
{guillaume.ghienne,odile.bellenguez,guillaume.massonnet,
maria-isabel.restrepo-ruiz}@imt-atlantique.fr

Keywords: Unambiguous finite automaton, Biclique decomposition, Regret heuristic, Mathematical programming, Personnel scheduling.

1 Introduction

Inspired by the Constraint Programming (CP) `regular` constraint [8], [3] proposed a similar approach in a Mixed Integer Programming (MIP) context: a `MIP regular` formulation. Their method proved its efficiency to solve several Personnel Scheduling Problems (PSP) [3,7]. The main idea is to consider employee's schedules as words of formal languages to encode a subset of working regulations. Flow formulations in the network structures of these languages can be derived and possibly combined with a classical MIP formulation to entirely model a PSP. In such a formal language, words have a fixed length n equal to the number of time periods in the planning horizon and can be represented by Finite Automata with $n+1$ levels (hereinafter denoted as n-FA) in which each accepting path represents a valid schedule between the first and the last level. The number of decision variables in the resulting network flow formulation corresponds to the number of transitions in the n-FAs.

When each word in the language has exactly one accepting path, we say that an n-FA is unambiguous. Note that deterministic automata (i.e., there is at most one transition per symbol for a given state) are unambiguous by definition. This characteristic allows to break possible symmetries in the mathematical programming formulation of the problem. Therefore, to efficiently obtain small and asymmetric `MIP regular` formulations for PSPs, we are interested in computing an unambiguous n-FA representing a given set of working regulations and with as few transitions as possible. This task faces three main challenges. First, n-FA minimization is NP-hard [1], and our aim is to develop a procedure performed in a negligible amount of time when compared to MIP solving time. Second, unambiguity is not guaranteed in a FA by usual minimization procedures. Third, existing works in the literature mainly focus on reducing the number of states in the automata, while we are more interested in reducing the number of transitions which in turn limits the number of variables in the final MIP formulations.

2 Reduction Approach

[6] describes a procedure to obtain deterministic (hence unambiguous) n-FAs representing a set of given PSP working regulations. We now seek to reduce the size of such automata. The classical minimization of deterministic n-FAs [5] is performed in linear time and is optimal in terms of both state and transition reduction. Although this obvious approach answers the three challenging points previously discussed in an efficient manner, nondeterministic n-FAs allow for an even more compact representation. In particular, [1] generalize Nerode's equivalence to derive an n-FA state reduction heuristic as a sequence of level minimization. This procedure allows to maintain unambiguity during the reduction and we observe short enough computation times when we test it on practical PSPs n-FAs. We propose to adapt this procedure for transition reduction.

Extending the results presented in [1], we show that minimizing the number of transitions entering and exiting a single level of an unambiguous n-FA is equivalent to solving an NP-hard problem that we call Weighted Vertices Biclique Decomposition Problem (WVBDP). Given a bipartite graph B, a biclique decomposition [2] of B is a set of bicliques (i.e., complete bipartite sub-graphs) whose respective edges partition B's edges. Given a positive weight function over B's vertices, we define the WVBDP as the computation of a biclique decomposition of B with minimum total weight (i.e., the sum of the weights of the vertices of each biclique). Our reduction procedure is therefore a succession of such level reductions (see [1] for more details) where each step is equivalent to solve a WVBDP.

The direct MIP formulation of the WVBDP that we propose fails to solve the intermediate steps of our procedure on practical `MIP regular` n-FAs in a reasonable amount of time. Therefore, we develop a heuristic to quickly obtain good approximate solutions for large problems. We build bicliques by starting from single uncovered edges and successively add vertices under some conditions. We define $\texttt{regret}(v, c)$ as the regret of adding a vertex v to the open (i.e. currently being built) biclique c rather than to the trivial biclique composed of v and its neighborhood. A biclique c is considered closed when no vertex v have $\texttt{regret}(v, c) \leq 0$ and open bicliques are extended with the vertex with smallest regret value. We establish some properties of our algorithm guaranteeing the convergence of the n-FA transition reduction procedure. The heuristic allows to obtain solutions close to the optimal on random WVBDP instances with small bipartite graphs. Section 3 shows an insight of the performance of the heuristic when used as an intermediate step in the n-FA reduction, which transfers in larger WVBDPs.

3 Numerical Experiments

We test our approach on instances for the Nurse Rostering Problem (NRP) presented in [4]. These instances represent a planning period of 4 weeks in 9 different hospital services. We model five types of working regulations (forward rotation, minimum and maximum consecutive working shifts, minimum consecutive days-off, maximum number of weekends and fixed days-off) as `MIP regular` constraints, while additional constraints are integrated as classical linear inequalities, as presented in [4]. We compute automata in Python3 and solve optimization problems with a time limit of one hour with the version 22.1.1 of CPLEX on one thread of a CPU Intel Xeon e5-2630-v4 from the CCIPL cluster (https://ccipl.univ-nantes.fr/).

Table 1 shows, for each service, the total number of states and transitions in all unambiguous n-FAs (one by employee) obtained with three different reduction procedures: a classical deterministic minimization [5] (`DeterMin`), the Nerode's equivalence generalization for state reduction [1] (`StateRed`), and our adaptation of [1] for transition reduction using the proposed regret heuristic as intermediate step (`TransRed`). Non-deterministic n-FAs allow a more compact representation of the language and our procedure results in a significant reduction in the number of transitions at the expense of a (mild) increase in the number of states when compared to the state reduction procedure.

Service		`DeterMin`			`StateRed`			`TransRed`		
#w	#s	#states	#trans	time	#states	#trans	time	#states	#trans	time
10	2	3155	6322	**0.48**	**2972**	6175	1.01	3027	**5601**	2.34
16	2	4726	9069	**0.58**	**4450**	8856	1.26	4553	**8085**	2.87
18	3	6691	15287	**1.03**	**6324**	14994	1.97	6404	**13144**	6.29
20	3	7679	18290	**1.30**	**7250**	18009	1.73	7465	**15905**	5.89
30	4	12897	32978	**1.91**	**12155**	32424	3.39	12390	**27814**	11.0
36	4	15798	44473	**2.98**	**15290**	44087	4.33	15461	**36957**	14.7
40	5	18754	53676	**3.37**	**17461**	52728	5.14	17924	**44500**	16.6
50	6	22196	74041	**4.20**	**20653**	72973	6.65	21369	**61935**	15.7
60	10	32693	149246	**8.37**	**30383**	147820	13.8	31676	**122263**	31.1

Table 1: Total number of states (#states) and transitions (#trans) for each instance: #w and #s are the number of nurses and shift types in the hospital service, computation time is indicated in seconds.

For each instance, we create 30 additional ones with the same working regulations and minor variations in the demand, vacation, or preferences to represent new planning periods in the same hospital service. Table 2 compares the solving performances for three MIP formulations: the classical model presented in [4], a `MIP regular` formulation with minimal deterministic n-FAs (`DeterMin MIP regular`) and the same `MIP regular` formulation using n-FAs reduced with `TransRed` (`TransRed MIP regular`). We observe that `MIP regular` formulations are generally more efficient when compared to a classical MIP formulation. Also, by comparing `DeterMin MIP regular` and `TransRed MIP regular`, the results show how the proposed unambiguous n-FA transition reduction leads to better MIP solving performances.

4 Conclusion

We propose a procedure for reducing the number of transitions in unambiguous n-FAs. This procedure is adapted from an existing n-FA state reduction approach and requires to solve a WVBDP as an intermediate step. The WVBDP is an NP-hard problem for which we present a regret heuristic in order to quickly find good solutions and therefore

Service		Compact Assignment		`DeterMin MIP regular`		`TransRed MIP regular`	
#w	#s	#sol(gap)	#opt(time)	#sol(gap)	#opt(time)	#sol(gap)	#opt(time)
10	2	30(<1)	29(93.44)	30(<1)	30(12.49)	30(<1)	**30(12.08)**
16	2	30(1.2)	25(615.1)	30(<1)	30(91.60)	30(<1)	**30(67.41)**
18	3	30(<1)	24(945.6)	30(<1)	**30(137.8)**	30(<1)	30(169.4)
20	3	30(5.6)	10(987.6)	30(<1)	30(289.3)	30(<1)	**30(207.0)**
30	4	30(16)	0(—–)	30(4.2)	0(—–)	**30(3.2)**	**1(1791)**
36	4	**28(9.9)**	**11(262.9)**	19(28)	6(1459)	27(23)	8(745.1)
40	5	30(2.7)	19(960.5)	30(<1)	30(231.5)	30(<1)	**30(133.1)**
50	6	30(<1)	29(309.8)	30(<1)	30(757.0)	30(<1)	**30(600.3)**
60	10	**16(18)**	3(1503)	3(<1)	3(1840)	6(<1)	**5(1508)**

Table 2: Solving performances: #sol(gap) = $n(x)$ indicates n integer solutions with a mean optimality gap of x %, #opt(time) = $n(x)$ indicates n optimal solutions with a mean solving time of x seconds (including `MIP regular` n-FAs reduction).

be able to use our n-FA transition reduction to improve `MIP regular` formulations for practical PSPs.

Results on an NRP show that our approach efficiently reduces the number of transitions in the `MIP regular` n-FAs, which leads to mathematical models with less variables. This globally translates into better MIP solving performances when using this type of reduction rather than a classical deterministic minimization approach.

References

1. Amilhastre, J., Janssen, P., Vilarem, M.C.: Fa minimisation heuristics for a class of finite languages. In: International Workshop on Implementing Automata. pp. 1–12. Springer (1999)
2. Amilhastre, J., Vilarem, M.C., Janssen, P.: Complexity of minimum biclique cover and minimum biclique decomposition for bipartite domino-free graphs. Discrete applied mathematics **86**(2-3), 125–144 (1998)
3. Côté, M.C., Gendron, B., Quimper, C.G., Rousseau, L.M.: Formal languages for integer programming modeling of shift scheduling problems. Constraints **16**, 54–76 (2011)
4. Curtois, T., Qu, R.: Computational results on new staff scheduling benchmark instances. Tech. rep., ASAP Research Group, School of Computer Science, University of Nottingham (2014), www.schedulingbenchmarks.org/papers/computational_results_on_new_staff_scheduling_benchmark_instances.pdf
5. Katsirelos, G., Narodytska, N., Walsh, T.: Reformulating global grammar constraints. In: International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research. pp. 132–147. Springer (2009)
6. Menana, J., Demassey, S.: Sequencing and counting with the multicost-regular constraint. In: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: 6th International Conference, CPAIOR 2009 Pittsburgh, PA, USA, May 27-31, 2009 Proceedings 6. pp. 178–192. Springer (2009)
7. Pandey, P., Gajjar, H., Shah, B.J.: Determining optimal workforce size and schedule at the retail store considering overstaffing and understaffing costs. Computers & Industrial Engineering **161**, 107656 (2021)

8. Pesant, G.: A regular language membership constraint for finite sequences of variables. In: International conference on principles and practice of constraint programming. pp. 482–495. Springer (2004)

Metaheuristic techniques for automated traffic light scheduling to minimize commute time

Uran Lajçi[1], Atlantik Limani[1], Elvir Misini[1], Fjolla Gashi[1], Kadri Sylejmani[1], and Arben Ahmeti[2]

[1] Faculty of Electrical and Computer Engineering, University of Prishtina, Bregu i Diellit n.n, 10000, Prishtina, Kosova
{uran.lajçi, atlantik.limani, elvir.misini, fjolla.gashi19}@student.uni-pr.edu, kadri.sylejmani@uni-pr.edu
[2] Department of Software Engineering, Riinvest College, Lidhja e Prizrenit, No.56, 10000, Prishtina, Kosova
arben.ahmeti@riinvest.net

Abstract. Keywords: Traffic Light Scheduling, Iterated Local Search, Evolutionary Algorithm

1 Introduction and problem description

The Intersection Traffic Signal Control Problem (ITSCP) [5] serves as a model for optimizing traffic signaling in urban areas. Broadly, ITSCP comprises three primary components: the cycle, representing the overall time for a signalization period; the phase sequence, indicating the order of allocation of green time for individual incoming streets at the intersection; and the green time duration, encapsulating the total timings for a specific phase (i.e., incomming street), during which signaling remains constant [11]. In this extended abstract, we focus on a variant of ITSCP as defined in the Google Hash Code Competition [9], a competition that has been held annually from 2014 to 2022 [9]. This variant is characterized by a general network type that considers single vehicle types with a fixed scheduling strategy employing an offline scheduling mechanism. Additionally, the objective function aims to minimize delay for all vehicles, with no constraints imposed on cycle length, order of phase sequence, or green timing length for individual phases.

In summary, the problem revolves around intersections, streets, and cars, with intersections having at least one incoming and one outgoing street, streets connecting intersections without intersecting each other, and cars following predefined paths of streets, passing through each intersection at most once. Several hard constraints apply: green time is assigned to incoming streets one at a time, forming a repeating cycle until the simulation ends; cars wait at red signals and move through green signals, with only one car able to pass through an intersection per second during the green phase; the traffic light schedule determines the order and duration of green lights for incoming streets, ensuring each street appears at most once; all streets are one-way, preventing duplicate connections between intersections in the same direction; and cars start at the end of initial streets, adhering to traffic signals to reach their final destinations. The objective is to optimize traffic signal control, where each car earns points based on completing its route before the simulation ends, with a fixed reward for completion and additional points for early completion, and the overall solution score is the sum of all cars' scores. For a detailed problem description, please refer to the Traffic Signaling Problem in [9].

2 Solution approach

Our approach to solving the problem involves two main methods: one utilizes single-state metaheuristic techniques, specifically Iterated Local Search (ILS) [7], while the other employs population-based techniques, specifically a variant of the Evolutionary Algorithm (EA) introduced by Pham and Castellani [8]. ILS is well-known for its ability to avoid getting stuck in local optimal solutions, and it is also praised for being fast and memory-efficient, which has made it successful in solving various scheduling problems in transportation, such as last-mile routing [2] and the colored traveling salesman problem [12]. On the other hand, EA is known for its ability to escape from local optima, making it suitable for solving various combinatorial optimization problems [6], such as wireless sensor networks [1], the team orienteering problem with time windows [3], and shortest path problems with fuzzy arc weights [4].

A candidate solution of the envisioned ITSCP problem variant is represented by an array, with its length matching the number of intersections in the given problem. Within this array, each element is a tuple object containing information about the phase order and signaling time (green time) for each incoming street at the respective intersection. The search process always operates within the feasible part of the search space, where solutions are evaluated using a single objective evaluation function defined in the respective problem description [9].

Table 1: Implemented neighborhoods

Name	Description
ShufflePhases(i)	Rearrange the order of phases for the green signal timing for all incoming streets at intersection i
SwapPhases(i,s_1,s_2)	Swap the order of phases for the green signal timing for incoming streets s_1 and s_2 of intersection i
ChangeSignaling (i,s)	Change the green time for incoming streets s of intersection i

We have a set of three neighborhoods already implemented, listed in Table 1. In each iteration, these neighborhoods compete with each other based on specific probabilities assigned beforehand, which are fine-tuned based on initial experimentation.

