Iterated Local Search for the examination timetabling problem with constructive-based initial solution[*]

Synim Selimi[1], Labeat Arbneshi[1], Kadri Sylejmani[1]⊠, and Nysret Musliu[2]

[1] Faculty of Electrical and Computer Engineering, University of Prishtina, Kosova
[2] Databases and Artificial Intelligence Group, TU Wien, Austria
synim.selimi@student.uni-pr.edu,
[labeat.arbneshi,kadri.sylejmani]@uni-pr.edu,
musliu@dbai.tuwien.ac.at

Abstract. In this extended abstract we present an approach and solution for the examination timetabling problem based on the Iterated Local Search metaheuristic. Initially we introduce a flat data remodeling of the given problem instances. The proposed constructive approach then uses precalculated heuristic information to construct a feasible initial solution from the refined instance data. The neighborhood structure consists of two operators, one to reallocate only rooms and the other to change room-period tuples for course examination events. The algorithm also applies a perturbation mechanism, which is tuned to guide the search for optimal solutions out of the local optima. The presented approach has been preliminarily tested against some smaller test sets existing in the literature, where it has shown that it is able to produce optimal results for some of the instances.

Keywords: Examination Timetabling, Constructive Heuristic, Iterated Local Search.

1 Introduction and problem formulation

The examination timetabling process at universities is an overly complex combinatorial problem that has been extensively researched, resulting in a substantial body of literature. Qu et al. [1] present a survey of the algorithmic methodologies and the respective variants of the problem formulation. Many of the algorithms in literature solve the variant of the examination timetabling problem that was introduced in the International Timetabling Competition in 2007 (ITC2007), which is described thoroughly by McCollum et al. [2]. ITC2007 is mainly inspired by the model of British universities for examination timetabling, and it introduces 12 test instances (tagged as ITC2007 dataset) which are quite challenging. To date, none of the instances have been solved to optimality. In this paper, we tackle a recent reformulation of the problem by

[*] The work on this paper was supported by the HERAS+ program within the project entitled "Automated Examination Timetabling in the Faculty of Electrical and Computer Engineering - University of Prishtina".

Battistutta et al. [3], which is based on the practicalities of the examination timetabling process at Italian universities.

The original formulation of a real-world examination timetabling problem carried out by Battistutta et al. [3] is the variant that we take on in this paper. In the following section we outline the essential problem entities and constraints, but we refer the reader to Battistutta et al. [3] for more in-depth details on the problem formulation.

The problem defines the following entities:
Courses, Exams and Events – Each course requires scheduling one or more exams within an examination term. In addition, each exam can be organized multiple times and can consist of one or two parts, depending on whether it has a written part, an oral part or both. Two-part exams must be scheduled sequentially.
Rooms - The respective exam events might require rooms of specified size. Rooms are classified into three size categories: small, medium, and large. Room combinations are annotated as room sets and are predefined as fixed sets in the problem instances.
Days, Timeslots, Periods - The number of available periods is the total number of time slots per day multiplied by the number of days in a term. For example, in a two-week time span (i.e., examination term) there are 20 available periods given two time slots per day (e.g., 09:00 and 14:00) during working days.
Curricula – Contains the courses with the same students (e.g., courses of a given study program). There are two categories of curricula, namely Primary and Secondary. The first holds the set of courses in the current semester, while the latter has the set of courses in previous semesters. The level of conflicts between assignments of events belonging to Primary and Secondary set is outlined below.

The hard constraints are:
Room request - For each exam event (written or oral) there is a specific number and type of the room/s that must be assigned. Also, if any oral exam requires a room, then a single room of any type must be assigned.
Room occupation - During each period, a given room cannot be assigned to more than one exam event.
Hard conflicts - Two events (written or oral) cannot be assigned to the same period, if they: (1) are primary courses in the same curricula (i.e., same semester), (2) have the same teacher, or (3) have an explicitly written down constraint forbidding them to be assigned in the same period.
Precedence – Requires that two events of the same course are strictly scheduled one after the other when: (1) events are part of two separate exams of the same course and (2) events are part of the same two-part exam (written and oral).
Unavailability - An exam event or room might be explicitly declared not to be scheduled in some specific periods.

The soft constraints are:
Soft conflicts - Two events that are assigned to the same period are in soft conflict if they: (1) belong to the same set of courses either in a primary-secondary or secondary-secondary relationship and (2) have an explicitly declared undesirable constraint not to be scheduled at the same time.

Algorithm 1 Construction of the initial solution

```
procedure Solve(instance)
    solution <- {}
    cInstance <- InstanceLevelHeuristic(instance) // Algorithm 2
    for course in cInstance.courses
        cCourse <- CourseLevelHeuristic(course) // Algorithm 3
        solution[course] <- AssignRoomPeriod(cCourse)
        cInstance <- PropagateConstraints(cInstance)
    return solution
```

Preferences - There is a set of constraints explicitly declaring that specific combinations of events and periods or rooms and periods are undesired or preferred.
Distances - For some pairs of events there are constraints requiring certain minimum and/or maximum distances between their respective assigned periods, such as: (1) distinct parts of the same exam have a minimum and a maximum distance, declared explicitly for each course, (2) different exams of the same course must be separated for a minimum number of periods, (3) if two courses are part of the same curriculum, there should be a minimum separation between the first event of the respective exams, and (4) there could be explicit constraints requiring certain distances between specific events.

Note that the weights of the violation for distinct types of soft conflicts are left to be specified by the end user.

2 Solution approach

2.1 Preprocessing, search space and cost function

In the preprocessing phase we do a complete flat data remodeling of the given instances to ensure that all relevant data is reduced and contained within its corresponding course instance. This is accomplished by moving and restructuring information about courses and constraints from scattered entities into a self-contained course event entity with information about allowed rooms/periods, number of events, event dependencies, etc. This step makes the use of constructive heuristics and constraint propagation techniques much easier as described in Section 2.2 (Initial solution).

A state in the search space is represented by two vectors, where, for each exam event, the first one stores the rooms and the second one stores the assigned period. An exam can only be placed in a certain room and period if it satisfies all hard constraints that are related to periods, rooms, curricula, and teachers.

The cost (fitness) function is the weighted sum of all penalties that occur when a room-period assignment does not fulfill the associated soft constraints.

Algorithm 2 Applying heuristic information at the instance level

```
procedure InstanceLevelHeuristic(instance)
    inst <- Copy(instance)
    if instance has more courses with multiple exams
        inst.courses <- GroupAndAssignByExamNumber(inst.courses)
    else if instance has more courses with multiple parts
        inst.courses <- GroupAndAssignByParts(inst.courses)
    else if instance has more flat courses
        inst.courses <- ReorderComplexCoursesFirst(inst.courses)
    return inst
```

Algorithm 3 Applying heuristic information at the course level

```
procedure CourseLevelHeuristic(course)
    c= Copy(course)
    if course required simple rooms
      c.possibleRooms <- Shuffle(c.possibleRooms)
    if course has multiple exams
      c.possiblePeriods <-
            DistributeExamPeriods(c.possiblePeriods)
    else if course has multiple parts
      c.possiblePeriods <-
            DistributePartPeriods(c.possiblePeriods)
    return c
```

2.2 Initial solution

Following the preprocessing phase, our approach ensures that we can consistently generate an initial solution for all instances in reasonable time by using multiple instance-level and course-level constructive heuristics. These heuristics guide the assignment of courses, periods and rooms based on situational information, for e.g., the number of exams per course, the type of exams (written, oral), the course event complexity (composite rooms, combination of multiple two-part exams), etc. While the approach proposed by Battistutta et al. [3] looks through a search space with infeasible states, our approach always starts with and explores within the feasible region of the search space.
Algorithm 1 displays the general approach to generating an initial solution. Allowed room-period tuples per course are defined during preprocessing and then updated from *PropagateConstraints* after each allocation. *AssignRoomPeriod* selects the first allowed room-period tuple for a given course event. Algorithm 2 illustrates the use of constructive heuristics to guide the generation of the initial solution.

The heuristic mechanisms described below guide the order of course assignments and the order of period-room allocations to always satisfy hard constraints promptly with little to no backtracking.

DistributeExamPeriods(possiblePeriods) – distributes and reorders assignable periods for course events with a calibrated distance between event exams.

Algorithm 4 Iterated Local Search

```
procedure SolveWithILS (instance, maxIterations,
hillClimbingIterations, operatorRate, changeRate, changeHomeRate,
perturbRate)
    current <- Solve(instance) // Algorithm 1
    best <- home <- current
    for n from 1 to maxIterations
        p <- random (0,1)
        for h from 1 to hillClimbingIterations
            if p < operatorRate
                neighbor <- ChangeRoom(current)
            else
                numCourses <- changeRate * instance.totalCourses
                for c from 1 to numCourses
                    neighbor <- ChangeRoomPeriod(current)
            if neighbor better than current
                current <- neighbor
        if current better than best
            best <- current
        h <- random (0,1)
        if h < changeHomeRate or current better than home
            home <- current
        numCourses <- perturbRate * instance.totalCourses
        for i from 1 to numCourses
            current <- ChangeRoomPeriod(home)
    return best
```

DistributePartPeriods(possiblePeriods) – distributes and reorders assignable periods for two-part course events with a calibrated distance between event parts.

GroupAndAssignByExamNumber(courses) – groups and reorders by exam number, whereafter preceding exams will be assigned before subsequent exams by default.

GroupAndAssignByParts(courses) – groups and reorders courses by their respective parts (Written or Oral), whereafter Written exams will be always assigned before Oral exams.

ReorderComplexCoursesFirst(courses) – groups and reorders courses by their complexity first, where complexity is defined as the number of events attached to a course due having multiple exams, multiple parts, or both.

2.3 Neighborhood structure

Battistutta et al. [3] define the *MoveEvent* operator, which changes the period-room tuple of an event (i.e., a part of an exam). In our case, the neighborhood structure is a variation of *MoveEvent*, which consists of two operators, namely *ChangeRoom* and *ChangeRoomPeriod*, as described below.

ChangeRoom – randomly selects a given exam (including all its events) and moves all corresponding events from their existing room to a new room, provided that all hard constraints are satisfied.

ChangeRoomPeriod – extends the *ChangeRoom* operator with the ability to move exam events from a given room and period to another randomly selected available room and period.

2.4 Iterated Local Search

In Algorithm 4, we present the pseudocode of the proposed approach that is based on the Iterated Local Search (ILS) metaheuristic. The initial solution is constructed by using constructive heuristics and constraint propagation as discussed in Section 2.2. This ILS-based procedure runs a form of hill climbing iteratively and explores the search space using the neighborhood structure described above. Within the iterative phase of ILS, we distinguish between three main steps, as follows.

The exploitation phase runs a hill climbing algorithm, which exploits the search space using our neighborhood structure guided by the parameter *operatorRate*. This parameter defines the selection of one of the existing neighborhood operators.

The selection phase selects the new home depending on the parameter *changeHomeRate*, which determines whether the algorithm has more of an explorative nature (where the current solution is accepted as the new home base, regardless of quality) or exploitative nature (where the current solution becomes the new home base only if its quality is better than the quality of current home base).

The perturbation mechanism, which, based on the parameter `perturbRate`, applies the operator `ChangeRoomPeriod` multiple times to perturb the home solution and consequently avoid the local optima.

3 Computation results

3.1 Data set and parameter tuning

The formulation of the examination timetabling problem from Battistutta et al [3] that we have tackled in this paper has a dataset of 40 instances that is divided into 7 groups. Each group presents timetabling requirements from different study department at selected Italian universities. For our preliminary computational study, we have used the revised version of the dataset [3] and have only selected two of the smaller groups in the dataset (namely group D2 and group D3) that have at most 89 exams, 188 periods and 15 rooms. The experiments have been conducted using an Apple M1 machine with 16 GB of memory.

3.2 **Comparison results**

During experimentation sampled from 100 attempts we managed to generate initial feasible solutions for all instances in 0.3537s on average, with 0.01s being the shortest and 2.58s the longest duration.

The preliminary computational experiments were conducted by running the algorithm 10 times for 30 seconds for each of the instances belonging to the test subsets, namely D2 and D3. All results and generated solutions have been validated with the solution validator (named *examtt toolbox*) provided by Battistutta et al [3]. Table 1 displays the preliminary results, which show promising indications that the algorithm can generate comparable and satisfactory solutions. For the D2 subset, our approach falls behind the approach of Battistutta et. al [3], while for the D3 subset, our approach finds the optimal solutions for 6 instances and is outperformed on the remaining 3 instances.

We cannot draw a direct comparison of the computation time against the approach of Battistutta et. al [3], due to different computing environments, however as shown in Table 1, the approach of Battistutta et. al [3] on average solves the D2 subsets for about 93 seconds and the D3 subsets on average for about 27 seconds.

4 Conclusion and future work

This paper presents an ongoing work about the design and implementation of an Iterated Local Search metaheuristic that can find optimal solutions for a subset of instances existing in the literature. We believe this approach is worth exploring further because of promising preliminary results and the ability to produce feasible solutions quickly, albeit not notably better than the existing ones for all instances in the test set.

Table 1. Comparison of the proposed approach against state-of-the-art methods.

Instance name	Simulated Annealing (SA)			ILS algorithm		
	Avg	Best	Time	Avg	Best	ILS vs. SA (%)
D2-1-18	427.77	426	94.7	630	570	25.26
D2-2-18	22.00	22	88.7	151	140	84.29
D2-3-18	22.00	22	95.0	169	97	77.32
D3-1-16	0.00	0	61.9	201	169	100
D3-1-17	0.00	0	83.5	472.5	401	100
D3-1-18	0.00	0	83.9	460	375	100
D3-2-16	0.00	0	0.8	1.7	0	0
D3-2-17	0.00	0	3.1	2.9	0	0
D3-2-18	0.00	0	3.5	0	0	0
D3-3-16	0.00	0	1.0	0	0	0
D3-3-17	0.00	0	2.3	3.1	0	0
D3-3-18	0.00	0	2.2	0	0	0

As part of future work, along with the development of new neighborhood operators, we aim to upgrade the existing operators with heuristic features that choose exam reallocations based on partial solution penalties and restrict the selection of periods and rooms to only those promising improvements. We will also explore applying our flat entity approach to address the practical challenges with *MiniZinc* modeling mentioned by Battistutta et. al [3]. Lastly, in addition to improving experimental benchmarking, we intend to modify our heuristics approach to also explore the infeasible part of the search space.

References

1. Qu, Rong, Edmund K. Burke, Barry McCollum, Liam TG Merlot, and Sau Y. Lee. "A survey of search methodologies and automated system development for examination timetabling." Journal of scheduling 12, no. 1 (2009): 55-89.
2. McCollum, Barry, Andrea Schaerf, Ben Paechter, Paul McMullan, Rhyd Lewis, Andrew J. Parkes, Luca Di Gaspero, Rong Qu, and Edmund K. Burke. "Setting the research agenda in automated timetabling: The second international timetabling competition." INFORMS Journal on Computing 22, no. 1 (2010): 120-130.
3. Battistutta, Michele, Sara Ceschia, Fabio De Cesco, Luca Di Gaspero, Andrea Schaerf, and Elena Topan. "Local Search and Constraint Programming for a Real-World Examination Timetabling Problem." In International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research, Springer, Cham, (2020): 69-81.

Multi-neighbourhood Simulated Annealing for the Capacitated University Examination Timetabling Problem (ITC-2007)

David Van Bulck[1][0000−0002−1222−4541], Dries Goossens[1,2][0000−0003−0224−3412] and Andrea Schaerf[3][0000−0001−6965−0536]

[1] Faculty of Economics and Business Administration, Ghent University, Tweekerkenstraat 2, 9000 Ghent, Belgium
{david.vanbulck,dries.goossens}@ugent.be
[2] FlandersMake@UGent – core lab CVAMO
[3] Polytechnic Department of Engineering and Architecture, University of Udine, via delle Scienze 206, 33100 Udine, Italy, andrea.schaerf@uniud.it

Keywords: Examination Timetabling · Simulated Annealing · ITC-2007-ETT

1 Introduction and problem description

Many variations of the Examination Timetabling Problem (ETT) exist, each with their own resources, constraints, and objectives. In this extended abstract, we consider the ETT in the classical version as proposed for the Second International Timetabling Competition (ITC-2007). This variant consists of allocating each exam to a single period (time slot) and room, allowing exams to share a room. Some of the hard constraints are for instance that no two conflicting exams (i.e. exams with at least one student in common) are assigned to the same period, or that the capacity of the rooms is at least the number of students allocated to the exams assigned to that room in a given period. The main soft constraints are to spread the students' exams as much as possible over time, while avoiding that exams with a different length are scheduled together in the same room. The assignment of exams to periods thus has some similarities with graph colouring, whereas the assignment of exams to rooms resembles a packing problem, given that a room can be shared among exams. For the sake of brevity, we do not report the full problem description here, but instead we refer to [10].

2 Solution method

Following the spirit of the work by Bellio et al. [2] for the uncapacitated ETT, we developed a multi-neighbourhood Simulated Annealing (SA) algorithm. The choice for SA is motivated by the fact that SA has already proven to be very effective for this [1, 9] and a number of other timetabling problems (see, e.g., [3, 6]). Our search space is composed by an array of pairs that assigns to each exam a period and a room, and also includes solutions that may violate hard

Table 1. Considered neighbourhoods

`Move(e,p,r)`	Move exam `e` to period `p` and room `r`.
`Swap(e1,e2)`	Swap the period and room assigned to exams `e1` and `e2`.
`Kick(e1,e2,p,r)`	Move exam `e1` to the period and room assigned to `e2`. Move exam `e2` to period `p` and room `r`.

constraints such as conflicts or room capacities. These violations are included in the cost function, along with the soft constraints, but with a suitably larger weight.

The portfolio of neighbourhoods that we already implemented is given in Table 1. These neighbourhoods were originally proposed for the uncapacitated version to the ITC-2007 problem by Bellio et al. [2], and were adapted to deal with the assignment of rooms which is not considered in the uncapacitated problem.

3 Preliminary experimental results

Preliminary results can be found in Table 2, which compares the best found solutions over 30 runs with some of the best results found by algorithms previously published in the literature. Runtimes are set approximately according to ITC-2007 specifications. Although the development and the experimentation with our algorithm is still ongoing, at present we reach results quite comparable with the state-of-the-art approaches albeit still inferior to the best known. Final results will be discussed at the conference.

Table 2. Preliminary results. Best available solutions are from https://opthub.uniud.it

No.	Best available	[4]	[5]	[8]	[1]	Us
1	3488	3691	3787	4128	3752	3579
2	380	385	402	380	385	385
3	7041	7359	7378	7769	8175	7975
4	11806	11329	13278	13103	13681	14106
5	2327	2482	2491	2513	2544	2539
6	25145	25265	25461	25330	25560	25265
7	3424	3608	3589	3537	3522	3513
8	7356	6818	6701	7087	7505	7405
9	904	902	997	913	941	935
10	12878	12900	13013	13053	13582	13288
11	22465	22875	22959	24369	26114	24921
12	5095	5107	5234	5095	5153	5423

4 Future work

The present work is an initial step toward a more comprehensive final goal. First of all, we will develop other, more elaborate, neighbourhood relations, specifically designed for this problem. Secondly, we plan to investigate SA variants (see [7]) , alternative metaheuristics, and hybrid techniques. Finally, we aim at performing an instance space analysis and a corresponding algorithm selection procedure for this problem.

Regarding the first point, we are currently developing a *Kempe chain* neighbourhood, such that possible conflicts generated by a movement are repaired. In detail, in order to repair any new conflict introduced by moving exam `e` to period `p` and room `r`, the move `KempeChain(e,p,r)`, reassigns all exams in `p` in conflict with `e` to the period originally assigned to `e` and to the cheapest room (greedily determined), and so on until there are no newly introduced conflicts (see also [2]).

References

1. Battistutta, M., Schaerf, A., Urli, T.: Feature-based tuning of single-stage simulated annealing for examination timetabling. Annals of Operations Research **252**(2), 239–254 (2017)
2. Bellio, R., Ceschia, S., Di Gaspero, L., Schaerf, A.: Two-stage multi-neighborhood simulated annealing for uncapacitated examination timetabling. Computers and Operations Research **132**, 105300 (2021)
3. Bellio, R., Ceschia, S., Di Gaspero, L., Schaerf, A., Urli, T.: Feature-based tuning of simulated annealing applied to the curriculum-based course timetabling problem. Computers & Operations Research **65**, 83–92 (2016)
4. Burke, E.K., Bykov, Y.: An adaptive flex-deluge approach to university exam timetabling. INFORMS Journal on Computing **28**(4), 781–794 (2016)
5. Burke, E.K., Bykov, Y.: The late acceptance hill-climbing heuristic. European Journal of Operational Research **258**, 70–78 (2017)
6. Ceschia, S., Di Gaspero, L., Schaerf, A.: Design, engineering, and experimental analysis of a simulated annealing approach to the post-enrolment course timetabling problem. Computers & Operations Research **39**, 1615–1624 (2012)
7. Franzin, A., Stützle, T.: Revisiting simulated annealing: A component-based analysis. Computers & Operations Research **104**, 191–206 (2019)
8. Gogos, C., Goulas, G., Alefragis, P., Kolonias, V., Housos, E.: Distributed scatter search for the examination timetabling problem. In: McCollum, B., Burke, E.K., White, G. (eds.) 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT-2010). pp. 211–223. PATAT, Belfast (2010)
9. Leite, N., Melício, F., Rosa, A.C.: A fast simulated annealing algorithm for the examination timetabling problem. Expert Systems with Applications **122**, 137–151 (2019)
10. McCollum, B., McMullan, P., Burke, E.K., Parkes, A.J., Qu, R.: The second international timetabling competition: Examination timetabling track. Tech. Rep. QUB/IEEE/Tech/ITC2007/Exam/v4.0/17, Queen's University, Belfast (UK) (September 2007)

Improving the Dynamic Programming Algorithm for Nurse Rostering

Jeffrey H. Kingston

School of Information Technologies, The University of Sydney, Australia
jeff@it.usyd.edu.au
http://jeffreykingston.id.au

Abstract. A dynamic programming algorithm for optimally timetabling one nurse appears in papers that perform nurse rostering using column generation. This paper generalizes this algorithm, allowing it to assign an arbitrary (but small) subset of the nurses on an arbitrary subset of the days of the timetable, and handle every nurse rostering constraint used in practice. The paper also presents work in progress on speeding up the algorithm, mainly by improving dominance testing, its key step. The aim is to fit the algorithm for use as the reassignment operator of a VLSN search for nurse rostering.

Keywords: Nurse Rostering · Dynamic Programming · VLSN Search

1 Introduction

Nurse rostering is the problem of assigning shifts to the nurses of a hospital ward, so as to satisfy a given set of hard constraints and minimize the cost of violating a given set of soft constraints.

Papers that use column generation to solve nurse rostering problems have long used a polynomial-time dynamic programming algorithm for finding an optimal timetable for a single nurse. This becomes one column of a set covering integer program which is solved to produce a timetable for the whole ward.

This paper generalizes this dynamic programming algorithm, allowing it to optimally assign an arbitrary (but small) subset of the nurses on an arbitrary subset of the days of the timetable, and handle every nurse rostering constraint used in practice. The paper also describes work in progress on speeding up the algorithm, mainly by improving dominance testing, its key step.

The improved algorithm aspires to be used as the reassignment operator of a very large-scale neighbourhood (VLSN) search for nurse rostering. In VLSN search, a given initial solution is repeatedly improved by unassigning a large part of it and reassigning that part, hopefully in a better way. The dynamic programming algorithm can carry out this reassignment optimally, making it a direct competitor for integer programming, the usual choice here. At present it is not competitive, but the work is ongoing.

Implementing the algorithm presented here turned out to be a major task, running to over 13,000 lines of C code. A detailed 100-page description appears online [11]. This paper focuses on the key points, omitting many details.

Section 2 defines the problem, and presents the relevant literature. Sections 3 and 4 present the original algorithm using our terminology. Sections 5 and 6 present our generalizations and optimizations. Section 7 presents experiments.

2 The nurse rostering and single nurse rostering problems

Nurse rostering is the problem of assigning shifts to the nurses of a hospital ward. Hospitals run 24 hours a day, so nurse rostering problems usually have at least three *shift types*: morning, afternoon, and night. Each *shift* (one shift type on one day) demands a certain number of nurses, often with specified skills. There may be some flexibility in how many nurses to assign to each shift, and the number typically changes from day to day. Nurses are not interchangeable in general, because they have individual contracts (full-time, part-time, and so on), skills (senior nurse, assistant, and so on), and requests for days off.

Perhaps the most characteristic feature of the problem is the large array of constraints that each nurse's timetable must or should satisfy. In practice there are always hard constraints requiring each nurse to work at most one shift per day, and constraints (hard or soft) on each nurse's total workload over the weeks that the problem spans. There are also rules such as 'no day shift immediately following a night shift', 'at most 5 consecutive busy days', 'at most 2 busy weekends', and so on. These vary from one formulation of the problem to another, and from one nurse contract to another.

Nurse rostering is one of the most-studied problems in the discipline of automated timetabling. There are survey papers [1, 4] but they are rather old now. Much of the recent work is inspired by the Second International Nurse Rostering Competition (INRC-II) [2, 3].

The general nurse rostering literature is large, but the problem of optimally assigning a single nurse seems to have been studied only by researchers who use integer programming with column generation. Each column represents a complete timetable for one nurse; to generate a column is to solve a single nurse problem, for which these researchers mostly use dynamic programming. The integer program selects a subset of the columns that solves the full problem. For a survey of this work, going back to 1998, we refer the reader to [12].

These column generation papers often use the resource constrained shortest path problem [6] as a stepping stone. In our experience this does not yield any useful insights into nurse rostering, and so in this paper we move directly from nurse rostering to the dynamic programming algorithm.

The author is aware of one paper which uses dynamic programming to solve a full resource assignment problem, for security guards [5]. As the number of resources increases, the search space expands rapidly, making solving full problems with dynamic programming only viable for small instances.

3 Overview of the dynamic programming algorithm

This section presents our starting point, the existing dynamic programming algorithm for rostering a single nurse r.

On each day, nurse r can be assigned a shift of a given type (morning, afternoon, etc.), or nothing. Let these choices be $\{s_0, s_1, ..., s_a\}$, where s_0 is a special shift type that denotes doing nothing (a day off). We are assuming here, as is usual, that a nurse may take at most one shift per day.

A solution may be an arbitrary set of assignments, but in this paper we consider only solutions of a particular kind. Let the sequence of days for which assignments are required be $\langle d_1, ..., d_n\rangle$. A *solution* is a sequence of k shift types representing assignments for the first k days, where $0 \leq k \leq n$. For example,

$$\begin{array}{|c|c|c|c|c|}\hline s_1 & s_1 & s_1 & s_0 & s_0 \\ \hline\end{array}$$

represents a timetable in which r is assigned shifts of type s_1 on the first three days, followed by two days off; and

$$\begin{array}{|c|c|c|c|c|}\hline s_2 & s_2 & s_2 & s_0 & s_0 \\ \hline\end{array}$$

is similar, with s_2 replacing s_1. A *complete solution* is a solution for all n days.

Let us first consider a tree search algorithm. Each node of its search tree is one solution. For its root, the tree has the unique solution of length 0. Then, for the first day, we try assigning a shift of each type for that day, omitting types for which no shift is available, or for which r is not qualified. This produces one child for each available shift type, being a solution of length 1. For each of these we try assigning a shift of each type for the second day, and so on until all solutions of length n have been tried. The cost of each solution is calculated and the overall result is the best solution of length n.

When all shift types $\{s_0, s_1, ..., s_a\}$ are available on all days $\langle d_1, ..., d_n\rangle$, the tree search tries $(a + 1)^n$ complete solutions, an impossibly large number when a timetable is needed over several weeks, as is common in practice. The idea of dynamic programming is to show that some solutions *dominate* others, that is, always produce better complete solutions in the end. Dominated solutions can be discarded, and this can produce major savings. In fact, it leads to running times which are polynomial in n [11], as is well known.

Define P_k to be the set of undiscarded solutions for $\langle d_1, ..., d_k\rangle$. For example, the two solutions above might lie in P_5. For the tree search, P_k contains up to $(a + 1)^k$ solutions.

The dynamic programming optimization explores the tree in breadth-first order: it first builds P_0 (just the length 0 solution), then P_1, then P_2, and so on. To proceed from P_k to P_{k+1}, for each solution x in P_k, it constructs all solutions y consisting of x plus one assignment of an available shift type to r on d_{k+1}. Before inserting y into P_{k+1}, it checks whether P_{k+1} contains a solution that dominates y. If so it discards y instead of inserting it. If not, it first removes from P_{k+1} and discards all solutions that are dominated by y, and then inserts y. In this way, P_{k+1} contains only undominated solutions.

For reasons that will become clear later, on the last day, d_n, there will be just one solution in P_n, and that is the desired optimal solution. That completes the algorithm. But we still need a definition of dominance that allows the algorithm to safely discard a large number of solutions.

4 Signatures and dominance

This section defines a dominance relation between solutions which allows many solutions to be safely discarded. As it turns out, this relation is determined by the constraints of the problem instance.

The *signature* of constraint c in solution x, written $s(c, x)$, is a concise but complete representation of x as it affects c, excluding parts of x for which c has already yielded a cost.

Let us consider some examples. Suppose c is a constraint on the total number of shifts worked by nurse r. Then $s(c, x)$ would be the number of shifts worked by r within x. It does not matter to c which shifts they are.

Suppose c is a constraint on the total number of busy weekends (weekends where r works at least one shift). Then for $s(c, x)$ we may choose the number of busy weekends during x, but there is a catch when x's last day is a Saturday: the signature must record whether r works a shift on that day, because that determines whether working a shift on the immediately following Sunday adds to r's number of busy weekends or not. So $s(c, x)$ is an integer plus a Boolean when x's last day is a Saturday, and an integer on other days.

Suppose c is a constraint on the number of consecutive night shifts worked by r. Then $s(c, x)$ is the number of consecutive night shifts ending on the last day of x, or 0 if r is not assigned a night shift on the last day of x. Sequences of consecutive night shifts that terminated earlier have already yielded a cost and do not influence $s(c, x)$.

The reader familiar with history in nurse rostering will have noticed a strong connection between signatures and history values [9]. A history value records what r did that affects c before the first day; a signature records what r did that affects c before and during the last day of x. The idea is the same, although [9] assumes that cases like the Saturday treated above do not occur at the start of the timetable. Here, we cannot assume such cases away.

Suppose solution x is for days $d_1, ..., d_k$. Suppose a constraint c depends only on the assignments on those days. Then c yields its final cost when d_k is assigned, so $s(c, x)$ is empty. Or suppose c depends only on the assignments on days $d_{k+1}, ..., d_n$. Then there is nothing to remember about c on $d_1, ..., d_k$, so again $s(c, x)$ is empty. The only constraints for which $s(c, x)$ is non-empty are those which are affected both by the assignments on at least one of the days $d_1, ..., d_k$, and also by the assignments on at least one of the days $d_{k+1}, ..., d_n$.

The *signature* of a solution x, written $s(x)$, is the concatenation, over all constraints c, of $s(c, x)$. Concretely, it is an array of integers (Booleans are encoded as 0 and 1). The cost of solution x, written $c(x)$, is the sum of all costs already yielded by constraints up to and including x's last day. Each solution x needs just four fields: a pointer to its parent solution in the search tree; the assignment that x adds to that parent; $s(x)$; and $c(x)$.

Given our definition of $s(c,x)$ as a complete representation of x as it affects c, it is not surprising that as solutions are built up, we can keep $s(c,x)$ up to date, and then on c's last day, convert it into a final cost for c. For example, if c is the number of shifts worked we can take this number from x's parent and add 1 or 0 to it, depending on whether x's assignment is for a shift or a free day. What is perhaps more surprising is that signatures support dominance testing as well as cost calculations, as we will see shortly.

We say that solution x *dominates* solution y if the best complete solution that can be derived from x by further assignments has cost no larger than the best complete solution that can be derived from y by further assignments. Clearly, in that case y can be discarded without risk of losing optimality.

In practice we use a more restricted definition of dominance: we require x and y to have the same last day, we require $c(x) \leq c(y)$, and for each constraint c, we require x to dominate y at c. By this last condition we mean that for each combination of assignments t for later days, when we add those further assignments to x and y, resulting in complete solutions x_t and y_t, the cost of c in x_t must not exceed the cost of c in y_t. It should be clear that every case of this more restricted definition of dominance is a case of the general definition.

There is no need to identify every case of dominance. All that matters is that for each case that we do identify, it is indeed safe to discard y. Of course, the more cases we identify, the faster the algorithm runs.

It is clear now why there is at most one solution on the last day, d_n. The signature array is empty because all constraints have reported their final cost by then, so x dominates y when $c(x) \leq c(y)$, so only one solution will be kept.

To understand dominance, then, it remains to understand how solution x can dominate solution y at constraint c. We answer this question for some kinds of constraints now; in Section 5 we'll see that we have in fact covered all cases.

Many constraints calculate a value that we call the *determinant*: the number of shifts, the number of consecutive night shifts, or whatever. The cost of such a constraint c is a monotone non-decreasing function of the amount by which the determinant exceeds a maximum limit or falls short of a minimum limit. The limits are fixed attributes of c.

The determinant is used as c's signature value on each day a signature value is needed. If c has a maximum limit only, then x dominates y at c when $s(c,x) \leq s(c,y)$. This is because when we add the same further assignments t to x and y, producing complete solutions x_t and y_t, the determinant increases by the same amount in both. So $s(c,x) \leq s(c,y)$ implies $s(c,x_t) \leq s(c,y_t)$, and the cost has this same relation too, because the cost when there is a maximum limit only is a monotone non-decreasing function of the determinant.

If c has a minimum limit only, x dominates y at c when $s(c,x) \geq s(c,y)$. This is because in this case the cost is a monotone non-increasing function of the determinant. If c has both a maximum limit and a minimum limit, then both analyses apply and x dominates y at c when $s(c,x) = s(c,y)$.

As expressed, this argument applies only to constraints on the total number of something, not to constraints on the number of consecutive things. But the argument is easily adapted to constraints on consecutive things, by changing x_t and y_t to the solutions on the first day after the sequence ends. On that day, the cost of the sequence is finalized and added to $c(x_t)$ and $c(y_t)$.

5 Generalizing to arbitrary nurses, days, and constraints

In this section we generalize the dynamic programming algorithm so that, starting from an initial timetable that we want to improve, it can unassign and reassign any (small) number of *selected nurses* on any number of *selected days*. We don't require the selected days to be consecutive, because we want to try oddities like reassigning the eight weekend days of a four-week timetable. We also generalize to arbitrary constraints.

To generalize from a single nurse to a set of selected nurses, the algorithm begins by unassigning the selected nurses on the selected days, then proceeds much as before. Each solution extends its parent solution by adding one assignment (possibly of a free day) on its last day to each selected nurse. If there are m selected nurses and $a + 1$ shift types, there are up to $(a + 1)^m$ ways to do this. The signature of each solution is the concatenation of the signatures of the selected nurses, in some fixed order, plus another signature for cover constraints, which we will come to shortly.

To generalize from all days to selected days, we redefine $\langle d_1, ..., d_n\rangle$ to be the sequence of selected days (in chronological order), not all days. A set of undominated solutions P_k is built for each selected day d_k, not for each day.

The other issue in generalizing this algorithm is to make sure that it can handle every kind of constraint. We do this by supporting the constraints of the XESTT nurse rostering model [7, 8], which have been shown in [8] to encompass everything that occurs in practice.

Nurse rostering constraints are either *cover constraints* (XESTT calls them *event resource constraints*), which require each shift to have a suitable number of suitably qualified nurses, or *resource constraints*, which require individual nurses' timetables to follow the many rules of nurse rostering: at most one shift per day, limits on total shifts, limits on consecutive shifts, and so on.

Although we illustrated the signature idea using resource constraints only, it applies equally well to cover constraints. Cover constraints always seem to constrain the shifts of a single day, and so do not need signature values. But if there was a multi-day cover constraint, for example a constraint requiring the staffing of at least four of the week's night shifts to include a senior nurse, then that could be handled, using a signature value saying how many of the current week's night shifts have been assigned a senior nurse.

XESTT has four event resource constraints, only three of which are used in nurse rostering, and seven resource constraints, only five of which are used. These small numbers are owing to the generality of the constraints: they may reference arbitrary sets of times and nurses. For example, no constraints are inherently concerned with weekends; instead, an instance would define the set of weekend times and reference that in its constraints.

We do not have space to define all these constraints in detail and explain how they are handled. Instead, we'll divide them into three groups and explain how each group is handled.

The first group contains constraints which find the number of occurrences of something (e.g. busy weekends) and compare that number, which we have called a determinant, with lower and upper limits. As explained in Section 4, these constraints are handled by storing the determinant as the signature value. Assignments on unselected days are constant throughout the solve; their effect here is to add a constant to the signature value. For example, if c constrains the number of shifts worked by nurse r, then at the start of the solve, the signature $s(c, x)$ is the number of shifts worked on unselected days, and it increases from there as the solve proceeds.

The second group contains constraints which do not have determinants and limits. These we transform so that they do. For example, consider a constraint requiring nurse r to be free on Tuesday. Let the determinant be the number of shifts that r works on Tuesday, and apply maximum limit 0.

The third and final group contains constraints which find the total number of *consecutive* occurrences of something and compare that number with lower and upper limits. The signature is the number of consecutive occurrences in the current run, as explained earlier. Handling unselected days is quite complicated, since between any two selected days there may be a sequence of unselected days. Their effect is pre-calculated so that it can be included quickly. For example, suppose we are constraining nurse r's number of consecutive night shifts, and the selected days are the weekend days. Then for each run of consecutive week days we pre-calculate the number of r's consecutive night shifts a at the start of the week, and the number b at the end of the week. Then as the solve proceeds, if the preceding Sunday has a night shift we count that not as 1 night shift but as $1 + a$ night shifts, and so on. For full details we refer the reader to [11].

The definition of dominance at c given above ('$\leq$' when c has a maximum limit only, '$\geq$' when c has a minimum limit only, and '$=$' when c has both) we call *basic dominance*. Our algorithm actually uses *strong dominance*, which is basic dominance with three changes. First, XESTT constraints have an *allow zero flag*, which when set specifies that a determinant with value 0 produces cost 0 regardless of the minimum and maximum limits. Its main use in nurse rostering is in 'complete weekends' constraints, which require a nurse to work both days of a weekend or neither. Strong dominance takes this into account. Second, strong dominance understands that when c has a maximum limit only, $s(c, x)$ may be so small that no further assignments can increase it to the maximum limit, and so x dominates every y at c because the cost of c in every x_t must be 0. The third change is similar, understanding that when c has a minimum limit only, if $s(c, x)$ has already reached it, then x dominates every y at c.

The algorithm discards solutions x such that $c(x)$ equals or exceeds the cost of the original solution (the solution from before the initial unassignments). If all solutions are discarded, the algorithm has been unable to improve on the original. In that case, the original solution (recorded separately) is reinstated before the algorithm returns. There are two points to note here.

First, constraints do not wait for their last day to report a cost; they report a cost on every day that they are affected by. Each day, they report their cost on that day minus the cost they reported on the previous day. This makes $c(x)$ as large as possible on every day, causing as much discarding as possible.

Second, every cost difference must be non-negative. A negative difference means that there might have been an unjustified discard on the previous day. Cost differences are naturally non-negative when they derive from maximum limits. But costs may decrease as determinants grow towards minimum limits. All constraints know this and only report costs that cannot go away later. Detailed formulas showing how to do this appear online [11].

This algorithm runs in time polynomial in the number of selected days but exponential in the number of selected nurses, according to a straightforward worst-case analysis based on the fact that the solutions in each P_k have distinct signatures [11]. The algorithm will usually do much better than the worst case. For example, the analysis does not take into account the fact that solutions whose cost equals or exceeds the cost of the original solution are discarded, but testing shows that this has a pronounced effect, especially during the last few days. However, testing also shows that selecting a few more days is not particularly costly, but selecting just one more nurse can dramatically increase the running time, confirming the general thrust of the analysis.

6 Optimizations

Our algorithm contains many minor optimizations. But here we focus on three major ones, each of which significantly reduces running time, as we will show.

Generating fewer solutions. For each solution x there are up to $(a+1)^m$ solutions y that extend x for one more day, where a is the number of shift types and m is the number of selected nurses. Our first optimization, which is really several optimizations combined, aims to reduce this large number, using ideas that we expect have been tried before on other problems.

One idea that does not work is to identify equivalent nurses and avoid generating solutions that are symmetrical in those nurses' assignments. As mentioned earlier, nurses have several individual characteristics, and when those are combined with the different timetables that nurses have on unselected days, finding symmetries among the nurses is hopeless.

On the other hand, there are symmetries among the *tasks* that nurses may perform (the variables that nurses may be assigned to) on each day. We have already informally partitioned these tasks into classes—the 'shift types.' But there are subtleties here. There may be a task in the night shift reserved by a hard constraint for a senior nurse; some tasks may have hard constraints requiring them to be assigned; others may have soft constraints requesting that they remain unassigned. We analyse the event resource constraints that apply to each task, and group the tasks into classes of equivalent tasks, placing the most preferred tasks first in each class. When assigning nurses, there is no need to try all combinations of tasks from one class; instead, each nurse tries only the most preferred of the remaining unassigned tasks of the class.

The analysis may identify some tasks for which assignment is compulsory, because a hard constraint requires it. As nurses are assigned to tasks on one day, if the number of remaining unassigned nurses falls below the number of remaining unassigned compulsory tasks, that line of generation of assignments is abandoned. Compulsory tasks are common so this can be very effective.

As each decision to try assigning a particular nurse to a particular task is made, the cost of that decision is calculated immediately and added to the growing solution cost. If the result equals or exceeds the cost we are trying to improve on, again that line of generation of assignments is abandoned.

We have not included experiments to show the effect of these optimizations on running time. They are cheap to carry out, and some are hard to turn off.

Tradeoff dominance. Our second optimization, which we have not seen elsewhere, improves the dominance test, finding more cases of dominance and shrinking the sets P_k. We previously improved basic dominance into strong dominance (Section 5); now we improve strong dominance into *tradeoff dominance*.

Suppose that solution x fails to dominate solution y, but only at one point along the signature, and only by 1. Suppose that the corresponding constraint has weight w. Then the effect of this failure is that at some point in the future that constraint could have a cost in some x_t which is at most w greater than its cost in the corresponding y_t (assuming the cost function is not quadratic), and this is why dominance fails.

But if $c(x) + w \leq c(y)$, this extra w cannot make x_t cost more than y_t. In other words, x still dominates y even though dominance fails at one point.

This idea easily extends to differences greater than 1, and to multiple points along the signature. It is simply a matter, as we proceed along the signature, of adding to $c(x)$ the cost of ignoring each violation of dominance. Then, if $c(x)$ exceeds $c(y)$ at any point, dominance has failed even with this tradeoff.

Representing sets of solutions by tries. Our third optimization aims to speed up the insertion of a new solution x into the set of undominated solutions P_k. Recall that this involves testing whether P_k contains a solution that dominates x, and if so discarding x; and if not, finding and discarding all solutions in P_k that are dominated by x, then inserting x. The straighforward approach here, taken by all previous authors as far as this author can ascertain, is to take each element of P_k and see whether it dominates x. If all those tests fail, take each element of P_k again and see whether it is dominated by x, making two dominance tests for each element of P_k.

The author experimented with a dominance test he calls *weak dominance*, in which x dominates y if $c(x) \leq c(y)$ and the two signatures are equal. This finds many fewer cases of dominance than strong dominance does, but it allows P_k to be organized as a hash table indexed by signature, converting the search for dominating and dominated solutions into a single hash table retrieval. However, it turned out that the faster table handling did not compensate for the larger size of the sets P_k. The larger size did not slow down the hash table, but creating the many extra solutions took too much time.

Another failed optimization involved maintaining a *cache* of solutions C_k alongside each usual set of solutions P_k. All solutions y which were extensions of the same solution x were inserted into C_k with the usual dominance testing. When there were no more extensions of x, each surviving element of C_k was inserted into P_k, again with the usual dominance testing, and C_k was cleared ready to receive extensions of another x. The idea was that solutions which are extensions of the same parent are likely to have dominance relations with one another, and these relations can be found quickly within C_k. This is true, and the cache halved some running times, but the improvement disappeared when P_k was organized as a trie.

So let us turn now to an optimization that actually helped. The traditional trie is a symbol table representing some set of values, each associated with a key which is a sequence of characters. The root of the tree contains an array of subtrees. Each subtree contains all values whose keys have the same first character, and its index in the array is the integer value of that character. So to retrieve a value by key, one uses the first character of the key to index the root array to obtain a child, then the second character to index that child's array, and so on. When a subtree contains only a single value, it has a different format: the value and its key are stored, and retrieval compares the key it is looking for with the stored key to see whether the value is the one wanted. There are also null subtrees representing empty sets of values.

Solutions are a natural fit for tries. A solution's key is its signature, a sequence of small integers well suited to array indexing.

To decide whether x is dominated by any solution already in trie T, we proceed as follows. Suppose the first element of x's signature is v, and that it is associated with a maximum limit and so is dominated by any value less than or equal to v. We need to recursively search only those subtrees with indexes in the range 0 to v inclusive, not the whole trie. Similarly if v is associated with a minimum limit, we need to search all subtrees from v to the end of the array of children. If the test is equality, only the subree with index v needs to be searched. This applies at each level of the trie.

Deleting all solutions of T that are dominated by x is similar, with the array ranges swapped: if v is associated with a maximum limit, then all solutions dominated by v lie in the range from v to the end of the array, and so on. Insertion is just the usual trie insertion.

This description applies to basic dominance. Formulas saying exactly where to search under strong dominance are derived online [11]. Tradeoff dominance is a problem, because in principle the entire trie needs to be searched. But instead of that, the algorithm proceeds heuristically: at each level of the trie, it searches each position needed for strong dominance, plus up to two adjacent positions where a tradeoff of w is required.

A signature entry representing workload in minutes can cause efficiency problems for tries, because it is likely to lead to large arrays of children, mainly filled with null entries. The author has not investigated this in detail, but the answer is probably to find the greatest common divisor d of all the workloads and store the workload in minutes divided by d.

7 Experiments

This section offers experiments which show how the algorithm performs on a difficult instance from a standard data set, and how effective the optimizations are. We have not yet tried the algorithm on a wide variety of instances, or tried calling it repeatedly as the reassignment operator of a VLSN search.

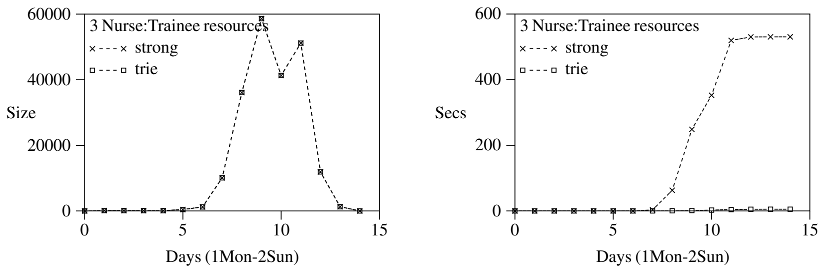


Fig. 1. The effect of introducing the trie data structure. The first graph shows the number of undominated solutions on each of the 14 days during which the selected resources are open to reassignment (almost 60,000 at the peak). This is the same with or without the trie data structure, because the same strong dominance test is used. The second graph shows the running time in seconds. For the ordinary array of undominated solutions, the running time is more than 500 seconds. The trie running time appears negligible. See Fig. 2 for a clearer view of the trie running time.

The experiments all use instance `INRC2-4-030-1-6291.xml`, the XESTT version of a 4-week instance derived from the Second International Timetabling Competition [2, 3] and tested by other authors [12, 13].

This instance divides into two independent parts, one for trainee nurses and one for non-trainee nurses. We show the trainee nurse results, because the algorithm runs more slowly on them. This is probably because trainee nurses can take each other's shifts, whereas non-trainee nurses may have varying skills, and can take some of each other's shifts but not all.