3 Preliminary experimental results

We conducted preliminary experiments to evaluate the performance of our ILS and EA approaches. Both methods were tested against upper bound values (the hypothetical scenario where no car ever waits at the traffic lights) and Shporer's Greedy Constructive Algorithm (SGCA) [10], which is recognized as a top-performing approach in the literature.

Our experiments involved 48 instances, including 5 from the original Google Hash Code competition and 43 newly generated synthetic instances. These instances featured a variety of intersection, street, and car configurations to provide a comprehensive assessment.

Our findings indicate that both ILS and EA are competitive with the state-of-the-art SGCA approach. Notably, our methods achieved new best results for 12 instances. When comparing ILS and EA, EA outperformed ILS in 26 out of 48 instances, with superior results by more than 1% in 10 instances. These results demonstrate the potential effectiveness of our metaheuristic approaches for traffic light scheduling.

4 Future work

The current study represents an initial phase towards achieving a broader final objective. Initially, we intend to combine EA with ILS to assess the potential for achieving improved results. Additionally, we plan to integrate heuristic functions to guide the selection of promising intersections for adjusting the phase order or signaling time. Two potential heuristic functions we are currently implementing involve selecting intersections based on either the average length of the queue of waiting cars or the number of cars passing through them.

Furthermore, we are currently working on generating a new test based on real-life data acquired from a taxi company in the city of Prishtina, Kosova. This test set will be used to evaluate the current variant of the algorithm against state-of-the-art solvers.

As a final goal, we aim to extend the current problem definition to a setting suitable for real-life applications. To demonstrate proof of concept, the extended algorithm variant will be incorporated into a prototype web system. This system will be capable of generating traffic light schedules for Prishtina, Kosova, which has 25 intersections with traffic lights and nearly 300 street lanes connecting them.

Acknowledgements This project is jointly financed by the German Federal Ministry for Economic Cooperation and Development (BMZ) and the European Union (EU), #Digital4Business is being implemented by GIZ Kosovo in the framework of the #DigitalTransformationCenter in the Innovation and Training Park (ITP) Prizren.

References

1. Ari, A.A.A., Yenke, B.O., Labraoui, N., Damakoa, I., Gueroui, A.: A power efficient cluster-based routing algorithm for wireless sensor networks: Honeybees swarm intelligence based approach. Journal of Network and Computer Applications **69**, 77–97 (2016)
2. Cook, W., Held, S., Helsgaun, K.: Constrained local search for last-mile routing. Transportation Science (2022)
3. Cura, T.: An artificial bee colony algorithm approach for the team orienteering problem with time windows. Computers & Industrial Engineering **74**, 270–290 (2014)

4. Ebrahimnejad, A., Tavana, M., Alrezaamiri, H.: A novel artificial bee colony algorithm for shortest path problems with fuzzy arc weights. Measurement **93**, 48–56 (2016)
5. Eom, M., Kim, B.I.: The traffic signal control problem for intersections: a review. European transport research review **12**, 1–20 (2020)
6. Kaya, E., Gorkemli, B., Akay, B., Karaboga, D.: A review on the studies employing artificial bee colony algorithm to solve combinatorial optimization problems. Engineering Applications of Artificial Intelligence **115**, 105311 (2022)
7. Lourenço, H.R., Martin, O.C., Stützle, T.: Iterated local search. Springer (2003)
8. Pham, D.T., Castellani, M.: The bees algorithm: modelling foraging behaviour to solve continuous optimization problems. Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science **223**(12), 2919–2938 (2009)
9. pierreavn: Google hash code archive. https://github.com/pierreavn/google-hashcode-archive (2021), accessed: December 6, 2023
10. Sagi Shporer: Solver for traffic signaling problem of google hash code competition - qualification round 2021. https://github.com/sagishporer/hashcode-2021-qualification (2021)
11. Urbanik, T., Tanaka, A., Lozner, B., Lindstrom, E., Lee, K., Quayle, S., Beaird, S., Tsoi, S., Ryus, P., Gettman, D., et al.: Signal timing manual, vol. 1. Transportation Research Board Washington, DC (2015)
12. Zhou, Y., Xu, W., Fu, Z.H., Zhou, M.: Multi-neighborhood simulated annealing-based iterated local search for colored traveling salesman problems. IEEE Transactions on Intelligent Transportation Systems **23**(9), 16072–16082 (2022)

Research on Optimization of Customized Feeder Bus for High-speed Railway Connection with Consideration of Travel Time Uncertainty

Yilin Chen[1,2], Jiemin Xie[1,2][0000-0002-0468-9445], Wei Huang[1,2][0000-0003-4377-8237], and Yingying Lin[1,2]

[1] Shenzhen Campus of Sun Yat-sen University, Shenzhen, China
[2] Guangdong Provincial Key Laboratory of Intelligent Transportation System, School of Intelligent Systems Engineering, Sun Yat-Sen University, Guangzhou, China
xiejm28@mail.sysu.edu.cn

Abstract. In order to cope with the inconvenience for passengers who need to take high-speed railway (HSR) due to the HSR stations in China being far away from the city center, this paper proposes a customized feeder bus (CFB) service for HSR connection. The research aims to optimize the comprehensive interests of the government, operators, and passengers, and constructs a mixed integer nonlinear optimization model that integrates customized bus line planning and timetable optimization with consideration of travel time uncertainty, and also considers the passenger assignment problem during the timetable optimization. In addition, the research also designs a heuristic decomposition algorithm to improve computation efficiency for solving large-scale cases. Finally, the results of test example and practical example show that the proposed model has better performance in reducing costs, passenger travel time deviations, government subsidies and energy consumption. Besides, it can provide more reliable services to passengers.

Keywords: Customized Bus, Feeder Service, Timetable Optimization, Passenger Assignment

1 Background

China high-speed railway (HSR) has developed rapidly in the past thirty years. However, in contrast to the efficiency and convenience of HSR, people generally spend a significant amount of time reaching and leaving an HSR station, which is primarily because most HSR stations in China have been built in suburban areas on the edge of cities (Wu and Han, 2022, Xu et al., 2023). Therefore, HSR feeder services have been brought into sharp focus to address the first and the last mile problem.

Customized bus (CB) service, as a superior feeder mode, integrates the characteristics of regular bus service with its low price and large capacity, along with the flexibility of taxi service. CB serves passengers with shared travel patterns fit for ride-sharing (Huang et al., 2020), which suits HSR passengers who typically plan trips in advance, ensuring that they reach stations on time. CB provides timely responses to passengers' needs with its flexibility and a fixed itinerary upon launch, ensuring punctual arrivals at HSR stations for travelers.

Research on CB as the HSR feeder mode is limited, highlighting two issues. First, supply-side models often unrealistically disregard vehicle empty runs, travel time uncertainty, and vehicle heterogeneity, affecting resource efficiency and service reliability. Second, passenger behavior models are oversimplified, neglecting how passengers' travel preferences influence optimization outcomes.

Based on this, this study formulates an integrated optimization model for Customized Feeder Bus (CFB) services that plans routes and schedules together, considering vehicle capacity and balancing the interests of operators, the government, and passengers for enhancing profitability, minimizing subsidies, and increasing satisfaction. It considers travel time uncertainty to enhance the robustness of CFB services. Moreover, passenger assignment model is included to refine timetable quality. To address the dispersion of HSR station passenger flows, this study advocates for hub stations to attract demand, improve transfer efficiency and boost operational profits.

2 Model and solution algorithm

Our CFB service consists of a set of candidate stops S, a set of routes P that contains all possible stop combinations and limited available services V provided by vehicles with a specific capacity. Each operational service will be assigned to a route, and all vehicles of operational services departing from an HSR station, passing through several stops on a particular route and final return to the HSR station. Vehicles will stop at intermediate points for passengers to board and disembark. Because varying traffic conditions on different routes during different time periods lead to travel time uncertainty, we have set different additional travel times for each time period based on the expected trip duration within each interval. That is to say, the actual travel time of the vehicle on the route will vary according to the different time periods in which it travels. The proposed model optimizes route design and timetable with consideration of passenger reaction, because passengers can choose to accept or reject services based on the deviation between the service timetable and their ideal time. A small example containing an HSR station, a hub station, five stops and two routes is illustrated in Fig.1.

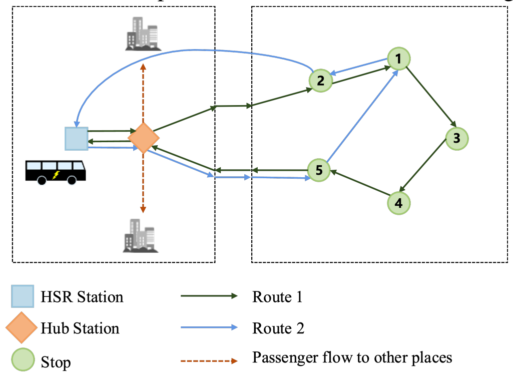


Fig. 1. An example of CFB service.

In terms of objective function, we consider minimizing the number of unserved passengers, the passenger travel time deviation, the government subsidy and the total electricity consumption and taking their weighted sum to transform into a single objective function. Therefore, the model can make reasonable decisions to balance passenger demand and economic feasibility, and save energy as well. In order to ensure that the model result meets the practical condition, we include CFB routing uniqueness constraint, planning horizon constraint, vehicle headway constraint, minimum boarding rate constraint, time deviation constraint, and UE condition considering vehicle capacity constraint.

The introduction of the UE condition leads to the nonlinearity of the model, which means that it is difficult to calculate by traditional methods in large-scale examples. Therefore, a decomposition algorithm (DA) is suggested to solve it and the algorithm flowchart is shown in Fig. 2. We split the original problem into two sub-problems, the timetable optimization problem and the passenger assignment problem, then solve them through an iterative process until the deviation of two successive iterations reaches the lower limit or the number of iterations reaches the upper limit. Specifically, in solving the passenger assignment problem, the timetable is given, while in solving the timetable optimization problem, the passenger service choices are given.

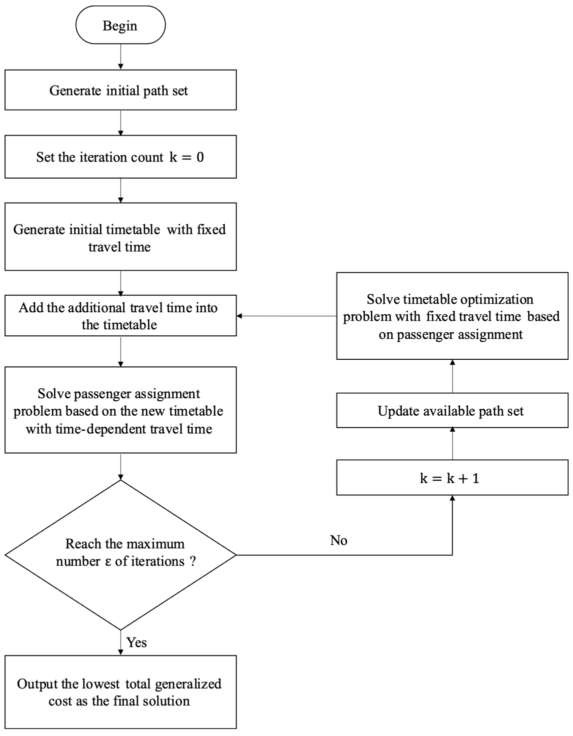

Begin
Generate initial path set
Set the iteration count k = 0
Generate initial timetable with fixed travel time
Add the additional travel time into the timetable
Solve passenger assignment problem based on the new timetable with time-dependent travel time
Reach the maximum number ε of iterations ?
Yes
Output the lowest total generalized cost as the final solution
No
k = k + 1
Update available path set
Solve timetable optimization problem with fixed travel time based on passenger assignment

Fig.2. Algorithm flowchart

In addition, considering that a set of routes P could get too large for big instances, in the large-scale algorithm, we gradually generate a set of paths by adding new routes for passengers who have not boarded the bus, which means that we do not have to generate all possible paths at the beginning.

3 Numerical examples

We conducted tests on two examples. In the small-scale example, we have demonstrated that compared to the benchmark result which used all-stopping services with the uniform vehicle type, the proposed model with consideration of differentiated capacities and skip-stop patterns could improve the passenger boarding rate by approximately 8.16%, reduce the travel time deviation by 30%, decrease the government subsidy by about 1.93%, and lower energy consumption by roughly 60.09%, as shown in Table 1.

Table 1. Comparison result.

Group	α	β	γ	δ	Obj
Control	62	42	5276	131.7	10134.5
Experimental	58	60	5380	330	13430
Results deviation	6.90%	-30.00%	-1.93%	-60.09%	-24.54%

(α: Number of boarding passengers, β: Travel time deviation, γ: Government subsidies, δ: Energy consumption)

Furthermore, considering travel time uncertainty can enhance service reliability and punctuality, preventing passengers from missing trains.

In the Guangzhounan Railway Station-Guangzhou Higher Education Mega Center calculation example, the CFB operation plan calculated by our model could reduce travel time per person, compared with the existing plan.

4 Conclusion

In pursuit of augmenting the convenience and comfort of feeder services for HSR commuters while improving service reliability with consideration of vehicle travel time uncertainty, this paper proposes a mathematical model to optimize CFB service. In order to improve the solution efficiency, we have designed a decomposition algorithm to iteratively solve the passenger assignment problem and the timetable optimization problem. The results of numerical examples demonstrate that the proposed model can effectively reduce costs, better match passenger flow demands, and provide passengers with more punctual and reliable services.

Acknowledgements The research described in this paper was supported by Shenzhen Science and Technology Program (Grant Nos. 202206193000001, 20220817201129001), Guangdong Basic and Applied Basic Research Foundation (Grant Nos. 2022A1515110235, 2023A1515012542), Science and Technology Planning Project of Guangdong Province (Grant No. 2023B1212060029) and National Natural Science Foundation of China (Grant No. 72371174).

References

1. Wu S, Han D. Accessibility of high-speed rail (HSR) stations and HSR–air competition: Evidence from China[J]. Transportation Research Part A: Policy and Practice, 166: 262-284 (2022).
2. Xu M, Shuai B, Wang X, et al. Analysis of the accessibility of connecting transport at High-speed rail stations from the perspective of departing passengers[J]. Transportation Research Part A: Policy and Practice, 173: 103714 (2023).
3. Huang D, Gu Y, Wang S, et al. A two-phase optimization model for the demand-responsive customized bus network design[J]. Transportation Research Part C: Emerging Technologies, 111: 1-21 (2020).

Scheduling EURO Conference Copenhagen 2024

Thomas Stidsen[1] [0000-0001-6905-5454] and Dario Pacino[2] [0000-0002-7255-004X]

[1] Technical University of Denmark, Kongens Lyngby 2800, Denmark thst@dtu.dk
[2] Technical University of Denmark, Kgs. Lyngby 2800, Denmark darpa@dtu.dk

Abstract. The EURO conference is the second largest Operations Research (OR) conference in the world, typically having more than 700 presentations belonging to one of 70 subject streams and more than 2000 participants. This article briefly explains how the EURO-2024 was scheduled.

Keywords: Scheduling, Conference planning, Mathematical Programming

1 EURO conference planning

The EURO series of conferences is the second largest OR conference in the world, only dwarfed by the INFORMS (US) conferences. The EURO conferences always take place in Europe, but are every third year replaced by the IFORS conference. Last year the IFORS conference took place in Santiago in Chile, hence no EURO conference 2023 was held. In 2024, the EURO conference took place in Copenhagen, 30/6-3/7. This abstract is on how the conference scheduling was optimized in 2024. A previous article has already been published on scheduling the EURO conferences, but it has been substantially improved since 2018, see [1].

1.1 Top level and low level planning

Given the size of the EURO conferences, the centralized scheduling only makes an overall schedule of the assignment of sessions for streams, to time-slots and rooms. A stream is an overall theme of research, with one or two assigned stream organisers. When an article is submitted, it is either submitted to a specific stream or assigned by the program chair to the most appropriate stream. Examples of streams are: "Behavioural OR", "Discrete Optimization and Algorithms (contributed)" or "ORAHS: OR in Health and Healthcare". Here the stream "Discrete Optimization and Algorithms (contributed)" consists of submissions, not to any specific stream, but allocated by the program chair to this stream.

The stream organisers are typically leading researchers in the topic and they are in charge of the detailed management of the stream. Each stream is allocated a number of sessions, corresponding to the number of submissions divided by 4 rounded up. In 2022 the stream "Behavioural OR" had 35 submissions and was assigned 9 sessions. The division of labor between the scheduling team, Thomas Stidsen and Dario Pacino (Local Chair of EURO-2024), is then to assign a timeslot out of the 12 possible timeslots and a room to each of the 9 sessions. The stream organiser will then have to assign each of the 35 presentations to one of the 9 sessions. This division of planning responsibilities have served the EURO conferences well for many years.

1.2 Schedule goals

The ultimate goals for the scheduling is to make the conference as pleasant and economical as possible for the participants. This is however a goal which needs to be quantified. Hence a number of different objectives exist:

- Sessions of the same stream in same room (making finding the sessions easier)
- Sessions of the same stream consecutive, i.e. if timeslot 3 has a session of a stream, if that stream has more sessions, there is either a session of the stream in timeslot 2 or 4.
- Streams which are similar in topic should not overlap in time. This cannot be avoided, but should be minimized. How similarity is defined, is detailed below.
- Streams belonging to the same Area should take place in nearby rooms
- Size of the rooms should be of sufficient size, but not too big.

2 Schedule Data

The data for the schedule is given as a set of streams $s \in S$, a set of timeslots $t \in T$ and a set of rooms $r \in R$. Each stream s has a number of submissions $SUB_s \in Z^+$ which leads to a number of sessions for a stream $Session_s = \lceil \frac{SUB_s}{T} \rceil$. Finally we need an estimate of the number of conference participants who will join the stream to listen to the presentations. This is a number which is hard to approximate and as a very simple approximation, we simply assume that all the presenters also participate in the stream.

At EURO-2018 in Valencia, we started using a new type of model, where the concept of a pattern was used. The idea is simple: Instead of simply assigning each session each own binary variable to decide when and where to place it, we instead generate a number of patterns for each stream, see Table 1 below, where 4 patterns are found for a stream that cannot be started up in time-slot 1. Given a stream with 8 sessions, where it is not possible to have a session in the first timeslot, these 4 patterns are the only possible patterns. The big advantage of this approach is that the two first requirements of the above list of objectives are automatically obtained. It can also lead to the need for more rooms, but usually this is not problematic.

Finally, we need to quantify the relatedness of different streams. All articles submitted to a stream are allocated up to 3 keywords. Then all the keywords used for the articles of a stream is saved in a set $K_s \in Z^+$ and the number of times each keyword of a stream is used is saved in $c_{s,k} \in Z^+ \; \forall \; k \in K_s$. Then the average number of times a keyword appears in a stream is calculated: $\overline{c_s} = \frac{\sum_{k \in K_s} c_{s,k}}{|K_s|}$. Finally, the co-variance of the number of times a keyword compared to the average no of keywords appears in two different streams is calculated: $CoV_{s_1,s_2} = \frac{\sum_{k \in K} \left(c_{s_1,k} - \overline{c_{s_1}}\right)\left(c_{s_2,k} - \overline{c_{s_2}}\right)}{\sqrt{\left(\sum_{k \in K} \left(cs_1,k - \overline{c_{s_1}}\right)^2\right)\left(\sum_{k \in K} \left(c^{-}s_2,k - \overline{c_{s_2}}\right)^2\right)}}$

Pat	MA	MB	MC	MD	TA	TB	TC	TD	WA	WB	WC	WD
1		X	X	X	X	X	X	X	X			
2			X	X	X	X	X	X	X	X		
3				X	X	X	X	X	X	X	X	
4					X	X	X	X	X	X	X	X

Table 1: Patterns for an 8 session stream

3 Schedule Model

The basic decision for the overall scheduling of the conference is hence to choose a pattern for a session s and select a room r. This is represented by the binary variable $x^s_{t,r} \in \{0,1\}$ which takes the value 1 if stream s uses the pattern with the first time-slot use t in room r.

This leads to the following, relatively simple model:

$$\min \quad \sum_{s1,s2,tp1,tp2,r} CoV_{s1,s2} \cdot pat\_overlap^{s1,s2}_{t1,t2} \cdot x^{s1}_{tp1,r} \cdot x^{s2}_{tp2,r}$$

Such that:

$$\sum_{tp,r} x^s_{tp,r} = 1 \qquad \forall\, s$$

$$x^s_{tp,r} \in \{0,1\}$$

The above model is quadratic, but is easily linearized. Since the actual planning is not yet finalized at the time of abstract submission, we expect more objectives and constraints to be added, and these will also be presented. At EURO-2022 in Finland additional constraints were implemented to optimize the co-location of related streams in nearby rooms. The selection of rooms was also optimized.

References

1. Stidsen, Thomas, David Pisinger, and Daniele Vigo. Scheduling euro-k conferences. *European Journal of Operational Research*, 270(3):1138–1147, 2018.

Industrial Production Scheduling in the Energy Deregulation Era

Panayiotis Alefragis[1] [0000-0002-1313-1750], Christos Gogos[2] [0000-0003-1113-8462], Christos Valouxis[3] [0000-0001-6832-2311], Michael Birbas[3] [0000-0002-6124-221X], and Alexios Birbas[3] [0000-0002-9468-7215]

[1] Electrical & Computer Engineering Department, University of Peloponnese, Patras, Greece
alefrag@uop.gr

[2] Informatics and Communications Department, University of Ioannina, Arta, Greece
cgogos@uoi.gr

[3] Electrical & Computer Engineering Department, University of Patras, Patras, Greece
{valouxis, mbirbas, birbas}@ece.upatras.gr

Abstract. This paper extends a preliminary model to address the very important problem of aligning industrial production with time periods where more renewable energy is available. Modern industries may use multiple energy sources, each having different temporal and quantitative availability. Our model uses forecasted day-ahead energy prices and energy production mix to generate an optimized production schedule. A toolset approach is applied where multiple solvers that share a common data model is implemented. The paper presents a production level Constraint Programming (CP) model and results from applying the toolset to a number of real world instances.

Keywords: Sustainability, Industrial Production Scheduling, Flexible Job Shop Scheduling

1 Introduction and related work

Emerging industrial sustainability domain dictate new production efficiency interventions driven by concerns related to energy costs and climate changes. Local energy production, renewable energy sources that introduce stochasticity in the availability and auxiliary energy markets effect the energy mix and prices creating a new deregulated era. Production scheduling is critical in the sustainability decision making process. Integrated production scheduling, maintenance planning and energy controlling for sustainable manufacturing systems using a hybrid of a Non-dominated Sorting Genetic Algorithm (NGSA-II) based multi-objective genetic algorithm and a mathematical model is used in [1]. A framework to allow collaboration between energy providers and manufacturing companies is proposed in [2]. Energy price forecasts are signaled to the manufacturers and an adaptive production scheduling approach considering the power usage of manufacturers in response to time-varying energy prices is presented. In [3] a Mixed Integer Linear Programming (MILP) stochastic programming model is proposed that simultaneously optimize production scheduling and electricity procurement. An energy aware scheduling model to optimize steel industry operations when multiple energy sources are available using a minimum-cost network flow for cost optimization is proposed in [4]. Recently, a flowshop scheduling problem to simultaneously minimize makespan and total energy cost using critical-path based local search methods is proposed in [5].

2 The EnerMan EAPS toolbox

The EnerMan Energy Aware Production Scheduling (EAPS) Toolbox supports the combined requirements collected from diverse problems from energy demanding production processes like automotive manufacturing and testing, semiconductor production, steel and aluminum production, food processing and 3D additive components manufacturing. A generic software component allows potential users to introduce new features in their production planning and scheduling. The toolbox implements a number of constructive heuristics, meta-heuristics and a Constraint Programming (CP) based solver. In the current paper, a version of the CP solver is presented.

3 Constraint Programming Solver

The current CP model extends a preliminary CP model[6]. Special constructs like interval variables, specialized global constraints (e.g., noOverlap, circuit, element) among others are employed. The current implementation uses the most performant open source solver (OR-Tools CP-SAT) and a commercial one (ILOG CP). Python is used for implementation as it was easier to manipulate the amount of data required. The toolbox is provided to the other services of the EnerMan platform as a OpenAPI RESTFul services, exchanging data model information as JSON based messages.

Let J/T_j represent the set of jobs/tasks that must be scheduled, A the set of task attributes and $a_{j,t}$ the attribute associated with each task. The set of factories and set of machines in a f factory are represented by F and M_f respectively. O_m and c_m represent the operation model and capacity of machine m. The setup time that will be needed if tasks $t1, t2$ will be scheduled at machine m, with task $t2$ processed as the next task after task $t1$ is represented by $S_{m,t1,t2}$. For each task t of job j and for all possible start times the task can be scheduled at factory f, machine m and operation mode o, vectors $\mathrm{C}_{\mathrm{j,t,o,m,f}}$ and $EC_{j,t,o,m,f}$ hold the energy consumption and cost that task t incurs. These values are calculated in advance by considering the machine characteristics and the energy cost components.

The main variables of the model $xvar_{j,t,o,m,f}$ are optional interval variables that represent if a task t of job j instance is performed on a machine m of factory f using operational mode o and have a start time $s_{j,t}$, an end time $e_{j,t}$ and a Boolean variable $is\_p_{j,t,o,m,f}$ that represent if they exist. $\mathrm{evar}_{\mathrm{j,t,o,m,f}}$ is an integer variable for the energy cost of a task. Additionally, auxiliary variables s_{j} , e_{j} are the start and end time of a job, $assigned\_to_{j,t}$ holds the machine it is processes on, $\mathrm{b}_{\mathrm{t1,t2,m,f}}$ are Boolean variables that assumes value 1 if both tasks $t1, t2$ are processed at the same machine and task $t1$ is processed immediately before task $t2$ on machine m of factory f.

The global constraint *noOverlap* is used to avoid simultaneous processing of multiple tasks at the same machine when the capacity of a machine equals to 1. When $c_m \geq 1$, the global constraint *Cumulative* is been used instead. For each $f \in F, m \in M_f, j \in J, t \in T_j, o \in O_m$

$$noOverlap(allxvar_{j,t,o,m,f}), c_m = 1 \tag{1}$$

$$Cumulative(allxvar_{j,t,o,m,f}, c_m), c_m \geq 1 \tag{2}$$

If a machine is not available for specific time periods across the horizon of the schedule, a set of dummy interval variables are defined with fixed starting times and durations corresponding to the time periods that this machine is not available. This set of dummy interval variables are used in the previous constraint, disallowing processing of tasks to this machine.

Each $t \in T_j$ of $j \in J$ must be scheduled exactly at one available machine.

$$\sum_{f \in F, m \in M_f, o \in O_m} is\_p_{j,t,o,m,f} = 1 \tag{3}$$

To impose a setup time, when a pair of incompatible tasks are scheduled in sequence at the same machine, the global constraint Circuit is used that defines a Hamiltonian path in a sequencing graph that visits each node exactly once. To determine the task sequence, a graph is defined for each machine and the nodes of this graph are all the tasks that can be executed at it. For each $f \in F, m \in M_f$,

$$Circuit(arcs_{m,f}) \tag{4}$$

$$EndOf(xvar_{j,t1,o,m,f}) + S_{m,t1,t2} <= StartOf(xvar_{j,t2,o,m,f}) \tag{5}$$

The energy cost of a task depends on the starting time of the corresponding interval variable. The global constraint Element determines the energy cost of a task t.

$$Element(StartOf(xvar_{j,t,o,m,f}), EC_{j,t,o,m,f}, evar_{j,t,o,m,f}) \tag{6}$$

The objective function coefficients c1, c2 determine the relative weights between energy consumption and energy cost.

$$\min \sum_{j \in J,\ t \in T_j,\ f \in F,\ m \in Mf,\ o \in Om} \left(c1 * C_{j,t,o,m,f} * is\_p_{j,t,o,m,f}\ c2 * evar_{j,t,o,m,f}\right) \tag{7}$$

4 Evaluation, Conclusions and future work

Fig. 1 represents a weekly solution from a semiconductor manufacturing industry, with colours representing different task types, the red line the variability of energy cost and the green line the cumulative energy cost. In the specific instance more than 9K tasks are scheduled. It was observed that it is possible to reduce the production cost by 6% while in parallel we reduced the generated CO_2 by 15% without sacrificing throughput. We intend to release the solvers along with the data model and problem data to the public when we manage to anonymize the related data and appropriate approvals are given by the problem owners.

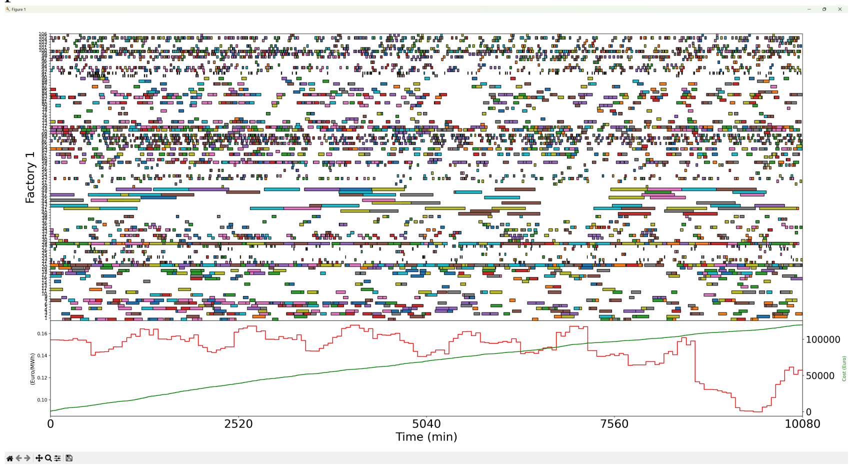


Fig. 1. Weekly schedule from a single factory of a semiconductor manufacturing industry.