Figures 1 and 2 report on tests that select 3 nurses for reassignment over the first 14 days. The 3 nurses are chosen at random but are the same in all tests, as is the initial solution. This test happens to find an improvement on the initial solution. Fig. 1 shows the effect of changing the way that the solutions are stored in each P_k from an ordinary unsorted array to a trie. The same solutions are found, but running time improves dramatically. Fig. 2 shows the effect of

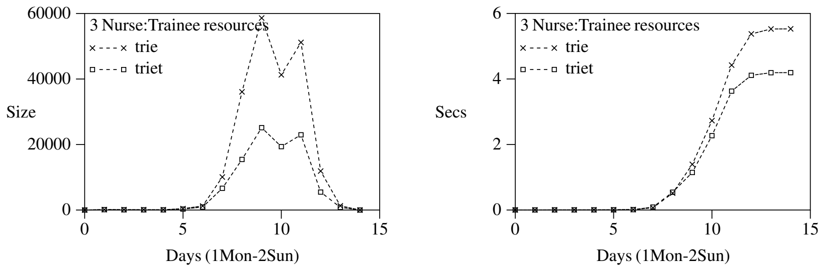


Fig. 2. The effect of introducing tradeoff dominance. This is the same instance as in Fig. 1, and the points labelled 'trie' are the same as in Fig. 1. The points labelled 'triet' are for the trie data structure with tradeoff dominance instead of strong dominance. The table size is reduced by more than half at the peak, and running time is reduced by about 30%.

changing from strong dominance to tradeoff dominance within the trie. The improvement is less dramatic but still useful.

The third figure, Fig. 3, shows what happens when the number of trainee nurses is increased to 4, using tries with tradeoff dominance. This run generates millions of undominated solutions and takes about 6000 seconds (100 minutes) to complete, a very bad result.

8 Conclusion

This paper has generalized the dynamic programming algorithm for optimal nurse rostering, and made some progress in optimizing it, in preparation for using it as the reassignment operator of a VLSN search. The implementation is available online [10] along with a detailed description [11].

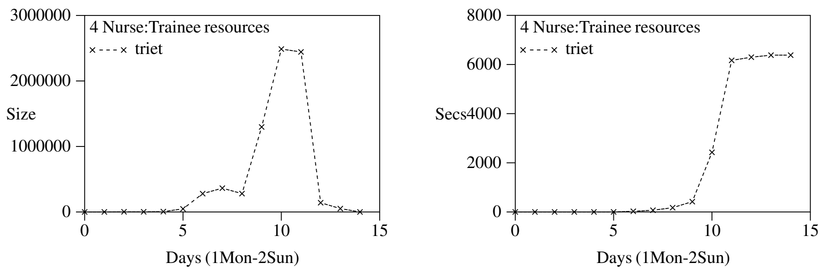


Fig. 3. As before, with tries and tradeoff dominance, but reassigning 4 trainee nurses.

The experiments have shown that the algorithm is able to reassign a fairly large number of days, for example 14. It is not able to reassign a large number of nurses. In this paper we have been able to reassign 3 nurses quickly, but not 4, so further improvement is needed.

Tries and tradeoff dominance do not interact well, and the implementation tested here does not find every case of tradeoff dominance when tries are used. The author has gathered statistics showing that the number of undominated solutions kept when both are used can be five times larger than when tradeoff dominance is applied to a simple list of undominated solutions. The trie is still faster, but it would be better if both ideas could perform at their best.

A well-known enhancement is to always extend a lowest cost solution next, chosen across all days. If a new best complete solution is found, all incomplete solutions whose cost exceeds its cost will then never be extended. The author has gathered statistics showing that 70% of the solutions created by the solve in Fig. 3 would either not be created or not be extended, if this was done.

Hand analysis shows that even tradeoff dominance is very conservative. By this we mean that there are many cases where one solution is morally certain to dominate another, but the dominance test does not succeed, because it cannot rule out the possibility that a set of highly improbable events will occur which allow the more costly solution to produce a superior extension. Further analysis of dominance testing could produce a major payoff.

As a last resort, we could keep, say, only the best 10,000 undominated solutions on each day. This would provide a very robust upper limit on the running time. In Fig. 3, if the solutions on each day are sorted by increasing cost, the solutions on the path to the optimal complete solution have remarkably low indexes in the sorted lists. For example, on the day with the largest number of undominated solutions, there are 2,486,741 undominated solutions, but the index of the solution on the path to the optimal complete solution is just 25. Of course, keeping only the best 10,000 solutions on each day would remove the guarantee of optimality. But then, the VLSN search that this algorithm is intended to support offers no guarantee of optimality either.

References

1. Edmund K Burke, Patrick De Causmaecker, Greet Vanden Berghe, and H. Van Landeghem, The state of the art of nurse rostering, Journal of Scheduling 7, pages 441–499 (2004). DOI 10.1023/B:JOSH.0000046076.75950.0b
2. Ceschia, S., Nguyen, T. T. D., De Causmaecker, P., Haspeslagh, S., Schaerf, A.: Second international nurse rostering competition (INRC-II), problem description and rules. oRR abs/1501.04177 (2015). http://arxiv.org/abs/1501.04177
3. Ceschia, S., Nguyen T. T. D., De Causmaecker, P., Haspeslagh, S., Schaerf, A.: Second international nurse rostering competition (INRC-II) web site, http://mobiz.vives.be/inrc2/
4. Cheang, B., Li, H., Lim, A., Rodrigues, B.: Nurse rostering problems – a bibliographic survey. European Journal of Operational Research **151**, 447–460 (2003)
5. Elshafei M., Alfares, H. K.: A dynamic programming algorithm for days-off scheduling with sequence dependent labor costs. Journal of Scheduling **11**, 85–93 (2008)
6. Irnich, S., Desaulniers, G., et al.: Shortest path problems with resource constraints. In: Column generation, chap. 2, 33âĂŞ65. Springer US (2005)

7. Kingston, J. H.: XESTT web site, http://jeffreykingston.id.au/xestt (2017)
8. Kingston, J. H., Post, G., Vanden Berghe, G.: A unified nurse rostering model based on XHSTT. In: PATAT 2018 (Twelfth International Conference on the Practice and Theory of Automated Timetabling, Vienna, August 2018), 81–96
9. Kingston, J. H.: Modelling history in nurse rostering. In: PATAT 2018 (Twelfth international conference on the Practice and Theory of Automated Timetabling, Vienna, August 2018), 97–111. Also Annals of Operations Research, https://doi.org/10.1007/s10479-019-03288-x
10. Kingston, J. H.: KHE web site (Version 2.7), http://jeffreykingston.id.au/khe (2022)
11. Kingston, J. H.: The KHE User's Guide (Version 2.7), Appendix C: Resource reassignment using dynamic programming. http://jeffreykingston.id.au/khe (2022)
12. Legrain, A., Omer, J., Rosat, S.: A rotation-based branch-and-price approach for the nurse scheduling problem. Mathematical Programming Computation **2019**, 1–34
13. Ceschia S., Schaerf, A.: Solving the INRC-II nurse rostering problem by simulated annealing based on large neighborhoods. In: PATAT 2018 (Twelfth international conference on the Practice and Theory of Automated Timetabling, Vienna, August 2018), 331–338

Predicting nurse rosters with machine learning techniques

Shayekh Hassan[1][0000-0002-6196-0318], Nadia Cissen[1][0000-0002-6792-8925] and Leendert Kok[1]

[1] ORTEC B.V., Houtsingel 5 2719 EA Zoetermeer, The Netherlands
shayekh.hassan@ortec.com

Abstract. Optimizing nurse rosters is a challenge in practice. A large number of labor rules, many individual preferences, and fuzzy objectives make it hard and cumbersome to create the right optimization model with all the relevant data. Since there are a lot of data patterns in nurse rosters, we tried a different approach using machine learning. We implemented supervised machine learning techniques to predict nurse rosters for a medical center by training our models on past rosters. The medical center uses as a rule of thumb that a roster is good if at least 80% of the roster is executed as planned. In our computational experiments, we found the best results with ensemble learning with an accuracy of over 90%. We consider this a remarkable result, given that the machine learning models have zero explicit knowledge of labor rules, preferences, roster objectives, occupancy requirements, and availabilities.

Keywords: Nurse rostering, Supervised learning, Roster prediction

1 Introduction

ORTEC Workforce Scheduling (OWS) is a leading employee rostering solution for various industries. Traditionally, OWS creates and optimizes rosters based on hard and soft constraints [1]. However, it has been observed that many users of OWS, especially in the healthcare industry, make significant changes to the optimized rosters or create rosters even entire manually [2]. The main reason for this is that there are typically many tacit roster preferences and criteria of planners and nurses that are too cumbersome for the users to include in OWS as constraints [2]. Next to that, there are many data patterns in healthcare rosters, such as working several night shifts in a row, having an entire weekend on or off, certain colleagues typically working together, etc. [2]. These two observations raised the question: would it be possible to predict rosters by learning from the past ones?

2 Model and solution approach

For this research, we obtained a dataset from a large medical center in the Netherlands. We choose the neurology department to create the supervised learning models, and we use two other departments, cardiology and IC Nurses, to verify the robustness of the models. We choose these departments because:

- They have a high number of nurses (Neurology: 186, Cardiology: 357, and IC Nurses: 258).
- The cardiology department is similar to the neurology department in terms of planning difficulty.
- According to planners, the IC nurses department is the most difficult one to plan.

We use data from 2019 and 2020, because further in the past, nurses and shifts were quite different from now, and therefore can contribute very little in predicting current rosters. We excluded data from 2021, as those rosters were not finalized yet at the time of data availability.

We categorized the shifts in the data by their start time into five different categories: day off, early, day, late, and night shift. The motivation for categorization is that the number of unique shifts for a department is very high, with only minor differences in start and end time. For example, the Neurology department has 49 shifts between 2019 and 2020. If it is possible to correctly predict the shift category a nurse will work, we will be able to further narrow down the unique shift the nurse works based on other deterministic methods (e.g., matching required and available skills) or further investigation using machine learning techniques. Therefore, the scope of this study has been limited to predicting nurse rosters based on the above-mentioned five categories of shifts only.

We choose random forest as the first supervised learning method for this study, primarily because of its ability to perform without much feature selection [3] and handle discrete data well [4]. We train the model by taking the roster of a certain day as the output and the preceding days as input. For input, we experimented with different horizons: 7 days, 30 days, 91 days, 183 days, and 366 days.

For predicting the whole roster of a month, we start with predicting the first day of that month using the preceding days as input. For the second and subsequent days of the month, we include the preceding predicted days in the input and iteratively construct this way for the entire month.

3 Computational experiments

We split the data such that we could use the roster of December 2020 as the test set and the preceding 12 months of data as the training set. Since this is a time-series data, we applied cross-validation on a rolling basis in 5 steps [5]. We found that the random forest model predicts the rosters the best when the whole preceding year is used as input. For this input (366 days), the weighted F_1 scores for the three departments are 0.8505, 0.8904, and 0.6598 (See Figure 1).

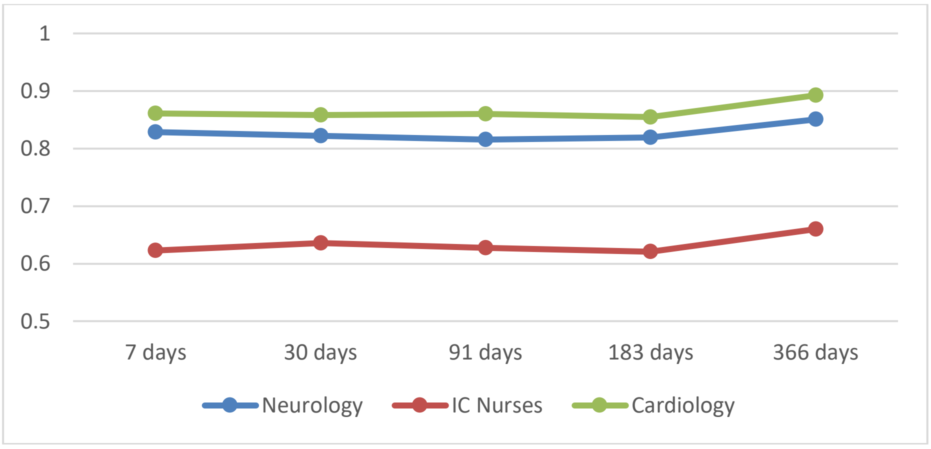


Fig. 1. Weighted average F_1 score of random forest model

We also developed gradient boosting and K-nearest neighbor (KNN) methods to compare the results. We used similar data processing methods as with the random forest model, and we used the same input and output structure. These models provided similar results: the highest weighted average F_1 scores are achieved when the input is 366 days, and the performance is the lowest for the department of IC nurses. Gradient boosting performed marginally better than the other two.

We also developed an ensemble learning model, combining the random forest, gradient boosting and KNN with maximum voting [6]. The results indicate that ensemble learning has a significantly higher weighted average F_1 score compared to the other models. For input of 366 days, the weighted F_1 scores for the three departments are 0.924 0.7432, and 0.9304.

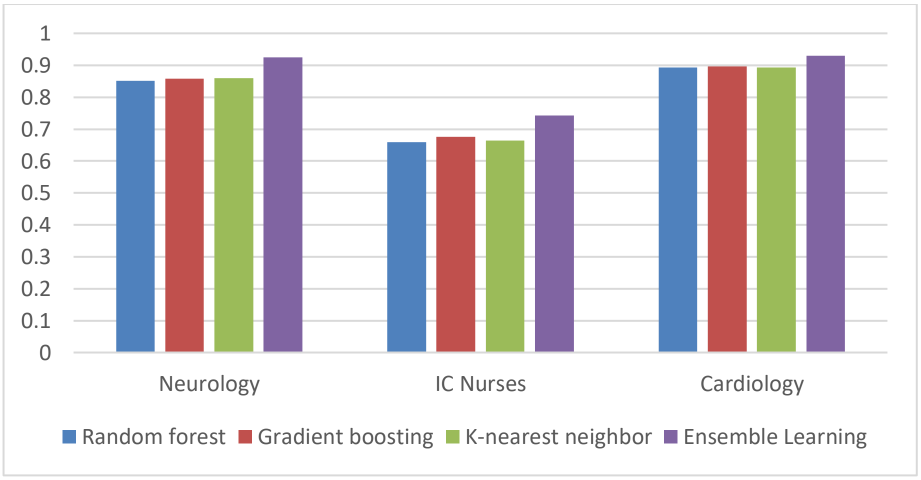


Fig. 2. Weighted average F_1 score of all models with 366 days as input

4 Conclusions

To conclude, supervised machine learning models, especially when multiple non-parametric models are combined, can predict nurse rosters to a high degree of accuracy based on past rosters of a year. Computational experiments indicate that accuracy and F_1 score of over 90% could be reached for certain departments. According to the planners, if the final roster is at least 80% similar to the planned roster, it is already considered good in practice. There is even more to win by including more information in the predictions, for example, labor rules, already scheduled holidays, etc. Further research will be conducted on applying these models to other customer data, improving these models' performances, and creating new models to predict more details (i.e., the exact shift a nurse will work).

5 References

1. Van Draat, L. F., Post, G., Veldman, B., & Winkelhuijzen, W: Harmonious personnel scheduling. Medium Econometrische Toepassingen, 14, 4-7 (2006).
2. Weersel, S. Preference learning: What defines an optimal shift schedule?. MS thesis. University of Twente, Twente (2019).
3. Fawagreh, K., Gaber, M.M., Elyan, E.: Random forests: From early developments to recent advancements. Systems Science & Control Engineering. 2, 602–609 (2014).
4. Janitza, S., Tutz, G., Boulesteix, A.-L.: Random Forest for ordinal responses: Prediction and variable selection. Computational Statistics & Data Analysis. 96, 57–73 (2016).
5. Hyndman, R.J., Athanasopoulos, G.: 5.10 Time series cross-validation. In: Forecasting: Principles and practice. Otexts, Lexington, Ky (2021).
6. Rokach, L.: Pattern classification using ensemble methods. World Scientific, Hackensack, New Jersey (2010).

A proven optimal result for a benchmark instance of the Uncapacitated Examination Timetabling Problem

Angelos Dimitsas[1], Vasileios Nastos[1], Christos Valouxis[2], Panayiotis Alefragis[3], and Christos Gogos[1]

[1] Dept. of Informatics and Telecommunications University of Ioannina, Arta, Greece
{a.dimitsas,vnastos,cgogos}@uoi.gr
[2] Dept. of Electrical and Computer Engineering, University of Patras, Greece
cvalouxis@upatras.gr
[3] Dept. of Electrical and Computer Engineering, University of Peloponnese,Patras, Greece alefrag@uop.gr

Abstract. Examination timetabling is a problem well known to the scheduling community. Its simplest version, which is the Uncapacitated Examination Timetabling Problem is easily described and comprehended. Nevertheless, proof of optimality is notoriously difficult even for moderate size problems. In this paper we describe the effort that our team exercised in finally proving the optimality of the sta83 instance of Carter's dataset. The problem was decomposed naturally in three parts and for each part a different approach managed to prove optimality of the currently best known solution. Several hours of computation were needed, but now we are confident that no solution exists with cost less than the proved optimal value. This work also presents optimal solutions to subproblems that exist in various public datasets problems and two best known solutions of such problems.

Keywords: Examination Timetabling · Mixed Integer Programming · Heuristics

1 Introduction

Timetabling problems arise in several domains including health-care, education, call centers, airlines and others. Rostering and scheduling are also commonly used terms to describe timetabling problems. In this paper we study the Uncapacitated Examination Timetabling Problem (UETP). UETP is the problem of scheduling university examinations to periods (time-slots) in such a way that no student should be examined at the same period for more than one course. Furthermore, the schedule of each student should allow enough time for studying between successive examinations. The problem is uncapacitated in the sense that no room capacities or availabilities are considered.

Our contribution to UETP is twofold. Firstly, we present a way of decomposing and reducing the sizes of the problems that results in obtaining two new best known solutions for benchmark instances. Secondly, and most noteworthy we propose a novel way of approaching a certain known instance of the Carter's dataset [6] of the UETP that results in actually proving the optimal value of the instance.

An outline of the paper follows. Section 2 provides a succinct description of the problem. Section 3 presents a glimpse of the broad bibliography for university examination problems capacitated or not. The next section describes our efforts to cleanse and decompose the problem instances so as to reduce their sizes, in an effort to feed various solution approaches with easier to digest problems. Section 5 is devoted to attacking the UETP problem instances with three specific methods that are later used in Section 6 to prove optimality for problem instance sta83 of the well known Carter's dataset. Next, our conclusions follow.

2 Problem Description

Each UETP instance contains information about the set of examinations that each student is enrolled in. Each instance has a specific number of periods that can be used to schedule the examinations to. The single hard constraint is that no student is allowed to participate in more than one examination per period. To allow time for each student to study between his examinations, for each student s, for each pair of examinations taken by s, a penalty of 16 is imposed if the two examinations occur in adjacent time slots (called distance 1), penalty 8 is imposed for distance 2, 4 for distance 3, 2 for distance 4, and 1 for distance 5.

The natural way to represent an instance is as an undirected weighted graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ where each vertex in $\mathbb{V}$ is an examination and each edge in $\mathbb{E}$ connects two examinations with common students. The weight of each edge is the number of common students for the examinations it connects.

3 Related work

The field of educational timetabling is very active. Several papers are typically published every year regarding course timetabling (post-enrollment and curriculum-based), examination timetabling, high school timetabling, thesis defense timetabling and others. Several surveys regarding the field have been published and present the challenges that such problems pose [12], [9]. The survey by Qu et al. [10] focuses on examination timetabling that is the subject of our work too. Recent surveys by Tan et al. [13] and Ceschia et al. [7] demonstrate the strong interest of the timetabling community for educational timetabling problems. Maybe this can be justified by the familiarity of such problems to academia circles. In [7] focus is given on "standard" formulations and benchmark instances that are also used in our work are presented. Another recent work for real world examination timetabling problems, this time, is the paper from Battistutta et al. [3].

Our team is also active in educational timetabling. In [8] we proposed a novel way of estimating lower bounds for UETP instances. Ideas about symmetry elimination, problem decomposition and cleansing of the instances were also presented there. The current paper serves as a follow-up and provides experimental results based on those ideas.

4 Preprocessing

Before solving each problem we perform a cleansing process through which we remove problem components that are insignificant and only add noise to the problem. The premise is that by solving the cleansed problem, we will still be able to find optimal solutions that will be optimal for the original problem. Furthermore, we identify independent subproblems that exist in each problem. Such subproblems can be solved independently and the solution to the original problem can be stitched by using the solutions of the subproblems. An exploration of the main ideas that we use for cleansing and decomposing the problems are more extensively described in our latest paper [8]. A synopsis follows.

Initially, we remove obvious noise students and examinations [2] (see lines 1 and 2 of Algorithm 1). Then, we identify subgraphs of the graph that can be handled independently. Note that the size of a subgraph refers to the number of its nodes which is equal to the number of the corresponding subproblem's exams. Subgraphs of size lower than $\lfloor \frac{P-1}{6} \rfloor + 1$ are identified as noise. This can be justified by the fact that we can spread the examinations of such subgraphs to the P available periods with zero penalty. Examinations with degree lower than $\frac{P}{11}$ are also noise since they can be always positioned with zero penalty. Then, any student that has a single non-noise examination and an arbitrary amount of noise examinations is also considered as noise. The process repeats until no more examinations or students can be marked as noise. A description of the procedure is given in Algorithm 1.

Another form of preprocessing involves the identification of interchangeable examinations that was proposed in [8]. These examinations have the same neighborhoods, as defined in graph $\mathbb{G}$, and the same number of common students for each neighbor. As these examinations are practically the same we can enforce them to either be in the same period if they are not in conflict or to follow a specific sequence of appearance in the final schedule if they have common students. By eliminating this type of symmetry of the problem, MIP/CP solvers are able to better explore the solution space.

4.1 Datasets

The standard benchmark dataset for UETP is Carter's dataset (a.k.a. Toronto dataset). Those instances were contributed in [6] back in 1996 and since then were used in many papers. Recently, 19 new instances that are modified versions of other more complex formulations, were added by Bellio et al. [4]. All of them are publicly available in https://opthub.uniud.it/ which is a site that

Algorithm 1: Remove noise examinations and students from an examination timetabling problem

```
   Input: An examination timetabling problem represented as a graph $G$
   Output: Graph $G$ with noise examinations and noise students removed
 1 Find students enrolled in a single exam, tag them as noise
 2 Find examinations with only noise students, tag them as noise
 3 Remove tagged examinations and students from $G$
 4 do              // loops until no more noise examinations are found
 5     done = True
 6     Let $S$ be the set of disconnected components (subgraphs) of $G$
 7     while $S \neq \emptyset$ do
 8         $G_i = \text{next}(S)$        // $i$ is the identifier of the subgraph
 9         if $|G_i| < \lfloor \frac{P-1}{6} \rfloor + 1$ then
10             Tag all examinations of $G_i$ as noise
11             Tag all students enrolled in examinations of $G_i$ as noise
12             Remove tagged examinations and students from $G_i$ and $G$
13             done = False
14         do
15             more_noise = False
16             for each examination $e$ in $G_i$ do
17                 if $deg_e < \frac{P}{11}$ then
18                     Tag $e$ as a noise
19                     Tag all students enrolled in $e$ as noise
20                     Remove tagged examinations and students from $G_i$ and $G$
21                     more_noise = True
22                     done = False
23         while more_noise
24         Remove $G_i$ from $S$
25 while not done
```

hosts definitions, datasets and solutions of several timetabling problems that have attracted the interest of the timetabling community.

The characteristics of the instances used in this paper are shown in Table 1. Conflict density is a metric that is computed by dividing the number of edges of the problem's corresponding graph by $n(n-1)/2$, where n is the number of vertices. Values for noise students and examinations are computed based on Algorithm 1. Moreover, the table presents the best known values that were obtained by solutions that we have downloaded from https://opthub.uniud.it/ in April 2022. Costs assume integer values and since the problem is of minimization nature, lower values are favored. Normalized costs are shown in the rightmost column of the table and are computed by dividing each integer cost by the corresponding number of students. The star symbol ($*$) in best known cost (95947) of instance sta83 indicates that this cost is optimal. At Section 6 we show that this is indeed the case. We consider it as the highlight of our work, since it is the first instance among the Carter's dataset for which it is proven that an optimal solution has been reached. It should be noted that the table has symbol † for the best known costs of two instances, ITC2007_9 and ITC2007_10. These best known values were contributed by our team and were obtained by exploiting the concept of noise examinations and students and the decomposition of problems to subproblems that enabled us to use optimal solutions to independent subproblems and search for good solutions using the Variable Neighborhood Search approach described in [2].

Table 1. Instances - descriptive statistics - noise examinations and noise students - best known costs

Instance id	Exams	Students	Periods	Conflict density	Noise exams	Noise students	Best known cost	Best known normalized cost
car92	543	18419	32	0.137986	10	3969	67084	3.6421
car91	682	16925	35	0.128386	13	3409	71727	4.2379
ear83	190	1125	24	0.266945	0	1	36473	32.4204
hec92	81	2823	18	0.420679	0	321	28325	10.0337
kfu93	461	5349	20	0.055579	33	276	68462	12.7990
lse91	381	2726	18	0.062592	3	99	26643	9.7737
pur93	2419	30029	42	0.029495	83	2627	120144	4.0009
rye93	486	11483	23	0.075279	1	2025	89999	7.8376
sta83	139	611	13	0.143989	0	0	*95947	*157.0327
tre92	261	4360	23	0.180696	3	667	33094	7.5904
uta92	622	21266	35	0.125557	5	6180	62675	2.9472
ute92	184	2749	10	0.084937	0	78	68090	24.7690
yor83	181	941	21	0.288889	0	1	32375	34.4049
ITC2007_1	607	7883	54	0.050495	25	227	5628	0.7139
ITC2007_2	870	12484	40	0.011695	238	2430	1538	0.1232
ITC2007_3	934	16365	36	0.026187	124	1306	20768	1.2690
ITC2007_4	273	4421	21	0.149968	0	4	47869	10.8276
ITC2007_5	1018	8719	42	0.008693	343	407	1567	0.1797
ITC2007_6	242	7909	16	0.061555	15	2622	30343	3.8365
ITC2007_7	1096	13795	80	0.019323	358	2620	262	0.0190
ITC2007_8	598	7718	80	0.045489	101	229	409	0.0530
ITC2007_9	169	624	25	0.078402	26	9	†2909	†4.6619
ITC2007_10	214	1415	32	0.049713	53	91	†12184	†8.6106
ITC2007_11	934	16365	26	0.026187	93	1306	54347	3.3209
ITC2007_12	78	1653	12	0.184482	4	684	10631	6.4313
D1-2-17	281	37	38	0.053254	21	1	2428	65.6216
D5-1-17	277	43	45	0.087166	53	0	3653	84.9535
D5-1-18	306	49	45	0.066560	47	0	3245	66.2245
D5-2-17	344	43	45	0.092447	2	0	8362	194.4651
D5-2-18	425	47	59	0.083629	6	0	6619	140.8298
D5-3-18	132	43	22	0.081309	2	0	1406	32.6977
D6-1-18	511	57	60	0.059975	74	0	9793	171.8070
D6-2-18	539	57	78	0.067639	10	0	7883	138.2982

4.2 Decomposed instances

After applying Algorithm 1 some problems are decomposed to subproblems. For most instances a number of examinations and students are removed since they are in effect noise. The resulting subproblems are presented in Table 2. The name of each subproblem follows the pattern d_i_(Ex_Sy_IDz), where d is the name of the originating instance, i is a number that assumes value 1 for the smallest subproblem and is incremented by 1 for each subsequent subproblem (subproblems are ordered by size = number of exams), x is the number of examinations, y is the number of students and z is the smallest examination number that exists in the subproblem. Number z is needed in order to differentiate among subproblems having the same number of examinations and same number of students. This is indeed the case for subproblems D1-2-17_1 and D1-2-17_2 that both have 8 examinations and 1 student but in the first case the identifying examination is 217 while for the second case the identifying examination is 257. Note that in Table 2 the number of examinations and the number of students exclude noise examinations and noise students respectively. Again, the presence of symbol $*$ denotes that the corresponding integer cost is optimal. It should be also noted that the normalized cost is computed by dividing the integer cost by the number of students (including noise ones) that exists in the originating instance.

5 Optimality proving tools

We have identified three different approaches to prove optimality for certain instances, and we present them below. Under certain conditions (number of exams, conflict density, current best known solution, number of periods) these approaches may be able to prove that a solution is indeed optimal.

5.1 Mixed Integer Programming

As optimality is our main concern the first thoughts that come to mind are Linear Programming and Mixed Integer Programming. The mathematical model described below can solve an UETP instance, provided that the instance size is manageable. For a graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ where vertices $\mathbb{V}$ serve as the exams, each edge in $\mathbb{E}$ means that two examinations have common students. The weight of an edge W_{v_1,v_2} connecting vertices v_1 and v_2 is equal to the number of common students these examinations have. P is the number of available periods.

The integer decision variables v_n in Equation 1 denote the period an examination will take place while the derived binary decision variables in Equation 2 help us to activate or deactivate penalties in the objective function in Equation 3. The constraint in Equation 4 forces examinations with common students to take place in different periods. Equation 5 forces binary decision variables in Equation 2 to indicate the distance between two exams. This constraint is not linear but capable solvers like IBM ILOG CPLEX using mathematical modeling tricks are able to linearize it out of the box. Equation 6 allows only one of the penalty indicating variables in Equation 2 to be active at any time. This constraint is redundant but its presence seems to help the solver in reaching better solutions.

$$v_n \in [0, P) \quad \forall n \in 1 \ldots |\mathbb{V}| \tag{1}$$

$$\begin{aligned} y16_{v_1,v_2} &\in \{0,1\} \quad \forall (v_1, v_2) \in \mathbb{E} \\ y8_{v_1,v_2} &\in \{0,1\} \quad \forall (v_1, v_2) \in \mathbb{E} \\ y4_{v_1,v_2} &\in \{0,1\} \quad \forall (v_1, v_2) \in \mathbb{E} \\ y2_{v_1,v_2} &\in \{0,1\} \quad \forall (v_1, v_2) \in \mathbb{E} \\ y1_{v_1,v_2} &\in \{0,1\} \quad \forall (v_1, v_2) \in \mathbb{E} \end{aligned} \tag{2}$$

Table 2. Problems resulted by decomposed instances of Table 1 - All noise examinations and noise students are removed

Instance id	Exams	Students	Conflict Density	Best known cost	Best known normalized cost
car92_1(E533_S18328_ID1)	533	18328	0.143139	67084	3.6421
car91_1(E669_S16750_ID1)	669	16750	0.133375	71727	4.2379
ear83_1(E190_S1125_ID1)	190	1125	0.266945	36473	32.4204
hec92_1(E81_S2823_ID1)	81	2823	0.420679	28325	10.0337
kfu93_1(E428_S5194_ID1)	428	5194	0.064326	68462	12.7990
lse91_1(E378_S2724_ID1)	378	2724	0.063576	26643	9.7737
pur93_1(E2336_S29766_ID1)	2336	29766	0.031566	120144	4.0009
rye93_1(E485_S11425_ID1)	485	11425	0.075590	89999	7.8376
sta83_1(E30_S162_ID1)	30	162	0.717241	*16002	*26.1899
sta83_2(E47_S210_ID3)	47	210	0.351526	*47250	*77.3322
sta83_3(E62_S239_ID4)	62	239	0.364357	*32695	*53.5106
tre92_1(E258_S4355_ID1)	258	4355	0.184840	33094	7.5904
uta92_1(E617_S21264_ID1)	617	21264	0.127539	62675	2.9472
ute92_1(E7_S20_ID30)	7	20	0.904762	*645	*0.2346
ute92_2(E177_S2729_ID1)	177	2729	0.090588	67445	24.5344
yor83_1(E181_S941_ID1)	181	941	0.288889	32375	34.4049
ITC2007_1_1(E582_S7798_ID1)	582	7798	0.054563	5628	0.7139
ITC2007_2_1(E9_S33_ID396)	9	33	0.888889	*0	*0.0000
ITC2007_2_2(E623_S9636_ID1)	623	9636	0.020856	1538	0.1232
ITC2007_3_1(E810_S15726_ID1)	810	15726	0.034214	20768	1.2690
ITC2007_4_1(E273_S4421_ID1)	273	4421	0.149968	47869	10.8276
ITC2007_5_1(E11_S9_ID434)	11	9	0.690909	*0	*0.0000
ITC2007_5_2(E13_S41_ID206)	13	41	0.487179	*0	*0.0000
ITC2007_5_3(E14_S263_ID120)	14	263	0.989011	189	0.0217
ITC2007_5_4(E637_S7559_ID1)	637	7559	0.018236	1378	0.1580
ITC2007_6_1(E4_S12_ID5)	4	12	1.000000	*33	*0.0042
ITC2007_6_2(E7_S75_ID122)	7	75	0.666667	*7	*0.0009
ITC2007_6_3(E27_S210_ID9)	27	210	0.293447	146	0.0185
ITC2007_6_4(E189_S7386_ID3)	189	7386	0.093662	30157	3.8130
ITC2007_7_1(E18_S143_ID178)	18	143	0.732026	*0	*0.0000
ITC2007_7_2(E720_S10034_ID2)	720	10034	0.040604	262	0.0190
ITC2007_8_1(E497_S7388_ID1)	497	7388	0.062764	409	0.0530
ITC2007_9_1(E143_S603_ID2)	143	603	0.105683	2909	4.6619
ITC2007_10_1(E7_S81_ID1)	7	81	1.000000	*196	*0.1385
ITC2007_10_2(E9_S91_ID78)	9	91	0.888889	*14	*0.0099
ITC2007_10_3(E11_S29_ID87)	11	29	1.000000	*54	*0.0382
ITC2007_10_4(E12_S111_ID121)	12	111	0.984848	1021	0.7216
ITC2007_10_5(E15_S59_ID200)	15	59	0.857143	292	0.2064
ITC2007_10_6(E16_S220_ID133)	16	220	0.958333	878	0.6205
ITC2007_10_7(E16_S124_ID166)	16	124	0.800000	338	0.2389
ITC2007_10_8(E16_S56_ID51)	16	56	0.550000	76	0.0537
ITC2007_10_9(E17_S143_ID149)	17	143	0.757353	836	0.5908
ITC2007_10_10(E19_S208_ID13)	19	208	0.964912	2356	1.6650
ITC2007_10_11(E23_S215_ID98)	23	215	0.909091	6123	4.3272
ITC2007_11_1(E841_S15857_ID1)	841	15857	0.031989	54347	3.3209
ITC2007_12_1(E5_S62_ID35)	5	62	0.900000	*22	*0.0133
ITC2007_12_2(E69_S1464_ID1)	69	1464	0.232310	10609	6.4180
D1-2-17_1(E8_S1_ID217)	8	1	1.000000	*5	*0.1351
D1-2-17_2(E8_S1_ID257)	8	1	1.000000	*5	*0.1351
D1-2-17_3(E10_S1_ID119)	10	1	1.000000	*17	*0.4595
D1-2-17_4(E11_S1_ID218)	11	1	1.000000	*26	*0.7027
D1-2-17_5(E12_S1_ID189)	12	1	1.000000	*36	*0.9730
D1-2-17_6(E13_S2_ID100)	13	2	0.538462	*0	*0.0000
D1-2-17_7(E14_S1_ID173)	14	1	1.000000	*62	*1.6757
D1-2-17_8(E18_S1_ID1)	18	1	1.000000	*150	*4.0541
D1-2-17_9(E18_S1_ID51)	18	1	1.000000	*150	*4.0541
D1-2-17_10(E28_S2_ID7)	28	2	0.592593	*190	*5.1351
D1-2-17_11(E120_S18_ID44)	120	18	0.164286	1787	48.2973
D5-1-17_1(E11_S3_ID98)	11	3	1.000000	*48	*1.1163
D5-1-17_2(E13_S3_ID99)	13	3	0.846154	*12	*0.2791
D5-1-17_3(E200_S34_ID5)	200	34	0.158945	3593	83.5581
D5-1-18_1(E9_S2_ID263)	9	2	1.000000	*8	*0.1633
D5-1-18_2(E13_S3_ID88)	13	3	0.846154	*12	*0.2449
D5-1-18_3(E14_S2_ID200)	14	2	0.736264	*10	*0.2041
D5-1-18_4(E223_S41_ID1)	223	41	0.118046	3215	65.6122
D5-2-17_1(E18_S1_ID199)	18	1	1.000000	*108	*2.5116
D5-2-17_2(E324_S42_ID1)	324	42	0.101307	8254	191.9535
D5-2-18_1(E18_S1_ID97)	18	1	1.000000	54	1.1489
D5-2-18_2(E56_S5_ID94)	56	5	0.318182	140	2.9787
D5-2-18_3(E345_S41_ID1)	345	41	0.116144	6425	136.7021
D5-3-18_1(E5_S2_ID40)	5	2	1.000000	*6	*0.1395
D5-3-18_2(E7_S1_ID59)	7	1	1.000000	*18	*0.4186
D5-3-18_3(E118_S40_ID3)	118	40	0.097349	1382	32.1395
D6-1-18_1(E12_S1_ID470)	12	1	1.000000	*7	*0.1228
D6-1-18_2(E22_S2_ID85)	22	2	0.636364	*32	*0.5614
D6-1-18_3(E403_S52_ID1)	403	52	0.092947	9754	171.1228
D6-2-18_1(E14_S1_ID1)	14	1	1.000000	*1	*0.0175
D6-2-18_2(E22_S1_ID343)	22	1	1.000000	*56	*0.9825
D6-2-18_3(E493_S54_ID3)	493	54	0.077904	7826	137.2982

$$\begin{aligned}\min \quad 16 * \sum_{v_1,v_2 \in \mathbb{E}} W_{v_1,v_2} * y16_{v_1,v_2} + 8 * \sum_{v_1,v_2 \in \mathbb{E}} W_{v_1,v_2} * y8_{v_1,v_2} \\ +4 * \sum_{v_1,v_2 \in \mathbb{E}} W_{v_1,v_2} * y4_{v_1,v_2} + 2 * \sum_{v_1,v_2 \in \mathbb{E}} W_{v_1,v_2} * y2_{v_1,v_2} \\ + \sum_{v_1,v_2 \in \mathbb{E}} W_{v_1,v_2} * y1_{v_1,v_2}\end{aligned} \tag{3}$$

$$\text{s.t.} \quad v_1 \neq v_2 \quad \forall (v_1, v_2) \in \mathbb{E} \tag{4}$$

$$\begin{aligned} y8_{v_1,v_2} &= (v_1 - v_2 = 2) + (v_1 - v_2 = -2) \quad \forall (v_1, v_2) \in \mathbb{E} \\ y4_{v_1,v_2} &= (v_1 - v_2 = 3) + (v_1 - v_2 = -3) \quad \forall (v_1, v_2) \in \mathbb{E} \\ y2_{v_1,v_2} &= (v_1 - v_2 = 4) + (v_1 - v_2 = -4) \quad \forall (v_1, v_2) \in \mathbb{E} \\ y1_{v_1,v_2} &= (v_1 - v_2 = 5) + (v_1 - v_2 = -5) \quad \forall (v_1, v_2) \in \mathbb{E} \end{aligned} \tag{5}$$

$$y16_{v_1,v_2} + y8_{v_1,v_2} + y4_{v_1,v_2} + y2_{v_1,v_2} + y1_{v_1,v_2} \leq 1 \quad \forall (v_1, v_2) \in \mathbb{E} \tag{6}$$

Finally, let $\mathbb{I}^+$ be the set of sets of interchangeable examinations as defined in [8]. In order to break a symmetry of the problem we enforce an order over the examinations belonging to each set. This is formulated in Equation 7, where members of each set $\mathbb{S}$ of the sets in $\mathbb{I}^+$ are ordered among each other.

$$v_i \leq v_{i+1} \quad \forall v_i \in \mathbb{S} : i \in 1 \ldots |\mathbb{S}| - 1, \quad \forall \mathbb{S} \in \mathbb{I}^+ \tag{7}$$

Other formulations of the mathematical model have been proposed in the past. An example is the work in [4] that uses the so-called channeling constraints that were originally proposed in [1]. A difference in our model is that we employ the concept of interchangeable examinations that are embedded in the formulation. Moreover, the objective function is constructed equivalently, but differently, in our case.

5.2 Intelligent enumeration

Some of the instances have a comparatively small number of available periods. It's noteworthy that even small sub-problems with a few periods and a relatively low number of examinations are hard to optimally solve by current state of the art mixed integer programming solvers. A new method was developed to handle instances, and this method depending on the number of examinations, available periods and the conflict density of the corresponding graph is able to solve some problems to optimality. Moreover, the same method can be exploited and reach good solutions for bigger instances.

To best describe this process we will use a toy example with its graph representation pictured in Figure 1. Let the available periods for this problem to be four. The problem consists of five examinations with a varying number of common students between certain pairs of exams. Note that exams $1, 2, 3$ form a non trivial clique e.g. they are a complete sub-graph of the graph. As no student is allowed to participate in more than one examination per period, those three examinations will end up in three different periods. Also, examination 5 with a weighted degree of just 2 doesn't seem to play a major part in the grander scheme of things.

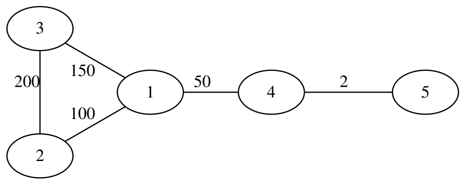


Fig. 1. Toy example for demonstrating the intelligent enumeration scheme.

The method can be used to either search for a good solution or to prove optimality, based on characteristics of the problem in question. The main idea remains the same for both cases. Firstly, we reduce the problem size by removing some of its exams. Then, we generate partial solutions, evaluate their cost and if it falls under some cut-off limit, which could be the cost of the best known solution, we fill the missing examinations to form a complete solution. This process is expected to act as a filter and has the potential to be computationally faster than a full enumeration.

The main idea of the method is to exploit a clique in the graph. In selecting a clique, it usually makes sense to choose the maximum clique. In the toy example, the maximum clique is the set of examinations $\{1, 2, 3\}$. It is guaranteed that the clique's examinations will end up on different periods which, for convenience, we name after them, $\{P_1, P_2, P_3\}$ correspondingly. Since we have four available periods we will name the period that will not be occupied by any of them as P_E. The remaining examinations $\{4, 5\}$ can be easily checked in this small example about their possible final positions. So, examination 4 can be placed in any of $\{P_2, P_3, P_E\}$ and examination 5 can join any period $\{P_1, P_2, P_3, P_E\}$.

Since examinations for the clique are fixed in periods $\{P_1, P_2, P_3\}$ the possible assignments for examinations 4 and 5 are $(4 : P_2, 5 : P_1), (4 : P_2, 5 : P_3), (4 : P_2, 5 : P_E), (4 : P_3, 5 : P_1), (4 : P_3, 5 : P_2), (4 : P_3, 5 : P_E), (4 : P_E, 5 : P_1), (4 : P_E, 5 : P_2), (4 : P_E, 5 : P_3)$ while $(4 : P_2, 5 : P_2), (4 : P_3, 5 : P_3), (4 : P_E, 5 : P_E)$ are infeasible as examinations 4 and 5 are in conflict. In total, there are 9 feasible schedules. If we had opted to leave examination 5 out, there would be just 3 feasible schedules $(4 : P_2), (4 : P_3), (4 : P_E)$.

Initially, we ignore examination 5 and we examine all possible permutations of $\{P_1, P_2, P_3, P_E\}$. We complement every permutation with each of the 3 possible partial schedules $(4 : P_2), (4 : P_3), (4 : P_E)$. Since each schedule and its reverse have exactly the same objective value, we can skip mirrored permutations, effectively cutting off half of the search space, thus eliminating this kind of symmetry. Nevertheless, for large numbers of periods, it is unrealistic to traverse all possible permutations, even by considering half of them. In the toy example, we evaluate $(4!/2) * 3$ partial solutions and we keep those that have cost under a cut-off barrier. The unscheduled examination 5 has a weighted degree of just 2, while other examinations have weighted degrees ranging from 52 to 350. So, most of the partial solutions should be filtered out.

Examination 5 of the toy example was initially ignored. A similar decision must be taken for each problem, about the examinations that will be initially ignored too. Unfortunately, this is not a trivial task. We cannot remove examinations of the chosen clique, should we wish to do so we should pick another clique. Intuitively, we want to initially ignore examinations with low degrees and weighted degrees, as they are able to appear in more periods. Consequently, they allow for more possible outcomes while at the same time their impact on the objective function is minor. It should be noted that not all partial solutions (solutions with ignored examinations still unscheduled) may lead to feasible solutions. So, for the case that full enumeration is unrealistic, quick feasibility checks can reveal unpromising partial solutions that are meaningless to be completed. The method is tuned by balancing the number of possible partial schedules generated with respect to the impact that the selected examinations have on the objective. The tuning is guided by selecting, through sampling, suitable examinations that will hopefully result in cutting-off many possible solutions. For the toy example the costs of these partial solutions are depicted in Table 3

Table 3. Permutations and partial solutions costs for the toy example in Fig. 1

P_1	P_2	P_3	P_E	$4 : P_2$	$4 : P_3$	$4 : P_E$
0	1	2	3	6800	6400	6200
0	1	3	2	4600	4000	4200
0	2	1	3	6800	7200	6600
0	2	3	1	5000	4800	5400
0	3	1	2	4600	5200	4800
0	3	2	1	5000	5200	5600
1	2	0	3	6400	6400	6000
1	2	3	0	6800	6400	6800
1	3	0	2	4400	4800	4800
1	3	2	0	6800	7200	7200
2	3	0	1	4400	4000	4400
2	3	1	0	6400	6400	6000

To further augment our filter while keeping computational cost low it's possible for partial solutions that are under the cut-off barrier to calculate the minimum cost each unscheduled examination can possibly introduce to the partial solution. If the sum of those minimum costs plus our partial solutions cost is under the cut-off barrier, the partial solution may lead to a desired complete solution. This process can be seen as a multi-layer filter like the one depicted in Fig. 2.

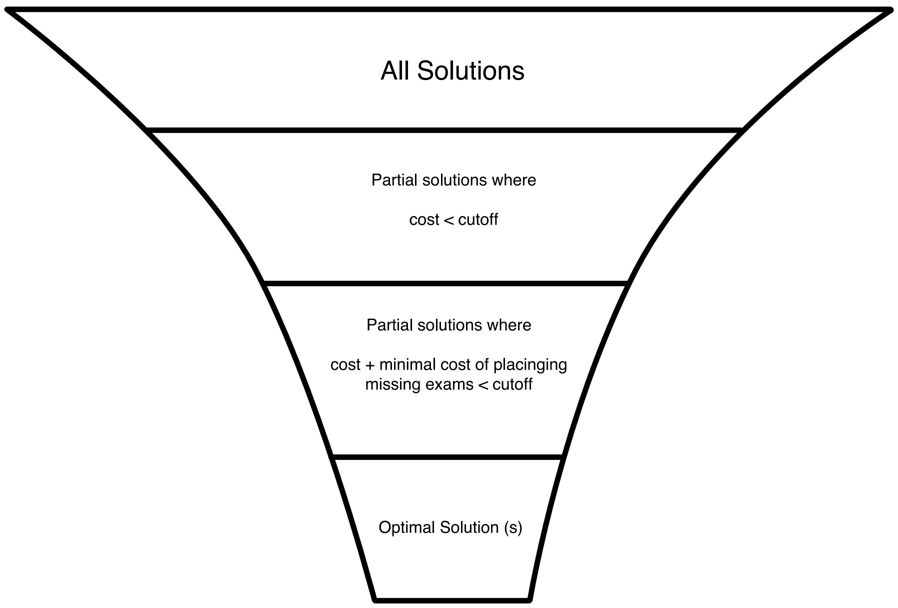


Fig. 2. Filter process.

5.3 Estimating lower bounds

Each students schedule is also an UETP sub-problem where his examinations are a complete graph where all edges have a weight of 1. This problem can be solved optimally for almost all instances, especially for those with a low number of periods.

Summing up those minimum penalties for all students can provide us with a lower bound. In the rare occasion that a solution's objective function is equal to this bound then this solution is optimal.

6 sta83 optimal solution

No optimality has ever been proved for any Carter's dataset instance until now. In this section we show that the solution for sta83 having value 95947 (95947/611=157.0327 in decimal value, where 611 is the total number of students for sta83) which appears in many papers is indeed optimal.