We intend to extend the toolbox with the ability to automatically generate what-if scenarios based on the forecasted prediction variability to calculate alternative solutions transitions that will allow the factory to during the implementation of a scenario if a demand response signal is observed the factory to participate in the demand response energy market without significantly sacrificing production performance.

Acknowledgements The work in this paper is partially financed by H2020-EnerMan project, https://enerman-h2020.eu/ and Horizon Europe FLEXIndustries project, https://flexindustries.eu/.

References

1. Cui W., Sun H., Xia B.: Integrating production scheduling, maintenance planning and energy controlling for the sustainable manufacturing systems under TOU tariff. Journal of the Operational Research Society 71(11), 1760–1779 (2020)
2. Mourtzis D., Boli N., Xanthakis E., Alexopoulos K.: Energy trade market effect on production scheduling: an Industrial Product-Service System (IPSS) approach. International Journal of Computer Integrated Manufacturing 34(1), 76–94 (2021)
3. Zhang Q., Cremer J. L., Grossmann I. E., Sundaramoorthy A., Pinto J. M.Q Risk-based integrated production scheduling and electricity procurement for continuous power-intensive processes. Computers & Chemical Engineering 86, 90–105 (2016)
4. Hadera H., Harjunkoski I., Sand G., Grossmann I. E., Engell S.: Optimization of steel production scheduling with complex time-sensitive electricity cost. Computers & Chemical Engineering 76, 117–136 (2015)

5. Fernandez-Viagas V., Prata B. de A., Framinan J. M.: A critical-path based iterated local search for the green permutation flowshop problem. Computers & Industrial Engineering 169, 108276 (2022)
6. P. Alefragis et al.: Sustainable energy aware industrial production scheduling. In: Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling, pp. 256–264(2022)

An interactive optimization method to promote ethics for Nurse Rostering

Vincent Bebien[1,2], Odile Bellenguez[1], Gilles Coppin[3], Anna Ma-Wyatt[2], and Rachel Stephens[2]

[1] IMT Atlantique, La Chantrerie, 4 rue Alfred Kastler, 44307, Nantes, France
[2] University of Adelaide, School of Psychology, SA 5005, Adelaide, Australia
[3] IMT Atlantique, Lab-STICC, Technopôle Brest-Iroise CS 83818, 29238, Brest, France

Keywords: Nurse Rostering Problem, Interactive methods, AI Ethics.

1 Introduction

In the field of AI Ethics, scholars have identified various kinds of ethical issues related to autonomous decision-making algorithms [6], which include Operations Research (OR) applications. While methods for addressing some ethical issues have been studied in scheduling contexts, especially fairness criteria [10], there are still research avenues for OR applications that have received limited attention. We consider that ethics cannot be efficiently integrated into a decision tool without considering the specific and dynamic aspects of the problem on the field [2]; thus we propose here a design for a decision tool for the Nurse Rostering Problem (NRP) that allows for a better integration of moral values and ethical considerations.

2 Computing moral values

In the field of ethics, multiple frameworks have been developed to describe the moral preferences of individuals by identifying core values one would wish to respect. Taking into account one of them, the basic human values theory [8], we try to address the main issues of NRP by identifying first which aspects may be related to which of the theory's ethical principles. For example, some constraints such as satisfying minimum personnel requirements may be related to the *conformity* value, while others such as balancing workload across employees can be considered as *benevolence* and *universalism*. This approach forms a basis for a moral compass of decision-makers.

Mathematically, these potentially conflicting values are represented with norms that may be either modeled as objective functions or constraints. Hard constraints may be used to represent a threshold that has to be attained regarding a certain norm, refusing all solutions that do not meet it. Alternatively, using soft constraints allows the consideration of such solutions as valid but of lesser fitness, depending on their assigned weights.

These weights implicitly create a hierarchy between soft constraints, where the ones with the highest penalties will be preferred to the others. Thus, an 'ethical profile' can be drawn from the way the objective function is modeled. Multi-criteria decision-making (MCDM) methods allow users to visualize the different set objectives and/or trade-offs between efficient solutions. For these methods, all objectives are considered equivalent in the model; only the end-user has the agency to either choose their preferred solution or decide how these objectives should be ranked or prioritized.

An important limitation of standard modeling approaches with hard and soft constraints with a single objective function when it comes to ethical decision-making is its static aspect. As moral values are constantly evolving [9], a mathematical model prioritizing some criteria and enforcing ethical constraints might become irrelevant and unreliable in the future, or for people with a different cultural background [1]. MCDM approaches such as interactive methods [4] offer some flexibility by including the decision-maker in the loop, allowing them to decide which criteria are most important in their current situation. Nevertheless, these criteria themselves as well as the problem structure typically remain the same, cannot be modified and might also lose relevance with time passing and context changing.

We argue that a human-in-the-loop decision-making process that gives more agency than standard MCDM interactive methods could be used to build a tool that better considers ethics. An interactive process offers some advantages that could benefit the whole nurse scheduling process. Incorporating human interaction allows for a better adaptation to new conditions, which helps to generate well-suited schedules and reinforces user agency as they may have a better comprehension of the whole process [7]. An open process also allows other stakeholders such as nurses to better grasp how a schedule has been designed, which might be regarded as a fair process [3].

3 Integrating ethical considerations into an interactive tool

We propose here to use an interactive reoptimization method [5] adapted to an NRP, where the decision-maker can iteratively modify the set of rules, which correspond to hard constraints, to obtain new solutions. These modifications may consist of either local changes (e.g. assigning a nurse to a certain shift on a specific day) or global changes (e.g. forbidding some shift patterns for all nurses). More specifically, users have access to a catalogue of ‘template’ rules that can be parameterized according to their preferences. For example, the catalogue contains a template called ‘Limit consecutive working days’ that can be parameterized by selecting the nurses and period for which this rule should be applied, as well as the limit value. Whenever a rule is added to (or deleted from) the model, a new solution is generated according to the changes, following a user-defined optimization criterion also chosen from a catalogue.

This design aims to provide an interactive tool that displays and allows changes to the main aspects of the mathematical models that are used. While the preliminary work that established the catalogue limits the decision-maker’s possibilities, it allows non-experts in OR to directly manipulate the NRP formulation. The proposed design may be especially useful when a clash between two ethical criteria arises and arbitration is needed to obtain a feasible solution. Through trial and error, the user may have a better understanding of the problem structure and the different trade-offs they should consider. While we focus on the scheduling process itself, such tool could also be used in a reoptimization context, when unplanned events may arise during the scheduled period.

A possible drawback of this method is that ethical aspects related to scheduling might be ignored or forgotten in the process, as scheduling tasks are often difficult for human decision makers. To help the user detect potential ethical flaws in a candidate schedule, we propose to implement the presentation of specific ethical recommendations that would highlight some of them. The set of presented recommendations may be determined by the user's ethical preferences, which could be assessed either beforehand or during the iterative process. This information could be used either to show recommendations that are preferred by the user or to nudge them towards other ethical criteria they would otherwise not likely consider.

References

1. Awad, E., Dsouza, S., Kim, R., Schulz, J., Henrich, J., Shariff, A., Bonnefon, J.F., Rahwan, I.: The moral machine experiment. Nature **563**(7729), 59–64 (2018). https://doi.org/10.1038/s41586-018-0637-6
2. Bebien, V., Bellenguez, O., Coppin, G., Ma-Wyatt, A., Stephens, R.: Ethical decision-making in human-automation collaboration: a case study of the nurse rostering problem. AI and Ethics (In press), https://hal.science/hal-04500402
3. Greenberg, J.: Organizational justice: Yesterday, today, and tomorrow. Journal of Management **16**(2), 399–432 (1990). https://doi.org/10.1177/014920639001600208
4. Korhonen, P.: Interactive methods. Multiple criteria decision analysis: state of the art surveys pp. 641–661 (2005). https://doi.org/10.1007/0-387-23081-5_16
5. Meignan, D., Knust, S., Frayret, J.M., Pesant, G., Gaud, N.: A review and taxonomy of interactive optimization methods in operations research. ACM Transactions on Interactive Intelligent Systems (TiiS) **5**(3), 1–43 (2015). https://doi.org/10.1145/2808234
6. Mittelstadt, B.D., Allo, P., Taddeo, M., Wachter, S., Floridi, L.: The ethics of algorithms: Mapping the debate. Big Data & Society **3**(2), 2053951716679679 (2016). https://doi.org/10.1177/2053951716679679
7. Onnasch, L., Wickens, C.D., Li, H., Manzey, D.: Human performance consequences of stages and levels of automation: An integrated meta-analysis. Human Factors **56**(3), 476–488 (2014). https://doi.org/10.1177/0018720813501549
8. Schwartz, S.H.: Are there universal aspects in the structure and contents of human values? Journal of Social Issues **50**(4), 19–45 (1994). https://doi.org/10.1111/j.1540-4560.1994.tb01196.x
9. Smith, K.B., Alford, J.R., Hibbing, J.R., Martin, N.G., Hatemi, P.K.: Intuitive ethics and political orientations: Testing moral foundations as a theory of political ideology. American Journal of Political Science **61**(2), 424–437 (2017). https://doi.org/10.1111/ajps.12255
10. Wolbeck, L.A.: Fairness aspects in personnel scheduling. Discussion Papers 2019/16, Free University Berlin, School of Business & Economics (2019). https://doi.org/10.17169/refubium-26050

Nurse Rostering with Strategic Planning of Skills for Sick-Leave Robustness

Björn Morén[1 [0000−0001−7191−5206)]] and Elina Rönnberg[1 [0000−0002−2081−2888]]

Department of Mathematics, Linköping University, Linköping, Sweden
`bjorn.moren@liu.se`, `elina.ronnberg@liu.se`

Keywords: Nurse, Uncertainty, Sick-Leave

1 Introduction and contribution

In hospitals, nurse schedules are sensitive to disruptions such as sick leave since the absence of nurses with the right skills can have a severe impact on healthcare quality. The literature on the nurse rostering problem (NRP) is extensive and addresses a variety of aspects, such as minimizing the cost of schedules, meeting work regulations, and satisfying the preferences of the staff. The nurse rerostering problem (NRRP), which models rerostering of a schedule due to the absence of nurses, was first defined by Moz and Pato [2]. It has, for example, been studied in [4] where the aim is to do the rerostering with as few changes as possible while respecting staffing demand and hard constraints. They schedule shifts and tasks, and in the rerostering, nurses are allowed to change shifts or change free days to work days. Strategies to achieve robust and cost-efficient schedules are considered in [5], by using capacity buffers and reserve shifts, which can be changed to a working shift at a later stage. There are also general studies of staff rerostering, e.g. [3] that proposes a large neighbourhood search enhanced with machine learning for solving the problem.

We consider nurse rerostering from a strategic perspective and study the case of having multiple skills and varying staffing demand. Our aim is to design a decision support tool to be used on a strategic level to analyse how the distribution of skills affects the sick-leave robustness of schedules. As a first step, we here propose a scheme to both schedule and evaluate the schedule with respect to sick leave, similar to the work by Wickert *et al*. [5], and we evaluate it on real data from a case study. See Figure 1 for an illustration of our scheme.

We compare three approaches for handling the distribution of skills, denoted (i) `case`, (ii) `mix`, and (iii) `min`, where `case` allows the use of all available skills, `min` minimizes the available skills in a base schedule while scheduling with a capacity buffer. The approach `mix` is based on the same base schedule as `min`, but for each generated scenario, it is possible to add skills to cover understaffing.

For the scheduling part, the aim is to minimize understaffing and add an additional buffer per day, shift and task, while respecting the work time of the available nurses. A prerequisite for sick-leave robustness to be of interest is to have more nurses than what is required to meet the minimum demand. Our objective is then to spread out the additional nursing resources in such a way that the schedule is robust with respect to sick leave. To

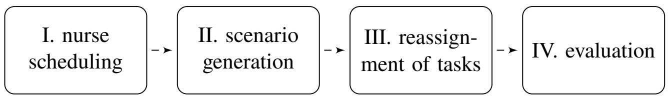


Fig. 1: Overview of the proposed scheme.

evaluate the robustness of a schedule with respect to sick leave, we generate scenarios. For each shift a nurse is assigned to, we assume that there is a probability that the nurse is absent and hence does not contribute to the staffing demand. This probability is based on one year of historical data of sick leave. For each scenario, we solve a restriction of the NRRP in which only tasks are reassigned. We impose this restriction as there are often limitations for changing a nurse's shift or a free day to a working day. For some understaffed shifts, it is possible to find a substitute nurse, but this is not known at the stage of scheduling. Hence, the only certain way to cover for an absent nurse is to reassign the tasks of the other nurses working that shift. If possible, the aim is to cover all the demands on tasks, or if not possible, at least cover the most important tasks. Finally, the updated schedule is evaluated with respect to understaffing.

2 Case study

Our case study is from a ward at a Swedish university hospital which has about 50 to 60 nurses and a scheduling period of 10 weeks. The ward requires nursing staff round the clock on both weekdays and weekends. There are three types of shifts—day, evening, and night—and for each shift, there is a task-based staffing demand. That is, a nurse is assigned a task for each scheduled shift. There are in total five tasks, requiring specified skills, except for one skill that is common for all nurses. The combination of skills is individual for each nurse and there is no general hierarchy between the skills. The demand varies over the scheduling period, for example, some tasks only appear every third week. The case can be classified as ASB|V3|LR based on the definitions from [1].

For the case study, we have formulated a straightforward mixed-integer programming model, based on binary variables indicating if to assign a nurse to a task at a specific shift, or not. There are additional variables concerning requirements of a task at a specific shift. They are used to model penalties for understaffing, lack of buffer, and presence of additional overstaffing. For the approaches including distribution of skills, there are binary variables for which tasks are available per nurse. The starting point is the skills of each nurse, and our approach tries to remove skills while still satisfying staffing demand, including a buffer. A summary of the included constraints is as follows.

- A nurse works at most one shift and is assigned at most one task per day.
- A nurse's maximum number of work days in a row is respected.
- A nurse's maximum number of night shifts is respected.
- The total work time of a nurse during the period is within an interval.
- A nurse works only according to permitted shift patterns, specifying which consecutive shifts that are allowed to work.
- On a weekend, a nurse is either free or works both Saturday and Sunday.

– A nurse works a specified number of weekends.
– A nurses should not work two weekends in a row and on-duty weekends should be spread out in the schedule.
– The work of different shift types is evenly distributed among the nurses.
– The staffing demand for each task at a specific shift is respected by either assigning a nurse to the task or by declaring a shortage.

3 Computational Experiments

Our evaluation is made on 100 simulated scenarios with a risk of absence of 6.4%, which is based on data from the ward. On average per scenario, there are absent nurses on 123 occasions. We show results for the three approaches `case`, `mix`, and `min`. The average understaffing for all shifts, weekdays, weekends is for approach `case` 72, 46, and 26, for approach `mix` 73, 46, 27, and for approach `min` 73, 46, 28. See Figure 2 for an illustration of understaffing per scenario. The computational times required for optimally solving the models for `case` and `min` are 1.5 h and 7 h, respectively.

Figure 3 shows the distribution of skills for the three approaches. Skill A is common among all nurses. It is clear that the number of nurses with skills C and D can be reduced to a large extent, while skill B and E appear to be more critical for the schedule and the sick-leave robustness. On average, the number of available skills (B-E) is 114, 69, and 67, for case, mix, and min, respectively.

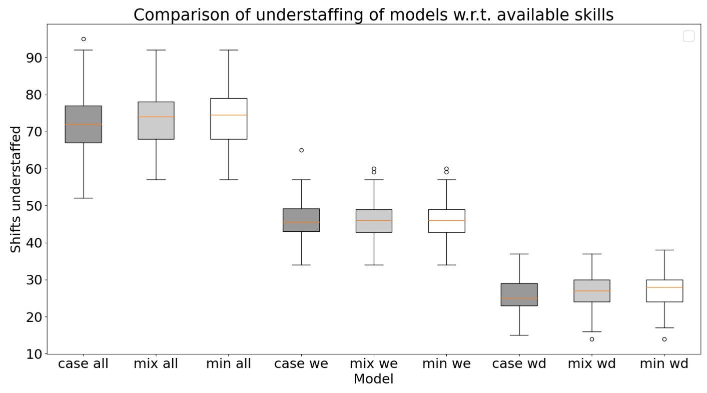


Fig. 2: Number of shifts understaffed per scenario for the three approaches, divided into all days (all), weekends (we), and weekdays (wd).

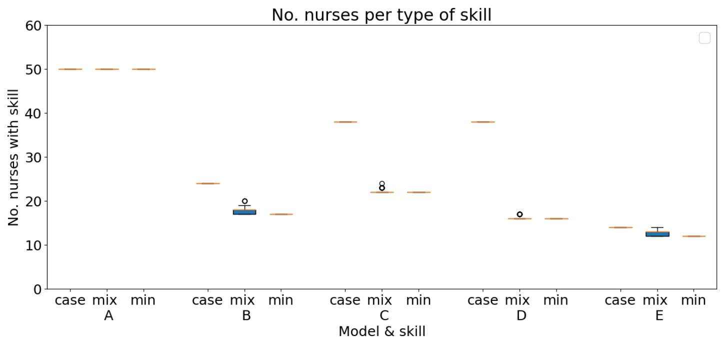


Fig. 3: Number of available skills of each type for the three approaches.

4 Discussion and conclusion

We have described a preliminary design for a decision support tool to plan the staff's distribution of skills. From our case study, we have shown that our tool can be used to indicate which of the available skills can be removed, with only a small impact on both the understaffing in the nominal scenario and scenarios with sick leave. These are results to be further discussed with the hospital.

There are simplifications in our approach, we consider a standard schedule period without individual nurse preferences, holidays, and vacation. The addition of nurse preferences could result in a less robust schedule and the presence of holidays and vacation would make more shifts understaffed in the initial schedule and also reduce the capacity to have buffers, limiting the possibilities to cover for sick leave.

References

1. De Causmaecker, P., Vanden Berghe, G.: A categorisation of nurse rostering problems. Journal of Scheduling **14**(1), 3–16 (2011)
2. Moz, M., Pato, M.V.: An integer multicommodity flow model applied to the rerostering of nurse schedules. Annals of Operations Research **119**(1-4), 285–301 (2003)
3. Oberweger, F.F., Raidl, G.R., Rönnberg, E., Huber, M.: A learning large neighborhood search for the staff rerostering problem. Lecture Notes in Computer Science **13292 LNCS**, 300–317 (2022)
4. Wickert, T.I., Smet, P., Vanden Berghe, G.: The nurse rerostering problem: Strategies for reconstructing disrupted schedules. Computers & Operations Research **104**, 319–337 (2019)
5. Wickert, T.I., Smet, P., Vanden Berghe, G.: Quantifying and enforcing robustness in staff rostering. Journal of Scheduling **24**(3), 347–366 (2021)

Timetabling Belgian youth field hockey competitions with an incomplete round robin tournament

Karel Devriesere[1,2] [0000−0002−6774−849X] and Dries Goossens[1,2] [0000−0003−0224−3412]

[1] Ghent University, Ghent, Belgium
[2] Core lab CVAMO, FlandersMake@UGent, Belgium
{karel.devriesere,dries.goossens}@ugent.be

Keywords: Sports scheduling, Matheuristic, Incomplete round robin.

1 Introduction

Field hockey in Belgium is rapidly growing in popularity. The recent success of the national hockey team (world champions in 2018, European champions in 2019 and Olympic gold in 2020) overwhelmed the Royal Belgian Hockey Association (RBHA) with a multitude of new clubs and an increased number of entries. Therefore, the construction of adequate schedules for hockey youth competitions has become increasingly more challenging.

Currently, teams are partitioned into leagues. The leagues are formed based on the travel time between the home venues of the teams, a problem that is known as the sports team grouping problem [4]. Given the assignment of teams to leagues, a schedule is constructed for each league with the objective of minimizing the number of venue capacity violations. This problem is also known as the multi-league sports scheduling problem [1,2].

Although this approach is widely used in practice, it also has some limitations. First, because these problems are handled sequentially, prioritizing travel times means that capacity violations often cannot be avoided. A solution method to integrate the partitioning and scheduling problems is proposed in [3]. Second, the requirement that teams can only face teams from the same league restricts the solution space. Although a solution might be found that performs well with respect to total travel time, close neighbors are sometimes partitioned into different leagues, leading to much disbelief by the teams.

In this extended abstract, instead of partitioning teams into leagues, we propose to allocate teams to one league only. Typically, the size of this league is too large to fit a round robin tournament. To resolve this issue, teams play a fixed number of games in which they can face any subset of the opponents. Hence, this format is called an incomplete round robin tournament and greatly extends the solution space, providing opportunities for reducing travel times.

2 Problem description

We consider hockey youth competitions of age categories under 7, 8, and 9 in the first half of the 2023/2024 season. Each age category is further split into a boys and girls competition. Moreover, each team is characterized by a strength level ranging from 1-3. This results in 18 categories, including 11 to 93 teams with a total of 808 teams. Each match is played at the home venue of one of the two teams. Whether a team plays home or away in a round is given by its home-away pattern (HAP).

Each club has a limited number of fields that are, in each round, available during a limited number of hours. The RBHA imposes that venue capacity requirements should be strictly met: a timetable should not require a club to host more matches than it physically can. As teams from the same club are often scattered over multiple categories, it is not possible to schedule the categories independently from each other without violating some of these constraints. Other constraints are that the same opponent can be seen at most twice: once at home and once away, the number of home and away games of each team should be balanced and three consecutive home or three consecutive away games are forbidden. Moreover, teams cannot see the same opponent twice in four consecutive slots and can have at most one bye. Finally, the objective is to reduce total travel time.

3 Solution approach and preliminary results

Since a single IP formulation to schedule all youth competitions simultaneously turns out too large to handle by CPLEX, we develop a fix-and-optimize based matheuristic. Neighborhoods are constructed by fixing either all variables related to teams from a subset of categories or all variables related to a subset of time slots. Next, a percentage of the HAPs is fixed for each neighborhood, depending on how difficult the neighborhood is to solve. Moreover, each neighborhood starts with a time limit of one minute. However, time limits are extended to five minutes for promising neighborhoods that are harder to solve. Neighborhoods are chosen based on repeatedly solving multi-armed bandit problems. We avoid being trapped in a local optimum by freeing a subset of the variables and maximizing the number of variables that can be changed from this set, while still guaranteeing feasibility.

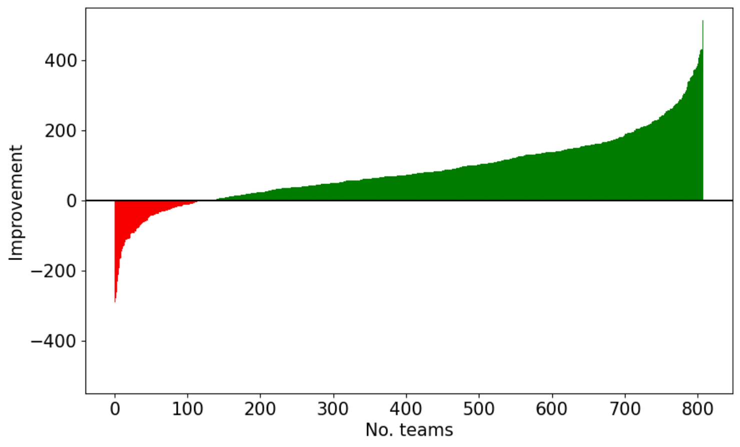
The proposed matheuristic is able to find a solution with a total travel time of 166,666 minutes (<1.5% of best found integer bound) in a reasonable time (12h). In contrast to the original schedule, all venue capacities are satisfied. Our approach also manages to reduce the total travel time by 25%, which corresponds to more than 971 hours. In total, 644 of the 808 teams are better off: the improvement in individual travel time of the teams are given in Figure 1.

Although the situation improves for most of the teams, we see that some teams are also considerably worse off. This is not surprising, as we only look at total travel time at the moment. Therefore, we are currently working on an approach that distributes travel times more fairly over the teams. The RBHA is very positive about our work and will adopt the incomplete round robin tournament in the second half of the 2023/2024 season.

References

1. Davari, M., Goossens, D., Beliën, J., Lambers, R., Spieksma, F.C.: The multi-league sports scheduling problem, or how to schedule thousands of matches. Operations Research Letters **48**(2), 180–187 (2020)

Fig. 1: Distribution of the improvement in team's travel time (in minutes) compared to the original schedule



Green depicts a reduction in travel time, while red depicts an increase in travel time, compared to the original schedule used for the first half of the 2023-2024 season.

2. Li, M., Davari, M., Goossens, D.: Multi-league sports scheduling with different leagues sizes. Eur. J. Oper. Res. **307**(1), 313–327 (2023)
3. Li, M., Goossens, D.: Grouping and timetabling for multi-league sports competitions. In: Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling-PATAT 2022. vol. 3, pp. 192–194 (2022)
4. Toffolo, T.A., Christiaens, J., Spieksma, F.C., Vanden Berghe, G.: The sport teams grouping problem. Annals of Operations Research **275**(1), 223–243 (2019)

Part IV

Demonstration Abstracts

Student Scheduling at Purdue University

Tomáš Müller

Purdue University, West Lafayette, Indiana, USA
`muller@unitime.org`

Abstract. This system demonstration presents the open-source system UniTime[1]. We will concentrate on the student scheduling process at Purdue University, which combines advising, pre-registration, and batch student scheduling, followed by open registration and wait-listing.

Keywords: Student Scheduling, Student Sectioning, UniTime

1 Introduction

Student scheduling, sometimes called *student sectioning*, involves assigning students to classes (course sections) based on their individual course demands. There are many hard constraints and optimization criteria, including course structure, class times and limits, reservations, student and course priorities, student preferences, course alternatives, travel times, and free-time requests. The student scheduling problem in UniTime was first presented in [2] as a proof of concept. It has gone a long way since then and was successfully turned into a software solution capable of scheduling tens of thousands of students, with a scalable web-based user interface, additional constraints, and optimization objectives, and interfacing with many other systems at the university.

Purdue University is the main contributor to the UniTime project, and it is used there for various educational timetabling and scheduling needs, including course timetabling [4], examination timetabling [1], instructor scheduling, and student scheduling. The student scheduling process starts with batch student scheduling available to all undergraduate students, followed by open registration. Graduate and professional students only use the open registration. During the batch scheduling process, a student is advised by their advisor, who fills in course recommendations, and the student submits their course requests. These contain courses the student wants to take, including alternatives and substitutes, preferences, and free-time requests. Based on these and the course timetable, the UniTime solver is run at the end of the pre-registration period, and all students are provided with an initial class schedule. Afterward, during the open registration, the students can go into the system and manually change their schedules, add and drop courses, or waitlist for courses or classes that are full.

The main campus at Purdue University has over 50,000 students. During the Spring 2024 registration, there were over 8,400 timetabled classes from 3,000 courses. Over 35,000 (out of roughly 40,000) undergraduate students filled in their pre-registration,

[1]https://www.unitime.org

which contained over 180,000 course requests, and were provided their initial class schedules.

The collected student pre-registration data can also be used during the course timetabling process, making sure that the created timetable will allow students to get the courses they need. See the ITC 2019 competition[2] for more details [3], as the competition problems were collected using UniTime, and the problem combines both the timetabling aspects (assignment of classes to times and rooms) with student scheduling (assigning students to classes). However, there are fewer student scheduling constraints during the course timetabling process as the solver only minimizes potential student conflicts as one of the optimization criteria, possibly only for a subset of the classes offered at the university at a time (at Purdue, each department builds its own course timetable on top of the centrally-timetabled large lecture room classes).

2 Student Scheduling Problem

The student scheduling problem consists of courses that have already been timetabled, students and their individual course demands, and producing a class schedule for each student. It is modeled as an assignment of student course requests with enrollments, i.e., valid combinations of classes that the student needs to take to enroll into a course. There are various hard constraints, such as students not having a time conflict (unless allowed, in which case the overlapping time is to be minimized), classes and courses being limited in size, or restrictions limiting who can attend a particular class or course. It is also an optimization problem, combining a long list of various criteria, such as maximizing the number of courses each student gets, considering student and course priorities, student preferences, penalizing alternatives and substitutes, minimizing distance conflicts or travel times, etc. In the rest of this section, the most interesting aspects of the student scheduling problem are discussed.

Course Structure Each course may be offered in multiple configurations, such as face-to-face or online, each with multiple components (subparts), such as a lecture and a lab. Each subpart may contain multiple alternative classes. A student requesting a course will get one class of each subpart of a single configuration. There can be additional parent-child relations between individual classes, which also must be followed. So, for example, a student may get Lec 1 - Lab 1, Lec 1 - Lab 2, Lec 2 - Lab 3, or Lec 2 - Lab 4 class combination if attending the course face-to-face, or Dist 1 when attending the course online. Each class can also have a limit, allowing only a certain number of students to get in.