Instance sta83 consists of 139 exams, 13 periods and has a relatively low conflict density of value 0.14. The instance has no noise examinations and no noise students as defined in Section4. The instance is comprised of 3 disconnected components as shown in Fig. 3.

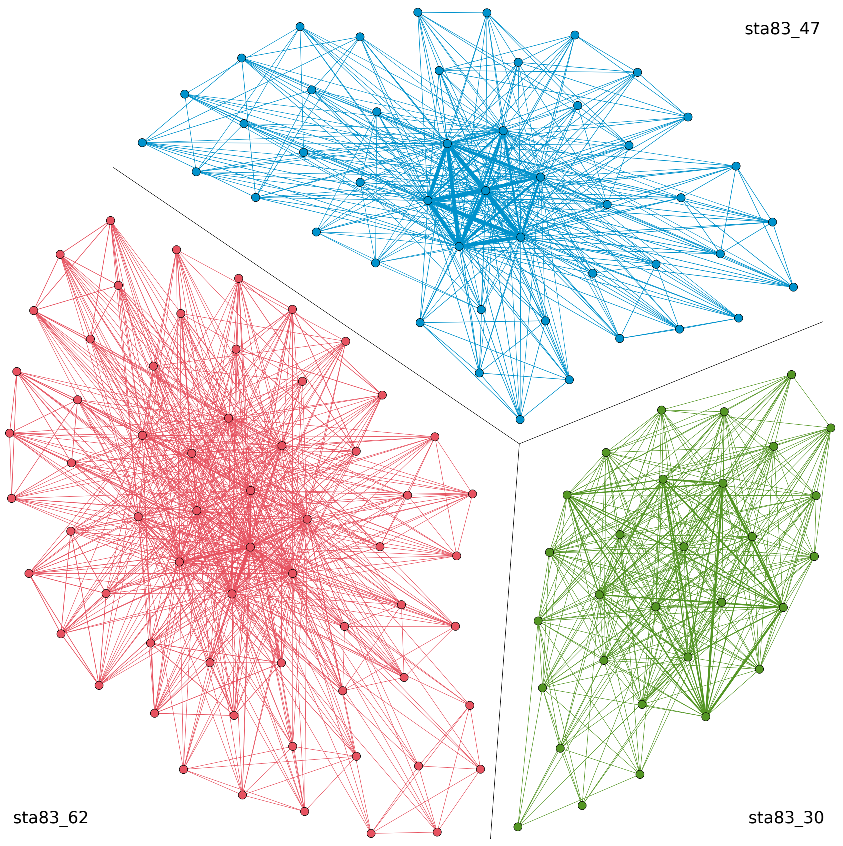


Fig. 3. Disconnected components of sta83. The weight of each edge is indicated by its thickness.

We can divide the problem into three independent subproblems because these components are disconnected. That is, there are three unique groups of students, each of which does not have an examination in common with the other two groups, allowing us to work on each component independently. The sum of these answers would be the optimal solution provided that all three of them are solved optimally. Motivated by the prospect of proving optimality for a Carter's dataset instance, we focused our attention on this task, and we managed to optimality solve each subproblem using a different approach, resulting in a novel way of handling high conflict-density components.

6.1 Component sta83_62

This is the largest component of sta83, having 62 examinations and a conflict density of 0.36. We tried to solve it using the model described in Section 5.1 using the IBM ILOG CPLEX IP solver. Unfortunately, after several hours the solver was unable to prove optimality. We tried to warm start the solution with the current best solution and have set the MIP emphasis parameter first to "emphasize optimality over feasibility" and then to "emphasize moving best bound". Both attempts were unsuccessful.

We noticed that the component has a special structure. It contains 10 sets of examinations with each set consisting of exactly 5 interchangeable examinations. These examinations amount for 50 of the 62 examinations that the component has in total. Details of these sets are presented in Table 4. Since interchangeable examinations can freely swap places with each other while keeping the objective value unchanged, the introduction of the symmetry breaking constraints of Equation 6 greatly improved the solver's efficiency in proving the optimal solution.

We also noticed that 3 examinations existed (72, 133, 136) in the graph that had connections with all other exams. So, we tried an approach that fixed these 3 examinations in specific periods and then tried to solve the remaining problem using IBM ILOG CPLEX. This time, the result was successful, the solver was able to return a result, either optimal or infeasible in a few minutes. It should be noted that infeasibility occurs because the cost of the best known solution is used as a cutoff constraint. So, we had only to try all possible places for positioning the 3 examinations and then solve the resulting problem. Since there are only 13 periods in instance sta83, this would mean that only $\binom{13}{3} = 286$ configurations existed that should be multiplied by $\frac{3!}{2}$ since the 3 examinations can occupy the fixed periods in any order (divided by 2 due to the inherent symmetry of the problem).

By exploiting the above observations, IBM ILOG CPLEX IP solver was able to solve each subproblem in a few minutes. After solving all subproblems, the optimal solution for sta83_62 was proved to be 32695. This solution occurred when examinations 72, 133 and 136 were fixed to periods 3, 6 and 8 respectively. The symmetric solution also exists and is produced by fixing examinations 72, 133 and 136 to periods 9, 6 and 4. Of course, many more symmetric solutions exist due to the interchangeable exams.

6.2 Component sta83_47

This component proved to be the easy part. It consists of 47 examinations and has a conflict density of 0.35. As described in subsection 5.3 we can estimate a lower bound by adding the minimum cost each student's schedule could possibly inflict. So, for each student in isolation, an IP model is formulated that given only the number of periods and the number of examinations that this student participates, decides about the schedule that results to the minimum possible cost. Of course, since each student is examined in isolation if two students share the same number of examinations then the problem needs to be solved just once. In practice, this is the case for several students. By adding minimum penalties of all students we have a lower bound for this component, which is 42750. The best known solution turns out to have cost equal to the lower bound obtained in this manner. Thus, the optimal solution for this component is 47250.

6.3 Component sta83_30

This was the last component to solve. It's the smallest one with just 30 examinations but a high conflict density of 0.72. With high hopes since just the smallest piece of the puzzle was missing, we were surprised to find out that to the best of our ability our MIP models were not able to prove an optimal solution. We have tried the same trick that we have used successfully in component sta83_62. We noticed that in the case of sta83_30 there is only one examination (134) that is connected to every other one. So, we tried to fix this examination to each period in turn and then to solve the remaining problems using IBM ILOG CPLEX. Unfortunately, this did not helped the solver to prove the optimality of the solution. Each subproblem seemed to run forever.

By observing closely the high density graph of this component we came up with the idea of separating examinations with high degrees and examinations with relatively low degrees. A similar idea has been exploited by [11] and others in constructing timetables giving precedence to high degree examinations and

Table 4. Component sta83_62, sets of interchangable examinations and their characteristics.

Set	Degree	Weighted Degree
{17, 38, 58, 85, 120}	8	8
{18, 39, 59, 86, 121}	16	240
{19, 40, 60, 87, 122}	16	264
{20, 41, 61, 88, 123}	15	168
{21, 42, 62, 89, 124}	12	88
{22, 43, 63, 90, 125}	16	160
{23, 44, 64, 91, 126}	15	160
{24, 45, 65, 92, 127}	16	264
{25, 46, 66, 93, 128}	16	280
{26, 47, 67, 94, 129}	16	280

leaving for a later phase the low degree ones. In our approach, we isolated the maximum clique, which for this particular instance comprises of 12 examinations and tried to arrange those examinations to the 13 periods leaving one period empty for each possible arrangement.

A significant observation is that irrelevant of the periods that the clique occupies, the possible placements for the remaining examinations will be the same because their possible positions are constrained by the examinations of the clique. By multiplying the number of those possibilities with the number of permutations of the periods we were able to count all possible solutions to be $13! * 109152$ where 13! is the number of possible period permutations and 109152 is the number of possible ways to schedule the remaining examinations for the specific component. This number is still quite large so we exploited the method described in Section 5.2. We aim to find a set of examinations that has minor impact on the cost but at the same time possible final positions of the sets' examinations might be disproportionate large. Fig. 4 which shows the degrees and weighted degrees of examinations was used as a visual aid for identifying the examinations needed. These examinations should reside at the lower left corner and should have the desirable characteristics. For sta83_30 a good set of examinations proved to be $\{5, 131, 28, 48, 76\}$ that manages to lower multiplier 109152 to just 47.

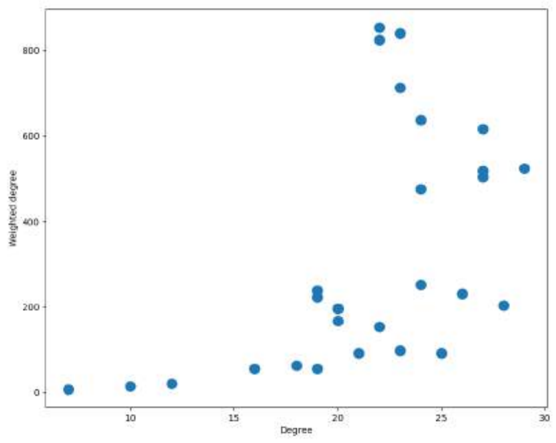


Fig. 4. Scatter plot of sta83_30 that gives insight about the set of examinations that should be scheduled last

The unscheduled examinations weighted degree is comparatively low and so the method has the potential of working effectively. By keeping in the set of initially unscheduled examinations, examinations that can easily move around the schedule, the number of possible partial solutions becomes quite low. Moreover, the low weighted degree that these examinations have prohibits from heavy impacts on the objective function. So, the filtering process is working. For the case of sta83_30 this "intelligent" search resulted in 13 distinct optimal solutions (and their symmetric ones) all having the same cost, 16002. The search was implemented in Julia [5] using its parallel computing features for the CPU. Five high end workstations were simultaneously running the experiment and the time needed was about 12 hours.

7 Conclusions

This work was about the uncapacitated examination timetabling problem. It continues previous work of our team. A key observation is that even for this rather simple scheduling problem that is only an abstraction of the corresponding real-life problem, the proof that a given solution is optimal is definitely not trivial. Nevertheless, our team succeeded in proving the optimality of a certain instance, namely sta83 of the Carter's dataset. In order for this to happen we had to decompose the problem into independent subproblems. Having 3 problems of moderate size gave us the opportunity of experimenting with various approaches. No method was able to solve all three subproblems. After many experiments and carefully analyzing the components, we finally discovered three approaches that were able to prove optimality. Each subproblem was solved by a different approach and the optimal solution for sta83 was proved. Furthermore, we contributed two new best solutions to public dataset problems, alongside with several optimal solutions to subproblems that exist in various instances.

Acknowledgements We acknowledge support of this work by the project "Dioni: Computing Infrastructure for Big-Data Processing and Analysis." (MIS No. 5047222) which is implemented under the Action "Reinforcement of the Research and Innovation Infrastructure", funded by the Operational Programme "Competitiveness, Entrepreneurship and Innovation" (NSRF 2014-2020) and co-financed by Greece and the European Union (European Regional Development Fund).

References

1. Aardal, K.I., Van Hoesel, S.P., Koster, A.M., Mannino, C., Sassano, A.: Models and solution techniques for frequency assignment problems. Annals of Operations Research **153**(1), 79–129 (2007)
2. Alefragis, P., Gogos, C., Valouxis, C., Housos, E.: A multiple metaheuristic variable neighborhood search framework for the uncapacitated examination timetabling problem. In: Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling-PATAT. vol. 1, pp. 159–171 (2021)

3. Battistutta, M., Ceschia, S., Cesco, F.D., Gaspero, L.D., Schaerf, A., Topan, E.: Local search and constraint programming for a real-world examination timetabling problem. In: International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research. pp. 69–81. Springer (2020)
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 (Aug 2021)
5. Bezanson, J., Edelman, A., Karpinski, S., Shah, V.B.: Julia: A fresh approach to numerical computing. SIAM Review **59**(1), 65–98 (2017). https://doi.org/10.1137/141000671
6. Carter, M.W., Laporte, G., Lee, S.Y.: Examination Timetabling: Algorithmic Strategies and Applications. Journal of the Operational Research Society **47**(3), 373–383 (Mar 1996)
7. Ceschia, S., Di Gaspero, L., Schaerf, A.: Educational timetabling: Problems, benchmarks, and state-of-the-art results. arXiv preprint arXiv:2201.07525 (2022)
8. 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. IEEE (2021)
9. Kristiansen, S., Stidsen, T.R.: A comprehensive study of educational timetabling, a survey. Department of Management Engineering, Technical University of Denmark.(DTU Management Engineering Report **8** (2013)
10. 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**(1), 55–89 (2009)
11. Rahman, S.A., Bargiela, A., Burke, E.K., Ozcan, E., McCollum, B.: Construction of examination timetables based on ordering heuristics. In: 2009 24th International Symposium on Computer and Information Sciences. pp. 680–685. Ieee (2009)
12. Schaerf, A.: A survey of automated timetabling. Artificial intelligence review **13**(2), 87–127 (1999)
13. Tan, J.S., Goh, S.L., Kendall, G., Sabar, N.R.: A survey of the state-of-the-art of optimisation methodologies in school timetabling problems. Expert Systems with Applications **165**, 113943 (2021)

Exam Scheduling With Hardship Minimization

Stephanie Hamilton and Donovan R. Hare

Department of Mathematics
University of British Columbia
Kelowna, British Columbia, Canada V1V 1V7
donovan.hare@ubc.ca

Abstract. The exam scheduling problem is a computationally difficult problem whose solution assigns exams to timeslots and rooms while satisfying a variety of examinee and institutional requirements. This paper proposes a generic specification of the problem that has wide applicability to real-world situations. In particular, the number of student *hardships* (i.e., students assigned multiple exams in a specified time interval) is defined in a way to encompass most contexts. Constraint programming (CP) implementations of the generic specification are then defined to model the timeslot assignment subproblem and a room assignment subproblem independently.
Two approaches are proposed and implemented to solve the overall problem from the subproblems: the use of a novel group of cuts, and the use of bin-packing global constraints. The cuts provide necessary conditions for a feasible solution of the timeslot assignment problem to have feasible room assignments. It is also shown that these cuts are sufficient conditions in certain general cases. A final section gives an empirical study using the data from the University of British Columbia.

1 Introduction

Educational timetabling is an interdisciplinary research area that has received a lot of attention over the last 20 years. In essence, the problem involves assigning university exams to timeslots and rooms in such a way that students do not write any of their own exams simultaneously, which can be elaborated on by imposing other institutional requests. Automated timetabling first caught the attention of mathematicians in the 1960's, at a time when most scheduling was done manually [6,8]. Fast forward 60 years to where the theory has advanced remarkably but in practice the scheduling software many institutions use do not implement these new research techniques, and thus institutions still struggle to get quality timetables that satisfy student and instructor needs [12]. Part of this is due to the research gap between theory and reality [10].

To address this gap, there have been three international timetabling competitions (ITCs) focusing on university timetabling. Since this problem is computationally difficult (i.e., NP-hard), researchers have tried various methods to tackle it. A sample of these methods include heuristics such as simulated annealing, population-based algorithms such as memetic algorithms, graph colouring heuristics, integer programming, and constraint-based methods (for recent literature reviews and surveys, refer to [1,3,7,13]). The winner of ITC 2007, who used a constraint satisfaction program with iterative forward search and a hill climbing heuristic, applied their model at Purdue University, with the possibility of it being utilized by other American universities [11]. In other real-life applications, constraint programming (CP) and integer programming appear to be the popular modeling techniques of choice [1,2,4,9], likely due to the flexibility these methods have in expressing constraints and the acceptable solutions provided. However, to further complicate the problem, many institutions have unique preferences, and therefore their scheduling needs are beyond the scope of the models that solve the exam scheduling problem for other universities and the benchmark datasets used in the ITCs. Therefore, even though there is an abundance of software within the academic community, institutions may be forced to continue producing schedules with rigid algorithms that must then be post-processed and fixed manually to meet complex scheduling requirements.

One of the complexities of the exam scheduling problem is that in most cases, for each exam, at least two assignments need to be made, a timeslot assignment and a room assignment, where the timeslot assigned to an exam influences the rooms available for that exam. Some authors have attempted to address this by decomposing the problem into different stages, typically a timeslot assignment phase and room assignment phase, though other decompositions have also been introduced [2,9]. This decoupling becomes increasingly complicated when multiple exams are allowed in one room. In this work we also explore decoupling the problem into timeslot and room phases. We introduce a necessary condition for the timeslot assignment model, where we generate cuts to direct the model towards producing feasible timeslot assignment solutions that will result in feasible room assignments in the next phase. These necessary conditions are sufficient for the case when each room can be assigned at most one exam at any given time.

The remainder of the paper is organized as follows. In Section 2, we provide a generic specification of the exam scheduling problem and the requirements that can be imposed on a solution. Here we introduce the student hardship requirement, which to the best of our knowledge, has not until now been given a formal specification in the literature. In Section 3.1, we describe the corresponding CP implementation of the generic specification for the timeslot assignment model. Here we describe the new necessary condition on the timeslot assignment that assists in finding a feasible room assignment. In Section 3.2, the CP requirements for the room assignment model are explained. Finally, in Section 4, we discuss the results from the experiments conducted with data from the Okanagan campus of the University of British Columbia (UBC), where we specifically focus on reducing student hardships of three types, and demonstrate the validity of the two phase model.

2 Generic Specification of the Exam Scheduling Problem

Let $\mathbb{Z}^+$ denote the set of nonnegative integers and $\mathbb{Z}^{++} = \mathbb{Z}^+ \setminus \{0\}$. All intervals throughout will be subsets of $\mathbb{Z}^+$ and for $k_1, k_2 \in \mathbb{Z}^+$, $k_1 \leq k_2$, denote the (integer) interval $[k_1, k_2] = \{k_1, \ldots, k_2\}$, and define $[k_2] = [1, k_2]$. For an interval $[a, b]$, define the *length* of $[a, b]$ to be $\ell([a, b]) = b - a$ if $b > a$, and $\ell([a, b]) = \ell(\emptyset) = 0$ otherwise. Moreover, for a finite monotonically increasing sequence of integers $r_0, \ldots, r_q$, define $r(i, j)$ to be the interval $[r_i + 1, r_j]$ for $0 \leq i < j \leq q$.

The *exam scheduling problem* is defined as the following: given a finite set of pairwise disjoint[1] integer intervals $T = \{[s_1, f_1], \ldots, [s_m, f_m]\}$ called the *timeslots*, a set P called the *persons*, a set $\mathcal{E}$ of subsets of P called the *exams*, a function $\varepsilon : \mathcal{E} \to \mathbb{Z}^{++}$ called the *exam sizes*, a function $\delta : \mathcal{E} \to \mathbb{Z}^{++}$ called the *exam durations*, a set $\mathcal{R}$ called the *rooms*, and a function $\sigma : T \times \mathcal{R} \to \mathbb{Z}^+$ called the *temporal room sizes*, find an assignment $\tau : \mathcal{E} \to T$ called the *timeslot assignment*, and an assignment $\rho : \mathcal{E} \to \mathcal{R}$ called the *room assignment*, such that for all $E, E' \in \mathcal{E}$, $E \neq E'$, the following four requirements are satisfied.

Requirement 1 (Person Single-Tasking) *$\tau(E) \neq \tau(E')$ if $E \cap E' \neq \emptyset$.*

Requirement 2 (Exam Duration) *$\delta(E) \leq \ell(\tau(E))$.*

Requirement 3 (Room Single-Tasking) *$\rho(E) \neq \rho(E')$ if $\tau(E) = \tau(E')$.*

Requirement 4 (Room Size) *$\varepsilon(E) \leq \sigma(\tau(E), \rho(E))$.*

Requirement 1 ensures that no person (including the invigilator) has two exams assigned to the same timeslot (see Requirement 6 for an extension to this requirement for the case of overlapping timeslots). Requirement 2 restricts the duration of an exam to not exceed the length of its assigned timeslot. Moreover, Requirement 3 ensures that no room is assigned two different exams during a timeslot (see Requirement 7 for a relaxation of this requirement). Finally, Requirement 4 ensures that the room assigned to an exam during the assigned timeslot can accommodate the exam's size requirement[2] (see Requirement 8 for a relaxation of this requirement).

In this specification, there is no distinction in the set of persons between students and instructors as they both are present during their exam. In the case of hardships, however, the students and instructors are separated as necessary (see Section 2.2). Moreover, for each person $p \in P$ and exam $E \in \mathcal{E}$, if $p \in E$, then we say p *writes* E, or E is *written* by p, even though p may be an instructor.

We will refer to $[s_k, f_k] \in T$ as timeslot k. As notational convenience, let $\sigma_k(R) = \sigma([s_k, f_k], R)$ for each timeslot k. The temporal room sizes function allows one to ensure that a room R does not use a timeslot k by specifying $\sigma_k(R) = 0$. This may be necessary if a room is closed during the timeslot, or to

[1] The pairwise disjoint restriction is relaxed in Requirement 6. Moreover, there is no loss of generality for exam scheduling in assuming that time can be discretized.

[2] Typically, for every exam E, $\varepsilon(E) = |E| - 1$.

limit timeslots that overlap for the room (see Requirements 7 and 8). Moreover, for each room $R \in \mathcal{R}$, let $T(R) = \{[s_k, f_k] \in T : \sigma_k(R) > 0\}$ be called the *effective* timeslots of room R.

A nonempty collection $\mathcal{A}$ of subsets of a set A is called *intersecting* if for all $A_1, A_2 \in \mathcal{A}$, $A_1 \cap A_2 \neq \emptyset$. Using the maximal intersecting subsets of exams for Requirement 1 is desired but finding them is computationally impractical (i.e., NP-hard). Instead, there are some natural intersecting subsets that can be used. For each person $p \in P$, let the intersecting subset of exams that p writes be denoted by

$$\mathcal{E}(p) = \{E \in \mathcal{E} : p \in E\}.$$

Moreover, the maximal intersecting subsets of timeslots – we will call this collection $\mathcal{B}^*$ – is the set of maximal cliques of the corresponding interval graph which can be found in linear time.

We will assume a fixed τ and ρ in the definitions that follow.

2.1 Extending the Exam Scheduling Problem

The exam scheduling problem can be extended to include a variety of other constraints from real-world contexts.

The first of these extensions further limits τ and ρ for particular situations.

Requirement 5 (Time and Room Specific) *An exam $E \in \mathcal{E}$ can be forced to be assigned a timeslot from a subset B of timeslots by the additional requirement $\tau(E) \in B$. Moreover, the exam can be forced to be assigned a room from a subset $\mathcal{Q}$ of rooms by the additional requirement $\rho(E) \in \mathcal{Q}$.*

In certain cases of modeling the exam scheduling problem, it may be necessary to allow the timeslots of the problem to overlap (i.e., not be pairwise disjoint). This occurs most commonly when modeling exams that are allowed varying lengths.

Requirement 6 (Overlapping Timeslots) *If the timeslots are allowed to overlap, then Requirement 1 is replaced with:*

$$\tau(E) \cap \tau(E') = \emptyset \text{ if } E \cap E' \neq \emptyset.$$

Moreover, Requirement 3 is replaced with:

$$\rho(E) \neq \rho(E') \text{ if } \tau(E) \cap \tau(E') \neq \emptyset.$$

Note that Requirement 6 generalizes Requirement 1 as non-overlapping timeslots satisfy $\tau(E) \cap \tau(E') = \emptyset$ if and only if $\tau(E) \neq \tau(E')$.

The third of these extensions allows for the use of a large room such as a gym to host several exams at once. For each $R \in \mathcal{R}$ and timeslot k, let

$$\mathcal{C}_{R,k} = \{E \in \mathcal{E} : \rho(E) = R, \tau(E) = [s_k, f_k]\}$$

be called the *concurrent* exams in room R during timeslot k.

If more than one exam is allowed in a room during a given time, then Requirement 3 is either removed entirely or it is replaced with the following requirement to limit the number of concurrent exams assigned to the room.

Requirement 7 (Room Task Limit) *During a timeslot k, and for a room $R \in \mathcal{R}$, to limit the room to be assigned at most $\gamma_k(R)$ concurrent exams, the following is required:*

$$|\mathcal{C}_{R,k}| \le \gamma_k(R).$$

Concurrent exams pose logistical challenges if timeslots overlap. In practice, concurrent exams assigned to a room can start at the same time but may finish at different times. These exams would be assigned different overlapping timeslots where one timeslot is contained in the other. Such a situation can be handled by using the larger timeslot for both exams. This is sufficient since it is not the case, in practice, that an exam would start in the middle of a timeslot of another exam in the same room and, say, end later. Thus, in order to simplify the model without losing applicability, we can effectively model concurrent exams for rooms that allow multiple tasks by limiting these rooms to have non-overlapping timeslots when they multi-task. More formally, this limitation is for all $R \in \mathcal{R}$ and all $[s_k, f_k], [s_{k'}, f_{k'}] \in T(R)$ with $k' \ne k$, if $\gamma_k(R) > 1$ and $\gamma_{k'}(R) > 1$, $[s_k, f_k] \cap [s_{k'}, f_{k'}] = \emptyset$. If $\gamma_k(R) = 1$, then other timeslots can intersect $[s_k, f_k]$.

With more than one exam allowed in a room at a given time, Requirement 4 is replaced with the following requirement.

Requirement 8 (Room Multitasking Size) *For a room $R \in \mathcal{R}$ to host concurrent exams, for every timeslot k,*

$$\sum_{E \in \mathcal{C}_{R,k}} \varepsilon(E) \le \sigma_k(R).$$

Requirement 9 (Coupled Exams) *To require two exams E and E' to be written*

1. *during the same timeslot, then require $\tau(E) = \tau(E')$, or*
2. *in the same room, then require $\rho(E) = \rho(E')$.*

Note that Requirement 9.2 only makes sense if we have Requirement 8 as well.

2.2 Measuring Hardships

We focus here on measuring, and later minimizing, the number of times persons have a certain number of examinations assigned to timeslots that are within a certain amount of time.

Let $d \in \mathbb{Z}^+$ represent a length of time. We first start by collecting the timeslots of T that are within d time units of each other as measured by the difference of the latest finish time with the earliest start time of its members. This set is defined by:

$$\mathcal{B}_d = \bigcup_{i=1}^{k} \{B \subseteq T : [s_j, f_j] \in B \text{ where } s_j \geq s_i \text{ and } f_j \leq s_i + d\}\,.$$

For a given person $p \in P$, the exams written by person p during a $B \in \mathcal{B}_d$ is given by

$$\mathcal{W}_{p,B} = \{E \in \mathcal{E}(p) : \tau(E) \in B\}.$$

For a given positive integer w, representing a minimum number of writes, the set of (w, d)-hardships (of τ) is defined to be

$$\mathcal{H}_{w,d} = \{(p, B) : p \in P, B \in \mathcal{B}_d, |\mathcal{W}_{p,B}| \geq w\}.$$

Thus $|\mathcal{H}_{w,d}|$ is the number of times persons are in at least w exams that are assigned by τ to be within any d time units of the schedule.[3]

Example 1. Three Exams in 27 Hours Hardships

If a student at UBC writes three exams that span at most 27 hours from the start of the first exam to the end of the last exam, then a *3-in-27-hours* hardship occurs and the student has the right to request another time to write one of the exams. This creates many issues ranging from exam calibration and fairness to exam security. It also puts an extra demand on the administrative and faculty resources. Exam schedules having zero 3-in-27-hours hardships are thus clearly valued.

UBC's examination period spans 12 days and there are four timeslots per day: 8:30 a.m., 12:00 p.m., 3:30 p.m., and 7:00 p.m. However a Sunday only has two timeslots: 12:00 p.m., and 3:30 p.m. Each exam is assumed to be two and a half hours long. Discretizing time so that 8:30 a.m. on the first day is at time $17 = 8.5 \times 2$, and is a Monday, the set of timeslots T have first week time intervals Monday: $[17, 22]$, $[24, 29]$, $[31, 36]$, $[38, 43]$; Tuesday: $[65, 70], \ldots$; Sunday: $[312, 317]$, $[319, 324]$; along with second week time intervals $[353, 358], \ldots, [566, 571]$. To measure those students having a hardship of 3 exams in 27 hours, the required timeslot sets are represented by $\mathcal{B}_{54}$ (half hour increments). See Table 1 for some examples.

Thus to minimize the number of persons having a 3-in-27-hours hardship, τ is chosen so as to minimize $|\mathcal{H}_{3,54}|$. It is also possible to restrict that no persons have such a hardship by constraining $|\mathcal{H}_{3,54}|$ to be zero. □

Requirement 10 (Hardships) *For a positive integer w, representing a number of writes, and nonnegative real number d, representing a length of time, to ensure that no persons have w examinations assigned to timeslots that are within time d (i.e., no (w, d)-hardships), then require:*

$$|\mathcal{H}_{w,d}| = 0.$$

[3] The set of persons P can be reduced here to just include students or just instructors depending on the application of the hardship.

Table 1. Example Timeslot Sets of $\mathcal{B}_{54}$ for 3-in-27-hours Hardships

Day	Timeslot Sets Involving Day
First Monday	$\{[17,22],[24,29],[31,36],[38,43],[65,70]\}$,
	$\{[24,29],[31,36],[38,43],[65,70],[72,77]\}$,
	$\{[31,36],[38,43],[65,70],[72,77],[79,84]\}$,
	$\{[38,43],[65,70],[72,77],[79,84],[86,91]\}$
First Sunday	$\{[264,269],[271,276],[278,283],[312,317]\}$,
	$\{[271,276],[278,283],[312,317],[319,324]\}$,
	$\{[278,283],[312,317],[319,324]\}$,
	$\{[312,317],[319,324],[353,358]\}$,
	$\{[319,324],[353,358],[360,365],[367,372]\}$
Last Friday	$\{[497,502],[504,509],[511,516],[518,523],[545,550]\}$,
	$\{[504,509],[511,516],[518,523],[545,550],[552,557]\}$
	$\{[511,516],[518,523],[545,550],[552,557],[559,564]\}$
	$\{[518,523],[545,550],[552,557],[559,564],[566,571]\}$

To ensure that these type of hardships are minimized, then include $|\mathcal{H}_{w,d}|$ *as a term in the minimizing objective function of the model.*

A special class of hardship are *back-to-back* exams. These hardships occur when a person writes two exams in two consecutive timeslots on the same day. Using the timeslots from Example 1, the back-to-back hardships are represented by $\mathcal{H}_{2,12}$. Given the regular nature of the timeslots of the example, a more efficient implementation is outlined at the end of the section entitled CP of Requirement 10.

3 CP Implementation of the Generic Specification

In what follows, we partition our CP implementation of the exam scheduling problem into a *timeslot assignment subproblem* and into one *room assignment subproblem* for each timeslot. The timeslot assignment subproblem is solved first and the result τ is used as input for each of the room assignment subproblems. The room assignment subproblem for timeslot k assigns rooms to only those exams assigned to timeslot k by τ. We say τ is *room-assignable* if the assignment has a feasible room assignment subproblem solution for each timeslot. The timeslot assignment subproblem contains constraints that are necessary and, in all but one scenario, are also sufficient (see Theorem 1) to ensure that all of its feasible solutions are also room-assignable. The scenario that cannot guarantee sufficiency occurs when Requirement 8 is specified. In this case, the timeslot assignment subproblem constraints are only necessary, although in practice do well in finding room-assignable solutions. An extension to the CP implementation of the timeslot assignment subproblem is also described in Section 3.3 that ensures all feasible solutions are room-assignable even with Requirement 8. However, it may not be possible to use this extension in practice, depending on the size of the input, as discussed in the Section 3.4. In order to highlight these performance considerations in the discussion, the sizes are calculated in this section for some of the relevant implementation options.

Throughout this section, we will use catalog of Beldiceanu, Carlsson and Rampon [5] as the source for definitions of known constraint programming global constraints. To facilitate the description of the implementation, the names of constraint programming variables will use a bold typeface.

3.1 Timeslot Assignment Subproblem

The timeslot assignment subproblem finds a timeslot assignment τ that ensures the implicit existence of a feasible room assignment ρ without actually determining ρ. In this section, we describe a CP implementation of the generic specification by first defining the decision variables and then defining the constraints for each of the requirements.

Primary Decision Variables In order to find a τ satisfying the requirements, for each $E \in \mathcal{E}$, we define a constraint programming integer-valued decision variable $\boldsymbol{t}_E$ whose domain is the set of indices of the timeslots, $[m] = \{1, \ldots, m\}$, with the understanding (to be encoded by the constraints) that if $\boldsymbol{t}_E$ is bound to k, then $\tau(E) = [s_k, f_k]$. For any subset $\mathcal{D}$ of exams, we let $\boldsymbol{T}_{\mathcal{D}} = \{\boldsymbol{t}_E : E \in \mathcal{D}\}$ be the corresponding set of decision variables.

CP of Requirement 1 (PERSON SINGLE-TASKING)**.**
Requirement 1 can be restated as follows: each intersecting subset of exams of size two must have the timeslot assignments of the exams different from each other. When this requirement is applied to any intersecting subset $\mathcal{I} \subseteq \mathcal{E}$ of exams, the timeslot assignments of any pair of exams from $\mathcal{I}$ must be different from each other, and so all of the timeslot assignments of exams from $\mathcal{I}$ must be different from each other. Thus we impose the global constraint `alldifferent`$(\boldsymbol{T}_{\mathcal{I}})$ (see [5, p. 434]) to implement Requirement 1 for all pairs of exams in $\mathcal{I}$.

The larger the size of the intersecting subset of exams, the greater the possible propagation power of the `alldifferent` constraint. That being said, using the maximal intersecting subsets of exams would be ideal but it can be computational too expensive to find such subsets. If such sets are not known, then it can be effective to use $\mathcal{E}(p)$ for each person $p \in P$. The key is that for all $E, E' \in \mathcal{E}$, $E \neq E'$ if $\{E, E'\}$ is intersecting, then $E \cap E'$ is a subset of at least one of the intersecting subsets used.

CP of Requirement 2 (Exam Duration).
The duration requirement for an $E \in \mathcal{E}$ is implemented by restricting the initial domain of $\boldsymbol{t}_E$ directly by removing each k such that $\delta(E) > \ell([s_k, f_k])$ (or indirectly and equivalently, by using the constraint $\boldsymbol{t}_E \neq k$).

CP of Requirements 3 and 4 (Room Single-Tasking) and (Room Size).
In order to ensure that τ is chosen to implicitly satisfy Requirements 3 and 4, we consider a sequence of constraints that are parameterized by a positive integer c which will ensure the necessary but not sufficient statement that the number of exams of size larger than c assigned to a timeslot is at most the number of rooms of size larger than c during the timeslot.

To implement this idea, define $\mathcal{E}_c^> = \{E \in \mathcal{E} : \varepsilon(E) > c\}$ and for $k \in [m]$, define $\mathcal{R}_{c,k}^> = \{R \in \mathcal{R} : \sigma_k(R) > c\}$. The format of the restrictions are provided by the set of triples $F_c = \{(k, 0, |\mathcal{R}_{c,k}^>|) : k \in [m]\}$ for the global constraint `global_cardinality_low_up`$(\boldsymbol{T}_{\mathcal{E}_c^>}, F_c)$ (see [5, p. 1040]). This constraint ensures that, for each k, the number of decision variables $\boldsymbol{t}_E$ where $E \in \mathcal{E}_c^>$ and that are bound to k is between 0 and $|\mathcal{R}_{c,k}^>|$. In other words, the number of exams E with $\varepsilon(E) > c$ that are assigned by τ to timeslot k is at most the number of rooms that have size larger than c throughout the timeslot.

A timeslot assignment τ satisfying the `global_cardinality_low_up` constraint for a fixed minimum size c does not guarantee that there is a feasible room assignment ρ. But a sequence of these `global_cardinality_low_up` constraints does. Let $[s_k, f_k]$ be a fixed timeslot. For this timeslot, we now consider the number of distinct room sizes and let $q = q_k$ denote this number. Define $r_0 = 0$ and consider the (non-multi-)set of room sizes $\{\sigma_k(R) : R \in \mathcal{R}_{0,k}^>\} = \{r_1, \ldots, r_q\}$ where the r_i's are labeled so that they are strictly increasing. Note that for an exam scheduling problem to be feasible, $\mathcal{E}_{r_q}^> = \emptyset$. The CP constraints for Requirements 3–4 are then

$$\texttt{global\_cardinality\_low\_up}(\boldsymbol{T}_{\mathcal{E}_{r_i}^>}, F_{r_i}) \quad \text{for } i \in [0, q-1]. \qquad \text{(Constraints 3–4)}$$

As described above, Constraints 3–4 must be necessarily satisfied by any feasible solution of the room assignment problem for timeslot k. The next theorem shows that they are also sufficient.

Theorem 1. *A feasible solution of the timeslot assignment problem with Requirement 3–4 is room-assignable if and only if the solution satisfies Constraints 3–4.*

Proof. Any feasible solution of the timeslot assignment problem with Requirements 3–4 that is room-assignable must satisfy Constraints 3–4 given the definition of the constraints described above.

On the other hand, consider a feasible solution τ to the timeslot assignment problem with Constraints 3–4 satisfied. We show that this solution is room-assignable for an arbitrary but fixed timeslot $[s_k, f_k]$. Let $q = q_k$, the number of distinct room sizes of the timeslot. For each $i \in \{0, ..., q-1\}$, let $S(i)$ be the statement that all exams assigned to $[s_k, f_k]$ by τ that have size larger than r_i can be assigned rooms in timeslot $[s_k, f_k]$. Note that all of the rooms assigned for these exams must have size larger than r_i.

Fix $k \in [m]$ and for $i \in \{0, ..., q-1\}$, let $K_i = \{E \in \mathcal{E}^{>}_{r_i} : \tau(E) = [s_k, f_k]\}$. With $i > 0$ note that, $K_i \subseteq K_{i-1}$, and, as well that $K_{i-1} \setminus K_i$ is precisely the set of those exams that are assigned to the timeslot with size in the range $r(i-1, i]$ (i.e., size larger than r_{i-1} but at most r_i). Moreover,

$$|K_i| = |\{\boldsymbol{t}_E \in \boldsymbol{T}_{\mathcal{E}^{>}_{r_i}} : \boldsymbol{t}_E \text{ is bound to } k\}|$$

since $E \in K_i$ if and only if $E \in \mathcal{E}^{>}_{r_i}$ and $\tau(E) = [s_k, f_k]$, if and only if decision variable $\boldsymbol{t}_E \in \boldsymbol{T}_{\mathcal{E}^{>}_{r_i}}$ is bound to k.

Rephrased, $S(i)$ is the statement that the exams of K_i can be assigned to rooms in the timeslot. The following proves that $S(0)$ is true by showing $S(q-1)$ is true and then inducting backwards to zero by proving that $S(i)$ implies $S(i-1)$ for all $i \in [q-1]$.

$S(q-1)$ is true. The constraint `global_cardinality_low_up`$(\boldsymbol{T}_{\mathcal{E}^{>}_{r_{q-1}}}, F_{r_{q-1}})$ ensures that $|\{\boldsymbol{t}_E \in \boldsymbol{T}_{\mathcal{E}^{>}_{r_{q-1}}} : \boldsymbol{t}_E \text{ is bound to } k\}| \leq |\mathcal{R}^{>}_{r_{q-1},k}|$. Thus $|K_{q-1}| \leq |\mathcal{R}^{>}_{r_{q-1},k}|$ and hence there are enough rooms of size more than r_{q-1}, that is those of size r_q, so that each of the $|K_{q-1}|$ exams can be assigned to a room of size at most r_q, one exam to a room, during the timeslot.

$S(i)$ implies $S(i-1)$. Suppose, for some $i \in [q-1]$, $S(i)$ is true. Thus the exams in K_i can be assigned to rooms during the timeslot. None of these exams can be assigned to rooms whose sizes are in the range $r(i-1, i]$ since they all have size larger than r_i. Assign each of these exams to their own room. There are then $|\mathcal{R}^{>}_{r_{i-1},k}| - |K_i|$ remaining rooms without assigned exams for the timeslot that have size larger than r_{i-1}. The set of remaining exams to be assigned for the timeslot that have size larger than r_{i-1} is $K_{i-1} \setminus K_i$. Moreover, the constraint `global_cardinality_low_up`$(\boldsymbol{T}_{\mathcal{E}^{>}_{r_{i-1}}}, F_{r_{i-1}})$ ensures that $|K_{i-1}| \leq |\mathcal{R}^{>}_{r_{i-1},k}|$ which in turn implies

$$|K_{i-1} \setminus K_i| = |K_{i-1}| - |K_i| \leq |\mathcal{R}^{>}_{r_{i-1},k}| - |K_i|.$$

Thus there are enough unassigned rooms of size larger than r_{i-1} (and hence size at least r_i) so the exams with sizes in $(r_{i-1}, r_i]$ in K_{i-1} can be assigned to their own room. Thus $S(i-1)$ is true.

By induction, we have that $S(0)$ is true and hence all of the exams assigned to the timeslot under τ have been assigned a room during the timeslot. □

From a given timeslot assignment τ satisfying these constraints, a greedy algorithm can then be used to assign rooms of a given timeslot. This is done by first ordering the rooms from largest to smallest in a list as well as ordering the exams largest to smallest in a separate list. Then repeat the following: assign the largest room to the largest exam, and delete each from their respective lists.

Typically, the number of rooms of a small size is much larger than the number of rooms of a larger size (e.g., in most universities there is only one gym). Moreover, many small size rooms do not get assigned an exam for a given timeslot. For these reasons, the room assignment problem is typically difficult for the larger sized exams. Thus in practice, not all minimum sizes need be considered above to ensure a feasible room assignment.

CP of Requirement 5 (Time and Room Specific).
Requirement 5 is straightforward to implement. For the timeslot assignment part, if $E \in \mathcal{E}$ and $B \subseteq T$, the requirement $\tau(E) \in B$ is implemented by reducing the domain of $\boldsymbol{t}_E$ to be $\{k : [s_k, f_k] \in B\}$ using not equal constraints or other constraint programming primitives.

CP of Requirement 6 (Overlapping Timeslots).
In order to extend Requirement 1 for overlapping timeslots, the `among_low_up` global constraint is used (see [5, p. 494]). For each $B \in \mathcal{B}^*$, let $V_B = \{k : [s_k, f_k] \in B\}$. Using an intersecting subset $\mathcal{I} \subseteq \mathcal{E}$ of exams, the `among_low_up`$(0, 1, \boldsymbol{T}_\mathcal{I}, V_B)$ constraint is used to require that at most one variable from $\boldsymbol{T}_\mathcal{I}$ can take a value from V_B. This will force that at most one timeslot from any two timeslots that overlap be chosen for each pair of exams from $\mathcal{I}$. See Requirement 1 for the discussion regarding which intersecting sets of exams to use.

Note that when the timeslots are pairwise disjoint, $|B| = 1$ and hence $|V_B| = 1$. For an intersecting subset $\mathcal{I}$ of exams, the constraint for this case, `among_low_up`$(0, 1, \boldsymbol{T}_\mathcal{I}, V_B)$, is logically equivalent to `alldifferent`$(\boldsymbol{T}_\mathcal{I})$. On the other hand, with overlapping timeslots, `alldifferent`$(\boldsymbol{T}_\mathcal{I})$ is weaker logically than `among_low_up`$(0, 1, \boldsymbol{T}_\mathcal{I}, V_B)$. Thus these `alldifferent` constraints should not be removed given the possibility that their inclusion could provide some additional propagative usefulness.

The room assignment restriction for Requirement 6 is discussed in Section 3.2.

CP of Requirement 7 (Room Task Limit).
We treat Requirement 7 in a similar way as we have for Requirements 3 and 4. In fact, Requirement 7 is a generalization of those requirements. Let $k \in [m]$, $q = q_k$, and $j \in [0, q-1]$. The maximum number of concurrent exams of size greater than r_j that can be scheduled during timeslot $[s_k, f_k]$ is given by

$$\gamma^{>}_{j,k} = \sum_{R \in \mathcal{R}^{>}_{r_j,k}} \gamma_k(R).$$

Note if Requirements 3 and 4 are in force, then $\gamma_k(R) = 1$ for all $R \in \mathcal{R}^{>}_{r_j,k}$, and hence $\gamma^{>}_{j,k} = \left|\mathcal{R}^{>}_{r_j,k}\right|$.

For a positive integer u, let $\mathcal{E}_u = \{E \in \mathcal{E} : \varepsilon(E) = u\}$ be the set of exams of size u. Consider the (non-multi)-set of exam sizes $U = \{\varepsilon(E) : E \in \mathcal{E}\}$. For each exam size $u \in U$ and each $k \in [m]$, introduce a CP variable $\boldsymbol{n}_{u,k}$ that is constrained to count the number of exams of size u assigned to timeslot $[s_k, f_k]$. By defining $F'_u = \{(k, \boldsymbol{n}_{u,k}) : k \in [m]\}$, the CP global constraint $\texttt{global\_cardinality}(\boldsymbol{T}_{\mathcal{E}_u}, F'_u)$ (see [5, p. 1034]) ensures $\boldsymbol{n}_{u,k}$ will equal the desired count.

Like Requirements 3 and 4, we need to use a sequence of constraints in order that a timeslot assignment with this requirement guarantees that there is a corresponding feasible room assignment ρ. For each $i \in [q]$ and each $k \in [m]$, use the standard global constraint representing the following sum of integer CP linear terms that counts the number of exams of sizes in $r(i-1, i]$ that are assigned to timeslot k:

$$\boldsymbol{n}^*_{i,k} = \sum_{\substack{u \in U \\ u \in r(i-1,i]}} \boldsymbol{n}_{u,k}.$$

Moreover, for $j \in [0, q-1]$,

$$\boldsymbol{n}^>_{j,k} = \sum_{i=j+1}^{q} \boldsymbol{n}^*_{i,k}$$

counts the number of exams of sizes larger than r_j that are assigned to the timeslot. Note that $\boldsymbol{n}^*_{q,k} = \boldsymbol{n}^>_{q-1,k}$ and if $i < q$, then $\boldsymbol{n}^*_{i,k} = \boldsymbol{n}^>_{i-1,k} - \boldsymbol{n}^>_{i,k}$. The CP constraints for Requirement 7 are then,

$$\boldsymbol{n}^>_{j,k} \le \gamma^>_{j,k} \quad \text{for } j \in [0, q-1] \text{ and } k \in [m]. \qquad \text{(Constraints 7)}$$

Constraints 7 must be necessarily satisfied by any feasible solution of the room assignment problem for timeslot k. They need to be combined with the constraints of Requirement 8 in order that a room with concurrent exams has enough space for the assigned examinees. In order to implement the mq constraints of Constraints 7, at most $m|U|$ new CP variables are required along with their defining $m|U|$ global constraints.

CP of Requirement 8 (Room Multitasking Size).
Let timeslot $[s_k, f_k]$ be fixed and $q = q_k$ in the following discussion. Continuing with the description of the CP implementation of Requirement 7, for a given room size r_i, a necessary (linear) constraint for feasibility when concurrent exams are allowed is:

$$\sum_{\substack{u \in U \\ u > r_i}} u \boldsymbol{n}_{u,k} \le \sum_{R \in \mathcal{R}^>_{r_i,k}} \sigma_k(R).$$

The left hand side represents the sum of the sizes of those exams of size larger than r_i scheduled in the timeslot, whereas the right hand side represents the sum of the sizes of the rooms larger than r_i. This constraint is not sufficient as it is possible that a collection of exams satisfies the constraint by collectively having enough room space but without a way to partition the rooms for the exams. For example, four exams of size 60 cannot fit into three rooms of size 80 even though the constraint is satisfied. To be precise, for each potential room size r_i and exam size u, let $\pi_k(r_i, u)$ be the maximum number of exams of size u that can be concurrent in rooms of size r_i during the timeslot:

$$\pi_k(r_i, u) = \sum_{\substack{R \in \mathcal{R} \\ \sigma_k(R) = r_i}} \left\lfloor \frac{r_i}{u} \right\rfloor .$$

The previous example provides the constraint that at most $\pi_k(80, 60) = 3$ exams of size 60 can be hosted by rooms of size 80. Note that $\pi_k(r_i, 1)$ is just the sum of the sizes of rooms of size r_i that can be assigned exams during the timeslot.

Let $u_q^{\triangledown}$ be the minimum size of the exams of $\mathcal{E}^{>}_{r_{q-1}}$.[4] The first necessary condition of Requirement 8 for a feasible solution of the timeslot assignment problem to be feasible for the room assignment problem requires that the number of exams assigned to the timeslot that are forced to be assigned to the largest size rooms must not exceed the rooms' collective capacity:

$$\boldsymbol{n}^*_{q,k} \leq \pi_k(r_q, u_q^{\triangledown}).$$

Continuing now with smaller sized rooms, without any assumption of room assignments for exams, one can at least represent the cumulative remaining space (residual capacity) of the size of rooms of a given minimum room and exam size for use in other necessary conditions. To this end, for $i \in [q]$, we define the CP variable $\boldsymbol{c}_{i,k}$ as:

$$\boldsymbol{c}_{i,k} = \sum_{R \in \mathcal{R}^{>}_{r_{i-1},k}} \sigma_k(R) - \sum_{\substack{u \in U \\ u > r_{i-1}}} u \boldsymbol{n}_{u,k}.$$

Consider all the exams of $\mathcal{E}^{>}_{r_{q-2}} \setminus \mathcal{E}^{>}_{r_{q-1}} = \{E \in \mathcal{E} : \varepsilon(E) \in (r_{q-2}, r_{q-1}]\}$ and $u_{q-1}^{\triangledown}$ to be the minimum size of these exams. Exams from this set must be assigned to rooms of size r_{q-1} or of size r_q. None of these exams of size larger than r_{q-1} can be assigned to rooms of size r_{q-1} so they must be assigned to rooms of size r_q. There are thus at most $\pi_k(r_{q-1}, u_{q-1}^{\triangledown})$ of these exams assigned to rooms of size r_{q-1}. The residual capacity for exams of size larger than r_{q-1} in rooms of size r_q is given by $\boldsymbol{c}_{q,k}$. Thus these rooms can be assigned to at most $\left\lfloor \frac{\boldsymbol{c}_{q,k}}{u_{q-1}^{\triangledown}} \right\rfloor$ of exams of size at most r_{q-1}. Therefore, a necessary condition for the number of exams of size in the range $r(q-2, q-1]$ assigned to the timeslot is:

$$\boldsymbol{n}^*_{q-1,k} \leq \pi_k(r_{q-1}, u_{q-1}^{\triangledown}) + \left\lfloor \frac{\boldsymbol{c}_{q,k}}{u_{q-1}^{\triangledown}} \right\rfloor .$$

[4] We may suppose the $\mathcal{E}^{>}_{r_{q-1}}$ is nonempty since otherwise we will just skip this step and not produce a constraint for $i = q-1$. This is also true for the remaining discussion.

For $i \in [q-2]$, let $u_i^{\triangledown}$ be the minimum size of the exams of $\mathcal{E}^{>}_{r_{i-1}} \setminus \mathcal{E}^{>}_{r_i}$. Now for such a fixed i, the exams from $\mathcal{E}^{>}_{r_{i-1}} \setminus \mathcal{E}^{>}_{r_i}$ must be assigned to rooms whose sizes are from $r_i, \ldots, r_q$. No exams of size larger than r_i can be assigned in rooms of size at most r_i. Using the same reasoning as in the last paragraph, rooms of size r_i and of size r_q can be assigned at most $\pi_k(r_i, u_i^{\triangledown}) + \left\lfloor \frac{\boldsymbol{c}_{q,k}}{u_i^{\triangledown}} \right\rfloor$ of these exams. For $j = [i+2, q-1]$, we wish to determine the residual capacity of the rooms of size r_j for exams whose sizes are in the range $r(j-1, j]$. Unfortunately, this number depends on a feasible room assignment. We will thus determine an upper bound, defined by $\boldsymbol{c}'_{j,k}$ in the next paragraph, of the residual capacity of rooms of size r_j from assigning exams of sizes in the range $r(j-1, j]$ for any feasible room assignment. Given this upper bound, the rooms of size r_j can be assigned at most $\left\lfloor \frac{\boldsymbol{c}'_{j,k}}{u_i^{\triangledown}} \right\rfloor$ exams during the timeslot. Thus the number of exams of sizes in the range $r(i-1, i]$ assigned to the timeslot must satisfy:

$$\boldsymbol{n}^*_{i,k} \leq \pi_k(r_i, u_i^{\triangledown}) + \sum_{j=i+1}^{q} \left\lfloor \frac{\boldsymbol{c}'_{j,k}}{u_i^{\triangledown}} \right\rfloor . \tag{1}$$

We now turn to determining an expression for $\boldsymbol{c}'_{j,k}$. Since exams of sizes in the range $(r_{q-1}, r_q]$ must be assigned to rooms of size r_q, $\boldsymbol{c}'_{q,k} = \boldsymbol{c}_{q,k}$. For $j \in [q-1]$, we determine $\boldsymbol{c}'_{j,k}$ by first considering an upper bound, $\boldsymbol{n}'_{j,k}$, of the maximum number of exams whose sizes are in the range $r(j-1, j]$ that can be assigned to rooms of sizes larger than r_j. Define

$$\boldsymbol{n}'_{j,k} = \min \left\{ \boldsymbol{n}^*_{j,k}, \left\lfloor \frac{\boldsymbol{c}_{j+1,k}}{u_j^{\triangledown}} \right\rfloor \right\} .$$

There are only $\boldsymbol{n}^*_{j,k}$ exams whose sizes are in the range $r(j-1, j]$ and assigned to the timeslot, and so those assigned to rooms larger than r_j cannot exceed this number. On the other hand, the term $\left\lfloor \frac{\boldsymbol{c}_{j+1,k}}{u_j^{\triangledown}} \right\rfloor$ is an upper bound for the maximum number of exams of sizes in the range $r(j-1, j]$ that can be assigned to rooms of sizes larger than r_j considering the residual capacity of the exams of size larger than r_j that are assigned to the timeslot. Thus $\boldsymbol{n}'_{j,k}$ is an upper bound to the stated maximum.

Define now $\boldsymbol{n}''_{j,k} = \boldsymbol{n}^*_{j,k} - \boldsymbol{n}'_{j,k}$. Then $\boldsymbol{n}''_{j,k}$ is a lower bound on the fewest number of exams of sizes in the range $r(j-1, j]$ assigned to rooms of size r_j. Thus the remaining capacity of rooms of size r_j from exams of sizes in the range $r(j-1, j]$ is at most $\boldsymbol{c}'_{j,k} = \pi_k(r_j, 1) - u_j^{\triangledown} \boldsymbol{n}''_{j,k}$.

For $i \in [q]$, and $k \in [m]$, the constraint given in (1) can be considered the most general form of a constraint of the CP implementation of Requirement 8 if we allow the empty summand to be disregarded when $i = q$ (i.e., set the summand to 0). As well, if for any i, $\mathcal{E}^{>}_{r_i} \setminus \mathcal{E}^{>}_{r_{i+1}} = \emptyset$, then the constraint is disregarded altogether for all $k \in [m]$. We label the entire collection of these constraints that includes the constraint given in (1) for each i and each k as Constraints 8. Our implementation of the mq constraints of Constraints 8 required $2mq$ new CP variables along with their defining $2mq$ constraints.[5]

Constraints 8 provide necessary constraints for a feasible solution of the timeslot assignment problem to be room-assignable, however they may not be sufficient. One of the issues that makes the constraints insufficient is that the terms $\boldsymbol{c}'_{i,k}$ depend on terms $\boldsymbol{c}_{i,k}$ that represent the residual capacity of the rooms of size at least r_i, and are thus cumulative. It is possible (as provided in the example) that they cannot be partitioned into the number of parts provided by the constraints in a way that ensures exams of the size being considered can be assigned to these rooms. Another issue is that the size of the parts for each constraint, $u_i^{\triangledown}$, does not guarantee there is enough space for the exams in $\mathcal{E}^{>}_{r_{i-1}} \setminus \mathcal{E}^{>}_{r_i}$ with sizes larger than $u_i^{\triangledown}$. To fix these issues, one can tighten the constraints but in doing so risk infeasibility of the timeslot assignment problem or, at best, risk removing feasible solutions to both problems. The first potential issue described is nontrivial to overcome. It turned out, however, to not be an issue for our target problems. The second issue can be fixed by replacing $u_i^{\triangledown}$ in the expressions with $u_i^{\triangle}$, the maximum size of the exams of $\mathcal{E}^{>}_{r_{i-1}} \setminus \mathcal{E}^{>}_{r_i}$. If this leads to infeasibility, any number between $u_i^{\triangledown}$ and $u_i^{\triangle}$ could be used (e.g., the average of the size of the exams). In practice, we used $u_i^{\triangledown}$ and achieved feasible timeslot assignment solutions that were also room-assignable.

We refer to Constraints 7–8 as the *room-cuts* as they provide redundant constraints for the number of tasks and sizes of the rooms in any solution to an exam scheduling problem.

CP of Requirement 9 (Coupled Exams)**.**
For Requirement 9.1, to require exams E and E' to have the same timeslot ($\tau(E) = \tau(E')$), a CP equality constraint for their corresponding decision variables $\boldsymbol{t}_E$ and $\boldsymbol{t}_{E'}$ is specified. The independent Requirement 9.2, requires that E and E' have the same room ($\rho(E) = \rho(E')$). As there is no corresponding decision variable in the timeslot assignment model for the room assignments, it is possible that if Requirement 9.1 is not also specified, then the timeslot assignment found may not allow for a feasible room assignment. On the other hand, a more usual requirement would be that both Requirement 9.1 and 9.2 are specified for the given pair of exams. In this case, the exams themselves could be considered as a single exam before the problem is specified (i.e., $E \cup E'$ replaces E and E'). Requirement 7 may then need to be adjusted as the two exams are now counted as one.

CP of Requirement 10 (Hardships)**.**
Given a subset $\mathcal{D}$ of exams, we wish to measure the spread of the timeslot assignments of exams from $\mathcal{D}$. Let the start and finish times of the timeslots

[5] Some of the intermediary variables described in this section simply store expressions and are thus not counted. The new variables counted here are the $\boldsymbol{c}_{i,k}$'s and the $\boldsymbol{n}'_{j,k}$'s. It may be possible to optimize this further.

be collected in $\dddot{s} = (s_1, \ldots, s_m)$ and $\dddot{f} = (f_1, \ldots, f_m)$ respectively. For each exam $E \in \mathcal{E}$, we define a new constraint programming variable $\boldsymbol{s}_E$ via the global constraint $\texttt{element}(\boldsymbol{t}_E, \dddot{s}, \boldsymbol{s}_E)$ (see [5, p. 958]) which ensures that $\boldsymbol{s}_E$ is bound to s_k if and only if $\boldsymbol{t}_E$ is bound to k (i.e., $\boldsymbol{s}_E$ is $s_{\boldsymbol{t}_E}$). Moreover, for each exam $E \in \mathcal{E}$, using $\texttt{element}(\boldsymbol{t}_E, \dddot{f}, \boldsymbol{f}_E)$ defines $\boldsymbol{f}_E$ so that $\boldsymbol{f}_E$ is bound to f_k if and only if $\boldsymbol{t}_E$ is bound to k (i.e., $\boldsymbol{f}_E$ is $f_{\boldsymbol{t}_E}$).

A new constraint programming decision variable $\boldsymbol{s}_{\mathcal{D}}^{\triangledown}$ is defined via the global constraint $\texttt{minimum}\big(\boldsymbol{s}_{\mathcal{D}}^{\triangledown}, \{\boldsymbol{s}_E : E \in \mathcal{D}\}\big)$ (see [5, p. 1378]) that represents the minimum start time of the intervals indexed by the variables assigned to exams from $\mathcal{D}$. A similar decision variable $\boldsymbol{f}_{\mathcal{D}}^{\vartriangle}$ is defined through the global constraint $\texttt{maximum}\big(\boldsymbol{f}_{\mathcal{D}}^{\vartriangle}, \{\boldsymbol{f}_E : E \in \mathcal{D}\}\big)$ that represents the maximum finishing time of the intervals indexed by the variables assigned to exams from $\mathcal{D}$ (see [5, p. 1348]). The constraint programming variable $\boldsymbol{l}_{\mathcal{D}}$, representing the length of the spread of the timeslot assignments of exams from $\mathcal{D}$, is thus the difference of decision variables $\boldsymbol{f}_{\mathcal{D}}^{\vartriangle} - \boldsymbol{s}_{\mathcal{D}}^{\triangledown}$. Moreover, when a solution τ is found from binding decision variables $\boldsymbol{t}$, $\boldsymbol{f}$, $\boldsymbol{s}$, $\boldsymbol{f}_{\mathcal{D}}$, $\boldsymbol{s}_{\mathcal{D}}$ and $\boldsymbol{l}_{\mathcal{D}}$, the variable $\boldsymbol{l}_{\mathcal{D}}$ will be bound to $\ell(I_B)$ where $B = \{\tau(E) : E \in \mathcal{D}\}$. Note that the domain of $\boldsymbol{l}_{\mathcal{D}}$ is a subset of $[0, \ell(I_T)]$.

In order to measure the number of times persons are in at least w exams that are assigned by τ to be within any d time units of the schedule (i.e., $|\mathcal{H}_{w,d}|$), we consider a subset of exams $\mathcal{D}$ such that $|\mathcal{D}| = w$ and such that $\cap\mathcal{D} \neq \emptyset$. The number of persons writing all of the w exams in $\mathcal{D}$ is $|\cap\mathcal{D}|$. Thus if $\boldsymbol{l}_{\mathcal{D}} \leq d$, then these w exams will be written within d time units and hence will contribute $|\cap\mathcal{D}|$ to $|\mathcal{H}_{w,d}|$, but 0 otherwise.

Since time is discretized, we define d^+ to be the next time unit after d. We use a combination of global constraints to map the decision variable $\boldsymbol{l}_{\mathcal{D}}$ to d if $\boldsymbol{l}_{\mathcal{D}} \leq d$, and to d^+ otherwise. The constraint programming variable $\boldsymbol{i}_{\mathcal{D},d}$ defined by $\texttt{maximum}\{d, \texttt{minimum}\{\boldsymbol{l}_{\mathcal{D}}, d^+\}\}$ encodes this understanding since if $\boldsymbol{l}_{\mathcal{D}} \leq d$, then $\texttt{minimum}\{\boldsymbol{l}_{\mathcal{D}}, d^+\}$ will have the value of $\boldsymbol{l}_{\mathcal{D}}$ and hence $\texttt{maximum}\{d, \boldsymbol{l}_{\mathcal{D}}\}$ will be d. On the other hand, if $\boldsymbol{l}_{\mathcal{D}} > d$, then $\boldsymbol{l}_{\mathcal{D}} \geq d^+$ and so $\texttt{minimum}\{\boldsymbol{l}_{\mathcal{D}}, d^+\}$ will be d^+ and hence $\texttt{maximum}\{d, d^+\}$ will also be d^+. Let $\dddot{d}$ be a $(d+1)$-tuple with the first d values as $|\cap\mathcal{D}|$ and the last value as 0. The new constraint programming variable $\boldsymbol{h}_{\mathcal{D},d}$ defined by the global constraint $\texttt{element}\big(\boldsymbol{i}_{\mathcal{D},d}, \dddot{d}, \boldsymbol{h}_{\mathcal{D},d}\big)$ will be equal to $|\cap\mathcal{D}|$ if $\boldsymbol{l}_{\mathcal{D}} \leq d$, and 0 otherwise.

Finally, $|\mathcal{H}_{w,d}|$ is represented by the CP variable $\boldsymbol{h}_{w,d}$ that is defined by the standard global constraint representing the sum of integer CP variables:

$$\sum_{\substack{\mathcal{D} \subseteq \mathcal{E} \\ |\mathcal{D}| = w}} \boldsymbol{h}_{\mathcal{D},d}.$$

Note that this sum need only be taken over those $\mathcal{D} \subseteq \mathcal{E}$ with $|\mathcal{D}| = w$ that have $\cap\mathcal{D} \neq \emptyset$ since $\boldsymbol{h}_{\mathcal{D},d}$ is zero otherwise.

Note also that this sum could have an exponential number of terms if $w \approx \frac{1}{2}|\mathcal{E}|$. This, however, does not happen in practice since for a subset of exams $\mathcal{D}$ of size w to have non-empty intersection means that some person is taking w exams and so typically this number is at most seven. Moreover, w is usually three as in Example 1 and restricting the number of this type for a variety of

d's will also restrict the hardships with $w > 3$ for larger time widths. Thus there are usually less than $|\mathcal{E}|^3$ terms in the sum. As well, the number of d's used is conventionally small given the small number of exams possible in a day and the fact that "hardship" loses its meaning with larger d's.

Adding a CP constraint that sets $\boldsymbol{h}_{w,d}$ to 0 for some w and d will ensure that no (w, d)-hardships of τ will occur in a feasible solution. On the other hand, the variables $\boldsymbol{h}_{w,d}$ can be weighted and added to a minimizing objective function if hardships are necessary for a feasible solution to be found.

Many exam scheduling problems have hardship constraints that involve each day of the schedule. Two common examples are *back-to-back* exam hardships and *2-in-1-day* exam hardships. When such constraints are required and the timeslots of the problem have a simple and regular structure, these hardships can be implemented in a more straightforward way than the above, as the following example illustrates. By adding the 8:30 a.m. and 7:00 p.m. timeslots to Sunday and setting the temporal room sizes to zero for these timeslots, the index of a timeslot in Example 1 can be used to determine the day index of the timeslot. Suppose $E_1, E_2 \in \mathcal{E}$ are such that $E_1 \cap E_2 \neq \emptyset$. Let $\mathcal{D} = \{E_1, E_2\}$ and define the boolean CP decision variable $\boldsymbol{h}'_{\mathcal{D},d}$ to represent the truth value of the following logical expression that uses the exams' corresponding timeslot variables:

$$(\boldsymbol{t}_{E_1} - 1)/4 = (\boldsymbol{t}_{E_2} - 1)/4 \quad \text{and} \quad |\boldsymbol{t}_{E_1} - \boldsymbol{t}_{E_2}| \leq d.$$

The expression uses the integer division and absolute value arithmetic functions of CP variables that are found in most CP solvers as well as the use of the truth value of a constraint arithmetically. Since there are four timeslots per day, $\boldsymbol{h}'_{\mathcal{D},d}$ indicates if E_1 and E_2 are assigned the same day index, and at the same time if they are within d timeslots from each other. Thus a back-to-back hardship occurs when $\boldsymbol{h}'_{\mathcal{D},1}$ is true, while a 2-in-1-day hardship occurs when $\boldsymbol{h}'_{\mathcal{D},3}$ is true.

For d at least one and less than the number of timeslots in a day, if $\boldsymbol{h}'_{\mathcal{D},d}$ is true, then we say a *day d-hardship* has occurred. Given $p \in P$, the number of day d-hardships occurring for person p can be represented by a CP variable $\boldsymbol{d}_{p,d}$ defined by:

$$\boldsymbol{d}_{p,d} = \sum_{\substack{\mathcal{D} \subseteq \mathcal{E}(p) \\ |\mathcal{D}|=2}} \boldsymbol{h}'_{\mathcal{D},d}.$$

The total number of day d-hardships is thus $\boldsymbol{d}_d = \sum_{p \in P} \boldsymbol{d}_{p,d}$ which can be minimized or constrained. It may be of interest as well to minimize or constrain $\boldsymbol{d}^{\triangle}_d = \max_{p \in P} \boldsymbol{d}_{p,d}$ so as to load balance the day d-hardships between all exam writers. See Section 4 for some examples of the use of these constraints.

3.2 Room Assignment Subproblem

The room assignment subproblem is relatively straightforward compared to the timeslot assignment subproblem. This is especially the case when the timeslots are not allowed to overlap, and we will make this assumption in what follows.

The necessary modifications will be discussed in the CP of Requirement 6 section if timeslots are allowed to overlap.

For each of the timeslots $[s_k, f_k]$, $k \in [m]$, the room assignment subproblem finds room assignment ρ_k using the exams assigned to $[s_k, f_k]$ by τ from a feasible solution of the timeslot assignment subproblem. In what follows, a timeslot $[s_k, f_k]$ will be considered fixed and let $\mathcal{E}_k = \tau^{-1}([s_k, f_k])$ be the set of exams assigned to the timeslot. Moreover, let the rooms of $\mathcal{R}^{>}_{0,k}$ be labeled $R_1, \dots, R_v$.

Primary Decision Variables For each $E \in \mathcal{E}_k$, we define a CP integer-valued decision variable $\boldsymbol{r}_E$ whose domain is the set of indicies of the rooms, $[v]$, with the understanding that if $\boldsymbol{r}_E$ is bound to i, then $\rho_k(E) = R_i$. For any subset $\mathcal{D} \subseteq \mathcal{E}_k$ of exams, we let $\boldsymbol{R}_{\mathcal{D}} = \{\boldsymbol{r}_E : E \in \mathcal{D}\}$ be the corresponding set of decision variables.

CP of Requirement1–2 (Person Single-Tasking and Exam Duration).
Requirements 1–2 are satisfied by a feasible τ and thus have no corresponding constraints in this section. We now focus on the remaining ones.

CP of Requirement 3 (Room Single-Tasking).
Requirement 3 forces every exam to be assigned its own room. This can be achieved by imposing the global constraint `alldifferent`$(\boldsymbol{R}_{\mathcal{E}_k}, [v])$.

CP of Requirement 4 (Room Size).
Requirement 4 ensures that the room assigned to an exam $E \in \mathcal{E}_k$ is the appropriate size. This is achieved by setting the domain of E equal to $\{i \in [v] : R_i \in \mathcal{R}^{>}_{\varepsilon(E),k}\}$.

CP of Requirement 5 (Time and Room Specific).
Requirement 5 is also straightforward to implement for the room assignment part. If $E \in \mathcal{E}_k$ and $\mathcal{Q} \subseteq \mathcal{R}$, the requirement $\rho(E) \in \mathcal{Q}$ is implemented by reducing the domain of $\boldsymbol{r}_E$ to be $\{i : R_i \in \mathcal{Q}\}$ by requiring the CP variable to be not equal to each value in $\{i : R_i \in \mathcal{R} \setminus \mathcal{Q}\}$, or by other domain-reducing constraint programming primitives.

CP of Requirement 6 (Overlapping Timeslots).
If timeslots are allowed to overlap, then the replacement of Requirement 3 with this requirement for the room assignment problem ensures that a room cannot be assigned to an exam from each of two overlapping distinct timeslots. In order to model this, several room assignment models have to be solved as a single model, or the extension to the timeslot assignment implementation should be used (see Section 3.3). To simplify the discussion, we will assume that the single model consists of all $\boldsymbol{r}_E$ for all exams $E \in \mathcal{E}_1 \cup \dots \cup \mathcal{E}_m = \mathcal{E}$. The constraints of this requirement then are $\boldsymbol{r}_E \neq \boldsymbol{r}_{E'}$, for $E \in \mathcal{E}_k$ and $E' \in \mathcal{E}_{k'}$ of all $k, k' \in [m]$, $k < k'$, with $[s_k, f_k] \cap [s_{k'}, f_{k'}] \neq \emptyset$.

CP of Requirement 7 (Room Task Limit).
The requirement is encoded in the set of triples $G_k = \{(i, 0, \gamma_k(R_i)) : i \in [v]\}$ for the global constraint `global_cardinality_low_up`$(\boldsymbol{R}_{\mathcal{E}_k}, G_k)$ (see [5, p. 1040]). This constraint ensures that, for each $i \in [v]$, the number of decision variables $\boldsymbol{r}_E$ with $E \in \mathcal{E}_k$ that are bound to i is between 0 and $\gamma_k(R_i)$. In other words, the number of exams assigned to room R_i is at most $\gamma_k(R_i)$ during the timeslot. Note that Requirement 7 has the limitation that no other timeslots overlap with $[s_k, f_k]$ and thus cannot be assigned to rooms during $[s_k, f_k]$.

CP of Requirement 8 (Room Multitasking Size).
For each $E \in \mathcal{E}_k$ and $i \in [v]$, let $\boldsymbol{b}_{E,i}$ be a CP decision variable that is one if the constraint $\boldsymbol{r}_E = i$ is true and zero otherwise. This requirement can then be modeled by the following constraints:

$$\sum_{E \in \mathcal{E}_k} \varepsilon(E) \boldsymbol{b}_{E,i} \leq \sigma_k(R_i) \text{ for each } i \in [v].$$

CP of Requirement 9 (Coupled Exams).
Requirement 9.1 is completely satisfied by a feasible solution to the timeslot assignment problem. Requirement 9.2 can be modeled for exams E, E', using the constraint $\boldsymbol{r}_E = \boldsymbol{r}_{E'}$.

3.3 Extending the Timeslot Assignment Subproblem

Depending on the size of the input data (i.e., the sizes of the sets T, $\mathcal{E}$ and $\mathcal{R}$), it may be possible to solve the timeslot assignment problem and all of the room assignment problems simultaneously through the use of the global constraint `bin_packing_capa` (see [5, p. 600]). A bin packing constraint requires items to be packed (i.e., assigned) into bins so that all items get a bin and the sum of the weights of the items of a bin does not exceed the capacity of bin. The correspondence between the exam scheduling problem and a bin packing constraint has the exams as items, the rooms as bins, and the number of examinees as the weight of an item. Since the assignment of an exam to a room depends on a timeslot, this unified model specifies a bin packing constraint for each of the timeslots. To get around the requirement of the bin packing constraint that each item must be packed into a bin, a virtual room is added to each of the constraint's bins with a new unique value, v^*, for the room's index. Additional constraints will ensure that an exam that is scheduled in timeslot k is not scheduled in the virtual room in timeslot k, and vice versa. The virtual room has unlimited capacity in all timeslots so as to allow any combination of exams to be packed into it in any particular timeslot thereby not restricting the exams to use actual rooms if they are not assigned to the timeslot.

To be more precise, the model includes, for each exam $E \in \mathcal{E}$, the previously described timeslot assignment primary decision variable $\boldsymbol{t}_E$, as well as room assignment primary decision variables $\boldsymbol{r}_{E,k}$, where $k \in [m]$ and $\boldsymbol{r}_{E,k}$ is a modified room assignment primary decision variable $\boldsymbol{r}_E$ for timeslot k. Each $\boldsymbol{r}_{E,k}$ has its domain extended to include v^*, the index of the virtual room.

If an exam E is assigned timeslot k, then the exam is not assigned the virtual room in timeslot k, and vice versa. This is modeled through the set of constraints

$$\boldsymbol{t}_E = k \text{ if and only if } \boldsymbol{r}_{E,k} \neq v^*, \text{ for all } k \in [m], \tag{2}$$

which are easily expressed in CP solvers. We store each pair of the index of a bin (i.e., room) along with its capacity (i.e., room size) in the set

$$\beta_k = \big\{\big(1, \sigma_k(R_1)\big), \ldots, \big(v, \sigma_k(R_v)\big)\big\} \cup \{(v^*, \infty)\}.$$

Moreover, each pair of a bin (i.e., room) assignment decision variable for item (i.e., exam) E at timeslot k along with the item's weight (i.e., number of examinees of E) is stored in the set $\iota_k = \{(\boldsymbol{r}_{E,k}, \varepsilon(E)) : E \in \mathcal{E}\}$. For $k \in [m]$, the constraint `bin_packing_capa`(β_k, ι_k) then ensures that all the exams get assigned to rooms in timeslot k without exceeding any room's size. Combined with (2), an exam gets assigned to an actual room in timeslot k if and only if the exam is assigned to timeslot k.

The collection of all these m global constraints along with the $m|\mathcal{E}|$ constraints from (2) thus implement Requirement 8.

The $\boldsymbol{r}_{E,k}$ variables can be reused to implement Requirement 7 in the global constraint of its CP implementation in Section 3.2. Each of the m timeslots requires a single global constraint and has at most $|\mathcal{R}|$ new CP counter variables.

We refer to this CP implementation of Requirements 7–8 as the *room-packings* constraints.

3.4 Remarks on Usage of Implementations

In this section, we discuss several possible scenarios regarding the use of the CP implementations of the requirements of the exam scheduling problem. A few factors go into deciding which scenario to use. The main factor is the sizes of the inputs to the problem, specifically the number of:

1. timeslots, m,
2. exams, $|\mathcal{E}|$,
3. different sizes of exams, $|U|$,
4. rooms, $|\mathcal{R}|$, and
5. maximum different sizes of rooms, $q = \max\limits_{k\in[m]} q_k$.

The main consideration of the implementation is whether rooms are prohibited from hosting concurrent exams (Requirement 3) or not (Requirement 8). If the rooms are prohibited from hosting concurrent exams, then Theorem 1 ensures that any feasible solution from the timeslot assignment CP implementation will be room-assignable. Thus the room-cuts and room-packings are not required.

When rooms are allowed to host concurrent exams, room-cuts or room-packings must be specified in the timeslot assignment implementation. An analysis of the sizes of the inputs to the problem might be required to determine whether to use room-cuts, room-packings or both. The following table lists the number of variables, non-global constraints, and global constraints required by each implementation option. The other constraints of the timeslot assignment implementation are not considered.

Table 2. Implementation Metrics of Model Options for Concurrent Exams

Model Option	Timeslot Assignment Implementation		
	# Variables	# Non-Global	# Global
room-cuts	$(\|U\| + 2q)m$	$(3q)m$	$\|U\|m$
room-packings	$(\|\mathcal{E}\| + \|\mathcal{R}\|)m$	$\|\mathcal{E}\|m$	$2m$

To compare different cells of Table 2, observe that $|U| \leq |\mathcal{E}|$ and $q \leq |\mathcal{R}|$. If room-cuts are used alone, then the room assignment implementation must be run for each timeslot. Also, the room-cuts are necessary but not sufficient for a feasible solution of the timeslot assignment implementation, without the room-packings, to be room-assignable. The real-world data sets we have encountered (see next section), however, have shown to always provide room assignable solutions. Moreover, if there are many rooms of small sizes to choose from, only larger rooms need be used in the room-cuts so that the number of distinct room sizes can be reduced to be much smaller than $\frac{1}{2}|\mathcal{R}|$ and $\frac{1}{3}|\mathcal{E}|$.

4 Experiments

The driving purpose of the models presented here was to produce workable schedules for the Okanagan campus of University of British Columbia (UBC) for the 2021/2022 academic year. Other than producing the final exam schedules, but within the context of using this real data, our experiments are designed to compare the room-packings model with the room-cuts model. A follow up study will provide a more complete analysis.

The data provided by UBC includes anonymized student enrollment data, instructor requests, and room sizes and availabilities. The exam scheduling problem at this institution includes a few exceptional scenarios that require special handling, as listed below.

Cross-listed and Common Exams: A course that has different titles because it is shared between different programs is called *cross-listed*. Cross-listed courses are required to have their exams scheduled at the same time and location. For such courses, the data of their exams are merged during a preprocessing stage so that the merged exam is the union of the cross-listed exams. The merged exam is treated as a single exam both in the timeslot and room assignment models. Though this scenario could be handled by implementing Requirement 9, we chose to merge the exams so that when implementing Requirements 7 and 8 in the timeslot assignment model, the cumulative size of the grouped exams is accounted for.

A course that has multiple sections but a common exam is treated similarly.

Double-Seating Requirement: Every room's effective size is half of its true capacity, excluding the gym. We call this the *double-seating requirement.* The requirement is implemented through appropriate σ input data.

Gym: The gym is a special room. First of all, it is the only room that does not require double-seating (its effective size is the true capacity of the room). Furthermore, while most rooms require single-tasking, the gym uses multi-tasking, allowing at most three exams to take place concurrently. The use of the gym is also minimized, since it is a large room and it is not desirable to have multiple exams take place in the same location. This is handled by creating a variable to track if an exam is assigned the gym, and minimizing the sum of these variables in the objective function of the room assignment model.

UBC's examination period follows the structure shown in Example 1. We are considering two datasets: the (Winter) Term 1 and 2 data from the 2021/2022 academic year. Statistics about these datasets are listed in Table 3.

Our primary focus with the experiments is to test the effectiveness of the proposed models while meeting the university's 3-in-27-hours hardship requirement (see Example 1). To do so, we consider a *room-packings* model and a *room-cuts* model for each data set. The room-packings model implements Requirements 7–8 in the timeslot assignment model as described in Section 3.3 (the room-packings constraints). The room-cuts model uses the CP implementation of Requirements 7–8 for the timeslot assignment model as explained in Section 3.1 (the room-cuts). The minimum size of the exams of $\mathcal{E}^{>}_{r_{i-1}} \setminus \mathcal{E}^{>}_{r_i}$, $u_i^{\triangledown}$, was used for these constraints.

For both approaches the models were configured as follows. The constraints included in the timeslot assignment model are listed in Table 4. We consider two variations of objective functions. The first objective function only minimizes the number of times students have back-to-back exams, that is, we minimize $\boldsymbol{d}_1$. The second objective function minimizes a weighted sum, which includes the number of back-to-back and 2-in-1-day hardships, where the former hardship has a higher priority/weight. The latter metric will be referred to as *2-in-1s.* When searching for solutions, the exam timeslot assignment variables, $\boldsymbol{t}_E$, are branched on first, using CP Optimizer's default settings. Then the default settings of CP Optimizer's search engine choose the remaining variables and values to search on.

Table 3. Data properties for the UBC 2021/2022 data sets.

Data Property	Term 1	Term 2
Number of exams	358	378
Number of exams requiring a room	217	275
Number of exams representing cross-listed sections	23	33
Number of exams representing common exams	41	41
Number of students enrolled in exams	10031	9369
Average number of exams per student	3.4	3.4
Average number of students per exam	96.4	84.9
Number of instructors	256	269
Number of regular rooms[a]	32	32
Number of gyms	1	1
Number of computer rooms	8	8
Number of restricted rooms	4	6
Number of triples of exams with students in common	15848	15457
Number of pairs of exams with students in common	9345	9033
Concurrent exam periods constraints (Req. 9.1)	20	19
Different period constraints	0	0
Concurrent exam rooms constraints (Req. 9.2)	0[b]	0
Different room constraints	0	0
Time specific constraints (Req. 5)	9	8
Room specific constraints (Req. 5)	7	25

[a] All rooms are available for all timeslots.
[b] All sets of exams with this requirement were amalgamated into a single exam in preprocessing.

Table 5 lists the constraints for the room assignment model. A lexicographic objective function is used to first minimize the number of exams assigned to the gym and then minimize the excess space in the room. The room assignment model takes in the solution from the timeslot assignment model and assigns the exams to rooms. The room assignment model is run once for each timeslot and uses the default search settings in CP Optimizer. The time taken to solve the room assignment problem given a feasible timeslot assignment is negligible, as finding the room assignment is trivial for the CP solver.

All experiments were conducted on an Intel i9-10900K @ 3.70GHz (10 cores–20 threads/workers) with 32.0 GB memory using IBM ILOG CPLEX Optimization Studio 20.1.0.0, with the timeslot assignment model having a time limit of 24 hours. The search was always terminated by this limit and the solutions never reached optimality.

Table 6 shows the results for minimizing only back-to-backs. In this scenario the room-cuts outperform the room-packings model for both terms. However,

Table 4. List of constraints included in the timeslot assignment model for the experiments.

Requirement	Description	Constraint Type
1	No student/instructor can have more than one exam per timeslot	Hard
2	Exam duration requirement is met implicitly by input data	Hard
5	Set domain for the exam timeslot based on instructor requests	Hard
7, 8	Room-cuts (necessary condition to satisfy room constraints)	Optional, Hard
7, 8	Room-packings	Optional, Hard
9	Cross-listed or common exams required to be assigned the same timeslot	Hard
10	Zero 3-in-27-hours hardships ($\boldsymbol{h}_{3,54} = 0$)	Hard
10	Maximum 2-in-1-day hardships per student is 1 ($\boldsymbol{d}_3^{\triangle} \leq 1$)	Hard
10	Minimize 2-in-1-day hardships ($\boldsymbol{d}_3$)	Optional, Optimized
10	Minimize back-to-back hardships ($\boldsymbol{d}_1$)	Optional, Optimized

there is a trade-off: minimizing this metric comes at a cost of increasing the 2-in-1s (even though a back-to-back counts as a 2-in-1-day). For example, in Term 2, the room-cuts approach results in 4 back-to-backs but 1486 2-in-1s, while the room-packings approach leads to 17 back-to-backs but only 1275 2-in-1s. This observation led us to explore a weighted objective for our comparisons.

Table 7 shows the results from minimizing the weighted objective, where back-to-backs were given a 20-to-1 priority over 2-in-1s. For these experiments the room-packings approach leads to less back-to-backs exams for both terms.

Table 5. List of constraints included in the room assignment model for the experiments.

Requirement	Description	Constraint Type
3, 7	Gym hosts at most three exams; all other rooms host at most one exam	Hard
4, 8	Number of students writing exams scheduled in a room is at most the effective room size[a]	Hard
5	Set domain for the exam room based on instructor requests	Hard

[a] The constraint may be softened for exams with specific room requests.

Table 6. Results from experiments minimizing the number of back-to-back ($\boldsymbol{d}_1$) hardships for the UBC 2021/2022 Term 1 and Term 2 data sets. All experiments required $\boldsymbol{h}_{3,54} = 0$ and $\boldsymbol{d}_3^{\triangle} \leq 1$.

Model Option	back-to-backs	
	Term 1	Term 2
room-cuts	38	4
room-packings	58	17

On the other hand, room-cuts have a lower number of 2-in-1s for Term 1. We can see from the overall objective that the room-packings approach performs better for the weighted objective.

One downside to the 2021/2022 data sets is that they were produced during the COVID-19 pandemic. This resulted in courses having the option of holding their exams online in Term 1 or Term 2 if the course's instructor required special accommodations. Therefore not all exams needed rooms, and so that the exam schedule had more flexibility due to having less constraints. To further push the model and gauge how well it may perform in future years without online courses, we modified the data sets so that all exams were written in-person and needed to be scheduled in a room on campus. The resulting data sets had 65% and 35% more exams needing rooms for Term 1 and Term 2, respectively. The results

Table 7. Results from experiments minimizing a weighted function of the number of back-to-back ($\boldsymbol{d}_1$) and 2-in-1-day ($\boldsymbol{d}_3$) hardships for the UBC 2021/2022 Term 1 and Term 2 data sets. All experiments required $\boldsymbol{h}_{3,54} = 0$ and $\boldsymbol{d}_3^{\triangle} \leq 1$.

Model Option	back-to-backs		2-in-1s		Objective Value	
	Term 1	Term 2	Term 1	Term 2	Term 1	Term 2
room-cuts	146	91	735	728	3655	2548
room-packings	115	67	771	646	3071	1986

for these runs with these are shown in Table 8. For Term 1, these results agree with those seen in Table 7. For Term 2, we get a surprising outcome. Using the room-cuts approach here led to a better objective in terms of both back-to-backs and 2-in-1s compared with the room-packings approach. Furthermore, these metrics were also better than the results from the original data set, for both the room-cuts and room-packings approaches. When comparing the statistics for the original and modified versions of the Term 2 data, it is not obvious why this is the case. However, due to the nature of these experiments and because these results are not proven to be optimal, it is easily possible for this scenario to occur due to differences in the search trees explored by CP Optimizer.

Table 8. Results from experiments minimizing a weighted function of the number of back-to-back ($\boldsymbol{d}_1$) and 2-in-1-day ($\boldsymbol{d}_3$) hardships for the augmented UBC 2021/2022 Term 1 and Term 2 data sets. All experiments required $\boldsymbol{h}_{3,54} = 0$ and $\boldsymbol{d}_3^{\triangle} \leq 1$.

Model Option	back-to-backs		2-in-1s		Objective Value	
	Term 1	Term 2	Term 1	Term 2	Term 1	Term 2
room-cuts	197	58	827	745	4767	1905
room-packings	172	117	1021	723	4461	3063

These experiments serve to illustrate the robustness of our model for moderate-sized universities, as well as the difficulty in choosing an approach for ensuring room-assignable solutions in the exam scheduling problem. Our results illustrate that even for the same data sets, the choice between using room-cuts versus room-packings constraints varies depending on the objective function used. Furthermore, there are several other considerations to take into account. First, the selected CP solver may not have built-in bin-packing constraints available and it may be too onerous to build them from scratch. In this case using room-cuts would be the only choice. If the university is large, then exploring room-cuts may be necessary to reduce the number of variables and constraints in the CP model. If there are several room restrictions that cannot be softened and should not be ignored in the timeslot assignment model, then the room-packings approach would be most appropriate. Altogether, both approaches presented here are valuable tools that provide modeling options for researchers and practitioners in various scenarios.

Future work will be done to establish lower bounds on the objective functions to narrow the optimality gap. Furthermore, more tests will be conducted to study each model's performance on various benchmark and real-world data sets.

5 Conclusion

In this paper we presented a generic specification of the exam scheduling problem that encompasses a large variety of institutional requests. We also described in detail how this specification can be implemented using constraint programming. In particular, we introduced a general definition and implementation of student hardships, which we illustrated with three real-world examples: the 3-in-27-hours, back-to-back, and 2-in-1-day hardships. As well, we introduced a novel set of cuts that establish necessary conditions for a timeslot assignment solution to emit feasible room assignments. These room-cuts allow the exam scheduling problem to be decoupled into separate timeslot and room assignment phases.

The CP implementation of these models demonstrated its capabilities in the production of the 2021/2022 final exam schedules at the Okanagan campus of the University of British Columbia. Moreover, our testing compared the room-cuts model with the room-packings model using variations of the most recent data from UBC. The room-cuts implemented in the timeslot assignment model ensured feasibility in the room assignment model for all tests conducted. On the other hand, the room-packings model uses bin-packing constraints to join the timeslot and room assignment subproblems ensuring such feasibility intrinsically. Furthermore, the room-cuts model achieved competitive solutions when compared with the room-packings model. In conclusion, we have illustrated that both approaches are robust and can tackle the exam scheduling problem instances explored in this work, thus advancing the theory and tools for researchers and practitioners alike.

Acknowledgements The authors are grateful for the generous support from the following sources: the Natural Sciences and Engineering Research Council of Canada; the Provost and Vice-President Academic, University of British

Columbia, Okanagan; and the Department of Computer Science, Mathematics, Physics and Statistics, University of British Columbia, Okanagan. The authors would also like to thank each of the anonymous referees for their helpful comments.

References

1. Abou Kasm, O., Mohandes, B., Diabat, A., El Khatib, S.: Exam timetabling with allowable conflicts within a time window. Computers and Industrial Engineering **127**, 263–273 (2019). https://doi.org/10.1016/j.cie.2018.11.037, http://www.sciencedirect.com/science/article/pii/S0360835218305771
2. Al-Hawari, F., Al-Ashi, M., Abawi, F., Alouneh, S.: A practical three-phase ILP approach for solving the examination timetabling problem. International Transactions in Operational Research **27**(2), 924–944 (Mar 2020). https://doi.org/10.1111/itor.12471, https://onlinelibrary.wiley.com/doi/abs/10.1111/itor.12471
3. Babaei, H., Karimpour, J., Hadidi, A.: A survey of approaches for university course timetabling problem. Computers and Industrial Engineering **86**, 43–59 (2015). https://doi.org/10.1016/j.cie.2014.11.010, http://dx.doi.org/10.1016/j.cie.2014.11.010
4. 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)
5. Beldiceanu, N., Carlsson, M., Rampon, J.X.: Global Constraint Catalog (2005), https://hal.archives-ouvertes.fr/hal-00485396/
6. Broder, S.: Final examination scheduling. Communications of the ACM **7**(8), 494–498 (Aug 1964). https://doi.org/10.1145/355586.364824, https://dl.acm.org/doi/abs/10.1145/355586.364824
7. Cataldo, A., Ferrer, J.C., Miranda, J., Rey, P.A., Sauré, A.: An integer programming approach to curriculum-based examination timetabling. Annals of Operations Research **258**(2), 369–393 (Nov 2017). https://doi.org/10.1007/s10479-016-2321-2, https://link.springer.com/article/10.1007/s10479-016-2321-2
8. Cole, A.J.: The preparation of examination time-tables using a small-store computer. The Computer Journal **7**(2), 117–121 (Feb 1964). https://doi.org/10.1093/comjnl/7.2.117, https://academic.oup.com/comjnl/article/7/2/117/335177
9. Genc, B., O'Sullivan, B.: A Two-Phase Constraint Programming Model for Examination Timetabling at University College Cork, vol. 12333 LNCS. Springer International Publishing (2020). https://doi.org/10.1007/978-3-030-58475-7_42, http://dx.doi.org/10.1007/978-3-030-58475-7_42
10. McCollum, B.: A perspective on bridging the gap between theory and practice in university timetabling. In: Burke, E.K., Rudová, H. (eds.) Practice and Theory of Automated Timetabling VI. pp. 3–23. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
11. Müller, T.: Real-life examination timetabling. J Sched **19**, 257–270 (2016). https://doi.org/10.1007/s10951-014-0391-z, http://www.unitime.org.

12. Oude Vrielink, R.A., Jansen, E.A., Hans, E.W., van Hillegersberg, J.: Practices in timetabling in higher education institutions: a systematic review. Annals of Operations Research **275**(1), 145–160 (2019). https://doi.org/10.1007/s10479-017-2688-8, https://doi.org/10.1007/s10479-017-2688-8
13. 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**(1), 55–89 (2009). https://doi.org/10.1007/s10951-008-0077-5

Scheduling Worker Timetables in Flowshops with Multi-Skill Workers

Ehud Ikar[1][0000−0002−9626−9324], Elad Shufan[2][0000−0001−7960−5798], Hagai Ilani[2][0000−0003−3548−1572], and Tal Grinshpoun[1][0000−0002−4106−3169]

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

Abstract. This work is inspired by a production line for manufacturing identical products (jobs) by multi-skilled workers. The production of each job consists of a set of pre-ordered successive operations, similar to the problem of a flow shop. However, different from the standard flow shop model, each operation may be performed only by a subset of workers who have the required skills for that operation. The workers are non-identical in the sense that each may posses a different set of skills. The objective is to schedule the timetables of the workers in a manner that minimizes the makespan.
There are two known formulations in the literature that capture the extremes of the multi-skilled workers at focus. The first, flexible (a.k.a. hybrid) flow shop, deals with scenarios in which several single-skilled workers can perform the same operation. The second, reentrant flowshop, deals with scenarios in which multi-skilled workers can perform different tasks but without overlaps, i.e., each operation can be performed by exactly one specific worker.
In a previous study we focused on the reentrant flowshop extreme; we formulated the problem as an integer program (IP) and obtained deep insights on two simple heuristics that solve the problem, in some cases even to optimality. Here, we study extensions of the IP formulation to the multi-skilled flow shop problem (MSFFP). The IP formulation is inspired by modelling the problem as a multi-mode resource constrained project scheduling problem. This model also enables adaptation of heuristics for solving the MSFFP along with several lower bounds that help guaranteeing a reasonable approximation to the optimal makespan.

Keywords: Production Line · Worker Timetabling · Multi-Skilled Workers · Identical Jobs · Resource-Constrained Project Scheduling.

Noname manuscript No.
(will be inserted by the editor)

A knowledge-based approach to detecting and explaining conflicts in timetabling problems

Kylian Van Dessel · Joost Vennekens

Received: date / Accepted: date

Keywords timetabling · conflict detection · declarative programming · knowledge representation and reasoning

1 Introduction

Traditional research on combinatorial optimization problems, such as scheduling, focuses on computational efficiency. The typical goal is to compute better solutions within given time constraints, or to compute optimal solutions faster. Over the years, the field of Operational Research (OR), and more recently also the field of Artificial Intelligence (AI) put numerous techniques forward that can solve combinatorial optimization problems in a highly efficient manner. The benchmarks used in this research are often artificially generated. Moreover they typically consist of well-defined problem definitions over consistent problem instances, i.e. there are no contradicting hard constraints and instances are not constrained to the point that there is no longer a solution. This has lead to a mismatch between academic result and real-life applications, in which problems are most often ill-defined.

To overcome this gap, both research fields have put a substantial effort in solving optimization problems for real-world benchmarks. E.g., the biannual international timetabling competition gathers a large number of real-world applications for its tracks [7] and more and more conferences in the field include

K. Van Dessel
Jan Pieter De Nayerlaan 5, 2860 Sint-Katelijne-Waver
Tel.: +32-15-688191
E-mail: kylian.vandessel@kuleuven.be

J. Vennekens
Jan Pieter De Nayerlaan 5, 2860 Sint-Katelijne-Waver
Tel.: +32-15-688235
E-mail: joost.vennekens@kuleuven.be

talks on real-world applications or even have dedicated tracks for them, such as PATAT or ICAPS.