Reservations and Restrictions Access to courses or certain components of courses, such as configurations or individual classes, may be restricted with reservations. A reservation reserves a certain number of seats in the course, or some of its components, for a particular group of students, e.g., identified by their study program. A restriction does not reserve any space, but the students must follow it. For example, a student of an online program may be restricted to the online course configuration, while other

[2]https://www.itc2019.org

students may freely choose between the two configurations, assuming space is available. Similarly, there can be 100 spaces in a course reserved for students of a Computer Science major, or the space in one of the Labs may be reserved for a particular cohort.

Alternative and Substitute Courses A student may provide alternatives to each course. The aim is to get a given number of courses, and one or more alternative courses can be provided for each course. It is also possible to provide substitute courses that can act as alternatives to any other course requests except those that have been marked as no-sub by the advisors (typically those that the student must take). The order in which the courses are requested is also important, as it helps us to break the ties, e.g., when it is not possible to get both courses because they are overlapping in time, or when there are more students requesting the course than the space available.

Student Preferences and Requirements A student can indicate which course configurations and/or classes are preferred for each course. It is also possible to provide free time requirements, which can act as unavailabilities (i.e., a student cannot get a class at the time) or preferences (i.e., an overlapping time with the free time should be minimized), depending on the position of the free time among the courses.

Students may also indicate whether they prefer online or face-to-face classes and whether back-to-backs are preferred or discouraged. For the Summer terms, which are organized into three four-week modules, it is also possible to indicate during which modules the student can have classes and whether they can attend classes on campus.

Student and Course Request Priorities As Purdue is constantly growing its student population, there can be many courses with more students requesting them than space available. A number of priorities have been added to help students graduate on time. First of all, advisors may indicate courses as **vital** to the student, which means that the student absolutely needs the course (or one of the provided alternatives) to progress towards their degree. Moreover, students are divided into priority students (such as athletes and students in university bands and orchestra), students near graduation (with 100 or more credits earned), senior students (60 or more credits), and the rest. Vital course requests take priority over student priorities.

3 Student Scheduling Solver

Student scheduling uses the same hybrid solver as other UniTime modules, combining constraint-programming primitives with local-search-based algorithms and various heuristics. While the solver can work with incomplete solutions, it does not allow the breaking of hard constraints. The search consists of various phases. During the first phase, a schedule for each student is constructed, with vital course requests assigned first and students taken in the order of their priorities. A branch-and-bound algorithm is used for each student to find the best possible schedule using the remaining available space. After the first phase, various other heuristics and neighborhoods are employed, e.g., looking for a swap between two students. Or assigning a student to a course while some other student (or students) is bumped out, e.g., due to the class limits. A branch-and-bound with a limited depth, stochastic hill climbing, and great deluge is used at various search phases. The solver uses multiple CPU cores, benefiting from the fact that individual student schedules do not interact with each other that much, typically only when the last spot in a course, a class, or a reservation is involved.

The same constraint-based model, optimization criteria, and some algorithms are also used in the other problems of student scheduling in UniTime. For example, when a student enrolls in a new course or courses during the open registration period, the same branch-and-bound algorithm creates the best possible class schedule, given the current availability and the student's existing schedule. Or providing suggestions when moving classes around in an existing class schedule.

The student scheduling solver is also used when there is a course timetabling change. For example, UniTime will automatically move students enrolled in a canceled class to other classes of the course or put them on a waitlist when there are no other possible enrollments into the course that are both available to the student and do not conflict with the rest of their schedule.

4 System Demonstration

Student scheduling is an important hard optimization problem of a huge size. In this abstract, a number of aspects that were required to bridge the gap between theory and practice were outlined. The demonstration will present the UniTime system and its user interface, covering the whole student scheduling process at Purdue, step by step. It starts with student advising and pre-registration. It follows with the batch student scheduling, but the solver is also used to provide nightly test runs, which are used to monitor pre-registration progress and to catch any potential issues early. The demonstration will conclude with open registration, showcasing a student making a schedule change, swapping a course, and/or wait-listing for another course or class that is currently full.

References

1. Müller, T.: Real-life examination timetabling. Journal of Scheduling **19**, 257–270 (2016). https://doi.org/10.1007/s10479-014-1643-1
2. Müller, T., Murray, K.: Comprehensive approach to student sectioning. Annals of Operations Research **181**, 249–269 (2010)
3. Müller, T., Rudová, H., Müllerová, Z.: Real-world university course timetabling at the International Timetabling Competition 2019. Journal of Scheduling (To Appear)
4. Rudová, H., Müller, T., Murray, K.: Complex university course timetabling. Journal of Scheduling **14**(2), 187–207 (2011)

Team Formation with Diversity and Similarity Goals

Marco Chiarandini[1 [0000−0002−6136−8496]] and Zhiru Sun[2 [0000−0002−9028−5905]]

[1] Department of Mathematics and Computer Science
[2] Department of Design, Media, and Educational Science
University of Southern Denmark
`{march,zhiru}@sdu.dk`

Abstract. Collaborative learning has been widely used to foster students' communication skills and facilitate joint knowledge construction. However, there exists ongoing debate regarding the optimal formation of teams to maximize the development of these competencies. This work aims to provide educational managers and teachers with a practical tool for team formation, allowing for the control of member diversity *within* teams and similarity *across* teams based on pre-selected student characteristics. The tool takes input in the form of individual student assessments across various characteristics, alongside specifications for team size ranges. Additionally, for each characteristic of the students, it takes a definition of its order of importance and a diversity goal to achieve within the teams, that is, heterogeneity or homogeneity. The output is a distribution of students into teams that satisfies the specified sizes and optimizes diversity goals in the given order while promoting similarity across teams. The tool solves a lexicographic mixed integer linear programming problem. A notable feature of this approach is its ability to accommodate diversity criteria for both numerical and categorical characteristics. Through experimentation with six real-life cases involving up to 151 students per case, the tool demonstrates swift problem-solving capabilities using state-of-the-art solvers. This efficiency renders the tool readily applicable in practical educational settings.

Keywords: Team formation, Mixed integer linear programming, Lexicographic optimization.

1 Introduction

Collaborative learning is an effective method for engaging learners by facilitating communication and idea exchange among team members to construct knowledge together [10]. However, simply putting learners in teams does not guarantee the success of collaborative learning. Therefore, the classification of learners into well-functioning teams is one of the most challenging tasks in the field of collaborative learning. A line of research regarding team composition has categorized collaborative teams into two major types based on the *within*-team composition, which is *homogeneous team* (i.e., learners within a team having similar ability levels) and *heterogeneous team* (i.e., learners within a team having dissimilar ability levels) [13]. Various studies have compared the two types of teams on learners' achievement and social interaction and there seems to be a slight prevalence of the heterogeneous team as the best choice [13]. For example,researchers believe that, compared to homogeneous teams, learners in heterogeneous teams tend to coordinate and create common ground faster and easier because the diverse skills and characteristics of team members might be complementary to each other and to the team as a whole [11,12]. In addition, from the economics perspective, the level of equality or fairness in heterogeneous teams is higher than that in homogeneous teams because resources (e.g., time, knowledge) are more likely to be equally distributed to each team member instead of being collected by a single or limited number of team members [3].

In practice, three primary approaches are employed for team formation: *random grouping* (i.e., assigning learners in the teams by chance), *self-selected grouping* (i.e., the learners choose with whom they want to work), and *controlled grouping* (i.e., assigning learners in the teams by instructors or computing systems based on certain criteria) [1,6,2]. The random grouping method commonly lacks control over team homogeneity or heterogeneity, potentially leading to unequal participation and the formation of teams with varying characteristics. This can result in disparities among teams, fostering a sense of unfairness [7]. The self-selected grouping method tends to produce homogeneous teams characterized by shared interests and amicable relationships among members, albeit often leading to decreased task orientation and engagement in off-task behaviors [6]. The controlled grouping method addresses these issues by facilitating the creation of teams with desired levels of diversity or similarity *within* teams while also controlling for variation *among* teams. However, achieving these objectives complicates the assignment process both in terms of formalization and optimal solution finding. Consequently, research has increasingly focused on algorithmic approaches to achieve controlled team formation.

In the past decade, a variety of algorithm-based team formation methods has been proposed to form controlled teams, that is, to create teams that are as similar among themselves as possible (*inter-homogeneous*), while maximizing the learners' individual differences within such teams (*intra-heterogeneous*). The majority of team formation algorithms are based on population-based metaheuristics such as ant colony optimization [4], particle swarm optimization [9], and genetic algorithm [2]. Local search-based heuristics such as random restart hill-climbing [8] and variable neighborhood search are also employed to form collaborative teams [15]. There is no standard definition of the optimization criteria in these references. Perhaps the most flexible tool available is CATME [8], that allows instructors to define their own characteristics of interest, their weight of importance in the team formation and whether within-team similarity or dissimilarity should be promoted. Information about the pre-selected characteristics can be collected directly from the students using the web application built around the tool. Characteristics are handled by discretizing them. One interesting characteristic modeled in this way is the student-schedule compatibility to favor the creation of teams that can actually meet. CATME then assigns students to teams by maximizing the minimum of a compliance measure computed on each team. The assignment is found starting by a random assignment and improving it by swapping students in a hill climbing fashion. However, in studies that aim at assessing team creation policies (e.g., which characteristics are relevant to consider, whether they should be similar or dissimilar) finding heuristic solutions to the team formation problem is undesirable because it adds a confounding element to the analysis, namely, the unknown degree of approximation of the optimal solutions.

In our work, we formalize the problem in a way that can be solved to proven optimality by mixed-integer linear programming (MILP) solvers. Similarly to [8], we define a compliance measure for each characteristic within the teams. We distinguish between numerical characteristics (e.g., a real number from $[0, 1]$) and categorical ones (e.g., the nationality) and define different measures for these types. For numerical characteristics we use the largest range of values within the team, while for the categorical ones we use the number of different categories represented in the team. For each characteristic in the order of importance we solve an optimization problem that tries to adjust the measure so that within-team compliance to similarity or dissimilarity is maximized and among-team similarity is also maximized. It is easy to extend this approach with side constraints like ranges on the size of the teams or student incompatibilities of the type "two persons cannot be in the same team" or similar. Our tests conducted on instances of the problem involving up to 151 students indicate that the MILP approach exhibits notable efficiency and pratical utility. Leveraging this approach, we have successfully formed heterogenous groups and examined the efficiency of such diversified teams in enhancing students' achievement and fostering positive emotions during collaborative learning [14].

2 Problem Formulation

We want to team up a set S of students indexed by s. Each student is characterized by a set of characteristics (or factors or features) $F = \{1..m\}$ indexed by f. Some of these characteristics, $F^q \subseteq F$, are quantitative or numerical, that is, they take values in $\mathbb{R}$; others, $F^c \subseteq F$, are categorical and can be mapped to take values in $\mathbb{N}$ or $\mathbb{B}$. For example, the gender of a person can be mapped into the integer numbers 0 and 1. A categorical characteristic $f \in F^c$ takes values from a finite set of categories (or levels) $L_f = \{1..v_f\} \subset \mathbb{N}$ indexed by ℓ. Thus, a student $s \in S$ is characterized by a vector $\vec{c}(s) = [c_{s1}, \dots, c_{sm}]$ with $c_{sf} \in \mathbb{R}$ for $f \in F^q$ and $c_{sf} \in \mathbb{N}$ for $f \in F^c$. Further, let $\pi : F \longrightarrow F$ be a permutation of the characteristics such that the permutation $\pi(1)..\pi(m)$ induces a strict total order on the characteristics (from most to least important).

We aim at combining the students in S into a set of teams $\mathcal{T} \subset 2^S$. We can denote such a team formation as a mapping $\sigma : S \longrightarrow \mathcal{T}$. Thus, $\sigma(s) = T$, if student $s \in S$ is assigned to team $T \in \mathcal{T}$. We want the team formation to be a partition of $\mathcal{T}$, that is, $T_1 \cap T_2 = \emptyset$ for any $T_1, T_2 \in \mathcal{T}$ and $\bigcup_{T \in \mathcal{T}} T = S$, and such that the size of each team T in $\mathcal{T}$ under σ is $\{\lfloor |S|/|T| \rfloor, \lceil |S|/|T| \rceil\}$, i.e., as equal as possible. Among all team formations satisfying these requirements, Σ, we want to find those that maximize within-team compliance and among-team similarity with respect to the characteristics under the order induced by π.

We formulate the preference criterion above in the following way. For a team formation σ, let $\delta_{f,p,T}$ be the absolute difference in the values of the characteristic f for any pair of students $p = (s, r)$ in T, that is, $\delta_{f,p,T} = |c_{sf} - c_{rf}|$ for all $f \in F^q, T \in \mathcal{T}$ and $\{p = (s, r) \mid \sigma(s) = \sigma(r) = T\}$. Then, let $\underline{\theta}_{f,T}$ and $\overline{\theta}_{f,T}$ for $f \in F^q$ be the smallest and the largest of these differences within each team T and $\underline{\theta}_f$ and $\overline{\theta}_f$ the minimum and

	Within-team heterogeneity	Within-team homogeneity
Categorical factor	$\min \overline{\eta}_f$ $\max \underline{\eta}_f$	$\min \overline{\eta}_f$ —
Numerical factor	$\min \overline{\theta}_f$ $\max \underline{\theta}_f$	$\min \overline{\theta}_f$ —

Table 1: The optimization applied for the two types of factors under the two different expression of within-team compliance (heterogeneity or homogeneity)

maximum difference throughout all teams, that is, for $f \in F^q$:

$$\underline{\theta}_f = \min_{T \in \mathcal{T}} \underline{\theta}_{f,T} = \min_{T \in \mathcal{T}, p \in T} \delta_{f,p,T}$$

$$\overline{\theta}_f = \max_{T \in \mathcal{T}} \overline{\theta}_{f,T} = \max_{T \in \mathcal{T}, p \in T} \delta_{f,p,T}$$

Similarly, for a team formation σ, let $\mu_{f,T}$ be the number different of categories of the characteristic $f \in F^c$ represented by the members of T and let $\underline{\eta}^c_f$ and $\overline{\eta}^c_f$ for $f \in F^c$ be, respectively, the smallest and largest number of categories present in any $T \in \mathcal{T}$, that is, for $f \in F^c$

$$\underline{\eta}_f = \min_{T \in \mathcal{T}} \mu_{f,T}$$

$$\overline{\eta}_f = \max_{T \in \mathcal{T}} \mu_{f,T}$$

We use $\underline{\eta}_f$ and $\underline{\theta}_f$ as measures of the within-team dissimilarity that we may want to maximize or minimize and $\overline{\theta}_f$ and $\overline{\eta}_f$ as measures of the among-team dissimilarity that we want to minimize. In Table 1, we consider the different cases. Accordingly, if we aim for within-team heterogeneity and among-team homogeneity, we aim at the following: for each categorical factor, first, we maximize the smallest number of categories in the teams, thus promoting within-team heterogeneity, and, second, we minimize the largest number of categories, thus promoting the range between minimum and maximum number of categories among the teams to be small and consequently favoring among-team homogeneity; for numerical factors, first, we maximize the smallest difference within the teams, thus promoting within-team heterogeneity, and, second, we minimize the largest value of the differences within the teams, thus aiming at the smallest range between these values and consequently promoting among-team homogeneity. If we aim at homogeneity within and among the teams we only minimize the largest number of categories and the largest overall difference.