In this work we aim to identify if and, more importantly, why a problem specification is ill-defined. If we can explain in a comprehensible manner why a conflict occurs, then the inconsistency can be reported to the user. The user can then use this information to construct a consistent specification that can be handled by the system. We focus on inconsistencies in the problem instance, rather than in the problem definition itself. Existing systems typically try to detect or prevent such inconsistencies in a rather ad hoc manner, e.g., by means of a user-friendly interface or by performing input data checks. These techniques are only usable for shallow inconsistencies such as typing errors and miscounts. For the detection of more profound conflicts, more complex inference is required.

In this paper, we propose a knowledge-based approach for detecting and explaining conflicts in instances of the School Timetabling Problem (STP) in the context of Belgian secondary schools. We do this in an effort to tackle the problem from a Knowledge Representation and Reasoning (KRR) point of view as opposed to the more traditional research from the field of OR. We believe that the declarative aspect of our proposed system will allow conflicts to be explained in a more comprehensible manner.

An additional contribution will be the compilation of a data set with realistic conflicts based on actual STP instances, which will also be used for the validation of our work.

2 The IDP knowledge base system

In order to explain inconsistencies in a comprehensible manner, we start from a purely declarative knowledge base. In this approach, a problem specification consists of a set of hard and soft constraints much like in OR techniques. However, we focus on writing down the knowledge base in an intuitive language, which is understandable for non-experts, as opposed to the mathematical models often used in OR techniques.

The STP has been studied in such a context of declarative programming before, e.g., in ASP [1]. State-of-the-art answer set programming (ASP) systems typically use a two-step ground-and-solve process to unite a declarative specification language with an efficient solver. A traditional ASP system first grounds the problem specification to a propositional program. This step reduces complex language constructs and shorthands, added to the language for expressiveness and ease of modeling, to their most basic, variable-free form. In a second step an ASP solver is then used to generate answer sets (solutions) for the ground program [3].

The knowledge-based system we propose for studying conflict detection and explanation in the context of the STP, is the IDP system [2]. This system is similar to ASP systems, but uses a language based on First Order Logic (FO), extended with additional language features such as types and aggregates (as opposed to the non-monotonic reasoning ASP-language used by traditional ASP systems).

Example 1 The trivial constraint that each class should be taught by a teacher can be expressed by the following FO sentence:

$$\forall c[Class] : \exists t[Teacher] : Teaches(t, c).$$

This sentence is grounded to a disjunction of propositions over each element of the type $Teacher$, and this for each element of the type $Class$. Consider the following structure:

$$\begin{aligned} Teacher &= \{Mary, John, Carl\} \\ Class &= \{Maths, Physics\} \end{aligned}$$

The constraint is grounded to the ground program:

$$Teaches(Mary, Maths) \lor Teaches(John, Maths) \lor Teaches(Carl, Maths).$$
$$Teaches(Mary, Physics) \lor Teaches(John, Physics) \lor Teaches(Carl, Physics).$$

A solution found by the solver would then look like this:

$$Teaches(John, Maths).\ Teaches(Mary, Physics).$$

For knowledge-based systems, several conflict detection techniques are known [4,6]. These techniques typically work by detecting an *unsatisfiable core*: a minimal subset of an inconsistent theory that is still inconsistent [5]. However, because existing techniques look for unsatisfiable cores in ground (i.e., propositional) theories, these cores tend to be very big and not comprehensible for a non-programmer. Because we aim to provide information to a timetabler, this makes these techniques unusable for us. We therefore develop a method which reduces an inconsistent theory to a smaller theory, that is still tractable in size and written in the same language as the original problem specification, i.e., the highly expressive FO(·)-language.

3 Diagnosis of an unsatisfiable problem

When representing a problem specification in a knowledge base, we first need to specify which symbols will be used for the representation (i.e. the vocabulary). The constraints then correspond to a theory over these symbols and a specific instance corresponds to a structure for part of the vocabulary. As such, the STP can be represented as a combination of a theory on all the hard and soft constraints of the problem (e.g., "Each class should be taught by a therefore qualified teacher") and a structure defining a specific STP instance (e.g., "Mary is a teacher", "teacher Mary is qualified to teach class Math").

For a given theory T and structure S, the problem of finding a solution is that of solving the model expansion problem $MX(S, T)$, which computes the set of all structures S' that extend S such that $S' \models T$.

We assume the problem to be unsatisfiable, thus $MX(S,T) = \{\}$, i.e., there are no solutions, and look for reductions S' of S for which $MX(S',T)$ is still empty. S' is considered a *reduction* of S if the domain of S' is a subset of the domain of S and, for each symbol P, the interpretation of P in S', denoted $P^{S'}$, is equal to the restriction of P^S to the domain of S'. If for each solution M to $MX(S,T)$, there exists a reduction M' of M such that M' is a solution of $MX(S',T)$, we call the reduction *T-safe*.

To help users understand why a model expansion problem $MX(S,T)$ is unsatisfiable, we might therefore look for reductions S' of S that are T-safe and for which $MX(S',T)$ is unsatisfiable. However, in the case that $MX(S,T)$ is empty, the notion of safeness is actually not very relevant, since each reduction is safe in that case.

Definition 1 *Let T be a theory and Σ a subset of the vocabulary of T. A* reduction policy *for T relative to Σ is a set of pairs $(P(x), \rho(\vec{y}, x))$ where P is a type in Σ and ρ is a Σ-formula. Given a Σ-structure S, we say that a reduction S' of S follows a reduction policy if it is the case that, for all pairs $(P(x), \rho(\vec{y}, x))$ in the policy and for each domain element d that is removed from type P, also all elements $e_1, e_2, \ldots, e_n$ are removed from their respective types for which $S \models \rho[x/d, \vec{y}/\vec{e}]$. A reduction policy is T-safe if, for each Σ-structure S, all reductions that follow the policy are T-safe.*

If we have a T-safe reduction policy, we can use the reductions that follow this policy to conclude that, if at least one of these is unsatisfiable, also the original model expansion problem is unsatisfiable.

Definition 2 *A diagnosis of an unsatisfiable problem $MX(S,T)$ is a T-safe reduction S' of S such that $MX(S',T)$ is unsatisfiable as well. Such a diagnosis is called minimal if S' itself is the only diagnosis of $MX(S',T)$.*

Example 2 Consider the following simple example. Each class must be taught, by a teacher who is qualified to teach it, but each teacher may teach only a single course:

$$\forall c[Class] : \exists t[Teacher] : Teaches(t,c).$$
$$\forall c[Class]\ t[Teacher] : Teaches(t,c) \Rightarrow Qualified(t,c).$$
$$\forall t[Teacher]\ c[Class]\ c'[Class] : Teaches(t,c) \land c \neq c' \Rightarrow \neg Teaches(t,c').$$

This theory is unsatisfiable in the following structure:

$$\begin{aligned} Teacher &= \{Mary, John, Carl\} \\ Class &= \{Maths, Physics, English\} \\ Qualified &= \{(Mary, Maths), (Mary, Physics), (John, English), (Carl, English)\} \end{aligned}$$

Obviously, the problem here is the fact that Mary is the only teacher qualified to teach both Maths and Physics. In other words, this problem will already manifest itself in the following reduction S' of S:

$$\begin{aligned} Teacher &= \{Mary\} \\ Class &= \{Maths, Physics\} \\ Qualified &= \{(Mary, Maths), (Mary, Physics)\} \end{aligned}$$

This reduction is T-safe. To prove this, it suffices to show that the reduction policy consisting of the single pair $(Teacher(x), Qualified(x, y))$ is T-safe. Because of the first formula, it is clear that $Qualified(t, c)$ covers the first formula. In order to be allowed to remove the teachers $John$ and $Carl$, this covering formula asks that we also remove all classes that $John$ and $Carl$ are allowed to teach. We see that, indeed, $Class^{S'}$ no longer contains the class $English$. Therefore, $MX(S', T)$ has at least as many solutions as the original problem $MX(S, T)$. In other words, removing a set of teachers together with all the classes they may teach only makes the problem easier. Because $MX(S', T)$ is unsatisfiable, it is a diagnosis of $MX(S, T)$. Since removing either $Mary$ or one of her two classes would render the problem satisfiable, $MX(S', T)$ has no more diagnoses and hence it is a minimal diagnosis. On the other hand, removing only the teachers but not the classes is not safe, since it makes the problem harder.

We are not interested in just any diagnosis. We are looking for those we can explain to a user. To this end, we set up a frame that dictates which safe reductions we want to apply to which elements in order to detect a certain kind of conflict. If we apply such a frame to an inconsistent problem and the reduction is inconsistent as well, then we have a diagnosis of that kind of conflict. We try to define these frames in a way that the reductions that follow the frame are as small as possible to still explain the conflict, but note that a correct diagnosis is not necessarily minimal.

4 Discussion and Future work

We presented a knowledge-based approach for detecting and explaining conflicts in optimization problems. The problem specification is written down in an intuitive, understandable language, unlike the typical mathematical models used in OR techniques. Additionally, as opposed to existing KRR research on computing unsatisfiable cores from the ground program, we propose a reduction on the level of the problem specification itself. In this way, we not only detect the conflicts, but also have the ability of explaining them in a comprehensible manner to a non-programmer.

We will put the proposed concepts to practice and study their correctness and performance in the context of the STP. We focus on ill-defined problem specifications. To this end, we compile a data set with conflicts from a real-life context. For our experiments we will use the IDP knowledge base system, which provides an expressive declarative language (FO(·)) for representing the problem specification.

References

1. Banbara, M., Soh, T., Tamura, N., Inoue, K., Schaub, T.: Answer set programming as a modeling language for course timetabling. Theory and Practice of Logic Programming **13**(4-5), 783–798 (2013). DOI 10.1017/S1471068413000495
2. Bruynooghe, M., Blockeel, H., Bogaerts, B., Cat, B.D., Pooter, S.D., Jansen, J., Labarre, A., Ramon, J., Denecker, M., Verwer, S.: Predicate logic as a modeling language: Modeling and solving some machine learning and data mining problems with IDP3. CoRR **abs/1309.6883** (2013). URL http://arxiv.org/abs/1309.6883
3. Kaufmann, B., Leone, N., Perri, S., Schaub, T.: Grounding and solving in answer set programming. AI Magazine **37**(3), 25–32 (2016). DOI 10.1609/aimag.v37i3.2672. URL https://www.aaai.org/ojs/index.php/aimagazine/article/view/2672
4. Liffiton, M., Sakallah, K.: Algorithms for computing minimal unsatisfiable subsets of constraints. J. Autom. Reasoning **40**, 1–33 (2008). DOI 10.1007/s10817-007-9084-z
5. Lynce, I., Silva, J.: On computing minimum unsatisfiable cores (2004)
6. Torlak, E., Chang, F.S.H., Jackson, D.: Finding minimal unsatisfiable cores of declarative specifications. In: J. Cuellar, T. Maibaum, K. Sere (eds.) FM 2008: Formal Methods, pp. 326–341. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
7. Van Bulck, D., Goossens, D., Belien, J., Davari, M.: The fifth international timetabling competition (itc 2021): Sports timetabling. In: MathSport International 2021, pp. 117–122. University of Reading (2021)

A Column Generation Approach for Solving the Fixed Route Dial-A-Ride Problem

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

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

Abstract. The fixed route dial-a-ride problem (FRDARP) is a demand-responsive transport solution in which passengers request to be transported at certain times between destinations that are located along a fixed route. The aim is to operate all the requests with a given fleet of vehicles and a bounded number of transports in a manner that minimizes the passengers' overall deviation from the requested times. The FRDARP is a variant of the famous dial-a-ride problem (DARP), in which the difficulty of finding the vehicle routes is neutralized by determining the route in advance. With a fixed route, the remaining problem is of grouping customers together and scheduling the timetable. The DARP solutions, and in particular the FRDARP, are suitable for a variety of transportation needs, including dedicated solutions for low-populated areas, as well as for customers with special needs, such as children or elderly.
Recently, we have presented a polynomial algorithm to solve the FRDARP by a reduction to the shortest path problem. Based on the problem input, we dynamically construct a graph; a shortest weighted path in this graph, which starts at a source node and ends at a goal node, corresponds to an optimal schedule. Though the presented method for solving the FRDARP is polynomial, its implementation involves construction and traversal of huge graphs. Therefore, finding an optimal solution for a large number of requests becomes a non-trivial challenge. FRDARP can be also modeled as an integer linear programming, yet with a huge number of variables. Here, we present a column generation approach for solving FRDARP using an integer linear programming formulation. We show that the sub-problem that defines the entering variable at each iteration is a coloring problem of an interval graph ,which is polynomially solvable. By using column generation, we expect to appreciably reduce the running time needed for finding an optimal solution and therefore increase the applicability of FRDARP for large problems.

Keywords: Demand Responsive Transport · Dial-a-Ride Problem · Fixed Route · Column Generation.

Local Search Techniques for a Medical Student Scheduling Problem

Eugenia Zanazzo[1], Sara Ceschia[1][0000−0003−1191−1929], Agostino Dovier[2][0000−0003−2052−8593], and Andrea Schaerf[1][0000−0001−6965−0536]

[1] DPIA, University of Udine, via delle Scienze 206, 33100 Udine, Italy,
{eugenia.zanazzo,sara.ceschia,andrea.schaerf}@uniud.it
[2] DMIF, University of Udine, via delle Scienze 206, 33100 Udine, Italy,
agostino.dovier@uniud.it

Keywords: Medical Student Scheduling · Local Search · Simulated Annealing

1 Introduction

We consider the Medical Student Scheduling (MSS) problem in the formulation proposed by Akbarzadeh and Maenhout [2], which is a simplified version of the general problem previously proposed by the same authors [1].

In the MSS problem, medical students have to be assigned in subsequent periods to a set of wards in designated hospitals, in order to complete their training by performing internships on the disciplines carried out in the specific wards.

This version of the problem takes into account, among other constraints and objectives, precedences among disciplines, student preferences, waiting periods, and hospital changes. The typical horizon considered is one year, split into either 12 periods of one month or 24 periods of two weeks.

The objective of the problem is to design a timetable that maximizes both students' desire and fairness among students, satisfying rules, regulations, and requirements for the medical school and the hosting hospitals.

We developed a local search technique for the MSS problem, based on a combination of two different neighborhood relations and guided by a Simulated Annealing procedure.

We also implemented an instance generator that was used to create challenging instances with up to 320 students. According to Akbarzadeh and Maenhout [1], such number of students represents a realistic size, though much larger than the ones in the original dataset (max 80 students). In addition, the generated instances also activate the constraint on the minimum number of students in a ward, which is included in the model but always set to 0 in the original instances. This constraint makes instances harder to be solved.

As customary, the generated instances are split into two sets, one used for the parameter tuning and the other one for the validation.

Our solution method has been able to find consistently the optimal solution value for all instances of the dataset proposed by Akbarzadeh and Maenhout [2], in much shorter runtime, though without any optimality guarantee, than their exact technique.

2 Solution method

For brevity, we do not report here the precise MSS formulation, which can be found in the original work [2].

Our local search technique uses for the search space an integer-valued matrix that assigns to each student in each period a specific ward of a specific hospital. That is, we use a single value that encodes a pair ⟨hospital, ward⟩. Since in the original data only one given discipline can be undertaken in any specific ward, this value specifies also the discipline. The conventional value -1 is used when the student is not assigned to any internship in the specific period.

We decided to include in our search space also solutions that may violate two hard constraints, namely the minimum and maximum number of students per ward per period and the precedences among disciplines. These constraints are taken care of by the cost function along with the soft constraints. As customary, the hard constraint violations are assigned a higher weight, in order to favor feasibility over optimality.

We consider the following two atomic neighborhood relations:

- Change (C). The move $\mathsf{C}\langle s, p, w, p', w' \rangle$ reassigns the student s from the period p at ward w to a new period p' and a new ward w'. The move has the precondition that s is currently idle in p', unless $p = p'$; in the latter case the move represents a reassignment of the ward in the current period p. It is also possible that $w = w'$, so that the student remains in the same ward, but at different time. It is not possible that $p = p'$ and $w = w'$, which would result in a null move.
- Swap (S). The move $\mathsf{S}\langle s, p, w, p', w' \rangle$ swaps the assigned wards w and w' of student s in the two distinct periods p and p'. The precondition here is that the student is assigned in both periods, i.e., $w \neq -1$ and $w' \neq -1$.

As guiding metaheuristic, we use Simulated Annealing, which already turned out quite effective in a number of timetabling problems (see, e.g., [6, 4, 3]). The neighborhood relation employed is the set union of Change and Swap, and the random move selection is guided by a parameter ρ_S (called *swap rate*), such that a Swap move is drawn with probability ρ_S and a Change move with probability $1 - \rho_S$.

3 Experimental results

The tuning procedure was performed on training instances of various sizes, created by our generator specifically for this purpose. We used the tool JSON2RUN [7], which performs the F-Race procedure [5] for selecting the best configuration. The winning configuration has swap rate ρ_S equal to 0.19.

Table 1. Comparative results between different solution methods and time limits.

	CPU (s)	Gap (%)	Opt (%)
MIP	8054	9	50
CP	2630	1	88
DP	166	0	100
SA-5	2.5	0.09	98.1
SA-10	4.9	0.07	99.3
SA-20	9.7	0.07	99.8
SA-50	24.1	0.06	99.9
SA-100	48.2	0	100

Table 1 shows the comparison between the methods presented by Akbarzadeh and Maenhout [2, Table 8], i.e. a Mixed Integer Programming (MIP) formulation, a Constraint Programming (CP) formulation and a Dynamic Programming (DP) method, and our method based on Simulated Annealing with increasing number of iterations (10^6k with $k \in [5, 10, 20, 50, 100]$), denoted by SA-$k$. The table reports for each method the average results obtained on all original instances in terms of computational time in seconds (CPU), final optimality gap (Gap) and percentage of instances solved to optimality (Opt) within the time limit[3]. These results show that we can obtain the optimal value already in 2.5 seconds on average, with a confidence of 98.1%. With 9.7 seconds we reach the near certainty given by the confidence of 99.8%.

The project is still ongoing, and the current work regards the experimentation on larger and more challenging instances, the development of exact methods, and the design of hybrid approaches.

Acknowledgements We thank Babak Akbarzadeh for answering all our questions about their work.

References

1. Akbarzadeh, B., Maenhout, B.: A decomposition-based heuristic procedure for the medical student scheduling problem. European Journal of Operational Research **288**(1), 63–79 (2021)
2. Akbarzadeh, B., Maenhout, B.: An exact branch-and-price approach for the medical student scheduling problem. Computers and Operations Research **129**, 105209 (2021)
3. Bellio, R., Ceschia, S., Di Gaspero, L., Schaerf, A.: Two-stage multi-neighborhood simulated annealing for uncapacitated examination timetabling. Computers and Operations Research **132**, 105300 (2021)

[3] The methods by Akbarzadeh and Maenhout [2] are implemented in the ILOG-OPL IBM environment and the time limit imposed is 5760 secs for instances with 40 students and 11520 secs for those with 80 students.

4. Bellio, R., Ceschia, S., Di Gaspero, L., Schaerf, A., Urli, T.: Feature-based tuning of simulated annealing applied to the curriculum-based course timetabling problem. Computers and Operations Research **65**, 83–92 (2016)
5. Birattari, M., Yuan, Z., Balaprakash, P., Stützle, T.: F-race and iterated F-race: An overview. In: Experimental methods for the analysis of optimization algorithms, pp. 311–336. Springer, Berlin (2010)
6. Ceschia, S., Di Gaspero, L., Schaerf, A.: Design, engineering, and experimental analysis of a simulated annealing approach to the post-enrolment course timetabling problem. Computers and Operations Research **39**, 1615–1624 (2012)
7. Urli, T.: json2run: a tool for experiment design & analysis. CoRR **abs/1305.1112** (2013)

Shift Scheduling in Interdependent Multi-stage Systems with Reallocation of Workforce

Seyed Mohammad Zenouzzadeh and Raik Stolletz[0000−0003−2725−9322]

Chair of Production Management, University of Mannheim, Germany
`zenouzzadeh@uni-mannheim.de`
`raik.stolletz@uni-mannheim.de`

Abstract. Many manufacturing and logistic systems (e.g., distribution centers) consist of serially organized stages. In each of these stages, a process is performed on each item of the demand (e.g., picking, consolidation/packing, and shipping in a distribution center). The number of items to be processed at each stage depends on the demand pattern and the assigned capacity to the predecessor stages. Hence, the capacity decisions for the stages are interdependent in such systems. We model the shift scheduling problem in systems with serially organized stages. We consider a daily planning horizon with a multi-skilled workforce who can be reallocated multiple times during their shifts. Each reallocation between stages results in a loss of capacity at the destination stage. The objective is to minimize the total workforce costs. We propose a column-generation algorithm to solve the problem. We solve various realistic instances to test the effectiveness of the proposed algorithm. Our results show that scheduling the shifts independently for stages will either result in suboptimal or infeasible solutions. We also show that taking into account the interdependency among the stages helps better utilize the reallocation flexibility.

Keywords: Shift Scheduling · Multi-skilled workforce · Worker Reallocation · Multi-stage systems

Noname manuscript No.
(will be inserted by the editor)

Enhancing Security via Deliberate Unpredictability of Solutions in Optimisation

Daniel Karapetyan · Andrew J. Parkes

Received: date / Accepted: date

Abstract The main aim of decision support systems is to find solutions that satisfy user requirements. Often, this leads to predictability of those solutions, in the sense that having the input data and the model, an adversary or enemy can predict to a great extent the solution produced by your decision support system. Such predictability can be undesirable, for example, in military or security timetabling, or applications that require anonymity. In this paper, we discuss the notion of solution predictability and introduce potential mechanisms to intentionally avoid it.

Keywords Unpredictability · Decision Support · Diversity of Solutions · Perfect Matching Problem

1 Introduction

A search algorithm, even non-deterministic, is likely to be biased to some solutions, and hence anyone knowing the input data and the algorithm might be able to predict much of the solution. This can be an issue if the solution has to be kept in secret. One of many examples of this is the scheduling of tasks in cloud computing. In this problem, computational tasks are assigned to various servers and time slots. While respecting the constraints and efficiency considerations, one may want to keep the schedule as unpredictable as possible to reduce the chances of the potential intruder to guess the server and/or time slot assigned to a specific task.

D. Karapetyan
School of Computer Science, University of Nottingham
E-mail: daniel.karapetyan@nottingham.ac.uk

A. J. Parkes
School of Computer Science, University of Nottingham
E-mail: andrew.parkes@nottingham.ac.uk

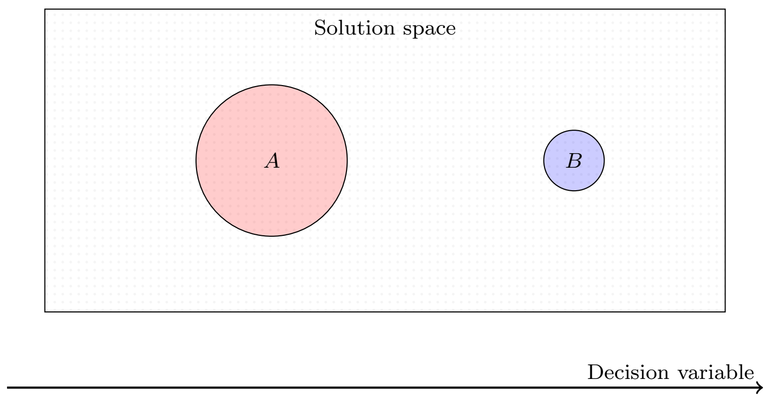


Fig. 1: Example of solution space with all the feasible solutions grouped into two clusters A and B. Cluster A contains many more solutions than cluster B; as a result, uniform sampling from the entire set of feasible solutions will be biased towards cluster A, hence the value of the 'decision variable' will be 'predictable'.

Solution unpredictability can be understood in many ways. In our example, one may be interested in predicting the exact server and time slot for a task, or may be interested in predicting only the server, or even an approximate location of the server (i.e., the specific data centre). Our aim here is not to give a generic formulation of the problem; but to point out potential interesting extensions of the classic decision support, and to provide some relevant results, and so to encourage further discussion of the topic.

In particular, we discuss diversity issues in the context of the assignment problem, or specifically of variations of Perfect Matching Problems in bipartite graphs, which is closely related to the task assignment in cloud computing. We want the locations/times of different tasks to be unpredictable (to make hacking harder) and so need diverse assignments to select from.

2 Generation of Unpredictable Solutions

A straightforward approach to achieve unpredictability of solutions is to randomly sample the set of feasible solutions. (A standard technique to do this is the "rapidly mixing Markov Chains", e.g. see [1,4]) However, to enhance security, the sampling should not necessarily be uniform. To illustrate this, we refer to Figure 1. in which the sets of feasible solutions form clusters, i.e. subsets of similar solutions (e.g. see [3]). In the example given in Figure 1, all the solutions are grouped into two clusters: A and B. Hence, solutions within each cluster share similar values of the 'decision variable'. Cluster A contains more solutions, and hence a solution selected with uniform sampling is likely to be from cluster A. This will make the value of the decision variable predictable; with high probability, it will correspond to the first cluster.

To address this issue, we may want to pre-select a subset of diverse feasible solutions and then pick one of them randomly. For example, such a subset may be obtained by selecting 10 feasible solutions such that the total Hamming distance between them is maximised. (In a loose sense, we are doing the exact opposite of the work on minimal perturbations, such as [2], which aimed to find nearby solutions, and instead are looking for “maximal perturbations”.)

This approach is likely to generate interesting optimisation challenges. Doing it directly by first enumerating all solutions is generally impractical, even for problems where finding a solution is easy. Indeed, just counting all the feasible solutions is generally #P-hard, and the solutions sets are typically exponential in size.

Also, there is the challenge of selecting diverse sets of solutions. This corresponds to a kind of “Maximum Diversity Problem” and again likely to be NP-hard. We expect that heuristic approaches can be used to address these complexity issues, though maybe with special cases for which efficient algorithms are available.

However, in this work-in-progress paper, naturally, we do not answer these questions. Instead we consider the relatively simple case of perfect matching problems, or assignment problems; though these problems are of interest in their own right. For enhanced security of systems using assignment problems, we might well want to increase the unpredictability of such solutions. Accordingly, in the next section, Section 3, we study the problem of finding diverse solutions to the perfect matching problem, and present relevant decision and optimisation problems, with some initial work on solution methods.

3 Diverse Solutions Sets for the Perfect Matching Problem

Firstly, a quick reminder of the base problem:

NAME: Perfect-Matching
INSTANCE: A bipartite graph $G = (U, V, E)$ over vertices (U, V) of sizes (n, n) and with edges E.
SOLUTION: A vertex-disjoint subset $M \subseteq E$ of n edges, i.e. a subset of the edges that cover every node, and that are disjoint (do not share any nodes).

Finding one solution (perfect matching) is well-known to be polynomial-time (e.g. using the Hungarian method). However, this does not mean that all questions about perfect matchings are necessarily easy. For example, counting the number of solutions is #P (sharp-P); a class that is (generally assumed) much harder than NP. In particular, we remark, that there may well be that there are questions, relevant to diversity, about the set of solutions (perfect matchings) that may also be harder than P (under usual assumptions, such as $P \neq NP$).

Firstly, suppose that we are given one perfect matching, and to promote diversity, we want to find another one that is “as different as possible”. For simplicity, we will just measure the difference or or distance between matchings, hence, maximising the number of edges that are different. (Since we are looking at perfect matchings then this is also equivalent to minimising the number of edges which are shared between matchings.) Specifically, we define the problem

DEFINITION: Distant-Perfect-Matching
INSTANCE:

- Bipartite (unweighted) graph G, on (n, n) nodes;
- Perfect matching M_1;
- Integer $0 \leq d \leq n$.

QUESTION: Does there exist another perfect matching M_2, such that M_1 and M_2 differ on at least d assignments? In other words, does there exist a matching M_2 such that $|M_1 \cap M_2| \leq n - d$?

The maximum distance, $d = n$, is easy because it means that no edges can be shared. Hence, we can simply solve this case by removing all the edges in M_1 from E, and then looking for a perfect matching in this reduced graph.

The problem of finding maximum d can also be solved in poly-time using the given solution M_1 to modify the weights of the edges, giving a new weighted graph and then doing a maximum weight perfect matching on this graph.

This approach has the drawback that we need to provide the first matching. Instead, generally we want to simultaneously find a pair of well-separated perfect matchings – ones differing on at least d edges. Using the usual distinction between "maximal" (local) and "maximum" (global), this leads to two problems, Firstly, the "maxim**al**" separation:

DEFINITION: Maximal-Separated-Perfect-Matchings
INSTANCE:

- Bipartite (unweighted) graph G on (n, n) nodes;

TASK: Find a pair of matchings M_1 and M_2, that are maximally separated. That is, no matching is further from M_1 than M_2 is, and vice versa.

This is in poly-time because we just iterate solution to Distant-Perfect-Matching, switching between which matching is considered the fixed one. Starting from any matching, call it M_1, then find the most distant, call it M_2, then find the most distant from M_2 etc, terminating when the distance no longer increases – which must happen within $O(n)$ iterations.

This problem is like finding local optima under the (large) move of finding the most distant matching. The corresponding global optimum version is to demand "maxim**um**" separation, equivalent to the following decision problem:

DEFINITION: Maximum-Separated-Perfect-Matchings
INSTANCE:

- Bipartite (unweighted) graph G on (n, n) nodes;
- Integer $0 \leq d \leq n$.

QUESTION: Do there exist perfect matchings M_1 and M_2, such that M_1 and M_2 differ on at least d assignments?

A special case of this is again a maximal separation, $d = n$, which requires a disjoint pair of perfect matchings. However, given two disjoint perfect matchings, then each vertex has two distinct edges to it; by following these we get disjoint cycles. So the $d = n$ case is equivalent to finding a "Disjoint Vertex Cycle Cover" – a set of disjoint cycles that contain all the vertices. This is known to be polynomial time by conversion to a matching problem[1]. Currently, we do not know whether the Separated-Perfect-Matchings problem for an arbitrary d is in P, or is NP-complete (or otherwise).

4 Conclusions

In this paper, we discussed the concept of unpredictability of solutions in automated decision support. As a motivating example, we consider a simple assignment problem, which could easily be part of task scheduling in cloud computing. The aim is that unpredictability of task assignments will increase security of the system, by making it harder for malicious agents to guess locations and time slots of tasks. The most obvious approach to achieving unpredictability, random sampling of solutions, turns out to be computationally hard and weak. Indeed, uniform sampling is both complex and would not necessarily give us the desired diversity of solutions.

Hence, we focus on finding a few diverse solutions; then we can select one of them randomly to achieve unpredictability. We model the task scheduling using the bipartite matching. We gave some initial definitions that relate to finding diverse pairs of matchings. In particular, observing that finding maximally separated matchings is possible in polynomial time. Though not (yet) answering the question of finding the pairs with maximum separation. Also whilst we found that it is easy to find a **pair** of non-overlapping solutions, we do not know whether this result generalises to a higher number of solutions.

This is still work in progress and more research is needed to establish efficient methods for increasing unpredictability of solutions in new and existing decision support systems. For example, here we have discussed only issues of selecting from 'feasible', but it could be that this includes quality being above some threshold.

References

1. Guruswami, V.: Rapidly mixing markov chains: A comparison of techniques (a survey) (2016)
2. Müller, T., Rudová, H., Barták, R.: Minimal perturbation problem in course timetabling. In: E. Burke, M. Trick (eds.) Practice and Theory of Automated Timetabling V, pp. 126–146. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
3. Parkes, A.J.: Clustering at the phase transition. In: Proceedings of the fourteenth national conference on artificial intelligence and ninth conference on Innovative applications of artificial intelligence (AAAI-97), pp. 340–345 (1997)

[1] See `https://en.wikipedia.org/wiki/Vertex_cycle_cover`

4. Sinclair, A., Jerrum, M.: Approximate counting, uniform generation and rapidly mixing markov chains. Information and Computation **82**(1), 93 – 133 (1989). DOI https://doi.org/10.1016/0890-5401(89)90067-9

Scheduling of an underground mine by combining logic-based Benders decomposition and a priority-based heuristic *

Emil Lindh[1], Kim Olsson[1], and Elina Rönnberg[1][0000−0002−2081−2888]

Department of Mathematics, Linköping University, SE-581 83 Linköping, Sweden
elina.ronnberg@liu.se

Abstract. Underground mining is a complex operation that requires careful planning. The short-term scheduling, which is the scheduling of the tasks involved in the excavation process, is an important part of the planning process. In this paper, we propose a new method for the short-term scheduling of cut-and-fill mines.

Our problem formulation includes a new aspect of the problem, which is to handle that different excavation locations of the mine can have different priorities. The inclusion of this aspect allows the user to control the output of the scheduling and to direct resources to the locations where they are most needed according to the long-term plans. Our solution method consists of two components: a priority-based heuristic that constructs a complete solution by iteratively solving partial scheduling problems containing subsets of tasks, and a logic-based Benders decomposition scheme for solving these partial problems.

The computational performance of the proposed method is evaluated on industrially relevant large-scale instances generated from data provided by the mining company Boliden. Comparisons are made both with applying a constraint programming solver instead of the logic-based Benders decomposition scheme and with applying a constraint programming solver directly on the complete problem. The results show that our method outperforms the other two methods and yields schedules with a shorter makespan. The used instances are made publicly available to support further research in this area.

Keywords: Underground-mine scheduling · Cut-and-fill mining · Logic-based Benders decomposition · priority-based heuristic

* The collaboration with Torbjörn Viklund and Rasmus Tammia at Boliden has been vital for this project. They have contributed to discussing the problem formulation, constructing relevant instances, and evaluating the schedules. The work is partly funded by the Center for Industrial Information Technology (CENIIT), Project-ID 16.05. Monolithic MIP and CP models for the studied problem were formulated and implemented in a student project by Daniel Cederberg, Jonatan Ek, Eric Felding, Emil Lindh, Kim Olsson, and Sam Olsson. The results from that project have been used for this paper with permission from the students.

1 Introduction

The excavation of an underground mine is a complex operation that requires careful planning. This planning is usually done in several phases with different time horizons. For an in-depth description of the planning phases, see [11]. The two components of the planning process that we address are the so-called *extraction plans* and the *short-term scheduling*. The extraction plans describe when a certain amount of ore from a certain part of the ore body should be excavated. The short-term scheduling then executes these plans by scheduling the activities and the machines involved in the excavation process.

In this paper, a new method is proposed for solving large-scale instances of a short-term scheduling problem for a cut-and-fill mine. Our problem formulation differs somewhat from previous work [11,13–15] as it includes a priority order between excavation locations. This extension of the formulation is proposed since, in practice, the urgency of the different excavation locations may differ. The reason for the differences can depend both on the nature of the extraction plan and on the progress made at each location, compared to what was expected. During excavation, activities are often postponed due to unforeseen events and the priority can therefore change between scheduling periods and it is important for the planners to be able to direct the resources to the locations where they are most needed.

Our solution method combines a priority-based heuristic with Logic-based Benders decomposition (LBBD). Computational results are provided for industrially relevant large-scale instances based on data from an operational mine. These instances have been made publicly available[1]. The project has been carried out in collaboration with the mining company Boliden and the paper is based on the master's thesis by the first two authors [6].

1.1 Problem definition

In cut-and-fill mining, the process of excavating a single volume of ore involves 11 tasks: *drilling*, *charging*, *blasting*, *ventilation*, *watering*, *loading*, *scaling*, *cleaning*, *shotcreting*, *bolting*, *facescaling* and *facecleaning*. In our problem formulation, the ventilation task is merged with the blasting task since none of them requires a machine; this results in having only 10 tasks to schedule. Together, these 10 tasks form an *excavation cycle*. A location where excavation takes place is called a *face*, and a mine can have several active faces in parallel.

Each task requires a specific type of machine and some tasks require the same machine type. There is a limited number of machines of each type and all machines of the same type are assumed to be identical. The tasks must be scheduled in the correct order at each face, and an excavation cycle cannot begin unless the previous cycle at the same face has been completed. Each day is divided into two work shifts, 06:30 - 14:30 and 15:30 - 00:00, and only the interruptible tasks can be started in one shift and finalised in the next. The

[1] https://gitlab.liu.se/eliro15/underground_mining_instances

shotcreting task requires a four-hour afterlag for the concrete to solidify before the next task can be performed. The blasting task can only occur during specific blasting windows between the work shifts, starting at either 15:00 or 00:42 and lasting 30 minutes. During a blasting window, the mine has to be evacuated due to safety reasons. Some tasks can be interrupted during the blasting windows while others cannot.

A task in a specific cycle that is to be scheduled at a specific face will henceforth be referred to as a *task instance*. Each machine type has an estimated velocity and when moving between two task instances at different faces, there is a travel time between the faces. The scheduling considers a number of excavation cycles at a number of faces and aims at minimising the sum of the individual makespans for the faces, while respecting the above-introduced constraints. A more in-depth problem description is found in [11] and [6].

1.2 Related work

The short-term scheduling problem considered in this work is not well-studied. There is however a series of papers from a PhD thesis [11] that addresses industrially relevant instances of underground mining problems. In this series of work, the scheduling of cut-and-fill mining is structured as a flow shop [13] and modelled by a CP formulation [14]. In [15], the problem formulation and the model of [13] and [14] are improved to better capture the characteristics of cut-and-fill mining, and the authors propose heuristics based on large neighbourhood search for solving the problem. The instances used in this series of work are unfortunately not available for further research or comparisons in this paper. Instead, we have made a re-implementation of their CP model to use for benchmarking. In [12], their results are extended to a more general underground-mining setting.

In [9] and [7], mixed-integer programming (MIP) approaches were applied to similar scheduling problems. In [9], the authors present a MIP model for a makespan minimizing mobile production fleet scheduling problem for a room-pillar mine. They solve small instances to optimality using a commercial solver and present heuristic methods for solving instances of larger sizes. In [7], the authors study a scheduling problem for a sublevel stoping mine, which includes the transportation of ore, and solve it using a MIP model and CPLEX. In [10], a scheduling support instrument is developed that schedules some of the activities in a production cycle.

To the best of our knowledge, there is no previous work where LBBD has been applied to this type of scheduling problem. However, LBBD has been successfully applied to other scheduling problems [1,2,5,8].

1.3 Contributions and outline

Today, the short-term scheduling of the excavation process at Boliden is done manually. As manual scheduling is a time-consuming and complex operation that does not guarantee consistency in production efficiency, a more autonomous way of scheduling can yield great improvements. Heuristic methods for solving realistic instances of this scheduling problem, with the objective to minimise makespan, have been developed [11]. However, many questions remain with respect to how to model and efficiently solve this problem. In particular, one crucial aspect for applicability in practice is the choice of objective function and how to take into account how the excavation progresses over time. Our work contributes to improved modelling of the problem and with a new strategy for solving it. These contributions are based on the following two fundamental observations about the problem that we have not seen addressed in previous work.

The first observation is that the problem has some inherent symmetries due to the machines being identical and the excavation cycles and the durations of tasks being the same for all faces. Even if the different travel times between faces contribute to breaking some of these symmetries, several solutions with the same or very similar makespan are likely to exist. A practical consequence of this property is that there is room for the decision-maker to choose between several plans with about the same efficiency in terms of length of makespan – which is the most important evaluation measure for a schedule.

Through discussions with practitioners, we learned that because of how the mining progresses and because of other aspects of the planning, the urgency of the faces can differ. This suggests that, in addition to the objective to minimise makespan, it is desirable to consider that there is a priority order between the faces. Therefore, we propose a priority-based heuristic to incorporate face priorities, while keeping the original makespan objective both in the models and as an evaluation measure for the solutions. Using this evaluation measure allows for a comparison with previous work. By using the priority-based heuristic, we both increase the decision-maker's control of the scheduling output and improve the computational times thanks to the symmetry breaking effect. This aspect of the model and method design is evaluated through a comparison between directly applying a CP solver using a carefully crafted model from previous work [11], referred to as M-CP, and applying an adaptation of this model in each iteration of the priority-based heuristic, referred to as H-CP.

The second observation is that by slightly reformulating the makespan objective, one obtains a formulation that lends itself to LBBD with a feasibility subproblem. This new objective was also developed in dialogue with Boliden. It is less granular than the standard objective function that directly aims to minimise the makespan and, instead, the focus is on which shifts to use to efficiently perform the tasks. Using the new objective, we introduce a LBBD scheme and tailored acceleration techniques to be applied in each iteration of the priority-based heuristic. The resulting approach is referred to as H-LBBD. Since the LBBD scheme is applied within the heuristic, its efficiency is evaluated in a direct comparison with approach H-CP.

The models and the heuristic are presented in Section 2 and the LBBD scheme, along with the acceleration techniques, is introduced in Section 3. Computational results are presented in Section 4. Conclusions and comments about possible future work are given in Section 5.

2 Modelling and priority-based heuristic

This section presents a complete CP model of the problem and the heuristic.

2.1 Monolithic CP-model

The CP model presented here is a re-implementation of the models presented in [11], and we refer to the previous work for a detailed motivation and description of the models. In previous work, the problem is described in a k-stage hybrid flow shop framework [13] where a job corresponds to an excavation cycle which is passed through k stages, each stage corresponding to a certain task within the excavation cycle. The model is adapted to the specific problem structure by including travel times and by removing time periods from the schedule in which the mine is evacuated or empty, i.e blasting windows and shift breaks. The latter results in a problem formulation in compressed time, and the solution is later post-processed to obtain the solution in the correct time. This compressed-time reformulation was introduced in [15], where a detailed description is found.

The notation used in the CP model is as follows. Let the set $\mathcal{M}$ index the machines, the set $\mathcal{F}$ index the faces, and the set $\mathcal{T} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ index the tasks. Further, let J be the maximum number of cycles to schedule and let the set $\mathcal{J}_f \subseteq \{1, \ldots, J\}$ index the cycles at face $f \in \mathcal{F}$. A task instance is specified by a task $t \in \mathcal{T}$ at a face $f \in \mathcal{F}$ in a cycle $j \in \mathcal{J}_f$. To simplify notation, we introduce a mapping of the indices of such task instance (tfj) to a single index a and let $\mathcal{A}$ contain all task instances. Let F_a denote the face at which task instance $a \in \mathcal{A}$ is to be scheduled. Furthermore, let the set $\mathcal{B}$ index the blasting windows b and let s_b be the time when blasting window $b \in \mathcal{B}$ starts and ends (these times coincides in the time-compressed model).

Let the set $\mathcal{M}_a \subseteq \mathcal{M}$ index the machines which can execute task instance $a \in \mathcal{A}$ and let the set $\mathcal{A}^{\mathrm{m}}_m$ index the task instances that machine $m \in \mathcal{M}$ can perform. Also, let the sets $\mathcal{A}^{(\mathrm{un})} \subseteq \mathcal{A}$, $\mathcal{A}^{(\mathrm{al})} \subseteq \mathcal{A}$, and $\mathcal{A}^{(\mathrm{bl})} \subseteq \mathcal{A}$ index the uninterruptible task instances, task instances that require afterlag, and blasting task instances, respectively. For each face $f \in \mathcal{F}$, let the ordered set $\mathcal{A}^{\mathrm{f}}_f$ index all the task instances, except the last one, in their execution order at face f.

Denote the duration of task instance $a \in \mathcal{A}$ by D_a and the duration of blasting window $b \in \mathcal{B}$ by $d^{(\mathrm{b})}$. Let the duration of the afterlag be denoted by $d^{(\mathrm{al})}$ and let the travel time between faces f, $f' \in \mathcal{F}$, be $l_{ff'}$. For each $a \in \mathcal{A}$ and $m \in \mathcal{M}_a$ the optional interval variable

$$I_{am} = (s_{am}, \mathbf{e_{am}}, D_a, \mathbf{o}_{am})$$

holds the start time s_{am}, end time $\mathbf{e_{am}}$, and duration D_a of task instance a. Also, it indicates by $\mathbf{o}_{am}$ whether machine m performs task instance a or not. To handle the compressed time, a variable $\mathbf{d}^{(\mathrm{al})}_a$ is introduced to calculate the correct afterlag. The variable $\mathbf{nextBlast}_{am}$ provides the next possible blast occasion following task instance $a \in \mathcal{A}$ executed by machine $m \in \mathcal{M}_a$. Lastly, a precedence variable for $a,\ a' \in \mathcal{A}^{\mathrm{m}}_{m},\ m \in \mathcal{M}$ is defined as

$$\mathbf{b}_{aa'} = \begin{cases} 1 & \text{if task instance } a \text{ precedes task instance } a' \text{ on machine } m, \\ 0 & \text{if task instance } a' \text{ precedes task instance } a \text{ on machine } m, \\ \perp & \text{if task instance } a \text{ and } a' \text{ are done by different machines.} \end{cases}$$

The complete CP-model is

$$\min \sum_{f \in \mathcal{F}} \max_{\substack{a \in \mathcal{A}^{\mathrm{f}}_f \\ m \in \mathcal{M}_a}} \Big(\mathbf{o}_{am} \times \mathbf{e}_{am} \Big),$$

$$\text{s.t.} \sum_{m \in \mathcal{M}_a} \mathbf{o}_{am} = 1, \quad a \in \mathcal{A}, \tag{1a}$$

$$s_{am} \in \{s_b | b \in \mathcal{B}\}, \quad a \in \mathcal{A}^{(\mathrm{bl})}, m \in \mathcal{M}_a, \tag{1b}$$

$$s_{am} \in \bigcup_{b \in \mathcal{B}} \{s_{b-1}, \dots, s_b - D_a\}, \quad a \in \mathcal{A}^{(\mathrm{un})}, m \in \mathcal{M}_a, \tag{1c}$$

$$\mathbf{b}_{aa'} = 1 \iff$$
$$\mathbf{o}_{am} \wedge \mathbf{o}_{a'm} \ \wedge s_{am} + D_a + l_{F_a F_{a'}} < s_{a'm} \quad m \in \mathcal{M}, a, a' \in \mathcal{A}^{\mathrm{m}}_{m}, \tag{1d}$$

$$\mathbf{b}_{aa'} = 0 \iff$$
$$\mathbf{o}_{am} \wedge \mathbf{o}_{a'm} \ \wedge s_{a'm} + D_{a'} + l_{F_{a'} F_a} < s_{am}, \quad m \in \mathcal{M}, a, a' \in \mathcal{A}^{\mathrm{m}}_{m}, \tag{1e}$$

$$\mathbf{b}_{aa'} = \perp \iff \neg(\mathbf{o}_{tm} \wedge \mathbf{o}_{t'm}), \quad m \in \mathcal{M}, a, a' \in \mathcal{A}^{\mathrm{m}}_{m}, \tag{1f}$$

$$\mathbf{nextBlast}_{am} \in \{s_b | b \in B\}, \quad a \in \mathcal{A}^{(\mathrm{al})}, \tag{1g}$$

$$\mathbf{d}^{(\mathrm{al})}_a = \begin{cases} d^{(\mathrm{al})} & \text{if } s_{am} + D_a + d^{(\mathrm{al})} < \mathbf{nextBlast}_{am} \\ d^{(\mathrm{al})} - d^{(\mathrm{b})} & \text{if } s_{am} + D_a + d^{(\mathrm{al})} > \mathbf{nextBlast}_{am} + d^{(\mathrm{b})} \\ d^{(\mathrm{al})} - \Delta d^{(\mathrm{al})} & \text{otherwise}, \quad a \in \mathcal{A}^{(\mathrm{al})}, \end{cases} \tag{1h}$$

$$s_{am} + D_a + \mathbf{d}^{(\mathrm{al})}_a \leq s_{(a+1)m}, \quad f \in \mathcal{F}, a \in \mathcal{A}^{\mathrm{f}}_f \cap \mathcal{A}^{(\mathrm{al})}, m \in \mathcal{M}_a, \tag{1i}$$

$$s_{am} + D_a \leq s_{(a+1)m}, \quad f \in \mathcal{F}, a \in \mathcal{A}^{\mathrm{f}}_f \setminus \mathcal{A}^{(\mathrm{al})}, m \in \mathcal{M}_a, \tag{1j}$$

where $\Delta d^{(\mathrm{al})} = s_{am} + D_a + d^{(\mathrm{al})} - \mathbf{nextBlast}_{am}$.

2.2 Priority-based heuristic

The main principle of the heuristic is to iteratively extend and solve the problem with more task instances and to fix a majority – but not all – of the decisions made in each iteration. To define the iterations of the heuristic, we introduce a batching of the tasks as illustrated in Figure 1. This batching is done such that there is a reasonable amount of decisions to be made for each batch and so that these decisions can be made in isolation for the current batch, even if they of course have dependencies to other decisions. Note especially that, for this reason, a batch always ends with an uninterruptible task.

In the first iteration of the heuristic, two batches are scheduled and thereafter, one batch at a time is added to the problem – and each such addition

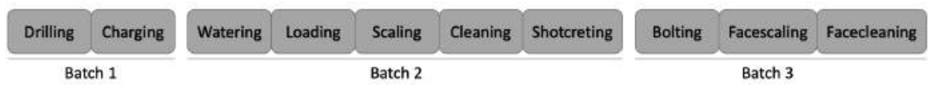


Fig. 1. The batching of tasks within a cycle

corresponds to one iteration of the heuristic. The order in which to add the batches is determined by the excavation cycle number, the priority of the face, and the batch number. Among these, the cycle number always has precedence, which means that all batches on all faces for a certain cycle number are scheduled before a next cycle is considered. For each cycle number, the order is primarily determined by the face priority, and for each face, its batches are added one at a time in accordance with the batch number.

A more formal description of the heuristic is presented through pseudo-code in Algorithm 1. Additional notation used in this description is that we let $\mathcal{F}^{(\mathrm{prio})}$ be an ordered set of face indices, given in the order of decreasing face priority, and that we let b_i contain the tasks in batch $i \in \{1, 2, 3\}$. The scheduling within each iteration is done either by applying a CP solver to the model introduced in Section 2.1 (but adapted to batch-wise scheduling) or by using the LBBD scheme to be introduced in Section 3. In Algorithm 1, such scheduling is referred to as applying a `scheduling_method`.

Algorithm 1 Priority-based heuristic

```
1: **Input:** $J$; $\mathcal{J}_f$, $f \in \mathcal{F}$; $\mathcal{F}^{(\mathrm{prio})}$; $b_1, b_2, b_3$; scheduling_method
2: for $j = 1, \dots, J$ do
3:     $k = 1$
4:     while $k \le |\mathcal{F}^{(\mathrm{prio})}|$ do
5:         if $k = 1$ and $j = 1$ then
6:             for $i \in \{1, 2, 3\}$ do
7:                 add batch $b_i$ of cycle $j = 1$ at faces $f_1, f_2 \in \mathcal{F}^{(\mathrm{prio})}$
8:                 apply scheduling_method
9:             $k = k + 2$
10:        else
11:            for $i \in \{1, 2, 3\}$ do
12:                add batch $b_i$ of cycle $j \in \mathcal{J}_f$ at face $f \in \mathcal{F}^{(\mathrm{prio})}$
13:                apply scheduling_method
14:            $k = k + 1$
```

In detail, a batch is scheduled by applying a `scheduling_method` on a current partial problem, which includes all of the previously scheduled batches as well as the current batch. The `scheduling_method` takes the scheduling decisions for the current batch, but the previously scheduled batches are included since the current batch has to be scheduled in relation to the already scheduled batches, e.g to be able to respect travel times. Once the `scheduling_method` has been applied, some scheduling decisions for the current batch will be fixed. When applying the LBBD scheme, these scheduling decisions are the machine assignments and the work shift assignments for each task instance. When applying a CP solver directly, these decisions are the machine assignments and the start times of each task instance.

3 Logic-based Benders decomposition

This section introduces an LBBD scheme designed to be used as a `scheduling_method` in the priority-based heuristic presented in Section 2.2. In the monolithic CP model, the objective is to minimise the sum of makespans of the faces, with the makespan of a face defined as the time when its last task instance is completed. In dialogue with Boliden, we concluded that this commonly used scheduling objective might not best reflect the decision problem to be solved in practice. Instead, we decided to focus on which shifts to schedule the task instances in, choosing as early shifts as possible, and then minimise the total duration of the task instances scheduled in the last shift used for a face. An important benefit of using this objective is that it allows for designing an LBBD scheme where the subproblem is a feasibility problem.

The decomposition is made such that in a MIP master problem, task instances are assigned to work shifts and machines are assigned to task instances. A CP subproblem is then used to schedule the tasks within the work shifts, respecting the machine assignments. Both problems are formulated in compressed time as introduced in [15]. The LBBD scheme iterates between solving the master problem and the subproblem, and if the subproblem is infeasible, a no-good cut or a set of no-good cuts are fed back to the master problem. These LBBD iterations are continued until the subproblem becomes feasible and then the master problem decisions for the current batch are fixed and returned to the heuristic. The values of the variables in the subproblem are never fixed since it is profitable if these can be adjusted depending on future master problem decisions. To enhance the performance of the LBBD scheme, cut-strengthening and problem-specific cuts are introduced, and the master problem is formulated to include a subproblem relaxation.

3.1 Master problem

The role of the master problem is to assign task instances to work shifts and machines to task instances. To formulate the MIP model, the following additional notation is used. Let the set $\mathcal{W}$ index all work shifts and let the set $\mathcal{W}^1$ index all work shifts except the first one. Also, let the set $\mathcal{M}_t \subseteq \mathcal{M}$ index the subset of machines that can perform task $t \in \mathcal{T}$.

To handle the batches, let the set $\mathcal{F}^{(\mathrm{ba})} \subseteq \mathcal{F}$ index the currently and previously scheduled faces, and let the set $\mathcal{J}_f^{(\mathrm{ba})} \subseteq \mathcal{J}_f$ index the currently and previously scheduled cycles at face $f \in \mathcal{F}^{(\text{ba})}$. Also, let the set $\mathcal{T}_{fj} \subseteq \mathcal{T}$ index the tasks in cycle $j \in \mathcal{J}_f^{(\text{ba})}$ at face $f \in \mathcal{F}^{(\text{ba})}$ that are included in the currently and previously scheduled batches and denote the last included task in each $\mathcal{T}_{fj}$ by $\bar{t}_{fj}$. The main decision variables of the master problem are, for $t \in \mathcal{T}_{fj}$, $f \in \mathcal{F}^{(\text{ba})}$, $j \in \mathcal{J}_f^{(\text{ba})}$, $w \in \mathcal{W}$ and $m \in \mathcal{M}_t$,

$$x_{tfjwm} = \begin{cases} 1 & \text{if task } t \text{ at face } f \text{ in cycle } j \text{ is assigned to work shift } w \text{ and} \\ & \text{performed by machine } m, \\ 0 & \text{otherwise.} \end{cases}$$

The duration of a task $t \in \mathcal{T}$ is D_t and each work shift $w \in \mathcal{W}$ has a length $D_w = e_w - s_w$, where e_w and s_w are the end and start times of the work shift, respectively. The second longest duration of all tasks is denoted p and used as an aid when fitting task instances into work shifts. Some constraints are specific to batch 2 and the parameter B_{fj} is used to indicate if batch 2 has been or is currently being scheduled in cycle $j \in \mathcal{J}_f^{(\text{ba})}$ at face $f \in \mathcal{F}^{(\text{ba})}$.

For $f \in \mathcal{F}^{(\text{ba})}$, $j \in \mathcal{J}_f^{(\text{ba})}$, $w \in \mathcal{W}$ and $m \in \mathcal{M}_6$ and given that $B_{fj} = 1$ the following variables are defined to account for afterlag in the master problem,

$$y_{fjwm} = \begin{cases} 1 & \text{if task 6 and task 7 are assigned to the same work shift,} \\ 0 & \text{otherwise.} \end{cases}$$

The face and cycle last added to $\mathcal{F}^{(\text{ba})}$ and $\mathcal{J}_f^{(\text{ba})}$, respectively, are denoted by $\bar{f}$ and $\bar{j}_{\bar{f}}$. The objective value is represented by the auxiliary variable $o_{\bar{f}}$ which is defined by constraint (2n) below. The master problem is

$$\min \; o_{\bar{f}},$$

$$\text{s.t.} \sum_{w \in \mathcal{W}, m \in \mathcal{M}_t} x_{tfjwm} = 1, \; t \in \mathcal{T}_{fj}, f \in \mathcal{F}^{(\text{ba})}, j \in \mathcal{J}_f^{(\text{ba})}, \tag{2a}$$

$$\sum_{w \in W, m \in \mathcal{M}_{t-1}} s_w x_{(t-1)fjwm} \leq \sum_{w \in W, m \in \mathcal{M}_t} s_w x_{tfjwm}, \; t \in \mathcal{T}_{fj}^{\text{l}}, f \in \mathcal{F}^{(\text{ba})}, j \in \mathcal{J}_f^{(\text{ba})}, \tag{2b}$$

$$\sum_{w \in W, m \in \mathcal{M}_{t-1}} e_w x_{(t-1)fjwm} \leq \sum_{w \in W, m \in \mathcal{M}_t} s_w x_{tfjwm}, \; t \in \tilde{\mathcal{T}}_{fj}^{\text{c}}, f \in \mathcal{F}^{(\text{ba})}, j \in \mathcal{J}_f^{(\text{ba})}, \tag{2c}$$

$$\sum_{w \in W, m \in \mathcal{M}_9} s_w x_{9f(j-1)wm} \leq \sum_{w \in W, m \in \mathcal{M}_0} s_w x_{0fjwm}, \; f \in \mathcal{F}^{(\text{ba})}, j \in \mathcal{J}_f^{(\text{ba}),\text{l}}, \tag{2d}$$

$$\begin{aligned} &\sum_{m \in \mathcal{M}_t, t \in \mathcal{T}_{fj}^{\text{c}}} D_t x_{tfjwm} + \sum_{m \in \mathcal{M}_1} (D_1 + p) x_{1fjwm} + \\ &\quad B_{fj} \Big(\sum_{m \in \mathcal{M}_7} p x_{7fjwm} + \sum_{m \in \mathcal{M}_6} d^{(\text{al})} y_{fjwm} \Big) \leq D_w + p, \\ &\quad f \in \mathcal{F}^{(\text{ba})}, j \in \mathcal{J}_f^{(\text{ba})}, w \in \mathcal{W}, \end{aligned} \tag{2e}$$

$$\sum_{m \in \mathcal{M}_t, t \in \mathcal{T}_{fj}} D_t x_{tfj(w-1)m} + \sum_{m \in \mathcal{M}_t, t \in \mathcal{T}_{fj}^{\text{c}}} D_t x_{tfjwm} + \sum_{m \in \mathcal{M}_1} (D_1 + p) x_{1fjwm} +$$

$$B_{fj}\Big(\sum_{m\in\mathcal{M}_6} d^{(\mathrm{al})}\left(y_{fj(w-1)m}+y_{fjwm}\right)+\sum_{m\in\mathcal{M}_7} p x_{7fjwm}\Big)\leq$$

$$e_w - s_{(w-1)} + p,\ f\in\mathcal{F}^{(\mathrm{ba})}, j\in\mathcal{J}_f^{(\mathrm{ba})}, w\in\mathcal{W}^{\mathrm{l}}, \tag{2f}$$

$$B_{fj}\sum_{t\in\mathcal{T}_s, m\in\mathcal{M}_t}(x_{tfj(w-1)m}+x_{tfjwm})\leq 3,\ w\in\mathcal{W}, f\in\mathcal{F}^{(\mathrm{ba})}, j\in\mathcal{J}_f^{(\mathrm{ba})}, \tag{2g}$$

$$B_{fj}\sum_{m\in\mathcal{M}_7}(x_{7fjwm}+x_{6fjw\tilde{m}}-y_{fjw\tilde{m}}-1)\leq 0,$$

$$f\in\mathcal{F}^{(\mathrm{ba})}, j\in\mathcal{J}_f^{(\mathrm{ba})}, w\in\mathcal{W}, \tilde{m}\in\mathcal{M}_6, \tag{2h}$$

$$B_{fj}(y_{fjwm}-x_{6fjwm})\leq 0,\ f\in\mathcal{F}^{(\mathrm{ba})}, j\in\mathcal{J}_f^{(\mathrm{ba})}, w\in\mathcal{W}, m\in\mathcal{M}_6, \tag{2i}$$

$$B_{fj}\sum_{m\in\mathcal{M}_7}(x_{7fjwm}-y_{fjw\tilde{m}})\geq 0,$$

$$f\in\mathcal{F}^{(\mathrm{ba})}, j\in\mathcal{J}_f^{(\mathrm{ba})}, w\in\mathcal{W}, \tilde{m}\in\mathcal{M}_6, \tag{2j}$$

$$\sum_{j\in\mathcal{J}_f^{(\mathrm{ba})}, f\in\mathcal{F}^{(\mathrm{ba})}} x_{7fjwm}\leq 1,\ w\in\mathcal{W}, m\in\mathcal{M}_7, \tag{2k}$$

$$[\text{no-good cuts}], \tag{2l}$$

$$[\text{problem-specific cuts}], \tag{2m}$$

$$\sum_{m\in\mathcal{M}_{\bar{t}}} s_w x_{\bar{t}\bar{f}\bar{j}wm}+\sum_{t\in\mathcal{T}_{\bar{f}\bar{j}}, m\in\mathcal{M}_t} D_t x_{t\bar{f}\bar{j}wm}\leq o_{\bar{f}},\ w\in\mathcal{W}, \tag{2n}$$

where for, $f\in\mathcal{F}^{(\mathrm{ba})}$ and $j\in\mathcal{J}_f^{(\mathrm{ba})}$, $\mathcal{T}^{\mathrm{c}}_{fj}=\mathcal{T}_{fj}\setminus\{2,8\}$, $\mathcal{T}^{\mathrm{l}}_{fj}=\mathcal{T}_{fj}\setminus\{0\}$ and $\tilde{\mathcal{T}}^{\mathrm{c}}_{fj}=\mathcal{T}_{fj}\cap\{2,8\}$. Additionally, $\bar{t}$, $\bar{f}$, and $\bar{j}$ denote the last task, face, and cycle, respectively, in the batches currently considered, and $\mathcal{J}_f^{(\mathrm{ba}),\mathrm{l}}$ is the set of cycles in $\mathcal{J}_f^{(\mathrm{ba})}$ excluding the first cycle and $\mathcal{T}_s=\{6,7,8,9\}\subset\mathcal{T}$.

The objective essentially directs the master problem to schedule each task instance in the earliest possible work shift, and to leave as short total task duration as possible for the last used shift. Constraint (2a) assigns exactly one machine to each task instance. Constraints (2b) and (2d) make sure that consecutive task instances are assigned to the same or to consecutive work shifts, while constraint (2c) prevents the pairs *charging* and *watering*, and *bolting* and *facescaling*, respectively, to be assigned to the same work shifts since it is impossible for those pairs of task instances to be scheduled in the same shift. Constraints (2e), (2f) and (2g) restrict which task instances that can be placed in the same work shifts, essentially removing some assignments that will be infeasible in the subproblem. Constraints (2h)–(2j) define the value of the afterlag variable. Constraint (2k) bounds the number of task instances a bolting machine can be assigned to in a work shift. Finally, the no-good cuts and the problem-specific cuts (to be introduced in Section 3.3) generated so far in the solution process are represented by (2l) and (2m), respectively. Note that constraints (2c) and (2h)–(2k) form a subproblem relaxation that strengthens the master problem.

3.2 Subproblem

A CP solver is applied to solve the subproblem, using a model derived from the monolithic model in Section 1. The adjustments made to account for the master problem decisions are as follows. Let the set $\mathcal{A}^{\text{curr}}$ index the task instances that are to be scheduled by the subproblem and let $\bar{w}_a$ denote the work shift assigned to task instance $a \in \mathcal{A}^{\text{curr}}$. The machine assigned to task instance $a \in \mathcal{A}^{\text{curr}}$ is denoted by $\bar{m}_a$ and the set that includes the indices of the used machines is denoted by $\bar{\mathcal{M}}$. Due to the fix machine assignments, the interval variables are no longer optional and constraint (1a) is not needed in the subproblem model. Furthermore, the domains of the interval variables will be restricted to their assigned work shifts by the constraints (3a) and (3b) and for this reason, constraint (1c) is not needed in the subproblem. Moreover, the scheduling of the blasting tasks does not need to be handled in the subproblem, since the master problem assignment of the charging and watering tasks imply when blasting will occur. Hence, constraint (1b) does not need to be included in the subproblem and the **nextBlast** variables are fixed, resulting in that also constraint (1g) is redundant to include in the subproblem model. Lastly, the subproblem has no objective function since the objective depends only on master problem decisions. Hence, the constraints to be included in the subproblem model are

$$\mathbf{startOf}(I_{a\bar{m}_a}) \in \{s_{\bar{w}_a}, \ldots, e_{\bar{w}_a} - D_a\},\ a \in \mathcal{A}^{(\text{un})} \cap \mathcal{A}^{\text{curr}}, \tag{3a}$$

$$\mathbf{startOf}(I_{a\bar{m}_a}) \in \{s_{\bar{w}_a}, \ldots, e_{\bar{w}_a} + D_a\},\ a \notin \mathcal{A}^{(\text{un})} \cap \mathcal{A}^{\text{curr}}, \tag{3b}$$

$$\mathbf{b}_{aa'} = 1 \iff \mathbf{s}_{a\bar{m}_a} + D_a + l_{F_a F_{a'}} < \mathbf{s}_{a'\bar{m}_a},\ m \in \bar{\mathcal{M}},\ a, a' \in \mathcal{A}^{\text{m}}_{m} \cap \mathcal{A}^{\text{curr}}, \tag{3c}$$

$$\mathbf{b}_{aa'} = 0 \iff \mathbf{s}_{a'\bar{m}_{a'}} + D_{a'} + l_{F_{a'} F_a} < \mathbf{s}_{a\bar{m}_a},\ m \in \bar{\mathcal{M}},\ a, a' \in \mathcal{A}^{\text{m}}_{m} \cap \mathcal{A}^{\text{curr}}, \tag{3d}$$

$$\mathbf{d}^{(\text{al})}_a = \begin{cases} d^{(\text{al})} & \text{if } \mathbf{s}_{a\bar{m}_a} + D_a + d^{(\text{al})} < \mathbf{nextBlast}_{a\bar{m}_a} \\ d^{(\text{al})} - d^{(\text{b})} & \text{if } \mathbf{s}_{a\bar{m}_a} + D_a + d^{(\text{al})} > \mathbf{nextBlast}_{a\bar{m}_a} + d^{(\text{b})} \\ d^{(\text{al})} - \Delta d^{(\text{al})} & \text{otherwise}, \quad a \in \mathcal{A}^{(\text{al})} \cap \mathcal{A}^{\text{curr}}, \end{cases} \tag{3e}$$

$$\mathbf{s}_{a\bar{m}_a} + D_a + \mathbf{d}^{(\text{al})}_a \leq \mathbf{s}_{(a+1)\bar{m}_{a+1}},\ f \in \mathcal{F}, a \in \mathcal{A}^{\text{f}}_{f} \cap \mathcal{A}^{\text{curr}} \cap \mathcal{A}^{(\text{al})}, \tag{3f}$$

$$\mathbf{s}_{a\bar{m}_a} + D_a \leq \mathbf{s}_{(a+1)\bar{m}_{a+1}},\ f \in \mathcal{F}, a \in \mathcal{A}^{\text{f}}_{f} \cap \mathcal{A}^{\text{curr}} \setminus \mathcal{A}^{(\text{al})}, \tag{3g}$$

where $\Delta d^{(\text{al})} = \mathbf{s}_{am} + D_a + d^{(\text{al})} - \mathbf{nextBlast}_{am}$.

3.3 Feasibility cuts

Whenever the subproblem is infeasible, that information is fed back to the master problem in form of a no-good cut [3,4] on the form

$$1 - \sum_{a \in \bar{\mathcal{A}}} x_{a\bar{w}_a\bar{m}_a} \geq 1,$$

using the mapping $a = (tfj)$ in the definition of the variable x and the set $\bar{\mathcal{A}}$ to index the task instances included in the cut. For a no-good cut that originates from solving the subproblem, $\bar{\mathcal{A}} = \mathcal{A}^{\text{curr}}$ holds. Since such cut is not very strong, we designed and applied the following two cut-strengthening methods.

The first cut-strengthening method, presented in Algorithm 2, is shift-based and applied in the first iteration of the heuristic when the subproblem is still of small size. It strengthens a no-good cut through three steps: (I) Start at work shift 0 and add one work shift at a time to the subproblem, until it becomes infeasible. (II) Apply a *greedy cut strengthening algorithm* [4] with respect to the added work shifts, starting at shift 0. (III) Apply a *deletion filter* [3, 4] on the task instances assigned to the remaining shifts. This method gives an irreducible cut since, in the last step, a deletion filter is applied to a subset of task instances that yielded an infeasible subproblem.

Algorithm 2 Shift-based method

```
1: let $w = 0$
2: **while** subproblem feasible **do**
3:     add all task instances assigned to work shift $w$ to the subproblem
4:     solve subproblem
5:     let $w = w + 1$
6: let $w = 0$
7: **while** subproblem infeasible **do**
8:     remove all task instances assigned to work shift $w$ from the subproblem
9:     solve subproblem
10:    let $w = w + 1$
11: apply a deletion filter on the remaining task instances
```

The cuts obtained by applying Algorithm 2 are also used to derive additional no-good cuts by creating permutations of the machine assignments, using the fact that the machines are identical. For example, if one of the drillrigs cannot execute two tasks instances, then neither can any of the other two drillrigs.

The second cut-strengthening method, presented in Algorithm 3, is applied in all but the first iteration of the heuristic. This method is batch-based and designed to efficiently handle the addition of new batches as they are added by the heuristic. It strengthens a cut by applying a deletion filter on the task instances of the current batch. Assignments from previous batches are already proven to be feasible before the addition of the current batch, but because of dependencies between the batches, the current one cannot be evaluated in isolation. However, to sustain a moderate size of the subproblems to be solved during cut strengthening also in later iterations of the heuristic, only a few work shifts and their task instances from previous batches are included. The risk of this is that the strengthening can fail and the original cut needs to be used, but in practice, this never became the case.

Algorithm 3 Batch-based method

```
1: let $\mathcal{B}^{(\mathrm{curr})}$ contain the task instances in the current batch
2: let $\tilde{w}$ be the earliest work shift of a task instance in $\mathcal{B}^{(\mathrm{curr})}$
3: add all task instances in work shifts $\tilde{w}-4,\ldots,\tilde{w}$ to the subproblem
4: add all task instances $\mathcal{B}^{(\mathrm{curr})}$ to the subproblem
5: if subproblem infeasible then
6:     apply a deletion filter with respect to the task instances in $\mathcal{B}^{(\mathrm{curr})}$
```

Problem specific cuts During the computational experiments, it was discovered that the master problem was particularly weak with respect to one aspect of the assignments and that many no-good cuts were required to reach feasibility due to this. The aspect originates from the limited number of machines to use for the tasks in batch 2. Specifically, if all task instances of two batches of batch 2 are scheduled within the same work shift, it is not possible to schedule all task instances of a next batch 2 within this shift.

During the solution process, it is possible to keep track of the work shifts that have been assigned two complete batches of batch 2. Let the set $\bar{\mathcal{W}}$ index such work shifts and let $\tilde{f}$ and $\tilde{j}$ be the face and cycle of the second batch that is currently scheduled. The problem-specific cut

$$\sum_{t\in b_2}\sum_{m\in\mathcal{M}_t} x_{t\tilde{f}\tilde{j}wm} \le 4, \qquad w\in\bar{\mathcal{W}},$$

ensures that, for each work shift with at least two batches of batch 2 already scheduled, at least one task instance from the currently scheduled batch 2 is assigned to another work shift.

4 Computational results

This section presents computational results from the implementation and evaluation of the following three solution methods.

- **M-CP:** Apply a CP solver directly on a monolithic model from previous work. The model is described in Section 2.1.
- **H-CP:** Apply the priority-based heuristic and, in each iteration, use a CP solver on an adaptation of the model in Section 2.1.
- **H-LBBD:** Apply the priority-based heuristic and, in each iteration, use the LBBD scheme introduced in Section 3.

The evaluation measures that we use are computational times and the sum of the makespans for the individual faces, henceforth simply referred to as makespan. Important to note is that the only method that actually uses makespan as the sole objective is M-CP, while both H-CP and H-LBBD include a priority order, not as part of the mathematical model but through the heuristic. In addition, H-LBBD uses a slightly changed objective function as part of the LBBD scheme design. This means that for H-LBBD, the makespan is computed after the solution is returned. In our comparison, M-CP is considered as the benchmark method, both because the method is exact and because the objective of the model is the same as our measure for evaluating the quality of the schedule. Furthermore, since M-CP stems from a re-implementation of a model in previous work, it is also the best comparison we could make in this respect.

For the evaluations, we used 6 industrially relevant instances constructed in dialogue with Boliden. These have been made publicly available and are described in more detail in Section 4.1 together with a description of computational settings. To evaluate the impact of the priority-based heuristic, each instance was solved for three different priority orders, denoted P1, P2, and P3. In P1, there is an increasing priority from the first to the last face, and for the other two, there is a randomly chosen order.

The first set of computational results, presented in Section 4.2, provides a comparison between H-CP and H-LBBD to evaluate the impact of applying the LBBD scheme instead of a CP solver in each iteration of the heuristic. This comparison is direct in terms of only evaluating the use of the LBBD scheme. Since the results of H-LBBD are much stronger than those of H-CP, the latter is omitted from further comparisons. The second set of computational results gives a comparison between H-LBBD and M-CP and is presented in Section 4.3.

4.1 Instances and computational settings

All our instances are constructed for a given mine topology that defines the distances between the different faces and for a given machine park. Each instance is characterised by the number of parallel excavation faces and the number of excavation cycles at each face. They have been encoded with #**F**:#**C**, e.g an instance with 6 parallel faces and 4 cycles at each face is encoded **6F:4C**, in line with notation used previous work [11]. Most of our instances consist of 24 cycles in total, since this gives a realistic instance size according to Boliden. An instance with a specific characteristic is **18F:XC** that has a total of 24 cycles, where faces 1 to 12 have one cycle and 13 to 18 have two cycles. Since the choice of the starting task at each face has a large impact on the objective value, we use the same starting task, drilling, for all faces and all instances.

The number of task instances that a problem instance contains is #**F** $\times$ #**C** $\times$ 10. For example, **6F:4C** include $6 \times 4 \times 10 = 240$ task instances. The machine park used in our instances consists of seven different kinds of machines, 2 *drillrigs*, 2 *chargers*, 2 *watering trucks*, 5 *loaders*, 3 *scalers*, 2 *shotcreters* and 4 *bolters*. Note that the choice of machine for each task instance also yields additional decisions to be made as part of the scheduling.

The CP models were solved by IBM ILOG CP Optimizer version 20.1.0.0 and the MIP model was solved using Gurobi Optimizer version 9.1.2. All tests were run on a PC using an Intel i7 2600k processor at 4.1 GHz and 16 GB of RAM. The industrially relevant instances have been generated in dialogue with Boliden and made publicly available at https://gitlab.liu.se/eliro15/underground_mining_instances where further details are found.

4.2 Comparison between H-LBBD and H-CP

The first set of experiments is made to determine which `scheduling_method` yields the best performance of the heuristic. Since the only difference between the methods H-CP and H-LBBD is how the scheduling is done in each iteration of the heuristic, it is possible to make an individual comparison for each pair of instance and priority order, providing 18 such pairs to evaluate.

Initial tests revealed that it would not be possible to let the CP solver run to optimality in each iteration of the heuristic, and for this reason we needed to put a time limit on each iteration. Since the initial tests also indicated that the LBBD scheme was consistent in producing reasonable schedules in each iteration, without time limits and with a zero-valued MIP-gap in the master problem, we decided to record the time used by H-LBBD in each iteration of the heuristic and then give the CP solver the same amount of time in each iteration. The results from the comparisons with these time limits are shown in Table 1.

Instance -priority		Time [s]	H-CP	H-LBBD
6F:4C	-P1	214	-	**71191**
	-P2	418	79718	**72156**
	-P3	221	-	**71191**
10F:3C	-P1	1005	114291	**98046**
	-P2	1281	112792	**96375**
	-P3	1646	98595	**97683**
12F:2C	-P1	1237	95777	**83321**
	-P2	1390	91255	**85279**
	-P3	835	93228	**82722**
F15:C2	-P1	2295	141988	**116935**
	-P2	2225	**118584**	119523
	-P3	2651	128531	**119784**
18F:XC	-P1	1159	99879	**98917**
	-P2	1034	103966	**99829**
	-P3	1092	105670	**104111**
24F:1C	-P1	2819	134634	**126304**
	-P2	2017	132404	**125704**
	-P3	2767	131204	**127434**

Table 1. The makespan (or ”-” when no solution was found) for the schedules produced by H-CP and H-LBBD, respectively, when using the same total computational time for an instance-priority pair

The results in Table 1 show that H-LBBD yields a better makespan than H-CP for all instances except **15F:2C** with priority P2. In particular, H-LBBD excels when the ratio of cycles to faces is high. Furthermore, when comparing the results for different priority orders for the same instance, it can be noted that the priority order has an impact on the makespan. Sometimes the differences are significant, especially for H-CP. For H-LBBD the results are more consistent. The impact of the priority order is further discussed in Section 4.3 when benchmarking with M-CP. Our conclusion from the first set of experiments is that H-LBBD performs better than H-CP and that we therefore can omit the latter in the further evaluations.

4.3 Comparison between H-LBBD and M-CP

The comparison between H-LBBD and M-CP is multifaceted since, even if in both cases we are interested in finding a solution with a short makespan quickly, both the methods and the objectives differ. Objective-wise, H-LBBD takes the priority order into account, while M-CP does not. However, in the comparison, the result is evaluated with respect to the makespan, that is, the objective used in M-CP. This means that for one result from M-CP, we have three results from H-LBBD to compare with and this comparison needs to include an analysis of the impact of taking the priority order into account in H-LBBD.

Method-wise, H-LBBD is a heuristic while M-CP is an exact method. However, as will be apparent in the results, the CP solver is far from capable of finding an optimal solution, and for this reason it becomes more reasonable to treat also M-CP as a heuristic in the sense that it is evaluated by the makespan produced after a certain time limit. For this purpose, we introduce two time limits. The first is *LBBD max time limit*, which is the maximum time spent by H-LBBD for the priority order that required the longest computational time, hence giving an advantage to M-CP in the evaluation. The second one, *Benchmark time limit*, is chosen to be significantly longer than the first to, if possible, provide a really short makespan to use as benchmark when assessing the impact of the priority order. The results are displayed in Table 2.

We begin by comparing H-LBBD with M-CP for the LBBD max time limit. Most striking is that for all instances, the makespans of the schedules from H-LBBD are, irrespective of the priority order used, shorter than the makespan of the corresponding schedule produced by M-CP. We interpret this as an effect of H-LLBD being a much more efficient solution method for the problem, and this to a magnitude that makes the priority order irrelevant. Comparing the relative improvement of the makespan between M-CP and H-LBBD for the priority order with the best makespan gives a relative difference of 22%, 6%, 2%, 14%, and 2%, respectively, for the instances solved by both methods.

When comparing the makespans for the LBBD max time limit and the Benchmark time limit for M-CP, we note that there is an improvement when allowing the CP solver additional computational time. For **6F:4C** it makes the difference between finding a feasible solution and not, and in this case the time limit was increased by a factor of about 9. For the other instances, the relative improvements of the makespans are 5.3%, 1.8%, 0.04%, 6.7%, and 0.1%, respectively, for the time limit being increased by a factor of 3, 3, 3, 5, and 3, respectively.

The final comparison is between H-LBBD and M-CP with the Benchmark time limit. There we see that for instance **15F:2C** with priority P2 and P3, the

Instance	Eval. measure	Method				
		M-CP		H-LBBD		
		Benchmark time limit	LBBD max time limit	P1	P2	P3
6F:4C	Makespan	83004	-	**71191**	72156	**71191**
	Time [s]	3600	418	214	418	221
10F:3C	Makespan	117249	123795	98046	**96375**	97683
	Time [s]	5400	1646	1005	1281	1646
12F:2C	Makespan	86577	88200	83321	85279	**82722**
	Time [s]	5400	1390	1237	1390	835
15F:2C	Makespan	119351	119863	**116935**	119523	119784
	Time [s]	7200	2651	2295	2225	2651
18F:XC	Makespan	107524	115294	**98917**	99829	104111
	Time [s]	5400	1159	1159	1034	1092
24F:1C	Makespan	126924	128273	126304	**125704**	127434
	Time [s]	7200	2819	2819	2017	2767

Table 2. The makespan (or ”-” when no solution was found) for the schedules produced by M-CP and H-LBBD, respectively, for different time limits

makespan is longer in the schedules from H-LBBD than in the schedule from M-CP. However, the relative differences are only about 0.1% and 0.3%, respectively, after about 3 times more computational time. For all other instances and priority orders, H-LBBD still provides the best makespans – despite the much longer runtimes for M-CP. Again, this means that the efficiency of the LBBD scheme dominates that of the effect of taking the priority order into account. Further analysis of the impact of priority orders is therefore left for future work. As a preliminary result in this direction, Figure 2 illustrates the schedules from H-LBBD for instance **10F:3C** with priority order P1 and P2, respectively. More illustrations of this kind are found in [6].

5 Concluding remarks

This paper addresses how to formulate and solve a short-term scheduling problem for a cut-and-fill mine. To enable a high degree of control on the scheduling output, a priority-based heuristic that adapts to different face priorities was proposed. Note that since the distances between different faces of a mine are not the same, the priority order is expected to have an impact on the best possible makespan that can be obtained. The heuristic was integrated with an LDDB scheme to solve the partial scheduling problems in each iteration of the heuristic. To enable an efficient LBBD scheme, a new objective for the problem was introduced. This objective indirectly aims for a short makespan but it uses the shift structure of the problem instead of considering the end times of tasks.

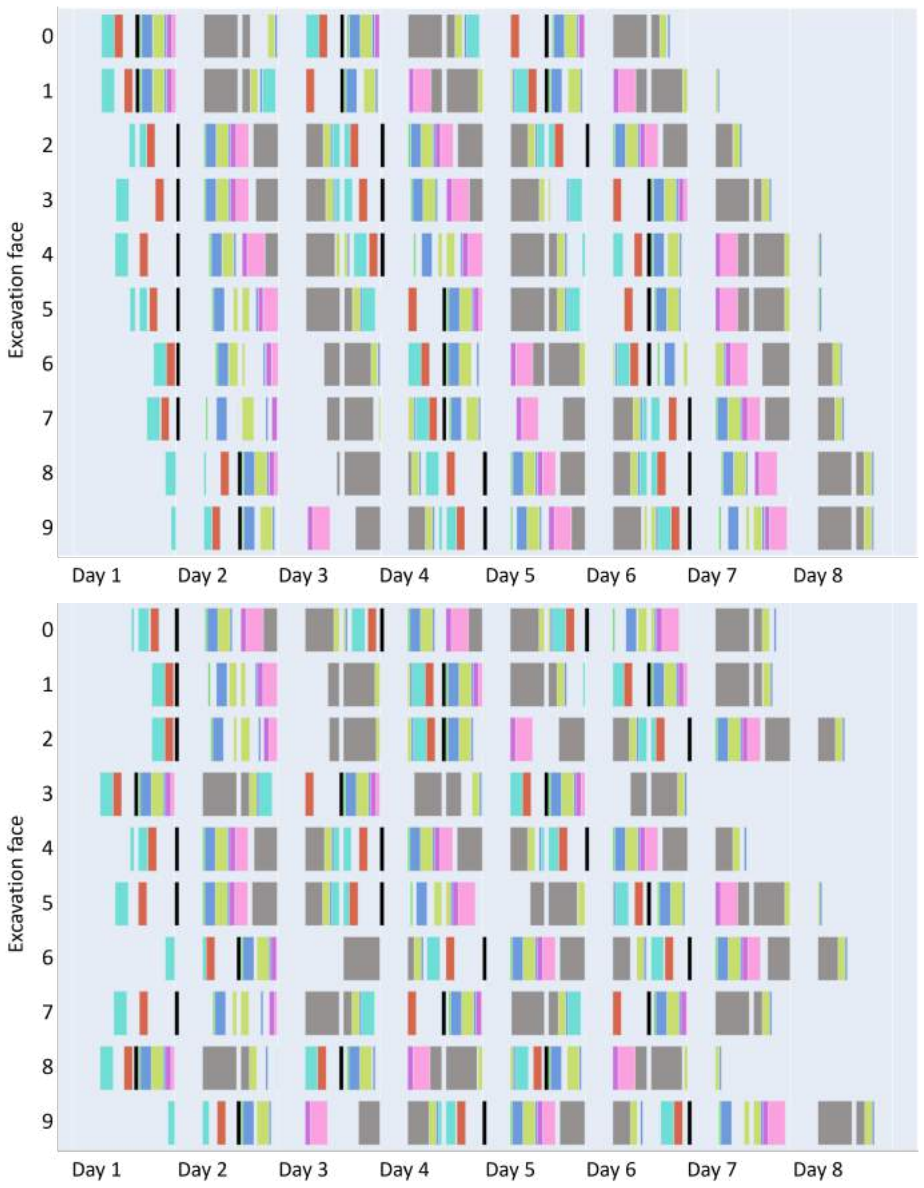


Fig. 2. Schedules for instance **10F:3C** for the priority order P1 (order: 1,2,3,4,5,6,7,8,9,10) and P2 (order: 4,9,6,5,1,8,3,2,7,10), respectively. Colours are: turquoise for drillrigs 1–2, red for chargers 1–2, green for watering trucks 1–2, blue for loaders 1–5, lime green for scalers 1–3, purple for shotcreters 1–2, grey for bolters 1–4, and pink for afterlag

The effect of including a priority order was evaluated by comparing with the makespans of schedules generated without considering a priority. The results showed that – thanks to computational efficiency – our schedules had a shorter makespan than that obtained when applying a CP solver on a monolithic CP model aiming only at minimising the makespan. Note that this was almost always the case also when the CP solver was given significantly longer computational times. The conclusion we draw from this is that applying an approach like the one we propose shows great promise for this type of scheduling problem and that the impact of including priorities needs to be further studied.

In other future work, it is relevant both to improve the modelling of the problem and to continue the development of more efficient solution methods. In particular, the heuristic can be further developed and so can the LBBD scheme. For the latter, there is potential to improve the master problem formulation and the cut-strengthening procedures.

References

1. Coban, E., Hooker, J.: Single-facility scheduling by logic-based Benders decomposition. Annals of Operations Research **210**, 245–272 (2013)
2. Emde, S., Polten, L., Gendreau, M.: Logic-based Benders decomposition for scheduling a batching machine. Computers & Operations Research **113**, 104777 (2019)
3. Hooker, J.N.: Logic-Based Benders Decomposition for Large-Scale Optimization, pp. 1–26. Springer International Publishing, Cham (2019)
4. Karlsson, E., Rönnberg, E.: Strengthening of feasibility cuts in logic-based Benders decomposition. In: Stuckey, P.J. (ed.) Integration of Constraint Programming, Artificial Intelligence, and Operations Research. pp. 45–61. Springer International Publishing, Cham (2021)
5. Karlsson, E., Rönnberg, E.: Logic-based Benders decomposition with a partial assignment acceleration technique for avionics scheduling. Computers & Operations Research (2022). https://doi.org/10.1016/j.cor.2022.105916
6. Lindh, E., Olsson, K.: Scheduling of an underground mine by combining logic-based Benders decomposition and a constructive heuristic. Master's thesis, Department of Mathematics, Linköping University, Sweden (2021)
7. Nehring, M., Topal, E., Knights, P.: Dynamic short term production scheduling and machine allocation in underground mining using mathematical programming. Mining Technology **119**(4), 212–220 (2010)
8. Roshanaei, V., Luong, C., Aleman, D.M., Urbach, D.: Propagating logic-based Benders' decomposition approaches for distributed operating room scheduling. European Journal of Operational Research **257**(2), 439–455 (2017)
9. Schulze, M., Rieck, J., Seifi, C., Zimmermann, J.: Machine scheduling in underground mining: an application in the potash industry. OR Spectrum **38**, 365–403 (2015)
10. Song, Z., Schunnesson, H., Rinne, M., Sturgul, J.: Intelligent scheduling for underground mobile mining equipment. PLOS ONE **10**(6), 1–21 (2015)
11. Åstrand, M.: Short-term Underground Mine Scheduling: An Industrial Application of Constraint Programming. No. 36, KTH, Automatic Control (2021), PhD thesis

12. Åstrand, M., Johansson, M., Feyzmahdavian, H.R.: Short-term scheduling of production fleets in underground mines using CP-Based LNS. In: Stuckey, P.J. (ed.) Integration of Constraint Programming, Artificial Intelligence, and Operations Research. pp. 365–382. Springer International Publishing, Cham (2021)
13. Åstrand, M., Johansson, M., Greberg, J.: Underground mine scheduling modelled as a flow shop : a review of relevant work and future challenges. The Southern African Journal of Mining and Metallurgy **118**(12), 1265–1276 (2018)
14. Åstrand, M., Johansson, M., Zanarini, A.: Fleet scheduling in underground mines using constraint programming. In: van Hoeve, W.J. (ed.) Integration of Constraint Programming, Artificial Intelligence, and Operations Research. pp. 605–613. Springer International Publishing, Cham (2018)
15. Åstrand, M., Johansson, M., Zanarini, A.: Underground mine scheduling of mobile machines using constraint programming and large neighborhood search. Computers & Operations Research **123**, 105036 (2020)

Solving an Industrial Oven Scheduling Problem with a Simulated Annealing Approach

Marie-Louise Lackner, Nysret Musliu, and Felix Winter

Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling, DBAI, TU Wien, Favoritenstraße 9, 1040 Vienna, Austria
marie-louise.lackner@tuwien.ac.at, nysret.musliu@tuwien.ac.at, felix.winter@tuwien.ac.at

1 Introduction

In times of a climate crisis, reducing industrial energy consumption has become all the more important. In this paper, we are concerned with the hardening process of electronic components in specialised heat treatment ovens, a highly energy-intensive task. The energy consumption of this process can be reduced by grouping compatible jobs into batches for simultaneous processing.

Recently, we formalized this problem as the Oven Scheduling Problem (OSP), an NP-hard parallel batch scheduling problem [4]. The key task of the OSP is to create a feasible assignment of jobs to batches and to find an optimal schedule of these batches on a set of ovens. The creation of batches must be done in such a way that jobs in the same batch have the same attribute as well as compatible minimal and maximal processing times. Moreover, the scheduling of batches needs to respect the jobs' earliest start times, their respective sets of eligible ovens as well as oven capacities. Furthermore, availability times of ovens as well as attribute-dependent setup times between batches need to be considered. The optimization goal of the OSP is to minimize a linear combination of three objectives: cumulative batch processing time, tardiness and setup costs. For a more formal definition of the OSP, we refer the reader to our previous publication [4].

A wealth of scientific papers has investigated batch scheduling problems throughout the past three decades (see. e.g., the surveys [6, 7, 2]). The problems studied so far in the literature typically minimize objectives related to makespan, tardiness or lateness. The OSP differs from these problems as one of its main objectives is to minimize the cumulative batch processing time across all ovens–which is directly related to the energy costs of running the ovens.

In our recent publications introducing the OSP [4, 5], we provided a benchmark set consisting of 80 instances of different sizes (up to 100 jobs), developed CP and ILP models and performed an extensive experimental evaluation of our proposed exact solution methods. Moreover, we developed theoretical lower bounds on the optimum value.

To increase the practical applicability of our previously developed methods for the OSP, two goals need to be pursued. Most of the large benchmark instances with 50 or 100 jobs could not be solved to optimality within the runtime limit of one hour. Firstly, we thus need to improve the solution quality for large instances. Secondly, in practice one often needs to obtain solutions – of not necessarily optimal, but sufficiently good quality – in shorter time than one hour, ideally in just a couple of minutes. In order to fulfill both of these goals, we propose a metaheuristic local search approach based on simulated annealing which we briefly describe in the following section. In Section 3, we then evaluate the results of our preliminary experiments with this new solution approach for the OSP.

2 The Simulated Annealing Approach

Simulated annealing is a metaheuristic local search technique targeted at finding an approximation of the global optimum for optimization problems with large search spaces. The name of this technique comes from the process of annealing in metallurgy and was introduced by Kirkpatrick, Gelatt and Vecchi [3]. Since, simulated annealing has been successfully employed to solve a variety of real-world optimization problems, including batch scheduling problems (see, e.g., [1]).

In the following, we briefly describe the core concept of simulated annealing and our implementation of this technique for the Oven Scheduling Problem. The simulated annealing algorithm starts off at an initial temperature and with an initial solution that we obtain using the construction heuristic we presented in our previous work [4, 5]. At every iteration step, one of four neighborhood types is chosen uniformly at random, and a candidate solution in this neighborhood is chosen by applying a random move to the current solution. The four neighborhood moves we propose are the following: (i) *Move Job to Batch:* select a job and add it to an existing batch that is compatible, (ii) *Create New Batch from Job:* select a job, remove it from its current batch, create a new batch consisting of this single job and insert this batch somewhere in the current schedule, (iii) *Move Batch:* select an entire batch and insert it at a new position of the current schedule, (iv) *Swap Consecutive Batches*: select two consecutive batches on the same oven and swap them.

The candidate solution obtained in this way is accepted if its solution cost is an improvement over the current solution. In case the candidate solution is a deterioration of the current solution, it is also accepted if the *acceptance criterion* is met. This acceptance criterion depends both on the current temperature and the relative size of the deterioration; we use the metropolis criterion [3] as acceptance function. After this step, the temperature is reduced according to a cooling scheme. This procedure is iterated until the runtime limit is reached and returns the best solution found.

3 Preliminary Experimental Evaluation

In order to assess the viability of the proposed local search approach using simulated annealing, we implemented a preliminary version of the algorithm. Based on manual parameter tuning, we set the initial acceptance rate to 50% and the minimum temperature to 10^{-10}. For a given instance, the initial temperature is chosen so that moves are accepted with probability equal to the initial acceptance rate at the beginning of the local search process. This is done by creating a sample of moves from the initial solution. Moreover, instead of using a fixed cooling rate, the cooling scheme is chosen adaptively for every run of the algorithm: the cooling is done in such a way that the minimum temperature is reached at the same time as the runtime limit. This is ensured by calculating a new, reduced temperature after every iteration step, based on the current average time required per iteration. We performed a series of experiments on our benchmark set [4] consisting of 80 randomly generated instances.[1] The experiments were run on single cores, using a computing cluster with 10 identical nodes, each having 24 cores, an Intel(R) Xeon(R) CPU E5–2650 v4 @ 2.20GHz and 252 GB RAM.

	exact methods	**simulated annealing**	
runtime limit per run	1 hour	30 seconds	5 minutes
# best results	54	56	**77**
# provably optimal results	**37**	36	**37**

Table 1: Overview of the preliminary computational results on the benchmark set consisting of 80 instances.

An overview of our preliminary results can be found in Table 1. In this table, we compare the quality of solutions obtained with the exact methods presented in our previous work [4] with those obtained with the simulated annealing approach. Our previous experiments (presented in [4]) were run in the same computing environment as the ones in this paper. For the exact methods, we use the overall best (minimal) solution cost per instance obtained in [4]. To be precise, this is the best result obtained by any of the 53 exact methods[2] within a runtime limit of one hour. For the simulated annealing approach, we chose to run our algorithm 5 times per instance and use the best result for the comparison. This repetition can be advantageous due to the non-deterministic nature of simulated annealing. We considered two runtime limits: 30 seconds per run (2.5 minutes in total) and 5 minutes per run (25 minutes in total).

The row labeled "# best results" in the table displays the number of instances for which overall best solutions could be achieved (out of 80). The row labeled "# provably optimal results" shows the number of obtained optimal solutions, i.e., solutions for which one of the exact solutions methods could prove optimality. As one can see, running the simulated annealing approach with merely 30 seconds

[1] The benchmark set is publicly available at https://cdlab-artis.dbai.tuwien.ac.at/papers/ovenscheduling/OSPrandominstances/.

[2] This number results from the combination of different models, solvers and search strategies as well as a warm-start option.

per run already allows us to obtain a similar solution quality as with the best exact methods and a runtime limit of one hour. Increasing the runtime of the simulated annealing approach to 5 minutes per run, leads to best results for nearly the entirety of the benchmark set (77 of 80 instances). Furthermore, regarding the instances for which provably optimal solutions could be found by the exact methods, the simulated annealing approach with 5 minutes runtime also finds optimal solutions for every one of these 37 instances; with 30 seconds runtime, optimal solutions can be found for all except one of these instances.