We solve this *multi-objective optimization problem* by lexicographic optimization using the strict order π of importance on the characteristics. For the case of aiming at within-team heterogeneity, with two objectives to optimize for every characteristics each optimization problem considers the following objective:

$$\operatorname{lex}\max_{\sigma\in\Sigma}(\varphi_1(\sigma),\ldots,\varphi_{2m}(\sigma))$$

where

$$\varphi_i(\sigma)=\begin{cases}\underline{\theta}_f & \text{if } i=2\pi(f)-1 \text{ and } f\in F^q\\ -\overline{\theta}_f & \text{if } i=2\pi(f) \text{ and } f\in F^q\\ \underline{\eta}_f & \text{if } i=2\pi(f)-1 \text{ and } f\in F^c\\ -\overline{\eta}_f & \text{if } i=2\pi(f) \text{ and } f\in F^c\end{cases}\qquad \text{for } i=1..2m.$$

This means that we consider first the characteristic that is first in the order induced by π, that is, $f \in F$ such that $\pi(f) = 1$, and maximize the value $\underline{\theta}_f$ or $\underline{\eta}_f$ depending on whether f is a quantitative or a categorical factor, respectively. Once the optimal team formation with respect to this objective has been found, we set that objective as a constraint and maximize $-\overline{\theta}_f$ or $-\overline{\eta}_f$, which corresponds to minimize $\overline{\theta}_f$ or $\overline{\eta}_f$. Then, we consider the next characteristic in the order, i.e., $f \in F$ such that $\pi(f) = 2$, and repeat the process while keeping all previously optimized objectives as constraints. We proceed in this way until all characteristics are considered.

Each optimization problem can be formulated as a mixed integer linear programming (MILP) problem (see Appendix A) and solved with one of the available general-purpose MILP solvers. Artificial restrictions on the set Σ of feasible team formations, such as "student s cannot be in the same team as student r" can be easily added within the same formalism.

Consider the example of Figure 1. We have four students s_1, s_2, s_3, s_4 described by two categorical characteristics C_1 and C_2 and four numerical characteristics, C_3, C_4, C_5, C_6. The order of importance of the characteristics is $\pi = (1, 2, 3, 4, 5, 6)$. We want to group the students in two teams T_1, T_2. The table on the left shows the values of the characteristics for the four students with columns in the same order of importance of the characteristics. The signs $+/-$ indicate whether we are interested in within-group heterogeneity or homogeneity, respectively, for the corresponding characteristic.

The assignment made by the algorithm is shown in the table on the right. It corresponds to $\sigma(s_2) = \sigma(s_3) = T_1$ and $\sigma(s_1) = \sigma(s_4) = T_2$. The last two rows show the measures of compliance among the teams for each characteristic. For C_1 both teams include two categories, which is the best possible in this case. For C_2 it is not possible to have two categories in both teams because of the restriction imposed on C_1. For the following categories it seems that the situation can not improved any further because of the constraints introduced on C_1.

3 Practical Experience

We used the tool in six real-life situations on 20, 21, 79, 99, 110, 151 students with 12 or 13 characteristics of both types giving rise to a maximum of 25 objectives. We used gurobi [5] as MILP solver, which can handle lexicographic optimization automatically. On all instances except one, the full lexicographic series could be solved in less then

	C1	C2	C3	C4	C5	C6
	+	−	+	+	−	+
s_1	1	2	0.3	0.4	0.2	0.3
s_2	0	3	0.5	0.3	0.7	0.8
s_3	1	4	0.1	0.7	0.3	0.2
s_4	0	2	0.8	0.9	0.4	0.5

Team 1	C1	C2	C3	C4	C5	C6
s_2	0	3	0.5	0.3	0.7	0.8
s_3	1	4	0.1	0.7	0.3	0.2
Team 2	C1	C2	C3	C4	C5	C6
s_1	1	2	0.3	0.4	0.2	0.3
s_4	0	2	0.8	0.9	0.4	0.5
Measures	C1	C2	C3	C4	C5	C6
$\overline{\eta}_f$ or $\overline{\theta}_f$	2	2	0.5	0.5	0.4	0.6
$\underline{\eta}_f$ or $\underline{\theta}_f$	2	1	0.4	0.4	0.2	0.2

Fig. 1: A numerical example. On the left, the input data for a case with four students (on the rows) and six characteristics (on the columns), of which the first two categorical. The order of priority on the characteristics is the same as their indices and the preference for within-team similarity (+) or dissimilarity (−) is indicated in the second row of the table. On the right, the teams produced by the solution to the model.

60 seconds of time. The instance with 110 turned out harder to solve. In 1000 seconds, only the first 3 objectives were solved, the fourth proved much more computationally demanding. We decided to halt the solution process and to use the best solution found up to that point.

4 Discussion

We have developed a tool designed for team formation, which considers relevant student characteristics. Emphasizing diversity with respect to these characteristics within teams can potentially foster competence development, while maintaining similarity across teams in their treatment of student characteristics promotes fairness. We formulated these goals in a mixed integer linear programming model accommodating both numerical and categorical characteristics. We dealt with the presence of multiple characteristics by asking teachers to prioritize them a priori, thus solving a series of lexicographic optimization problems. Other approaches for managing multiple characteristics, such as weighted sum and allowing partial degradation of previous objectives, are feasible avenues to explore. While Pareto optimization presents an intriguing alternative, its implementation entails greater complexity. Our tool has undergone testing solely on real-life instances involving up to 151 students, organized into teams of 5. The results indicate that the approach is generally computationally practicable and efficient. For instances that require more computational resources, a transition from an exact to a heuristic approach is possible by allocating a limited time budget for solving each objective in the lexicographic series. This budget allocation can prioritize objectives associated with higher priority characteristics, thereby facilitating computational tractability.

We are planning to conduct scalability tests on larger artificial instances. Moreover, we would like to deepen our understanding of the solution quality and the influencing factors' impact on outcomes. Notably, as illustrated in the numerical example of Fig. 1, the situation might become blocked very early. Finally, we are actively developing a web-based application to serve as an interface for out tool, facilitating its accessiblity and usability.

Appendix A – The MILP Model

Let x_{st} for $s \in S$ and $T_t \in \mathcal{T}$ be the binary variables that denote the assignment of s to T_t under σ. Let also y_t for $T_t \in \mathcal{T}$ be auxiliary binary variables indicating whether a team T_t contains students or not. A feasible team formation $\mathcal{X}$ satisfies:

$$\sum_{T_t \in \mathcal{T}} x_{st} = 1 \qquad \forall s \in S \tag{1}$$

$$\sum_{s \in S} x_{st} \le b_t y_t \qquad \forall T_t \in \mathcal{T} \tag{2}$$

$$\sum_{s \in S} x_{st} \ge a_t y_t \qquad \forall T_t \in \mathcal{T} \tag{3}$$

$$\sum_{s \in S} x_{s,t} \ge \sum_{s \in S} x_{s,t+1} \qquad \forall t = 1..|\mathcal{T}| - 1 \tag{4}$$

$$x_{st} \in \mathbb{B} \qquad \forall s \in S, T_t \in \mathcal{T} \tag{5}$$

$$y_t \in \mathbb{B} \qquad \forall T_t \in \mathcal{T} \tag{6}$$

Constraints (1) ensure all students are assigned to a team. Constraints (2)-(3) ensure that if a team is created it is assigned students between its lower and upper bound a_t, b_t, respectively. Constraints (4) are symmetry breaking constraints.

To compute the values $\overline{\theta}_f, \underline{\theta}_f$ we need to introduce auxiliary binary variables $z_{s_1,s_2,t}$ that are one if the two students s_1 and s_2 are in team T_t and zero otherwise. For an feasible formation $\vec{x} \in \mathcal{X}$:

$$x_{s_1,t} + x_{s_2,t} - 1 \le z_{s_1,s_2,t} \qquad \forall s_1, s_2 \in S, \forall T_t \in \mathcal{T} \tag{7}$$

$$x_{s_1,t} \ge z_{s_1,s_2,t} \qquad \forall s_1, s_2 \in S, \forall T_t \in \mathcal{T} \tag{8}$$

$$x_{s_2,t} \ge z_{s_1,s_2,t} \qquad \forall s_1, s_2 \in S, \forall T_t \in \mathcal{T} \tag{9}$$

$$\overline{\theta}_f \ge |c_{s_1,f} - c_{s_2,f}| z_{s_1,s_2,t} \qquad \forall f \in F^q, \forall s_1, s_2 \in S, \forall T_t \in \mathcal{T} \tag{10}$$

$$\underline{\theta}_f \le M_f(1 - z_{s_1,s_2,t}) + |c_{s_1,f} - c_{s_2,f}| z_{s_1,s_2,t} \qquad \forall f \in F^q, \forall s_1, s_2 \in S, T_t \in \mathcal{T} \tag{11}$$

$$z_{s_1,s_2,t} \in \mathbb{B} \qquad \forall s_1, s_2 \in S, \forall T_t \in \mathcal{T} \tag{12}$$

$$\overline{\theta}_f \in \mathbb{R}_0^+ \qquad \forall f \in F^q \tag{13}$$

$$\underline{\theta}_f \in \mathbb{R}_0^+ \qquad \forall f \in F^q \tag{14}$$

Constraints (7)-(9) ensure the z variable take the value described. Constraints (10)-(11) force $\overline{\theta}_f$ and $\underline{\theta}_f$ to stay above and below all realized differences, respectively. We set $M_f = \max_{s1,s_2 \in S}\{|c_{s_1,f} - c_{s_2,f}|\}$.

To compute the values $\overline{\eta}_f, \underline{\eta}_f$ we will slightly abuse of notation and use $\eta_{tf\ell}$ and η_{tf} to indicate for characteristic $f \in F^c$ and team $T_t \in \mathcal{T}$ whether the category ℓ is represented and the number of different categories represented in the team, respectively.

$$x_{st} \leq \eta_{tf\ell} \qquad \forall s \in \{s \mid s \in S \wedge c_{sf} = \ell\}, \ell \in L_f, f \in F^c, \forall t \in \mathcal{T} \tag{15}$$

$$\eta_{tf\ell} \leq \sum_{s \in S \mid c_{sf} = \ell} x_{st} \qquad \forall \ell \in L_f, f \in F^c, \forall t \in Gi \tag{16}$$

$$\eta_{tf} = \sum_{\ell \in F_f} \eta_{tf\ell} \qquad \forall f \in F^c, \forall t \in \mathcal{T} \tag{17}$$

$$\overline{\eta}_f \geq \eta_{tf} \qquad \forall f \in F^c, \forall T_t \in \mathcal{T} \tag{18}$$

$$\underline{\eta}_f \leq \eta_{tf} \qquad \forall f \in F^c, \forall T_t \in \mathcal{T} \tag{19}$$

$$\eta_{tf\ell} \in \mathbb{B} \qquad \forall \ell \in L_f, \forall f \in F^c, \forall t \in \mathcal{T} \tag{20}$$

$$\eta_{tf} \in \mathbb{Z}_0^+ \qquad \forall f \in F^c, \forall t \in \mathcal{T} \tag{21}$$

$$\overline{\eta}_f \in \mathbb{Z}_0^+ \qquad \forall f \in F^c \tag{22}$$

$$\underline{\eta}_f \in \mathbb{Z}_0^+ \qquad \forall f \in F^c \tag{23}$$

Constraints (15)-(16) ensure $\eta_{tf\ell}$ is either one or zero depending on whether any student among those who have that category are assigned to the team. Constraints (17) collect the number of different categories present in the team. Constraints (18) and (19) force $\overline{\eta}_f$ and $\underline{\eta}_f$ to stay above and below the number of categories over all teams, respectively.

We can finally state the overall MILP model with the objective function defined in the main text:

$$\begin{aligned} &\operatorname{lex}\max_{\sigma \in \Sigma} \; (\varphi_1(\sigma), \ldots, \varphi_{2m}(\sigma)) \\ &\text{subject to } (1) - (6) \\ &\qquad\qquad\; (7) - (14) \\ &\qquad\qquad\; (15) - (23). \end{aligned}$$

References

1. Chan, T., Chen, C.M., Wu, Y.L., Jong, B.S., Hsia, Y.T., Lin, T.W.: Applying the genetic encoded conceptual graph to grouping learning. Expert Systems with Applications **37**(6), 4103–4118 (2010)
2. Chen, C.M., Kuo, C.H.: An optimized group formation scheme to promote collaborative problem-based learning. Computers & Education **133**, 94–115 (2019)
3. Fallucchi, F., Fatas, E., Kölle, F., Weisel, O.: Not all group members are created equal: Heterogeneous abilities in inter-group contests. Luxembourg Institute of Socio-Economic Research (LISER) Working Paper Series **14**, 1–42 (2018)
4. Graf, S., Bekele, R.: Forming heterogeneous groups for intelligent collaborative learning systems with ant colony optimization. In: International conference on intelligent tutoring systems. pp. 217–226. Springer (2006)
5. Gurobi Optimization, L.: Gurobi optimizer reference manual (2021), https://www.gurobi.com

6. Hilton, S., Phillips, F.: Instructor-assigned and student-selected groups: A view from inside. Issues in Accounting Education **25**(1), 15–33 (2010)
7. Huxham, M., Land, R.: Assigning students in group work projects. can we do better than random? Innovations in Education and Training International **37**(1), 17–22 (2000)
8. Layton, R.A., Loughry, M.L., Ohland, M.W., Ricco, G.D.: Design and validation of a web-based system for assigning members to teams using instructor-specified criteria. Advances in Engineering Education (2), 1–28 (2010)
9. Lin, Y.T., Huang, Y.M., Cheng, S.C.: An automatic group composition system for composing collaborative learning groups using enhanced particle swarm optimization. Computers & Education **55**(4), 1483–1493 (2010)
10. Liu, C.C., Tsai, C.C.: An analysis of peer interaction patterns as discoursed by on-line small group problem-solving activity. Computers & Education **50**(3), 627–639 (2008)
11. Lou, Y., Abrami, P.C., Spence, J.C., Poulsen, C., Chambers, B., d'Apollonia, S.: Within-class grouping: A meta-analysis. Review of educational research **66**(4), 423–458 (1996)
12. Manske, S., Hecking, T., Chounta, I.A., Werneburg, S., Hoppe, H.U.: Using differences to make a difference: a study on heterogeneity of learning groups. In: Lindwall, O., Haäkkinen, P., Koschman, T., Tchounikine, P., Ludvigsen, S. (eds.) Exploring the Material Conditions of Learning: The Computer Supported Collaborative Learning (CSCL) Conference 2015. vol. 1, p. 8. International Society of the Learning Sciences, Inc.[ISLS]., Gothenburg, Sweden (2015)
13. Murphy, P.K., Greene, J.A., Firetto, C.M., Li, M., Lobczowski, N.G., Duke, R.F., Wei, L., Croninger, R.M.: Exploring the influence of homogeneous versus heterogeneous grouping on students' text-based discussions and comprehension. Contemporary Educational Psychology **51**, 336–355 (2017)
14. Sun, Z., Chiarandini, M.: An exact algorithm for group formation to promote collaborative learning. In: Warschauer, M., Lynch, G. (eds.) LAK21: 11th International Learning Analytics and Knowledge Conference (LAK21), April 12–16, 2021, Irvine, CA, USA. Association for Computing Machinery, New York, NY, USA (2021)
15. Takači, D., Marić, M., Stankov, G., Djenić, A.: Efficiency of using vns algorithm for forming heterogeneous groups for cscl learning. Computers & Education **109**, 98–108 (2017)

A Novel Potential-based Algorithm for the Vertex Coloring Problem and Its Application to Timetabling Online Platform DaAlgo

Hyunwoo Jung[1]

KSA of KAIST, Baegyanggwanmun-ro 105-47, Busanjin-gu, Busan, Republic of Korea
aryonhwjung@ksa.kaist.ac.kr

Abstract. Given an undirected graph $G(V, E)$, the Vertex Coloring Problem(VCP) requires to assign a color to each vertex in such a way that no two adjacent vertices have the same color and the number of colors used is minimized. A novel heuristic for graph coloring problem and its application to a timetable construction online platform is presented in this paper. It shows better results than the previous heuristic DSatur [9]. The online platform shows very fast timetable construction and very convenient UX. We want to demonstrate our platform in front of timetable specialists.