We note that the compared methods were all capable of finding solutions for the entire benchmark set (80 instances). Moreover, both the exact methods and the simulated annealing approach were always capable of improving the (initial) solution provided by the construction heuristic [4].

In Table 2, we take a closer look at the results obtained for the 43 instances of the benchmark set for which none of the exact methods could prove optimality within the runtime limit of one hour. For every one of these instances, we compute the relative improvement ri in % of the simulated annealing approach over the best exact approach. For the simulated annealing approach, we take the best result of 5 runs each having a runtime limit of 5 minutes. In this table, we group the instances by their number of jobs (20, 50 or 100 jobs) and by the value of the relative improvement. Overall, the improvement ri is between -0.05% and 1% for 32 (of 43) instances: for 15 instances, the simulated annealing approach finds solutions of the same quality as the exact approach ($ri = 0$), for another 15 instances, simulated annealing delivers slightly better results ($ri \in (0, 1]$), and for 2 instances slightly worse results ($ri \in [-1, 0)$). For the remaining 11 instances, an improvement of the solution cost by more than 1% was possible, for 7 of which the improvement was larger than 10%. Note that for none of the 43 instances, the solution quality of the simulated annealing approach was significantly worse than the best exact result: there are no instances with $ri < -1\%$.

It is important to note that the size of the instance has a major impact on the improvement made by the simulated annealing approach: for all of the smaller instances with 20 jobs, the solution quality could not be improved; for the instances with 50 jobs, an improvement was possible for 7 out of 17 instances; for the instances with 100 jobs, the solution quality was improved for 19 out of 20 instances. A possible explanation is that the solutions found by the exact methods for the smaller solutions might by optimal–even though no optimality proof could be delivered within the runtime limit of one hour–or very close to the global optimum.

4 Future Work

The preliminary results obtained with the proposed simulated annealing approach on our benchmark set already look promising. As a next step, we plan to configure our algorithm by using automated parameter configuration tools for parameters such as the probabilities of neighborhoods, initial acceptance rate

		# instances with relative improvement ri						
# jobs	# instances	$[-1, 0)$	$= 0$	$(0, 1]$	$(1, 10]$	> 10	avg ri	max ri
20	6	0	6	0	0	0	0%	0%
50	17	1	9	**6**	0	**1**	0.65%	10.45%
100	20	1	0	**9**	**4**	**6**	7.15%	28.01%
total	43	2	15	**15**	**4**	**7**	3.58%	28.01%

Table 2: Overview of the relative improvement ri (in %) achieved by the simulated annealing approach over the best exact approach. Numbers in bold font indicate instances for which an improvement over the best exact result could be achieved. Results are only displayed for the 43 benchmark instances for which no exact method could deliver an optimality proof within the runtime limit of one hour.

and final temperature. Then we plan to conduct an additional evaluation of our simulated annealing approach, including experiments with a longer runtime.

In practice, instances can be even larger than those included in our benchmark set; instances consisting of up to 1500 jobs can be expected. We will therefore also conduct experiments on larger instances than those in our benchmark set. Moreover, we will include the lower bounds obtained by the exact methods [4] as well as the theoretical lower bounds [5] in this evaluation in order to provide a more precise assessment of the solution quality. The theoretical lower bounds, which can be calculated in a few seconds, could also be integrated in a stopping criterion for the simulated annealing approach to reduce the runtime.

Another promising line of research would be to investigate adaptive neighborhood selection. In the current formulation of our simulated annealing approach, the probability of each of the four neighborhoods is constant throughout the solution process. It could however be advantageous to adapt these probabilities based on the current state of the search process. Moreover, it would be interesting to investigate domain-independent hyper-heuristic approaches.

Acknowledgments 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.

Bibliography

[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] 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)

[3] Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by Simulated Annealing. Science **220**(4598), 671–680 (May 1983). https://doi.org/10.1126/science.220.4598.671

[4] Lackner, M.L., Mrkvicka, C., Musliu, N., Walkiewicz, D., Winter, F.: Minimizing Cumulative Batch Processing Time for an Industrial Oven Scheduling Problem. In: 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)

[5] Lackner, M.L., Mrkvicka, C., Musliu, N., Walkiewicz, D., Winter, F.: Exact methods and lower bounds for the oven scheduling problem. Under review (2022), https://arxiv.org/abs/2203.12517

[6] 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)

[7] Potts, C.N., Kovalyov, M.Y.: Scheduling with batching: A review. European Journal of Operational Research **120**(2), 228–249 (Jan 2000)

Scheduling Satellite Timetables using DCOP

Shai Krigman, Tal Grinshpoun[0000−0002−4106−3169], and Lihi Dery[0000−0002−8710−3349]

Department of Industrial Engineering and Management
Ariel Cyber Innovation Center
Ariel University, Ariel 4070000, Israel
shai.krigman@msmail.ariel.ac.il, talgr@ariel.ac.il, lihid@ariel.ac.il

Abstract. Earth observation satellites (EOS) are satellites equipped with optical sensors that orbit the Earth to take photographs of specific areas at the request of users. With the development of space technology, the number of satellites increases continuously. Yet still, the number of satellites cannot meet the explosive growth of applications. Thus, scheduling solutions are required to satisfy requests and obtain a high observation efficiency. While the literature on multi-satellite scheduling is rich, most of the solutions are centralized algorithms. However, due to their cost, EOS systems are often co-funded by several agents (e.g., countries, companies, or research institutes) and central solutions require that these agents will share their requests for observations with others. To date, there is no solution for EOS scheduling that protects the private information of the interested parties. In this study, we model the EOS scheduling problem as a distributed constraint optimization problem (DCOP). This modeling enables generating timetables for the satellites in a distributed manner without a priori sharing private information of the users with some central authority. For solving the resulting DCOP, we use the Distributed Stochastic Algorithm (DSA), which is a simple DCOP algorithm that is known to produce efficient solutions in a timely manner. The modeling together with the solving of the resulting DCOP constitute our new solution method, which we term Distributed Satellite Timetable Solver (DSTS). Experimental evaluation reveals that the DSTS method provides solutions of higher quality than a commonly-used Greedy algorithm.

Keywords: Earth observation satellites · Satellite timetables · DCOP.

1 Introduction

Earth observation satellites (EOSs) are sensor-equipped satellites that are designated to take photographs of special areas at the request of a user [39]. The satellites perform a cycle of orbits around the Earth over a period of several days. Each orbit slightly changes with respect to the preceding one but its trajectory is cyclic in the sense that the satellite recovers its initial position after a predefined number of orbits. Furthermore, a full cycle enables the satellite to view each area of the planet. Most EOSs operate at low altitudes with the orbital periods varying from dozens of minutes to several hours. However, it takes several days for a single EOS to complete a full cycle and view the whole area of the Earth. During the course of one particular orbit, a satellite can take several photographs by rotating itself between consecutive shots. After capturing the photographs, the acquired data is stored in the on-board memory and transferred to a ground station when the satellites are in a feasible transferring range [49].

EOSs have been extensively employed in a wide range of tasks, such as Earth resource exploration, natural-disaster surveillance, environmental monitoring, and defense missions [6]. The demand for EOS services has risen over time due to some unique advantages, including an expansive coverage area, long-term surveillance, accurate and effective information access, and unlimited airspace borders [43].

With the development of space technology, the number of satellites continuously increases [43]. Yet, the number of EOSs still cannot meet the demand due to an explosive growth of applications that require observations. Consequently, scheduling solutions are needed to satisfy more requests and obtain a high observation efficiency. In particular, multi-satellite systems require dedicated scheduling solutions [3].

Due to their cost, EOS systems are often co-funded by several agents (e.g., countries, companies, or research institutes). Once constructed and made operational, the common property resource must be exploited and shared between the partners. Each party wants to fulfill its requirements for observations and a timetable should be developed to schedule all these requirements. Such a timetable needs to be (a) *efficient* in the sense that the satellites are maximally utilized with the highest priority tasks; and (b) considered *fair* by all parties [2].

While the literature on multi-satellite scheduling is rich, as summarized in [43], most of the solutions are centralized algorithms that assume all the requests reach a central entity that creates a timetable for them. Such solutions are prone to privacy issues since EOSs can be used for defense and security purposes and most of the parties do not want others to be informed of the way they are using the satellites. This raises the need for distributed scheduling solutions [35]. Since a given request can sometimes be satisfied by several satellites in more than one of their orbits, the problem is not separable by satellite nor by orbit. Instead, the scheduling process must be performed simultaneously for all satellites and orbits considered.

Distributed constraint optimization problem (DCOP) [15,9] is a powerful framework for representing and solving distributed combinatorial problems. DCOPs have been successfully applied in a variety of real-world problem domains, including meeting scheduling [28], traffic-light synchronization [18], sensor networks [7], and the Internet of Things [24]. Recently, Picard et al. [35] suggested the use of DCOPs as a potential approach for dealing with EOS scheduling problems. Following this, a DCOP-based solution has been proposed [34]. That work deals with the problem of coordinating users having reserved exclusive orbit portions and one central planner having several requests that may use some intervals of these exclusives. Their solution enables the exclusive users to independently plan their own tasks; however, the rest of users' tasks are still scheduled in a central manner.

Contributions: Our paper goes one step further. We present a novel Distributed Satellite Timetable Solver that we name DSTS. The DSTS method works in two phases – it starts by mapping the EOS scheduling problem to a DCOP according to our new proposed modeling, and then it solves the DCOP using a DCOP algorithm. The DSTS method is beneficial for two main reasons. First, the EOS scheduling problem is distributed by nature, and as such, it can be modeled and solved in a distributed manner, such as DCOP. In particular, users do not need to a priori share their private information with some central authority. Second, DCOP is a well-established model that provides a wide palette of algorithms, as well as common metrics and simulation environments. Specifically, our method is modular as it is not limited to a specific algorithm.

The rest of the paper is organized as follows. Section 2 provides an overview of relevant existing studies. Section 3 includes formal definitions of the EOS scheduling problem, which is at the focus of this study. Section 4 provides the definition of DCOP and the DCOP modeling of the EOS scheduling problem. An experimental evaluation is presented in Section 5, followed by the conclusions in Section 6.

2 Related Work

A number of EOS scheduling solutions have been proposed. A detailed literature review is presented in [43]. That review classifies the algorithms into four classes: exact methods, heuristics, meta-heuristics, and machine learning based algorithms. Exact methods can provide optimal or near optimal solutions but they are limited to relatively small scale instances. Heuristic methods are employed when exact methods cannot be used; these are further classified into constructive heuristics and time-efficient heuristics. The methods are easily implemented and have relatively short computational time. However, they are specifically designed and there is no guarantee of solution quality. Meta-heuristics are general procedures that find, generate, or select a search algorithm that may provide a sufficiently high-quality solution to an optimization problem. Evolutionary algorithms and single-point search algorithms are two main examples of this category. Machine learning methods have been recently proposed for solving the EOS scheduling problem. These include deep reinforcement approaches [44,17] and competitive learning strategies [26]. The downside of such methods is the requirement for large amount of data to provide good solutions.

All of the above are *static* solutions in the sense that all problem inputs are given in advance. Another research direction deals with the requirement for *dynamic* scheduling [41]. Some examples in which dynamic scheduling is necessary include incoming emergency tasks [36], impact of clouds [40,42], and impact of transition time between observations [45,46].

Most of the known solutions for EOS scheduling problems are centralized algorithms in which the assumption is that satellite constellations are shared resources managed by a central mission control that receives all the requests for observations and schedules them. Recently, a few distributed methods have been proposed [25,4]. These works distribute the scheduling problem between the satellites in attempt to provide rapid response to dynamic changes. However, such form of distribution does not deal with the privacy issue of the users. In a pioneer work, Picard [34] investigates the use of multi-agent allocation techniques for solving the EOS scheduling problem. One of the proposed techniques is DCOP-based. It works under the assumption that some users (termed exclusive users) have reserved exclusive orbit portions. A central planner collects the requests of non-exclusive users and sends them to the exclusive users. The exclusive users employ a distributed procedure that attempts to sequentially schedule those requests, each time creating a DCOP instance in an attempt to add a new request. In this case, the exclusive users retain their privacy while the non-exclusive users still need to share their tasks with a central mission control and thus lose their privacy. To the best of our knowledge, there are no studies that fully distribute the problem among all users.

In order to obtain a fully distributed solution for all users, we suggest to model the whole problem as a DCOP. A major benefit of modeling a problem as a DCOP is that once modeled, the problem can by solved by a wide variety of algorithmic approaches, either complete [31,32,11,48] or incomplete [50,8,20]. DCOPs are NP-hard. Therefore, the use of incomplete approaches is required when solving medium- to large-scale problems.

3 Problem Definition

This section provides the core definitions of the problem we investigate. Consider a set of independent users, each of which generates a set of observation requests. Each request requires viewing a specific area for a specific duration of time, i.e., a request for a specific latitude-longitude-altitude position (LLA) at a certain time interval. These requests potentially yield several observation opportunities per request. In order to define the set of opportunities for each request, the following parameters are required: the LLA to observe, the satellite's location (defined by its orbit plan), and the attitude adjustment capabilities of the satellite's camera. A user can generate a set of observation opportunities for each of its requests using these three parameters. Each such opportunity can be served by a specific satellite at a specific time. The duration of an opportunity is determined by the duration of the request with the addition of the setup time required by the satellite. The extent of the setup time depends on the previous viewing angle of the satellite before it attended to the opportunity at hand. Each user assigns a certain utility, herein termed reward, to its requests. The goal is to find a timetable in which the sum of rewards due to handled requests is maximal. Based on the core definitions in [34], we present the following definitions.

Definition 1 (Earth Observation Satellite Scheduling Problem). *An Earth Observation Satellite Scheduling Problem $\mathcal{P}$ is defined by a tuple*

$$\mathcal{P} = < \mathcal{S}, \mathcal{U}, \mathcal{R}, \mathcal{O} >$$

where $\mathcal{S}$ is a set of satellites, $\mathcal{U}$ is a set of users, $\mathcal{R}$ is a set of observation requests, and $\mathcal{O}$ is a set of observation opportunities for fulfilling the requests in $\mathcal{R}$.

Definition 2 (Satellite). *A satellite $s \in \mathcal{S}$ is defined as a tuple*

$$s = < OP_s, \kappa_s, ST_s >$$

where OP_s is the orbit plan of the satellite. $\kappa_s \in \mathbb{N}^+$ is the satellite capacity (i.e. the maximum number of observations during its orbit plan). The transition time between two given observations is $ST_s\colon \mathcal{O} \times \mathcal{O} \to \mathbb{R}$.

For simplicity, we assume that the timeline is divided into discrete steps; τ denotes the time period index, see [43].

Definition 3 (User). *A user $u \in \mathcal{U}$ is defined by its set of requests $\mathcal{R}_u \subset \mathcal{R}$.*

Definition 4 (Request). *A request $r \in \mathcal{R}$ is a tuple*

$$r = < t_r^{start}, t_r^{end}, \Delta_r, \rho_r, p_r, u_r, \Theta_r >$$

where the request validity time window is $t_r^{start} \in \mathcal{T}$ and $t_r^{end} \in \mathcal{T}$, $\Delta_r \in \mathcal{T}$ is the required observation duration, $\rho_r \in \mathbb{R}$ is the reward obtained once the request is fulfilled, p_r is the latitude-longitude-altitude position (LLA) to observe, $u_r \in \mathcal{U}$ is the requesting user, and finally, $\Theta_r \subseteq \mathcal{O}$ is the list of opportunities to fulfill the request.

Definition 5 (Opportunity). *An opportunity is defined as a tuple*

$$o = < t_o^{start}, \Delta_o, r_o, \rho_o, s_o >$$

where $t_o^{start} \in \mathcal{T}$ is the opportunity start time, $\Delta_o \in \mathcal{T}$ is the opportunity duration, r_o is the request to which this opportunity contributes, ρ_o is the reward that this opportunity will provide (it is based on ρ_r with modifications according to the observation angle, weather conditions, etc.), s_o is the satellite on which this opportunity can be scheduled.

Definition 6 (Solution). *A solution to a EOS scheduling problem is a feasible subset of opportunities $\mathcal{M} = \{o \in \mathcal{O}\}$, so there is at most one observation per request, and the overall reward (i.e., the sum of the rewards of all scheduled observations) is maximized:*

$$\arg\max_{\mathcal{M}} \sum_{o \in \mathcal{O}} \rho_o x_o$$

subject to

$$x_o \in \{0,1\} \; \forall o \in \mathcal{O} \tag{1}$$

$$\sum_{o \in \Theta_r} x_o \leq 1 \; \forall r \in \mathcal{R} \tag{2}$$

$$s_{o_i} \neq s_{o_j} \vee [int_{o_i} + ST_{s_{ij}}] \cap [int_{o_j} + ST_{s_{ji}}] = \emptyset \; \forall o_i, o_j \in \mathcal{M} \tag{3}$$

$$int_o := [t_o^{start}, t_o^{start} + \Delta_o] \tag{4}$$

where x_o is the decision variable that defines whether opportunity o is included in $\mathcal{M}$ or not. Equation 2 ensures at most one opportunity for each request is included in the solution. Finally, Equations 3 and 4 avoid overlaps between opportunities, by verifying that the opportunities are either in different satellites or their time intervals including transition time (int_o) do not overlap.

4 DCOP Modeling of the Problem

Now we model the problem as a DCOP. We first provide the formal definition of a DCOP, followed by our proposed DCOP modeling of the EOS scheduling problem. This modeling is the first phase of the proposed DSTS method.

4.1 DCOP Definition

A *DCOP* is a tuple $< \mathcal{A}, \mathcal{X}, \alpha, \mathcal{D}, \mathcal{R} >$, where: $\mathcal{A}$ is a finite set of agents $A_1, A_2, ..., A_n$; $\mathcal{X}$ is a finite set of variables $X_1, X_2, ..., X_m$; $\alpha : \mathcal{X} \rightarrow \mathcal{A}$ maps each variable to one agent; $\mathcal{D}$ is a set of domains $D_1, D_2, ..., D_m$, where each domain D_i consists of a finite set of values that can be assigned to variable X_i; $\mathcal{R}$ is a set of relations (constraints), where each constraint $C \in \mathcal{R}$ is a function $C : D_{i_1} \times D_{i_2} \times \ldots \times D_{i_k} \rightarrow \mathbb{R}_+$ that defines a non-negative *cost* for every possible value combination of a set of variables. A *complete assignment* consists of assignments to all variables in $\mathcal{X}$. An *optimal solution* to a DCOP is a complete assignment of minimal cost.

The advantage of DCOP representation over the classical constraint optimization problems is in the ability to solve the problem in a distributed manner. DCOP researchers have proposed a wide variety of solution approaches, such as search-based algorithms [10,31,48], logic programming [22], inference-based algorithms [33], and other methods [30]. In all of these approaches, each agent handles its own variables and exchanges messages with the other agents in order to determine the final variable values. No centralized entity takes part in the solving process.

In this work we used a search-based approach, specifically, the Distributed Stochastic Algorithm (DSA) [50]. DSA is a standard incomplete DCOP algorithm. This algorithm operates as follows. In each step, the state of every variable (the current value assignment) is shared with its neighbors (the variables with which it is constrained) by sending and receiving messages. After receiving the updated states of its neighbors, each agent decides whether to change the current state of the variable in an attempt to reduce its costs – this decision is the most fundamental step in DSA. If the agent cannot find a new value in its domain to improve its current state, it has no reason to change its current value. However, in case there is such a value that improves the state, the agent may or may not change to the new value based on a stochastic scheme that prevents scenarios of infinite alteration loops. The probability of parallelism, p, controls how frequently neighboring variables can change their values. The p probability is set as a parameter of the DSA algorithm. The algorithm stops its execution after pre-defined number of iterations.

4.2 EOS Scheduling as a DCOP model

We first provide the mapping of the EOS scheduling problem onto a DCOP model, and then present an example that illustrates the mapping. The mapping is performed in the following manner: A user is mapped to an agent in the DCOP: $\mathcal{A} := \{u \in \mathcal{U}\}$ Each user maps its requests to variables in the DCOP: $\mathcal{X} := \{r \in \mathcal{R}\}$. Because $\mathcal{X} = \mathcal{R}$, we use these notations interchangeably. The α mapping is performed according to the set of requests $\mathcal{R}_u$ of each user u. The domain $\mathcal{D}$ of each variable is defined by the set of available opportunities Θ_r to fulfill the request r with the addition of the value 0. $\mathcal{D}\colon = \{\Theta_r \cup \{0\} \mid \forall r \in \mathcal{R}\}$. Each value represents one available opportunity for this request. The additional 0 value represents situations in which this request cannot be fulfilled.

Two types of constraints are defined for each user for each of its variables:

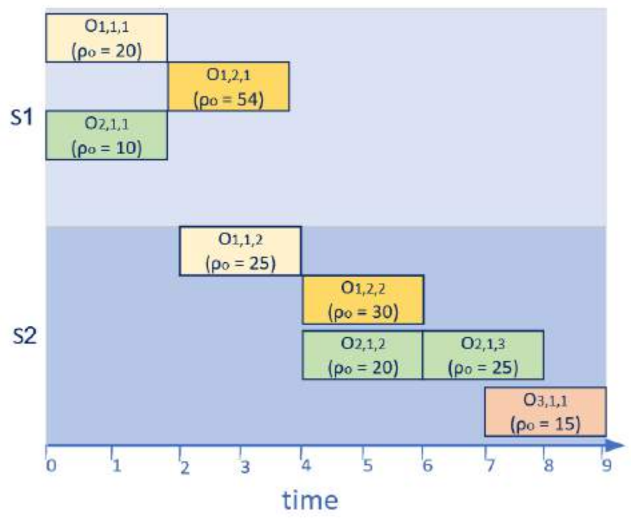
1. A **unary constraint** assigns a cost for each value of the variable based on the reward of the corresponding opportunity. DCOPs are commonly used as minimization problems and require non-negative constraints. Accordingly, we define the cost of opportunity o as follows:

$$cost_o = maxCost - \rho_o \tag{5}$$

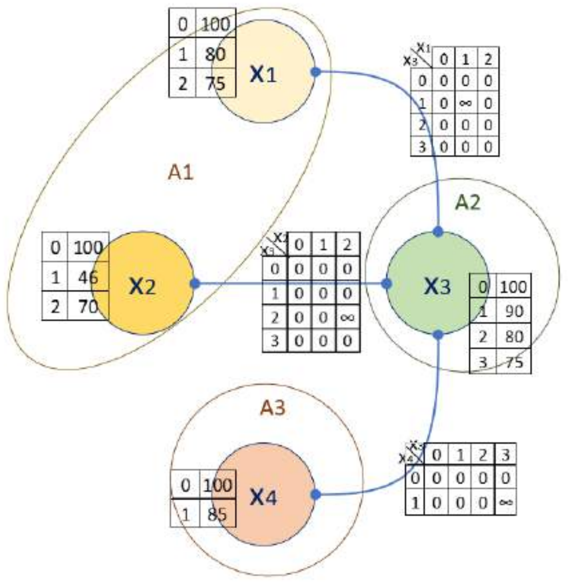
   where $maxCost$ is greater than the reward of any opportunity:

$$maxCost > \max_o \rho_o$$

   Specifically, the cost of 0 value is $maxCost$. Thus, the algorithm will assign a 0 only when there is no possibility to assign any of the opportunities, i.e., the request cannot be fulfilled.
2. **Binary constraints** are defined between two variables if there is overlap between the opportunities they represent. Overlap occurs when the two opportunities are served by the same satellite and their schedule times overlap. The cost of each combination of values is ∞ if there is overlap between the opportunities that these values represent; otherwise, the cost is zero. The setup time between the opportunities, as defined in the ST_s table of the satellite, is considered when computing the overlap. The cost of the 0 value is zero for any combination with the other variable's values.



(a) An EOS scheduling problem. S1 and S2 are two different satellites, rectangles represent opportunities, and the color of a rectangle relates to the request that the opportunity can fulfill.



0	100
1	80
2	75

0	100
1	46
2	70

0	100
1	90
2	80
3	75

0	100
1	85

x3 \ x1	0	1	2
0	0	0	0
1	0	∞	0
2	0	0	0
3	0	0	0

x3 \ x2	0	1	2
0	0	0	0
1	0	0	0
2	0	0	∞
3	0	0	0

x4 \ x3	0	1	2	3
0	0	0	0	0
1	0	0	0	∞

(b) The DCOP modeling of the above problem. A_1, A_2, and A_3 are the agents (users) and $x_1, \ldots, x_4$ are the variables (requests). The tables represent the constraints.

Fig. 1: An example of an EOS scheduling problem mapped to a DCOP.

An example of an EOS scheduling problem and its DCOP modeling are shown in Figure 1. Figure 1a depicts an EOS scheduling problem with two satellites (s_1 and s_2) and three users (u_1, u_2, and u_3). User u_1 has two requests: $r_{1,1}$ and $r_{1,2}$ each of which has two opportunities: $o_{1,1,1}$ and $o_{1,1,2}$ for $r_{1,1}$ and $o_{1,2,1}$ and $o_{1,2,2}$ for $r_{1,2}$. The second user, u_2, has a single request, $r_{2,1}$ with three opportunities: $o_{2,1,1}$, $o_{2,1,2}$, and $o_{2,1,3}$. The third user, u_3, has a single request, $r_{3,1}$ with a single opportunity $o_{3,1,1}$. In the figure, the opportunities are shown with their schedules and the rewards.

Figure 1b describes the DCOP modeling of this problem. This DCOP contains three agents (A_1, A_2, and A_3) representing the users (U_1, U_2, and U_3). Four variables (x_1, x_2, x_3, and x_4) correspond to the four requests and their domains are defined according to the number of opportunities that can fulfill each request. The domain of x_1 consists of three values: two values (1 and 2) for the opportunities that can fulfill it and the 0 value for the option of not scheduling this request. For the same reasons the domain of x_2 has three values, the domain of x_3 has four values, and the domain of x_4 has two values. Each variable has a unary constraint that assigns a cost to each value. The cost of the 0 value is always $maxCost$ ($maxCost = 100$ in this example). The costs of the other values are defined by Equation 5. For example, $o_{1,1,1}$ has a reward of $\rho_o = 20$, so the cost of value 1 of x_1 is 80. Binary constraints are defined between requests that their opportunities overlap. For example, there is a constraint between x_1 and x_3 because $o_{1,1,1}$ and $o_{2,1,1}$ are scheduled at the same time and on the same satellite (s_1). The cost of the combination is ∞. The other combinations have zero cost since there are no conflicts between these potential opportunities. Both constraint types are illustrated in the tables. The unary constraints tables contain rows for each of the opportunities and an additional row for the 0 value. Each row contains a single value: the unary cost of this opportunity. For example, the unary constraint table of X_1 contains three rows (two for the two opportunities and one for the 0 value) with the value 100 in the first row for the 0 value, 80 in the second row as the cost value of $o_{1,1,1}$ and 75 in the third row as the cost value of $o_{1,1,2}$. The binary constraints are illustrated by two-dimensional tables that are linked to an arc that connects two variables. Each cell in the tables represents the possibility of overlap between two opportunities. For example, in the binary constraints table of $x_1 - x_3$, all the cells have zero value, except for the cell that represents the combination of the $o_{1,1,1}$ and $o_{2,1,1}$ opportunities; that cell has the ∞ value since only this combination generates an overlap.

5 Evaluation

We first describe our experimental setup (Section 5.1) and then the obtained results (Section 5.2).

5.1 Experimental Setup

In order to evaluate DSTS, we generated EOS scheduling problems. For the first phase of DSTS, we mapped the generated problems into DCOPs, as described in Section 4.2. For the second phase of DSTS, in which the resulting DCOP model should be solved by some DCOP algorithm, we used the DSA algorithm (as detailed in Section 4.1). DSA is suitable for our purposes for several reasons. First, DSA has a low communication and computation overhead, which is required when dealing with problems with hundreds of variables. Second, DSA operates simultaneously by all agents with no pre-defined ordering or structure[1], which provides some sense of fairness.

We compare our DSTS method to a greedy benchmark algorithm (denoted Greedy), which is used by most satellite and constellation operators [5]. Greedy sorts all opportunities in increasing order according to their start time and then by their reward (in decreasing reward order). Then, for each opportunity in this sorted list, if there is a free slot for it on its satellite then it is scheduled and all other opportunities of the same request are deleted; otherwise, this opportunity is deleted.

Following [34], we generated two sets of experiments:

1. **Highly conflicting problems**: small-scale problems (5 minutes planning horizon) with 3 satellites, 8 users emitting 2 to 20 requests each ($|R_u| = 2, 4, \ldots, 20$), and 10 observation opportunities per request. The requests validity time window varied in the range $[10 : 20]$ and its duration was set to $\tau = 5$, with reward $\rho_o = [10 : 50]$ meaning a reward was sampled uniformly at random between 10 and 50. This set of problems yields many overlaps between observation opportunities, thus it produces tight problems.

2. **Realistic problems**: large-scale problems in a 6-hour planning horizon. We generated instances with 8 satellites, 6 users emitting 10 to 100 requests each ($|R_u| = 10, 20, \ldots, 100$), and 5 observation opportunities per request. The requests validity time window varied in the range $[40 : 60]$ and its duration was set to $\tau = 20$, with reward $\rho_o = [10 : 50]$. This set of problems has many observation opportunities (up to 3000) but with less overlaps between them (a sparse setting).

We generated 100 instances of each setting, which resulted in a total of 2000 generated problems. All experiments were performed on the 'AgentZero' simulator [27].[2] All experiments were run on a standard laptop (Lenovo T14 with Intel(R) Core(TM) i7-10610U CPU running at 1.80GHz) with Win10 OS and took a few minutes in total.

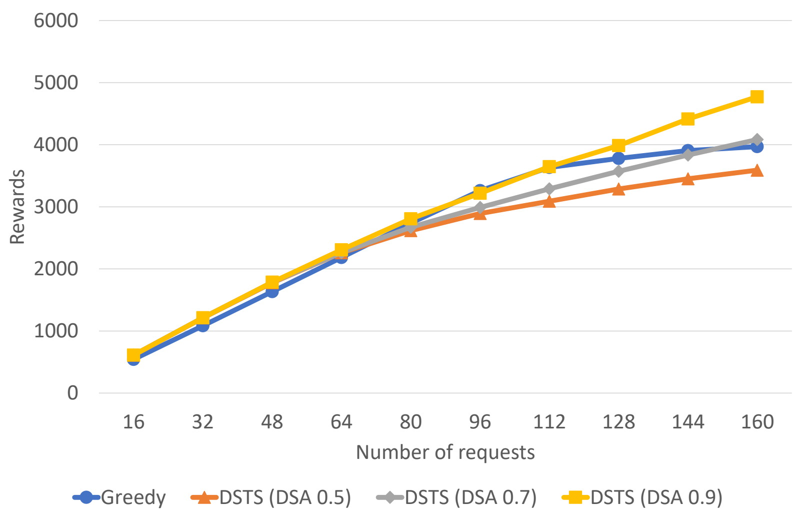


Fig. 2: Rewards of highly conflicting problems

5.2 Experimental Results

We first examined the rewards of highly conflicting problems. Figure 2 presents a comparison of DSTS and Greedy. We used the DSA algorithm to solve the DCOP. Specifically, we used the DSA-B version [50] with three different p values, $p = \{0.5, 0.7, 0.9\}$, and ran it for 10 iterations. Axis x presents the number of requests, from 16 (eight users with two requests each) to 160 (eight users with 20 requests each). Axis y presents the sum of the rewards over all scheduled requests.

For a low number of requests the results of the two compared methods are similar, but for a high number of requests DSTS (using DSA with $p = 0.9$) outperforms Greedy. The best results for DSTS are obtained with $p = 0.9$. These findings are consistent with known results regarding the probability of parallelism in DSA [50]; higher parallelism usually leads to higher quality solutions until a 'phase transition' is reached, after which the solution quality drops drastically. In particular, Zhang et al. [50] showed that for the DSA-B version the phase transition commonly occurs when $p > 0.9$, in consistence with our results.

[1] Some DCOP algorithms like SyncBB [15] and AFB [11] maintain a pre-defined ordering of the agents, while others, such as DPOP [32] and BnB-ADOPT [48], operate on a tree structure.

[2] AgentZero is a Java-based programming framework for research and implementation of multi-agent problems, and particularly DCOPs. It enables to generate various types of multi-agent problems, test distributed algorithms, and collect various performance statistics.

Table 1 shows the average number of messages exchanged during the run of DSTS using DSA with $p = 0.9$; Greedy is a centralized algorithm, thus it does not exchange messages. Table 2 compares the run-times of DSTS (using DSA with $p = 0.9$) and Greedy. As can be seen, DSTS is more affected by the size of the problem. Yet, it is still very fast (less than 100 milliseconds for the largest problems in this set).

Requests	16	32	48	64	80	96	112	128	144	160
DSTS	327	1782	5349	11390	19758	30054	42813	56748	73287	92281

Table 1: Number of messages in highly conflicting problems

Requests	16	32	48	64	80	96	112	128	144	160
DSTS	0.33	0.92	1.8	3.92	8.6	16.3	24.2	36	52	97
Greedy	0.15	0.12	0.13	0.16	0.22	0.32	0.36	0.42	0.49	0.69

Table 2: Run-time of highly conflicting problems (milliseconds)

Next, we examined realistic problems. We used the DSA algorithm to solve the DCOP with the same three p values as before. The results are displayed in Figure 3. Axis x presents the number of requests, from 60 (six users with ten requests each) to 600 (six users with 100 requests each). Again, axis y presents the sum of the rewards over all scheduled requests. Here, all three versions of DSA obtain similar rewards. However, these rewards are $\sim 6\%$ better than those of Greedy . These findings can be explained by the relative sparsity of constraints in this set, which results in problems that are more easily solved by both Greedy and the various DSTS versions. Still, DSTS outperforms Greedy, since even in such sparse settings there are a few conflicts that Greedy fails to resolve due to its simplistic and obviously greedy nature.

Table 3 shows the average number of messages exchanged during the run of DSTS using DSA with $p = 0.9$ and Table 4 presents the run-time comparison with Greedy. Here, the run-time of DSTS is only slightly higher than that of Greedy. Note that realistic problems produce less messages and run faster than the conflicting problems (cf. Tables 1 and 2); this further indicates that density is an extremely important factor. Another conclusion that can be drawn from these results is that due to their sparseness, realistic problems of much larger sizes (in terms of numbers of requests) could be solved in practice using DSTS.

6 Conclusions

In this work we proposed DSTS, a novel method that models *EOS* scheduling problems as DCOPs and solves them using standard DCOP algorithms. Our

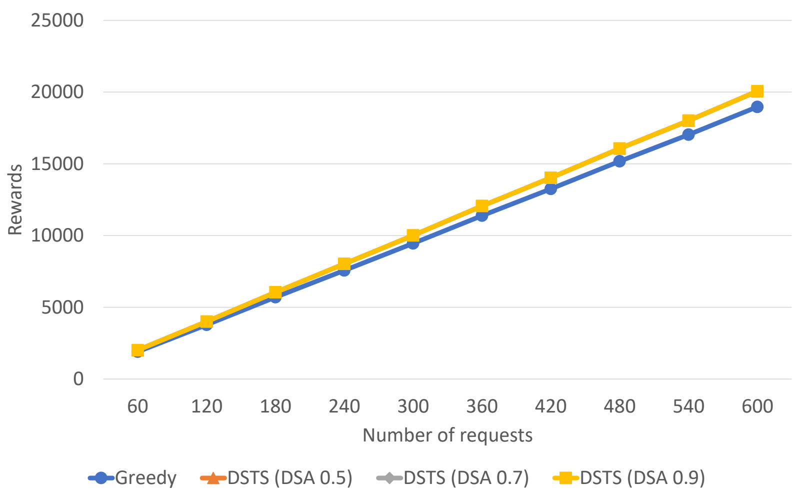


Fig. 3: Rewards of realistic problems

Requests	60	120	180	240	300	360	420	480	540	600
DSTS	18.6	77.7	181.5	321.4	514.6	765	1056	1393	1817	2283

Table 3: Number of messages in realistic problems

experiments reveal that DSTS fastly solves the EOS scheduling problems and provides higher quality solutions than the Greedy benchmark algorithm currently used.

Modeling the EOS scheduling problem as a DCOP is natural since the problem is inherently distributed. Moreover, by applying an algorithm with no predefined ordering or structure, such as DSA, all users "have the same starting point" when participating in the solution process, which is considered fair [1]. It should be noted that even though the users do not need to a priori share their private information with some central authority, some private information may be leaked during the DCOP solving process [29,12]. In situations where privacy is an important issue, one may resolve to using a privacy-preserving DCOP algorithm in phase two of DSTS. One may choose a complete privacy-preserving algorithm (e.g., [23,13]) or preferably an incomplete one (e.g., [38,14]) for appli-

Requests	60	120	180	240	300	360	420	480	540	600
DSTS	0.17	0.2	0.32	0.37	0.47	0.53	0.65	0.85	0.84	1.12
Greedy	0.23	0.16	0.24	0.26	0.29	0.35	0.43	0.52	0.58	0.72

Table 4: Run-time of realistic problems (milliseconds)

cability reasons, as privacy preservation comes with a price tag of considerably higher overheads.

An interesting feature of the problem at hand is that users do not know in advance with whom they are constrained, which is considered trivial information in other problem domains (e.g., meeting scheduling [28]). This problem can be handled in a privacy-preserving manner by employing standard multi-party computation methods. Basically, each pair of users has to construct a Boolean or arithmetic circuit that represents the time intervals of the observation opportunities. That circuit can be evaluated using techniques of cryptography (e.g., garbled circuits [47]) to obtain mutual information of the overlaps without learning anything else. Despite the cryptographic workload, such a preprocessing stage is performed only once by each pair of users ($\mathcal{O}(n^2)$) and, therefore, does not heavily influence the overall performance. Another solution for the preprocessing stage is to delegate these computations to a set of external mediators. Recent studies in the DCOP field showed that such mediators may perform the computations in an oblivious manner, without gaining access neither to the problem inputs nor to its outputs [37,21] (this is in contrast to the centralized approach in which the central authority is exposed to the inputs and outputs). An interesting direction for future work is to devise a mediation-based solution in which the mediators can perform both the preprocessing stage and the DCOP-solving stage.

In this work we do not assume that the satellites have limited capacity; however, some real-world satellites do have such limitation. Applying this limitation to our model is not trivial since it requires adding global constraints, which are known to impair the performance. Nonetheless, a solution to a similar problem has been successfully applied recently, including the development of new variation of DSA that focuses on problems with limited capacity [19]. Therefore, employing a similar solution in DSTS is another prospect for future work. Yet another issue is that of dynamic changes; satellites are often affected by environmental changes (e.g., clouds) or may receive emergency requests. We, therefore, plan to follow recent advances on dynamic DCOPs [16] to enable dynamic modifications of the timetables.

Acknowledgments This work was partially supported by the Ariel Cyber Innovation Center in conjunction with the Israel National Cyber Directorate in the Prime Minister's Office.

References

1. Abdulkadiroğlu, A., Sönmez, T.: Random serial dictatorship and the core from random endowments in house allocation problems. Econometrica **66**(3), 689–701 (1998)
2. Bataille, N., Lemaitre, M., Verfaillie, G.: Efficiency and fairness when sharing the use of a satellite. In: Artificial Intelligence, Robotics and Automation in Space. vol. 440, p. 465 (1999)

3. Bianchessi, N., Cordeau, J.F., Desrosiers, J., Laporte, G., Raymond, V.: A heuristic for the multi-satellite, multi-orbit and multi-user management of earth observation satellites. European Journal of Operational Research **177**(2), 750–762 (2007)
4. Braquet, M., Bakolas, E.: Greedy decentralized auction-based task allocation for multi-agent systems. IFAC-PapersOnLine **54**(20), 675–680 (2021)
5. Cho, D.H., Kim, J.H., Choi, H.L., Ahn, J.: Optimization-based scheduling method for agile earth-observing satellite constellation. Journal of Aerospace Information Systems **15**(11), 611–626 (2018)
6. Denis, G., Claverie, A., Pasco, X., Darnis, J.P., de Maupeou, B., Lafaye, M., Morel, E.: Towards disruptions in earth observation? new earth observation systems and markets evolution: Possible scenarios and impacts. Acta Astronautica **137**, 415–433 (2017)
7. Farinelli, A., Rogers, A., Jennings, N.R.: Decentralised coordination of low-power embedded devices using the max-sum algorithm. In: AAMAS. pp. 639–646 (2008)
8. Farinelli, A., Rogers, A., Petcu, A., Jennings, N.R.: Decentralised coordination of low-power embedded devices using the max-sum algorithm. In: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2. pp. 639–646 (2008)
9. Fioretto, F., Pontelli, E., Yeoh, W.: Distributed constraint optimization problems and applications: A survey. Journal of Artificial Intelligence Research **61**, 623–698 (2018)
10. Gershman, A., Meisels, A., Zivan, R.: Asynchronous forward-bounding for distributed constraints optimization. In: Proc. ECAI-06. pp. 103–107. Lago di Garda (August 2006)
11. Gershman, A., Meisels, A., Zivan, R.: Asynchronous forward bounding. Journal of Artificial Intelligence Research **34**, 25–46 (2009)
12. Greenstadt, R., Pearce, J.P., Tambe, M.: Analysis of privacy loss in distributed constraint optimization. In: AAAI. vol. 6, pp. 647–653 (2006)
13. Grinshpoun, T., Tassa, T.: P-SyncBB: A privacy preserving branch and bound DCOP algorithm. Journal of Artificial Intelligence Research **57**, 621–660 (2016)
14. Grinshpoun, T., Tassa, T., Levit, V., Zivan, R.: Privacy preserving region optimal algorithms for symmetric and asymmetric DCOPs. Artificial Intelligence **266**, 27–50 (2019)
15. Hirayama, K., Yokoo, M.: Distributed partial constraint satisfaction problem. In: CP. pp. 222–236 (1997)
16. Hoang, K.D., Hou, P., Fioretto, F., Yeoh, W., Zivan, R., Yokoo, M.: Infinite-horizon proactive dynamic DCOPs. In: Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems. pp. 212–220 (2017)
17. Huang, Y., Mu, Z., Wu, S., Cui, B., Duan, Y.: Revising the observation satellite scheduling problem based on deep reinforcement learning. Remote Sensing **13**(12), 2377 (2021)
18. Junges, R., Bazzan, A.L.: Evaluating the performance of DCOP algorithms in a real world, dynamic problem. In: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2. pp. 599–606 (2008)
19. Khakhiashvili, I., Grinshpoun, T., Dery, L.: Course allocation with friendships as an asymmetric distributed constraint optimization problem. In: 2021 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT). IEEE (2021)
20. Kiekintveld, C., Yin, Z., Kumar, A., Tambe, M.: Asynchronous algorithms for approximate distributed constraint optimization with quality bounds. In: AAMAS. vol. 10, pp. 133–140 (2010)

21. Kogan, P., Tassa, T., Grinshpoun, T.: Privacy preserving DCOP solving by mediation. In: International Symposium on Cyber Security Cryptography and Machine Learning. Springer (2022)
22. Le, T., Son, T.C., Pontelli, E., Yeoh, W.: Solving distributed constraint optimization problems using logic programming. Theory and Practice of Logic Programming **17**(4), 634–683 (2017)
23. Léauté, T., Faltings, B.: Protecting privacy through distributed computation in multi-agent decision making. Journal of Artificial Intelligence Research **47**, 649–695 (2013)
24. Lezama, F., Palominos, J., Rodríguez-González, A.Y., Farinelli, A., Munoz de Cote, E.: Agent-based microgrid scheduling: An ict perspective. Mobile Networks and Applications **24**(5), 1682–1698 (2019)
25. Liu, L., Dong, Z., Su, H., Yu, D.: A study of distributed earth observation satellites mission scheduling method based on game-negotiation mechanism. Sensors **21**(19), 6660 (2021)
26. Liu, Y., Chen, Q., Li, C., Wang, F.: Mission planning for earth observation satellite with competitive learning strategy. Aerospace Science and Technology **118**, 107047 (2021)
27. Lutati, B., Gontmakher, I., Lando, M., Netzer, A., Meisels, A., Grubshtein, A.: AgentZero: A framework for simulating and evaluating multi-agent algorithms. Agent-Oriented Software Engineering: Reflections on Architectures, Methodologies, Languages, and Frameworks pp. 309–327 (2014)
28. Maheswaran, R.T., Tambe, M., Bowring, E., Pearce, J.P., Varakantham, P.: Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. In: AAMAS. pp. 310–317. New York, NY, USA (2004)
29. Maheswaran, R.T., Pearce, J.P., Bowring, E., Varakantham, P., Tambe, M.: Privacy loss in distributed constraint reasoning: A quantitative framework for analysis and its applications. Autonomous Agents and Multi-Agent Systems **13**(1), 27–60 (2006)
30. Mailler, R., Lesser, V.: Asynchronous Partial Overlay: A New Algorithm for Solving Distributed Constraint Satisfaction Problems. Journal of Artificial Intelligence Research **25**, 529–576 (April 2006), http://mas.cs.umass.edu/paper/397
31. Modi, P.J., Shen, W., Tambe, M., Yokoo, M.: Adopt: asynchronous distributed constraints optimization with quality guarantees. Artificial Intelligence **161**(1-2), 149–180 (2005)
32. Petcu, A., Faltings, B.: A scalable method for multiagent constraint optimization. In: IJCAI. pp. 266–271. Edinburgh, Scotland, UK (2005)
33. Petcu, A., Faltings, B.: ODPOP: An algorithm for open/distributed constraint optimization. In: AAAI. pp. 703–708. Boston, MA, USA (2006)
34. Picard, G.: Auction-based and distributed optimization approaches for scheduling observations in satellite constellations with exclusive orbit portions. arXiv preprint arXiv:2106.03548 (2021)
35. Picard, G., Caron, C., Farges, J.L., Guerra, J., Pralet, C., Roussel, S.: Autonomous agents and multiagent systems challenges in earth observation satellite constellations. In: International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021). pp. 39–44 (2021)
36. Sun, H., Xia, W., Wang, Z., Hu, X.: Agile earth observation satellite scheduling algorithm for emergency tasks based on multiple strategies. Journal of Systems Science and Systems Engineering **30**(5), 626–646 (2021)
37. Tassa, T., Grinshpoun, T., Yanai, A.: PC-SyncBB: a privacy preserving collusion secure DCOP algorithm. Artificial Intelligence **297**, 103501 (2021)

38. Tassa, T., Grinshpoun, T., Zivan, R.: Privacy preserving implementation of the Max-Sum algorithm and its variants. Journal of Artificial Intelligence Research **59**, 311–349 (2017)
39. Walker, J.G.: Satellite constellations. Journal of the British Interplanetary Society **37**, 559 (1984)
40. Wang, J., Demeulemeester, E., Hu, X., Wu, G.: Expectation and saa models and algorithms for scheduling of multiple earth observation satellites under the impact of clouds. IEEE Systems Journal **14**(4), 5451–5462 (2020)
41. Wang, J., Zhu, X., Yang, L.T., Zhu, J., Ma, M.: Towards dynamic real-time scheduling for multiple earth observation satellites. Journal of Computer and System Sciences **81**(1), 110–124 (2015)
42. Wang, X., Gu, Y., Wu, G., Woodward, J.R.: Robust scheduling for multiple agile earth observation satellites under cloud coverage uncertainty. Computers & Industrial Engineering **156**, 107292 (2021)
43. Wang, X., Wu, G., Xing, L., Pedrycz, W.: Agile earth observation satellite scheduling over 20 years: Formulations, methods, and future directions. IEEE Systems Journal **15**(3), 3881–3892 (2020)
44. Wei, L., Chen, Y., Chen, M., Chen, Y.: Deep reinforcement learning and parameter transfer based approach for the multi-objective agile earth observation satellite scheduling problem. Applied Soft Computing **110**, 107607 (2021)
45. Wei, L., Xing, L., Wan, Q., Song, Y., Chen, Y.: A multi-objective memetic approach for time-dependent agile earth observation satellite scheduling problem. Computers & Industrial Engineering **159**, 107530 (2021)
46. Xiang, S., Xing, L., Wang, L., Zhou, Y., Peng, G.: Enhanced pigeon inspired optimisation approach for agile earth observation satellite scheduling. International Journal of Bio-Inspired Computation **17**(3), 131–141 (2021)
47. Yao, A.C.: Protocols for secure computation. In: FOCS. pp. 160–164 (1982)
48. Yeoh, W., Felner, A., Koenig, S.: BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm. Journal of Artificial Intelligence Research **38**, 85–133 (2010)
49. Yifang, B., Gong, P., Gini, C.: Global land cover mapping using earth observation satellite data: Recent progresses and challenges. ISPRS journal of photogrammetry and remote sensing (Print) **103**(1), 1–6 (2015)
50. Zhang, W., Wang, G., Xing, Z., Wittenburg, L.: Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. Artificial Intelligence **161**(1-2), 55–87 (2005)

Noname manuscript No.
(will be inserted by the editor)

A Constraint Language For University Timetabling Problems

Vincent Barichard · Corentin Behuet · David Genest · Marc Legeay · David Lesaint

Received: date / Accepted: date

Abstract We present a domain-specific modeling language for a class of university timetabling problems (UTP) that involve course scheduling, resource allocation and student sectioning. The UTP language combines a formal domain model and a rules formalism to state constraints. The model is based on a multi-scale schedule horizon (i.e., weeks, weekdays and daily slots), a hierarchical course structure (i.e., course parts, part classes and class sessions), and an extended set of resources (i.e., rooms, lecturers, students and student groups). Student groups must be formed to populate classes and class sessions are to be scheduled individually and allocated single or multiple rooms and lecturers. The model encodes sectioning constraints on classes, core scheduling constraints on sessions as well as compatibility, capacity and cardinality constraints on resource allocation. Rules allow to state conjunctions of constraints on selected sets of entities and sessions using a catalog of timetabling predicates and a syntax to group, filter and bind entities and sessions. As for implementation, the UTP language is based on XML and comes with a tool chain that flattens rules into constraints and converts instances to solver-compatible formats. We present here the abstract syntax of the UTP language and alternative constraint programming models developed in MiniZinc and CHR together with preliminary experiments on a real case study.