Keywords: Graph Coloring, Heuristic Algorithm, Online Platform

1 Introduction

Given an undirected graph $G(V, E)$, the Vertex Coloring Problem(VCP) requires to assign a color to each vertex in such a way that no two adjacent vertices have the same color and the number of colors used is minimized. The Vertex Coloring Problem is a well-known NP-hard Problem [1] with real world applications in many areas including scheduling [2], timetabling [3], register allocation [4], and communication networks [5]. Despite its importance to real-world applications, few exact algorithms for VCP have been found, and are able to solve only small instances up to 100 vertices for random graphs[6,7,8]. Many heuristic approaches have been proposed to deal with graphs of hundreds or thousands of vertices. The first heuristic approaches are derived based on greedy construction algorithms. The best-known greedy techniques are the maximum saturation degree(DSatur) and the Recursive Largest First(RLF) heuristics proposed by Brelaz [9] and by Leighton [10], respectively. Many meta-heuristic algorithms have been proposed for VCP based on tabu search [11], simulated annealing [12], genetic algorithm [13], linear programming [15], etc. In this paper, a novel heuristic for VCP with good performance is designed. In section 2, a novel heuristic **PAE** is shown. In section 3, a computational comparison result between **PAE** and **DSatur** is shown. In section 4, we show our timetable construction online platform that uses the algorithm **PAE**.

2 A Potential-based Algorithm for the Vertex Coloring Problem

In the following, a novel potential-based algorithm $\mathbf{PAE(G, k)}$ for VCP with exponential decay is described. The algorithm $\mathbf{PAE(G, k)}$ is checking whether the given graph $\mathbf{G}$ can be colored using at most k colors.

Algorithm 2: Potential-based Algorithm for the Vertex Coloring Problem with Exponential Decay(PAE)

```
Algorithm: PAE(G, k)
Data: G(V, E) and k
Result: colorability of vertices in G(V, E) using at most $k$ colors
$n \leftarrow len(V)$
$Stack \leftarrow []$
for $v \leftarrow 1$ to $n$ do
    $\phi_v = n$
end
$Q$ is a priority queue containing all the vertices with their priorities which means the
  number of distinct colors of adjacent vertices for each vertex
while there are any uncolored vertices do
    pop vertices from $Q$ until we get a vertex $v$ with the recent update
    color $v$ with the lowest usable color
    if the number of used colors is greater than $k$ then
        if $\phi_v$ is zero then
            return False
        end
        $\phi_v = \phi_v // 2$
        pop vertices from Stack uncoloring(updating) the vertices up to Stack[$\phi_v$]
        color $v$ with the lowest usable color
    else
        $\phi_v = min(\phi_v, len(Stack))$
    end
    append $v$ to Stack
end
return True
```

Intuitively, the potential ϕ_v for each vertex v means the highest possible order of v that leads to valid coloring using at most k colors. We do a binary search on the number of usable colors k to determine the tight k, this algorithm is called **PAE**. In this paper, as the priority of each element in the queue of the algorithm **PAE**, we are using the priority of DSatur [9] heuristic which is the number of distinct colors of adjacent vertices for each vertex. When the number of distinct colors of adjacent vertices for each vertex are same, the tie is broken by the higher degree of each vertex. The algorithm **PAE** is very simple and shows better performance than DSatur heuristic [9]. The time complexity of **PAE** is $O(|V|^2 \log(|V| + |E|))$. It can deal with graphs with thousands of vertices.

3 Experimental Results

We compare the performance of PAE and DSatur using the subset of the DIMACS benchmark graph coloring instances. The result is shown in the Table 1. We implemented the algorithms PAE and DSatur using Python. We run the program on a laptop machine with an Intel Core i7 CPU 2.30 GHz, and 24GB RAM. Large instances from the DIMACS benchmark whose number of vertices is greater than 100 are used for the comparison test. For all the cases, the algorithm PAE showed better or equal results on the aspect of the number of colors used. In the comparison table, boldfaced numbers mean that PAE shows better results than DSatur. The time in the table means the elapsed seconds until the program is finished. Note that the program is always finished with a coloring. Especially, in the class scheduling graphs shool1 and shool1_nsh, PAE showed almost optimal results. This is important since the algorithm **PAE** of this paper is used for an online platform that services the automatic construction of an exam timetable. χ^* means the best-known solution according to the paper by Malaguti et. al [16].

	PAE		DSatur [9]		
instance	k	time	k	time	χ^*
DSJC1000.1	**25**	277	27	0.214	20
DSJC1000.5	**113**	3309	115	0.968	83
DSJC1000.9	**296**	8162	310	2.156	224
DSJC125.1	6	0.093	6	0.016	5
DSJC125.5	**20**	2.223	22	0.028	17
DSJC125.9	**48**	9.056	51	0.033	44
DSJC250.1	**10**	2.672	11	0.025	8
DSJC250.5	**35**	24	37	0.074	28
DSJC250.9	**84**	107	90	0.125	72
DSJC500.1	15	39	15	0.054	12
DSJC500.5	**62**	319	66	0.248	48
DSJR500.1	**12**	0.095	13	0.019	12
DSJR500.5	**125**	163	132	0.279	122
latin_square_10	**124**	2220	132	1.066	99
le450_15a	**16**	2.423	17	0.048	15
le450_15b	16	1.586	16	0.039	15
le450_15c	**23**	28	24	0.076	15
le450_15D	**23**	35	24	0.083	15
le450_25c	**27**	34	29	0.081	26
le450_25d	28	20	28	0.079	26
le450_5a	9	3.435	9	0.028	5
le450_5b	**9**	5	10	0.022	5
le450_5d	**7**	3.721	8	0.063	5
school1	**14**	3.877	22	0.077	14
school1_nsh	**15**	4.303	25	0.059	14

Table 1: Performace Comparison between PAE and DSatur

4 Implementation of the Online Timetabling Platform DaAlgo

By generalizing the graph coloring idea of this paper, an online timetabling platform https://www.daalgo.org is implemented. The online platform services making exam timetables automatically using the customer's data about students' registrations for courses. We use an Excel file for input data. We process the input data using an algorithm that generalizes PAE, and then render the exam timetable to a browser after getting the constructed timetable from the backend. For the backend, we used FAST API, Postgres SQL, and Python. For the front end, we used React. For the cloud, we used the AWS cloud. We used AWS RDS, S3, Amplify, EC2, etc. See the Fig. 1 for the AWS architecture of our platform. Fig. 2 shows a result of an exam timetable construction which is rendered on the Chrome web browser. After rendering the exam timetable on the web browser, we can use drag and drop to change the exam schedules of some courses. Finally, we can download the exam timetable as the Excel format. We can make the exam timetable automatically and modify the timetable on the same platform at once.

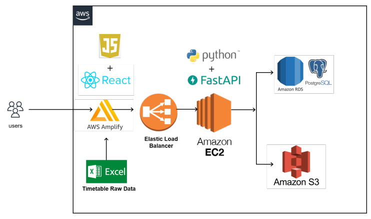
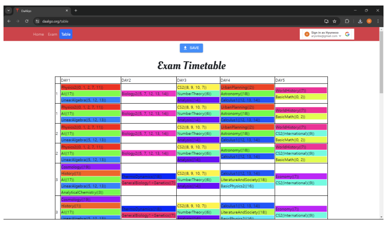


Fig. 1: AWS Architecture of DaAlgo Platform



Exam Timetable

	DAY1	DAY2	DAY3	DAY4	DAY5

1	Physics2((0, 1, 2, 7, 11))
	AI((17))
	LinearAlgebra((5, 12, 13))
		Biology2((5, 7, 12, 13, 14))	CS2((8, 9, 10, 7))
			NumberTheory((6))
			Analysis((14))
				UrbanPlanning((2))
				Astronomy((18))
				Calculus1((12, 13, 14))
					WorldHistory((7))
					BasicMath((0, 2))

1	Physics2((0, 1, 2, 7, 11))
	AI((17))
	LinearAlgebra((5, 12, 13))
		Biology2((5, 7, 12, 13, 14))	CS2((8, 9, 10, 7))
			NumberTheory((6))
			Analysis((14))
				UrbanPlanning((2))
				Astronomy((18))
				Calculus1((12, 13, 14))
					WorldHistory((7))
					BasicMath((0, 2))

2	Physics2((0, 1, 2, 7, 11))
	AI((17))
	LinearAlgebra((5, 12, 13))
		Biology2((5, 7, 12, 13, 14))	CS2((8, 9, 10, 7))
			NumberTheory((6))
			Analysis((14))
				UrbanPlanning((2))
				Astronomy((18))
				Calculus1((12, 13, 14))
					WorldHistory((7))
					CS2(International)((9))
					BasicMath((0, 2))

2	Physics2((0, 1, 2, 7, 11))
	AI((17))
	LinearAlgebra((5, 12, 13))
		Biology2((5, 7, 12, 13, 14))	CS2((8, 9, 10, 7))
			NumberTheory((6))
			Analysis((14))
				Astronomy((18))
				Calculus1((12, 13, 14))
					WorldHistory((7))
					CS2(International)((9))
					BasicMath((0, 2))

3	Cosmology((19))
	History((1))
	AI((17))
	LinearAlgebra((5, 12, 13))
	AnalyticalChemistry((3))
		ThermoDynamics((18))
		GeneralBiology1+Genetics((7))
			CS2((8, 9, 10, 7))
			NumberTheory((6))
			Analysis((14))
				Calculus1((12, 13, 14))
				LiteratureAndSociety((18))
				BasicPhysics2((16))
					Economy((7))
					CS2(International)((9))

3	Cosmology((19))
	History((1))
	AI((17))
		ThermoDynamics((18))
		GeneralBiology1+Genetics((7))
			CS2((8, 9, 10, 7))
			NumberTheory((6))
				Calculus1((12, 13, 14))
				LiteratureAndSociety((18))
					Economy((7))
					CS2(International)((9))

Fig. 2: The Result Exam Timetable

5 Concluding Remarks

In this abstract, we presented a novel heuristic for VCP which shows very good performance for school-related graphs. We apply the algorithm to implement an online service platform that constructs an exam timetable based on the data of students' registration information for courses. The online platform is very fast to make an exam timetable automatically and provides users with a very new UX. We will share our novel ideas and our creative timetable service platform.

Acknowledgements This research was supported by the research program at the Korea Science Academy of KAIST with funding from the Korean government (Ministry of Science and ICT).

References

1. M.R.Garey, D.S. Johnson.: Computers and Intractability: A Guide to the Theory of NP-Complelteness. W.H.Freeman & Co., New York (1979).
2. F.T.Leighton.: A Graph Coloring Algorithm for Large Scheduling Problems. Journal of Research of the National Bureau of Standards, 84(6), 489–503 (1979).
3. D. De Werra.: An Introductoin to Timetabling. European Journal of Operational Research, 19:151-162 (1985).
4. T.K. Woo, S.Y.W. Su, and R. Newman Wolfe.: Resource Allocation in a Dynamically Partitionable Bus Network using a Graph Coloring Algorithm. IEEE Trans. Commun., 39(12), 1794-1801 (2002).
5. F.C. Chow and J.L.Hennessy.: The Priority-based Coloring Approach to Register Allocation. ACM Transactions of Programming Languages and Systems, 12(4), 501-536 (1990).
6. T.J. Sager and S. Lin.: A Pruning Procedure for Exact Graph Coloring. ORSA Journal on Computing, 3(3), 226-230 (1991).
7. E.C. Sewell.: An Improved Algorithm for Exact Graph Coloring. In D.S. Johnson and M.A. Trick, editors, Cliques, Coloring, and Satisfiability: 2nd DIMACS Implementation Challange, 1993, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 359-373 (1996).
8. II. Méndez-Díaz, P. Zabala.: A Branch-and-Cut Algorithm for Graph coloring, Discrete Applied Mathematics 154, 826–847 (2006).
9. D. Brμelaz.: New Methods to Color the Vertices of a Graph. Communications of the ACM, 22(4), 251-256 (1979).
10. F.T. Leighton.: A Graph Coloring Algorithm for Large Scheduling Problems. Journal of Research of the National Bureau of Standards, 84(6), 489-503 (1979).
11. A. Hertz, D. de Werra.: Using Tabu Search Techniques for Graph Coloring, Computing 39, 345–351 (1987).
12. D.S. Johnson, C.R. Aragon, L.A. McGeoch, C. Schevon.: Optimization by Simulated Annealing: an Experimental Evaluation; part II, Graph Coloring and Number Partitioning, Operations Research 39, 378–406 (1991).
13. L. Davis.: Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York (1998).
14. P. Galinier, J.K. Hao.: Hybrid Evolutionary Algorithms for Graph Coloring, Journal of Combinatorial Optimization 3, 379–397 (1999).
15. A. Mehrotra, M.A. Trick.: A Column Generation Approach for Graph Coloring, INFORMS Journal on Computing 8, 344–354 (1996).
16. E. Malaguti, M. Monaci, P. Toth.: A Metaheuristic Approach for the Vertex Coloring Problem, INFORMS Journal on Computing 20(2), 302–316 (2008).

Index of Authors

Index of Keywords