Keywords University Timetabling · Domain-Specific Modeling Language · Constraint Programming · Resource Scheduling

This work has been funded by a research grant from Université d'Angers.

Vincent Barichard, Corentin Behuet, David Genest, Marc Legeay, David Lesaint
Univ Angers, LERIA, SFR MATHSTIC, F-49000 Angers, France
Tel.: +33 241-735-420
E-mail: vincent.barichard@univ-angers.fr
E-mail: corentin.behuet@univ-angers.fr
E-mail: david.genest@univ-angers.fr
E-mail: marc.legeay@univ-angers.fr
E-mail: david.lesaint@univ-angers.fr

1 Introduction

Course and exam organization in universities involves strategic, tactical and operational decisions relating to curriculum design, student sectioning, course staffing, room planning, class scheduling and resource allocation [28]. These computational tasks and their overall coordination vary between countries and educational institutions as does the level of process automation and decision tool support [35]. In French universities for instance (see Figure 1), curricula are conventionally revisited every 5 years and students enroll in courses prior to each teaching period in the course of the academic year. Demand is matched by sectioning courses into classes, partitioning students into fixed groups, and populating classes with groups. Eligible groups, lecturers, rooms and equipment are then identified for each course before class sessions get scheduled and allocated the necessary resources. Each stage involves different stakeholders with their own requirements (faculty departments, administrative units, course owners, lecturers, tutors, etc.) and the workflow naturally allows for deviations and contingencies (marginal amendments to curricula on a yearly basis, late student registrations, staff absences, etc.).

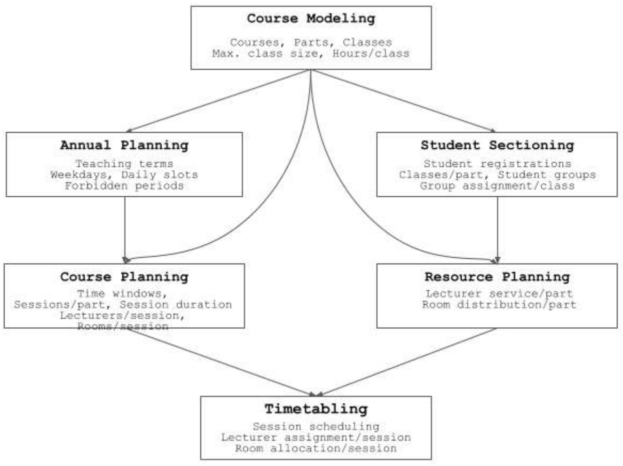


Fig. 1 Conventional workflow for course organization in French universities.

Various problem formulations together with data formats and algorithms have been proposed in the literature to tackle specific aspects of university timetabling including curriculum balancing [15,17,33], student sectioning [31, 34], examination timetabling [13,8,29], curriculum-based or post-enrolment-based course timetabling [29,10,27,12,22,16], tutor allocation [14], and minimal timetabling perturbation [28,26]. Modeling languages have also been developed, notably the `XML` language used in the 2019 international timetabling competition [30,24] (which we refer to as the `ITC-2019` language) which provides a catalog of constraints and supports model variability. We adopt a similar approach in this paper and introduce a class of university timetabling problems called UTP that involve course scheduling, resource allocation and student sectioning. We present a domain-specific language to model UTP instances (UTP language) which is designed around a formal domain model and a rules language to state constraints. Each instance is decomposed into a model of entities, a rule set and a solution component. Rules express collections of timetabling constraints on model entities and the solution component lists assignment decisions. The latter may be void, partial or inconsistent to accommodate different contexts (e.g., a solution for student sectioning to turn into a complete timetable, an outdated solution that must be revised or repaired).

Similarly to the ITC-2019 language, the UTP language adopts a multi-scale schedule horizon (i.e., weeks, weekdays and daily slots), a mixed set of resources (i.e., students, student groups, rooms and lecturers), and a hierarchical course structure (i.e., course parts, part classes and class sessions). In our approach however, class sessions (a.k.a., class meetings) are considered as first-class objects that must be scheduled individually alongside resources. The model supports single-resource sessions (e.g., single lecturer) as well as multi-resource sessions (e.g., hybrid teaching), and encodes core constraints relating to student sectioning, session scheduling and resource allocation. All resources are assumed cumulative (i.e., rooms, lecturers and students may host, teach and attend overlapping sessions) but this policy may be overridden with disjunctive scheduling rules. The rules language effectively allows to enforce additional constraints on selected sets of sessions and entities (i.e., resources and course elements). Rules are expressed using a catalog of timetabling predicates and a comprehension syntax to group, filter and bind sessions and entities. Specifically, each rule denotes a conjunction of UTP constraints sharing the same predicate (e.g., periodicity of all lecture classes of a course) and constraints are technically generated through a rule flattening process.

Note that all constraints are handled as hard constraints and each UTP instance is reduced to a hard constraint satisfaction problem (CSP). The ability to model preferences and multi-criteria objectives by the means of soft constraints is paramount in course timetabling and will be the subject of future extensions. Likewise, the catalog of UTP predicates still lacks important constraints (e.g., gap, distribution and pattern constraints - see e.g. [6,16]) which will be gradually added in future versions.

As for implementation, the UTP language is based on XML and embedded in two constraint modeling languages, namely, MiniZinc [32,3] and CHR [18]. We developed a tool chain consisting of a XML parser, a rule processor to flatten rules into constraints, and an encoder to convert the resulting instances to solver-compatible formats (see Figure 2). Beyond MiniZinc and CHR, constraint-based UTP instances may be used as inputs to any solver implementing the model and predicates of the UTP language. We do not discuss here the XML syntax of the language (the reader is referred to [1] which provides access to the detailed specification, the MiniZinc and CHR models, the

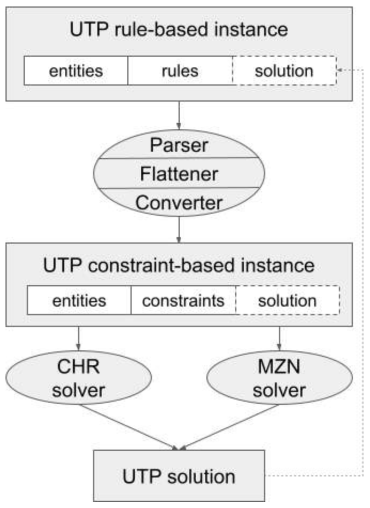


Fig. 2 The UTP toolchain.

tool suite, and a benchmark of instances). Rather, we present the abstract syntax of the UTP language and provide semantics for the key components.

The remainder of the paper is organized as follows. Section 2 introduces the UTP language and draws a comparison with the ITC-2019 schema. Section 3 presents a generic constraint-based UTP model. Section 4 discusses its implementation using MiniZinc and CHR and the cross-validation of the models on a real instance. Section 5 concludes and discusses extensions of this work.

2 University Timetabling Problem

A UTP instance is defined by an entity model and a rules set. A solution to a UTP instance is a list of choices made for all the decisions at stake that satisfies the core constraints of the entity model and the constraints expressed by the rules. We provide in this section an informal description and set-theoretic semantics for the UTP language components, namely the entity model (Section 2.1), constraints (Section 2.2), rules (Section 2.3) and solution (Section 2.4). Section 2.5 draws a comparison between the UTP language and the ITC-2019 schema.

2.1 Entity model

The entity model of a UTP instance defines its schedule horizon, course structure and resources, as well as properties of entities and relational maps (see Figure 3 for a sketch of the meta-model and Figure 4 for a toy example). First, the entity model uses a time grid that decomposes into weeks, weekdays and daily slots. Weeks share the same weekdays and weekdays the same daily slots. The latter make up 24 hours and have the same duration. Note that neither successive weeks nor successive weekdays are assumed to be consecutive. The schedule horizon is implicitly defined by the series of time slots mapping to week, weekday and daily slot combinations. Slots hence serve as time points to represent start and end times of course sessions and to measure session duration, travel time and any gap between sessions.

Courses have a tree-structure wherein each course (e.g., Algorithms) decomposes into parts (e.g., Lecture and Lab), parts into classes (e.g., lecture classes A and B), and classes into sessions (e.g., sessions 1 to 10 for each lecture class). Class sessions are the elementary tasks to schedule when solving a `UTP` instance and the model fixes their number, duration and sequencing. First, the classes of a course part are decomposed into an identical number of sessions of equal duration, both constants being part-specific. Although this approach forbids classes using different session durations in a course part, it is paramount to capture requirements that rely on clear-cut sessions (e.g., starting lab classes after 2 lecture sessions, synchronizing the 5th sessions of the lab classes for a joint examination). Second, the sessions of a class are ranked in the model and must be sequenced accordingly in any solution (session 1 before session 2 . . .). Note that sessions are considered uninterruptible and, in particular, may not overlap two days.

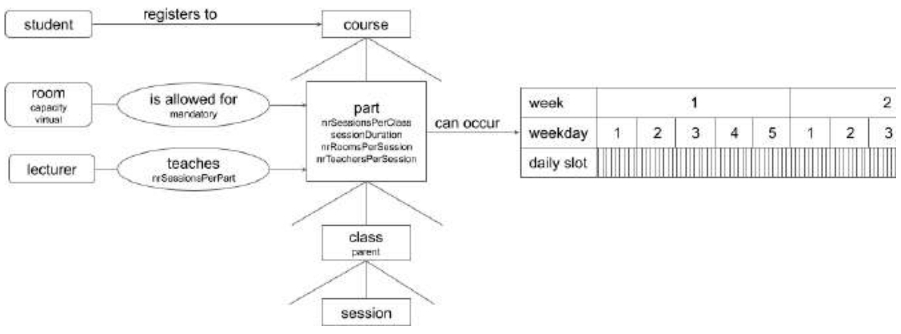


Fig. 3 Entity meta-model.

`UTP` resources fall into 4 types, namely, rooms, lecturers, students and (student) groups. All the resources of an instance, except groups (see Section 2.4), are declared and typed in the entity model. In practice, upstream processes and decisions determine the suitable rooms, eligible lecturers, candidate students and allowed times for the different courses (e.g., faculties prescribing degree-specific time grids, departments implementing room pooling policies and naming lecturers for courses, students registering to courses). These compatibility constraints are modeled by associating sets of possible start times, rooms and lecturers to each course part and a set of registered students to each course. Each session then inherits the sets of allowed resources from the course part and the course it belongs to.

The entity model also encodes flow constraints that govern the distribution of resources over courses based on student registrations and capacity planning decisions (e.g., workload distribution between lecturers). First, each lecturer is allocated a fixed number of sessions in each course part he is eligible for, leaving lecturer-to-session assignment decisions to solvers. Second, each room allowed in a course part may be freely allocated to any session of the part (possibly none) but the model provides the flexibility to mark a room as mandatory in which case it will host or co-host all the sessions. As for students, the sectioning policy is implicit and complies with the course structure, i.e., each student must be assigned to a single class in each part of a course he has registered to and attend all sessions of these classes. In addition, the model supports group nesting constraints between classes to implement course-specific policies (e.g., aggregating student groups bottom-up from labs to lectures) or cross-course sectioning (e.g., imposing the same groups between classes of different courses of a curriculum).

Resource utilization is naturally subject to demand and capacity constraints. Since modalities differ from one environment to the next, the language supports disjunctive and cumulative resources. The default policy is to consider all students, groups, lecturers and rooms as cumulative resources, i.e., they can attend, teach or host simultaneous sessions. Note though that rules may be stated to make some resources fully disjunctive or to prevent specific sessions from overlapping. Support for cumulative resources is paramount to address flexible attendance requirements (e.g., students assigned optional tutoring sessions that may overlap with compulsory courses) or to handle multi-class events (e.g., rooms hosting several classes for an exam or a conference). The model imposes no limits on the number of parallel sessions lecturers and students may attend. Rooms however may only host class sessions whose cumulated headcount is within their capacity. Upper bounds on room capacity and class size are encoded for all rooms and classes and the model also allows uncapacitated rooms to cater for the case of virtual rooms.

The language also supports sessions using multiple resources of the same type. The need for multiple rooms or lecturers arises in practical situations (e.g., multi-room sessions for hybrid teaching, joint supervision of practical work sessions, exams requiring several monitors). To this end, the model associates to each course part the number of lecturers required per session and indicates whether the sessions are single- or multi-rooms. Note that sessions without lecturers or rooms are allowed (e.g., unsupervised student project sessions). The model enforces specific constraints to handle multi-room sessions which override the default room allocation policy. Specifically, students attending the session may be freely dispatched in rooms irrespectively of the group structure, the cumulated capacity of the allocated rooms is taken into account for hosting, uncapacitated rooms cannot be allocated, and the allocated rooms are considered disjunctive for the time of the session.

Note finally that the language provides users with the ability to label resources and course elements to define their own classes of entities (e.g., teams of lecturers, blocks of rooms). Labels together with built-in entity types and identifiers are used to filter entities and to scope rules appropriately.

We formalize below the entity model and introduce notations that will be used thereafter. Let E denote the set of entities and S the set of sessions. E is partitioned into a set of courses C, a set of course parts P, a set of classes K, a set of students U, a set of lecturers L, a set of rooms R, and the singleton domain of courses C^* ($C^* = \{C\}$). Let $\mathcal{E} = \{C^*, C, P, K, U, L, R\}$ denote the set of entity types ($E = \cup_{X\in\mathcal{E}} X$) and $\prec = \{(C^*, C), (C, P), (P, K), (K, S), (U, C), (L, P), (R, P)\}$ denote the relation over $\mathcal{E} \cup \{S\}$ that models the course hierarchy and the distribution of resource types over course components.

$\prec^*$ denotes the transitive closure of $\prec$ over $\mathcal{E} \cup \{S\}$ and $d^{X,Y} : X \to 2^Y$ denotes the function mapping each element of X to its set of compatible elements in Y for each pair $X{\prec^*}Y$. For instance, $d^{R,P}$ represents the distribution of rooms over course parts, $d^{P,K}$ the decomposition of course parts into classes, $d^{K,S}$ the decomposition of classes into sessions, and $d^{R,S}$ the inferred distribution of rooms over sessions. The functions corresponding to the pairs of $\prec$ are directly encoded in the entity model and the remaining functions are defined inductively using recursive aggregation.

We shall denote by $d_i^{X,Y}$ the image of entity i of type X over 2^Y and by $d^{Y,X}$ the inverse of $d^{X,Y}$. Equation (1) below models the hierarchical decomposition of course elements[1], Equation (2) is the closure rule over $\prec^*$, and Equation (3) models inverse maps.

$$\forall (X, Y) \in \{(C^*, C), (C, P), (P, K), (K, S)\} : Y = \sqcup_{i\in X} d_i^{X,Y} \quad (1)$$

$$\forall X, Y, Z \in \mathcal{E} \cup \{S\} : X \preceq^* Y \preceq^* Z \Rightarrow (\forall i \in X : d_i^{X,Z} = \sqcup_{j\in d_i^{X,Y}} d_j^{Y,Z}) \quad (2)$$

$$\forall X, Y \in \mathcal{E} : X \preceq^* Y \Rightarrow (\forall i \in X, j \in Y : j \in d_i^{X,Y} \Leftrightarrow i \in d_j^{Y,X}) \quad (3)$$

Table 1 provides the full list of constants, sets, properties and relational maps encoded in the entity model.[2]

2.2 Predicates and constraints

`UTP` constraints apply to pairs, called e-maps, which associate an entity with a non-empty subset of its compatible sessions. Constraints are built with predicates whose signature includes e-map variables, the number of which is referred to as the arity of the predicate. Note that some predicates may also accept parameters. Let $F = \cup_{X\in\mathcal{E}}\{(e, S') \mid e \in X, S' \subseteq d_e^{X,S} \wedge S' \neq \emptyset\}$ denote the set

[1] $\sqcup$ denotes the disjoint union operation, i.e. set union over pairwise disjoint sets.

[2] The following rules apply. $H = \{i.d.m + j.m + k \mid 0 \le i < w, 0 \le j < d, 1 \le k \le m\}$. For each class k in part p, $\{rank_s^S \mid s \in d_k^{K,S}\} = \{1, \ldots, |d_k^{K,S}|\}$, and $parents_k^{K,K} \not\subset d_p^{P,K}$. For each pair of sessions s, s', $(s, s') \in O$ iff $d_s^{S,K} = d_{s'}^{S,K}$ and $rank_{s'}^S = rank_s^S + 1$. For each course part p, $team_p^P.|d_p^{P,S}| = \sum_{l\in d_p^{P,L}} service_{l,p}^{L\times P}$.

(w, d, m)	the number of weeks w, weekdays d and daily slots m
H	the time slots
E	the entities
$C^* \subseteq E$	the course domain
$C \subseteq E$	the courses
$P \subseteq E$	the course parts
$K \subseteq E$	the classes
$R \subseteq E$	the rooms
$L \subseteq E$	the lecturers
$U \subseteq E$	the students
$d_i^{X,Y} \subseteq Y$	the entities of type Y associated with entity i of type X
$\mathcal{L} \subseteq 2^E$	the labels
$G \subseteq 2^U$	the groups of students
S	the sessions
$d_i^{X,S} \subseteq S$	the sessions compatible with entity i of type X
$d_s^{S,X} \subseteq X$	the entities of type X compatible with session s
$d_s^{S,H} \subseteq H$	the start times allowed for session s
$length_s^S \in H$	the duration of session s
$rank_s^S \in \mathbb{N}^*$	the rank of session s in its class
$O \subseteq S \times S$	the pairs of sessions with consecutive ranks in a class
$parents_k^{K,K} \subseteq K$	the parent classes of class k if any
$maxsize_k^K \in \mathbb{N}$	the maximum size of class k
$capacity_r^R \in \mathbb{N}$	the capacity of room r
$virtual_r^R \in \mathbb{B}$	whether room r is virtual or not
$V \subseteq R$	the virtual rooms
$multi_p^P \in \mathbb{B}$	whether course part p is multi-room or not
$M \subseteq P$	the multi-room parts
$mandatory_p^P \subseteq R$	the mandatory rooms of part p
$team_p^P \in \mathbb{N}$	the number of lecturers required by every session of part p
$service_{l,p}^{L \times P} \in \mathbb{N}$	the number of sessions required by lecturer l in part p

Table 1 Entity model: constants, sets, maps and relations.

of e-maps, a UTP constraint has the form

$$c((e_1, S_1), \ldots, (e_m, S_m), p_1, \ldots, p_n) \tag{4}$$

where c is a predicate symbol of arity m, $(e_1, S_1), \ldots, (e_m, S_m)$ are e-maps ($(e_i, S_i) \in F$, $i = 1 \ldots m$) and $p_1, \ldots, p_n$ are values for the parameters of c ($n \geq 0$). Three constraints (C1, C2, C3) are illustrated in Figure 4.

Every predicate may be used indistinctly with e-maps defined on course elements or on resources. E-maps defined on resources are interpreted as conditional session-to-resource assignments when checking constraints whereas e-maps defined on course elements are unconditional assignments since they model constitutive sessions. In other words, a constraint is only evaluated on the sessions for which its e-map arguments and the considered solution propose the same entity assignment.[3]

[3] Formally, let $x_e^{E,S}$ be the variable denoting the set of sessions assigned to entity e and $S'_1, \ldots, S'_m$ be sets of sessions, the conditionality of a constraint c is stated as follows: $(x_{e_1}^{E,S} = S'_1 \wedge \ldots \wedge x_{e_m}^{E,S} = S'_m) \Rightarrow (c((e_1, S_1), \ldots, (e_m, S_m), p_1, \ldots, p_n) \Leftrightarrow c((e_1, S_1 \cap S'_1), \ldots, (e_m, S_m \cap S'_m), p_1, \ldots, p_n))$.

It follows that a constraint is evaluated on every session that is mapped to a course element by one of its e-map arguments. Constraints that apply exclusively to course elements are therefore unconditional. Note also that the use of e-maps that model the whole set of sessions compatible with an entity will necessarily constrain any session that may be assigned to this entity.

Name	Arity	Parametric	Semantics
same_daily_slot	1	no	Sessions start on the same daily slot
same_weekday	1	no	Sessions start on the same weekday
same_weekly_slot	1	no	Sessions start on the same weekly slot
same_week	1	no	Sessions start the same week
same_day	1	no	Sessions start the same day
same_slot	1	no	Sessions start at the same time
forbidden_period	1	yes	Sessions cannot start in the given time period
at_most_daily	1	yes	The number of sessions scheduled in the daily period is upper-bounded
at_most_weekly	1	yes	The number of sessions scheduled in the weekly period is upper-bounded
sequenced	≥ 2	no	Sessions are sequenced
weekly	1	no	Sessions are weekly
no_overlap	1	no	Sessions cannot overlap
travel	1	yes	Travel time is factored in if sessions hosted in the given rooms
same_rooms	1	no	Sessions are hosted in the same room(s)
same_students	1	no	Sessions are attended by the same student(s)
same_lecturers	1	no	Sessions are taught by the same lecturer(s)
adjacent_rooms	1	yes	Sessions are hosted in the given adjacent rooms
lecturer_distribution	≥ 2	yes	Distributes lecturer workload over classes

Table 2 Catalog of UTP predicates.

Table 2 lists the predicates of the language and indicates which are variadic or parametric. The first predicates same_daily_slot, ..., same_slot enforce common restrictions on the start times of the targeted sessions (e.g., sessions starting the same day). Additionally, any start time interval may be forbidden by passing its start and end points as parameters to predicate forbidden_period. Predicates at_most_daily and at_most_weekly upper-bound the number of sessions scheduled daily or weekly within the given time interval. sequenced is a n-ary predicate ($n \geq 2$) which constrains the latest session of the i-th e-map to end before the earliest session of $i+1$-th e-map ($i = 1..n-1$). Predicate weekly ensures sessions are scheduled weekly without presuming any particular sequencing. Predicate no_overlap ensures sessions do not overlap in time and is typically used to model disjunctive resources. Predicate travel factors in any travel time incurred between consecutive sessions hosted in distant rooms. The travel time matrix is a parameter of the predicate. same_rooms, same_students and same_lecturers require that sessions be assigned to the same set of rooms, students or lecturers. Predicate adjacent_rooms require that sessions be hosted in adjacent rooms based on an adjacency graph passed as a parameter. Lastly, predicate lecturer_distribution distributes the volumes of sessions represented by the different e-map arguments among different lecturers. Lecturers and session volumes are parameters of the predicate.

2.3 Rules

Rules are used to state conjunctions of constraints and in particular single constraints. Each rule is defined by a universally quantified formula which bounds the domains of the e-map variables of a given predicate. The collection of constraints hence represented is derived by instantiating the predicate with each tuple of e-maps belonging to the cross-product of the prescribed domains. E-map domains are not given in extension but represented using a language of selectors allowing to generate and filter e-maps. Let $\mathcal{F}$ denote the language of e-map domain selectors, a `UTP` rule has the form

$$c(F_1, \ldots, F_m, p_1, \ldots, p_n) \tag{5}$$

and is interpreted by the formula

$$\forall (e_1, S_1) \in [\![F_1]\!], \ldots, (e_m, S_m) \in [\![F_m]\!] : c((e_1, S_1), \ldots, (e_m, S_m), p_1, \ldots, p_n) \tag{6}$$

where c is a predicate symbol of arity m, $F_1, \ldots, F_m$ are selectors ($F_i \in \mathcal{F}$, $i = 1 \ldots m$), $[\![F_i]\!]$ denotes the domain of e-maps represented by selector $F_i \in \mathcal{F}$, and $p_1, \ldots p_n$ are values for the parameters of c ($n \geq 0$), .

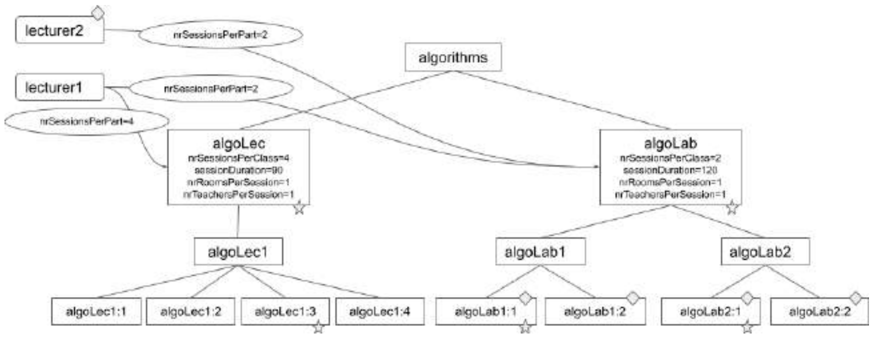
The language of selectors allows to target entities based on type, label or identifier and to filter their sets of sessions based on session rank and mutual compatibility with other entities. It is complete in the sense that it allows to construct any domain of e-maps whose entities share the same type. For instance, one may construct the e-maps which associate any of the rooms labeled `Building-A` with the compatible sessions of rank 2 or 4 that are also constitutive of course `course-1` or class `class-3`. A selector combines a generator and an optional list of filters. Generators and filters are triples (T_i, L_i, O_i) consisting of an entity type T_i, an entity label or identifier L_i and a subset of session ranks O_i (a.k.a., session mask), the latter two elements being optional. A selector matches any e-map whose entity satisfies the type, label and identifier constraints of the generator and whose image includes any compatible session satisfying the mask of the generator and one of the filters. Note that rules featuring null selectors are discarded during the flattening stage.

Let $\mathcal{O}$ denote the range of session ranks, $d^{\mathcal{O},S} : \mathcal{O} \rightarrow 2^S$ the rank-based partitioning of sessions ($s \in d_o^{\mathcal{O},S}$ iff $rank_s^S = o$), and $\mathcal{L}^* = \mathcal{L} \cup \{E\} \cup \{\{e\} \mid e \in E\}$ the set of labels completed with the whole set of entities to mock label optionality and singleton entities to support identity-based selection, the language of selectors is the set $\mathcal{F} = \cup_{n \geq 1}(\mathcal{E} \times \mathcal{L}^* \times 2^{\mathcal{O}})^n$. Each selector $d = ((T_1, L_1, O_1), \ldots, (T_k, L_k, O_k))$ decomposes into a generator (T_1, L_1, O_1) and a possibly empty list of filters $((T_2, L_2, O_2), \ldots, (T_k, L_k, O_k))$. d matches any e-map whose entity has type T_1 and label L_1 and whose image includes any compatible session satisfying mask O_1 and any of the filters. The set of e-maps

$[\![d]\!]$ matched by d is defined by

$$[\![d]\!] = \bigcup_{e \in T_1 \cap L_1} \Big\{ (e, S') \mid S' = d_e^{T_1,S} \bigcap \bigcup_{i=2\ldots k} \Big(d^{T_i,S}[L_i] \bigcap d^{O,S}[O_1 \cap O_i] \Big) \\ \wedge S' \neq \emptyset \Big\}$$

where $d^{X,Y}[X'] = \bigcup_{i \in X'} d_i^{X,Y}$ with $X' \subseteq X$.



```
forbidden_period((<(L,lecturer2,_)>,9120,9240)                          (R1)
sequenced(<(K,_,{3}),(P,algoLec,_)>, <(K,_,{1}),(P,algoLab,_)>)         (R2)

forbidden_period((lecturer2,{algoLab1:1,algoLab1:2,algoLab2:1,algoLab2:2}),
9120,9240)                                                              (C1)
sequenced((algoLec1,{algoLec1:3}), (algoLab1,{algoLab1:1}))             (C2)
sequenced((algoLec1,{algoLec1:3}), (algoLab2,{algoLab2:1})              (C3)
```

Fig. 4 Rules flattening and corresponding constraints on a toy example.

Figure 4 illustrates the rules flattening process on a toy example. Course `algorithms` is split into a lecture part `algoLec` and a lab part `algoLab`. The lecture part has a single class of 4 sessions taught by `lecturer1` and the lab part has 2 classes of 2 sessions each taught by `lecturer1` or `lecturer2`. Rule R1 requires that `lecturer2` has no session between slots 9120 and 9240, corresponding for instance to 8am and 10am on Tuesday of week 2. The selector includes no mask and no filter hence matches with all possible sessions of `lecturer2` as indicated with diamonds on Figure 4. The resulting domain of e-maps is the singleton $\{(lecturer2, d^{L,S}_{lecturer2})\}$ and the rule is flattened into a single `forbidden_period` constraint (C1). Rule R2 requires that the first sessions of the labs start after the third lecture. The two selectors include a filter. The first selector matches with all class sessions of rank 3 in part `algoLec`, and the second matches with all class sessions of rank 1 in part `algoLab` as indicated with stars on the figure. The rule is flattened into 2 `sequenced` constraints (C2 and C3) corresponding to the cross product of the e-map domains

$\{(algoLec1, \{s \in S \mid rank_s^S = 3\} \cap d_{algoLec}^{P,S})\}$ and $\{(algoLab1, \{s \in S \mid rank_s^S = 1\} \cap d_{algoLab}^{P,S}), (algoLab2, \{s \in S \mid rank_s^S = 1\} \cap d_{algoLab}^{P,S})\}$.

2.4 Solution

The solution component includes assignment decisions relating to the choice of slots and resources for sessions, the placement of students in groups and the assignment of groups to classes. The solution hence represented may be partial, even empty, and does not have to be consistent with the constraints built in the entity model or entailed by the rules. The support for partial solutions allows to tackle subproblems using separate `UTP` instances and solution seeds. For instance, a scheduling instance may be defined on the basis of partial and consistent solutions pre-generated for the student sectioning and resource allocation subproblems. Likewise, the support for inconsistent solutions is paramount to repair solutions that have become inconsistent due to unforeseen changes.

Student groups are considered a by-product of student sectioning. For this reason, groups may only be listed in the solution component, not in the entity model, and defined both by the students they include and the classes they are assigned to. This sectioning process is subject to different constraints. First, students are partitionned into groups and students are inextricably bound to their group. Second, a group may only include students with identical course registrations. Third, group-to-class assignments must comply with any subgroup inclusion constraint stated in the entity model.

2.5 Related work

We highlight here the main differences between the `UTP` language and the `ITC-2019` language (`ITC-2019` for short).

A first difference between the two frameworks lies in the representation of the possible times a class can meet. In `UTP`, a class is defined by a single sequence of sessions of equal duration and the problem is to schedule each session. In `ITC-2019`, a class is given alternative fixed session schedules (`times` elements in the `XML` schema) and the problem is to choose one of the schedules for the class. A schedule is the repetition over a set of weeks of one or more sessions that have the same duration and start on specific days of the week at the same predefined time (daily slot). The two representations are not reducible to one another. For instance, alternative schedules using different session durations cannot be modeled in `UTP`. Conversely, class schedules where sessions do not necessarily start on the same daily slot cannot be modeled in `ITC-2019`. Nevertheless, basic class schedules may be represented in either approach by stating `ITC-2019` constraints or `UTP` rules on classes. For instance, a class meeting every week on the same day and the same daily slot, both being subject to time restrictions, may be modeled using `same_daily_slot`, `weekly` and `forbidden_period` constraints. The implementation of a more comprehensive reduction method will be the subject of future work.

Second, `ITC-2019` represents alternative course configurations by introducing an intermediate layer in the course hierarchy that sits between courses and parts. The configurations of a course typically differ in their number of (sub)parts and are mutually exclusive from a student sectioning standpoint, that is, a registered student must be assigned a single configuration and attend all of its parts. This feature is not currently supported in `UTP`. As for resources, `UTP` explicitly represents lecturers on par with rooms whereas `ITC-2019` only models rooms. `UTP` also provides the flexibility to allocate different resources within a class (and specify lecturer workload in particular) whereas the same room must be allocated in `ITC-2019`. Additionally, `UTP` supports multi-resource sessions whereas `ITC-2019` is restricted to single-room sessions.

Lastly, the two constraint languages present important differences. While `ITC-2019` constraint predicates apply to classes, `UTP` predicates apply to any set(s) of sessions and may be used in particular on individual sessions, hence granting finer-grained control. Besides, `UTP` rules and the selector language allows to constrain any class of resources or course elements in a concise way.

Lastly, the `ITC-2019` schema addresses the timetabling problem as a combinatorial optimization problem. It includes a cost function weighting 4 criteria which respectively penalize the choice of sessions and rooms for the classes, the violations of constraints and the overlapping of sessions per student. In its current version, the `UTP` language addresses the problem as a hard constraint satisfaction problem. The integration of soft constraints and the possibility of aggregating penalties or preferences, either in solution generation or repair contexts, is under investigation.

3 A Constraint-Based Model for `UTP`

We introduce in this section a constraint-based model for `UTP` instances. The model of an instance combines the constraints associated to the entity model and the constraints resulting from the flattening of the rules, if any. The former are decomposed into 4 fragments relating to student sectioning, resource distribution, session scheduling and resource allocation. We present each fragment in turn by reusing notations of Table 1, illustrate the modeling of some predicates before discussing opportunities for model reformulations on a per-instance basis. Note that some constraints are given a naive formulation to clarify semantics and more efficient implementations using `MiniZinc` and `CHR` will be discussed in Section 4.

Table 3 lists the decision variables of the model. All, except time slot variables, are set variables.

$x_g^{G,U} \subseteq U$	the set of students assigned to group g
$x_k^{K,G} \subseteq G$	the set of groups assigned to class k
$x_s^{S,R} \subseteq R$	the set of rooms assigned to session s
$x_s^{S,L} \subseteq L$	the set of lecturers assigned to session s
$x_r^{R,S} \subseteq S$	the set of sessions assigned to room r
$x_s^{S,H} \in H$	the start slot assigned to session s

Table 3 The decision variables.

3.1 Student Sectioning

The sectioning constraints partition students into groups and assign groups to classes while satisfying sectioning rules and class size upper-bounds. Constraint (1) ensures the groups partition the set of students. Constraint (2) prevents the clustering of students who register to different courses. Constraint (3) ensures that classes of a course part have no shared groups and Constraint (4) that the group of a student attends each part of a course he is registered to. Constraint (5) implements the parent relationships between classes. Constraint (6) ensures maximum class size is never exceeded by the number of students in its groups. Note that expressions ($g \in x_k^{K,G}$) in this contraint denote pseudo-boolean variables. The same notation is used for convenience in other constraints.

$$U = \bigsqcup_{g \in G} x_g^{G,U} \quad (1)$$

$$\forall u, u' \in U, d_u^{U,C} \neq d_{u'}^{U,C}, g \in G : \quad \{u, u'\} \not\subseteq x_g^{G,U} \quad (2)$$

$$\forall p \in P, k, k' \in d_p^{P,K}, k \neq k' : \quad x_k^{K,G} \bigcap x_{k'}^{K,G} = \emptyset \quad (3)$$

$$\forall u \in U, g \in G : \quad (u \in x_g^{G,U}) \rightarrow \bigwedge_{p \in d_u^{U,P}} \bigvee_{k \in d_p^{P,K}} (g \in x_k^{K,G}) \quad (4)$$

$$\forall k \in K, k' \in parents_k^{K,K} : \quad x_k^{K,G} \subseteq x_{k'}^{K,G} \quad (5)$$

$$\forall k \in K : \quad maxsize_k^K \geq \sum_{g \in G} |x_g^{G,U}|.(g \in x_k^{K,G}) \quad (6)$$

3.2 Resource Distribution

Resource distribution involves domain, cardinality and basic summation constraints. Constraint (7) defines the allowed rooms and allowed lecturers per session. Constraint (8) models single-room sessions and Constraint (9) models mandatory rooms of course parts. Constraint (10) ensures sessions get assigned the right number of lecturers (possibly none) as defined in each course part and Constraint (11) ensures each lecturer is assigned the expected number of sessions.

$$\forall W \in \{R, L\}, s \in S: \quad x_s^{S,W} \subseteq d_s^{S,W} \tag{7}$$

$$\forall p \in P \setminus M, s \in d_p^{P,S}: \quad |x_s^{S,R}| = 1 \tag{8}$$

$$\forall p \in P, s \in d_p^{P,S}: \quad mandatory_p^P \subseteq x_s^{S,R} \tag{9}$$

$$\forall p \in P, s \in d_p^{P,S}: \quad team_p^P = |x_s^{S,L}| \tag{10}$$

$$\forall l \in L, p \in P: \quad service_{l,p}^{L \times P} = \sum_{s \in d_p^{P,S}} (l \in x_s^{S,L}) \tag{11}$$

3.3 Session Scheduling and Resource Allocation

Session scheduling and resource allocation involve positioning, sequencing, non-overlapping and capacity constraints. Constraint (12) defines the allowed slots per session and Constraint (13) ensures sessions do not span over two days. Constraint (14) sequences sessions if they are ranked consecutively in a class. Constraint (15) models multi-room class sessions and enforces exclusive access to their rooms. This constraint is formulated using auxiliary predicate $split(w, S_1, S_2)$ (18) which ensures no session of S_1 overlaps with a session of S_2 if both are assigned to resource w. We provide a naive decomposition of this predicate using Predicate (19). Constraints (16) and (17) model room utilization and capacity limits and use auxiliary variables $y_{r,k,s,h}$. $y_{r,k,s,h}$ models the number of students attending session s of class k in room r at time h and is defined using auxiliary constraint (20). Constraint (16) is the default cumulative constraint which applies to non-virtual rooms when allocated to single-room sessions. Constraint (17) handles the specific case of multi-room sessions and ensures the cumulated capacity of the rooms used by a multi-room session exceeds the number of students attending the session. Note that the constraint is purely quantitative and allows each individual group to be distributed over different rooms.

$$\forall s \in S: \quad x_s^{S,H} \in d_s^{S,H} \tag{12}$$

$$\forall s \in S: \quad x_s^{S,H}/m = (x_s^{S,H} + length_s^S)/m \tag{13}$$

$$\forall (s, s') \in O: \quad x_s^{S,H} + length_s^S \leq x_{s'}^{S,H} \tag{14}$$

$$\forall k \in d^{P,K}[M]: \quad \bigwedge_{r \in d_k^{K,R}} split(r, d_k^{K,S}, d_r^{R,S} \setminus d_k^{K,S}) \tag{15}$$

$$\forall r \in R \setminus V: \quad \bigwedge_{h \in H} capacity_r^R \geq \sum_{\substack{p \in d_r^{R,P} \setminus M \\ k \in d_p^{P,K} \\ s \in d_k^{K,S}}} y_{r,k,s,h} \tag{16}$$

$$\forall p \in M: \quad \bigwedge_{\substack{h \in H \\ k \in d_p^{P,K} \\ s \in d_k^{K,S}}} \sum_{r \in d_p^{P,R}} (r \in x_s^{S,R}).capacity_r^R \geq \max_{r \in d_p^{P,R}} y_{r,k,s,h} \tag{17}$$

Let $W \in \{R, L, U\}, w \in W, S_1, S_2 \subseteq S$:

$$split(w, S_1, S_2) \leftrightarrow \bigwedge_{\substack{s_1 \in S_1, s_2 \in S_2 \\ s_1 \neq s_2}} (w \in x_{s_1}^{S,W} \cap x_{s_2}^{S,W} \rightarrow split(s_1, s_2)) \tag{18}$$

Let $s, s' \in S$:

$$split(s, s') \leftrightarrow (x_s^{S,H} + length_s^S \leq x_{s'}^{S,H} \vee x_{s'}^{S,H} + length_{s'}^S \leq x_s^{S,H}) \tag{19}$$

Let $r \in R, k \in K, s \in S, h \in H$:

$$\begin{aligned} y_{r,k,s,h} = {} & (r \in x_s^{S,R}).(x_s^{S,H} \leq h \wedge h \leq x_s^{S,H} + length_s^S) \\ & .\sum_{g \in G} |x_g^{G,U}|.(g \in x_k^{K,G}) \end{aligned} \tag{20}$$

3.4 `UTP` Predicates

We present a subset of `UTP` constraint predicates, namely, `forbidden_period` (21), `same_weekday` (22), `same_rooms` (23), `no_overlap` (24) and `sequenced` (25). Note that `forbidden_period` accepts start and end point parameters. Predicate `no_overlap` uses auxiliary predicate *split* for resources (18) and a variant for course elements (26).
Let $X \in \mathcal{E}, e \in X, S' \subseteq d_e^{X,S}, h, h' \in H \ (h < h')$.

$$\begin{aligned} & forbidden\_period((e, S'), h, h') \\ & \qquad \leftrightarrow \bigwedge_{s \in S'} (x_s^{S,H} + length_s^S \leq h \vee h' < x_s^{S,H}) \end{aligned} \tag{21}$$

$$same\_weekday((e, S')) \leftrightarrow \bigwedge_{s \in S'} x_s^{S,H}/d = (x_s^{S,H} + length_s^S)/d \tag{22}$$

$$same\_rooms((e, S')) \leftrightarrow \bigwedge_{s, s' \in S'} (x_s^{S,R} = x_{s'}^{S,R}) \tag{23}$$

$$no\_overlap((e, S')) \leftrightarrow split(e, S', S') \tag{24}$$

Let $i \in \{1, \ldots, n\}, X_i \in \mathcal{E}, e_i \in X_i, S_i \subseteq d_{e_i}^{X_i,S}$:

$$\begin{aligned} & sequenced((e_1, S_1), \ldots, (e_n, S_n)) \\ & \qquad \leftrightarrow \bigwedge_{j=1 \ldots n-1} \max_{s \in S_j} (x_s^{S,H} + length_s^S) \leq \min_{s \in S_{j+1}} x_s^{S,H} \end{aligned} \tag{25}$$

Let $X \in \{C^*, C, P, K\}, e \in X, S_1, S_2 \subseteq S$:

$$split(e, S_1, S_2) \leftrightarrow \bigwedge_{\substack{s_1 \in S_1, s_2 \in S_2 \\ s_1 \neq s_2}} (e \in d_{s_1}^{S,X} \cap d_{s_2}^{S,X} \rightarrow split(s_1, s_2)) \tag{26}$$

3.5 Reformulation

The model presented above is generic and may be adapted on a per instance basis depending on the features and rules at stake. We discuss here a few variants of the `UTP` problem which provide opportunities for model reformulation.

When instances only involve single-room sessions ($M = \emptyset$), one may adopt integer or enumerated room allocation variables instead of set variables $x^{S,R}_s$ and rewrite constraints accordingly. In the same way, lecturer assignment variables and constraints may be adapted when a single lecturer is required per session. Note that hybrid models mixing single or multi-resource session variables may be considered too. The temporal model may also be simplified when the time grid is coarse-grain and guarantees no session can span over consecutive start times ($\forall s \in S, length^S_s \leq \min(\{h' - h \mid h, h' \in d^{P,H}[P] \wedge h < h'\})$. This situation occurs in institutions that impose a common time grid to ensure sessions (with any travel time incurred) necessarily fit in each time slot. If so, sessions may be handled as time points rather than time intervals and temporal predicates and constraints may be adapted. Capacity constraints may also be simplified for disjunctive rooms. A room is disjunctive if a `no_overlap` constraint is stated on the whole set of its compatible sessions ($r \in D \leftrightarrow \mathit{no\_overlap}(r, d^{R,S}_r)$ where $D \subseteq R$ denotes the set of disjunctive rooms). If so, the default cumulative constraint (16) may be overridden by Constraint (27).

$$\forall r \in D : \bigwedge_{\substack{h \in H \\ k \in d^{R,K}_r}} capacity^R_r \geq \max_{s \in d^{K,S}_k} y_{r,k,s,h} \tag{27}$$

4 Constraint Programming Implementation

In this section, we present two constraint-based models for `UTP` instances developed in `MiniZinc` and `CHR`. The two models use the same arrays, functions and constants for representing input data. We do not list them here but they are easily understandable such as `part_sessions` which gives the set of sessions constitutive of a part, `session_rooms` which gives the set of allowed rooms for a session, `week` which gives the week of a slot, and `nr_weekly_slots` which is the number of slots in a week.

4.1 `MiniZinc` model

`MiniZinc` is a high-level language to model constrained optimization problems [32,3]. `MiniZinc` models are translated into `Flatzinc` [4] which allows to interface different types of solvers including solvers on finite domain `CSPs` such as `Gecode` [2]. The `MiniZinc` model for `UTP` is presented in Table 5 and based on the decisions variables listed in Table 4. The model uses some of the global constraints supported in MiniZinc which are dedicated to scheduling problems.

array[U] of var G:	x_group	group assigned to a student
array[K] of var set of G:	x_groups	set of groups assigned to a class
array[S] of var set of R:	x_rooms	set of rooms allocated to a session
array[S] of var set of L:	x_lecturers	set of lecturers allocated to a session
array[S] of var H:	x_slot	starting slot of a session

Table 4 Decision variables (MiniZinc).

Sectioning constraints partition students into groups and assign each group to a class according to sectioning rules and class size thresholds. Constraint (1) allows students to be part of the same group only if they are registered to the same courses. (2) imposes that every student attends all the part of the courses to which he is registered. (3) ensures that classes from the same part do not have any common group. (4) implements the parent-child relation between classes. Lastly, (5) checks that the groups fit in the class they have been assigned to.

Resource distribution relies on domain, cardinality and sum constraints. Constraints (6) and (7) define available rooms and lecturers for each session. (8) forces the number of rooms allocated to a session according to the specific requirements of the course part (i.e., no room, single-room or multi-room). (9) allocates the required number of lecturers to a session and (10) checks that every lecturer has the right number of sessions in a part.

Session scheduling and resource allocation involves positioning, sequencing, non-overlaping and capacity constraints. Constraint (11) defines the allowed slots for each session. (12) forbids a session to be on two days. (13) sequences the sessions of a class according to their rank. Constraints (14) and (15) model multi-room sessions and the exclusive access to their rooms. (14) makes disjunctive any resource that is allocated to a multi-room session while it is hosting the session. (15) ensures that the number of students attending a multi-room session do not exceed the cumulated capacity of the allocated rooms. (16) models the mandatory rooms to be allocated. (17) models the default cumulative capacity constraint controlling the allocation of non-virtual rooms to single-room sessions. This constraint uses the `cumulative` global constraint of MiniZinc (see [9] for the Gecode implementation) which MiniZinc also reuses to rewrite the global `disjunctive` constraint.

Table 5 also presents some UTP predicates when the targeted resources are rooms. (18) implements the `forbidden_period` predicate that takes the start and end time slots of the period as parameters. (19), (20) and (21) model `same_weekday`, `same_rooms` and `sequenced` predicates, respectively. (22) implements the `no_overlap` predicate that relies on the `disjunctive` global constraint.

forall(u, v in U where u<v) (student_courses[u]!=student_courses[v] -> x_group[u]!=x_group[v])	(1)
forall(u in U, p in student_parts[u]) (exists(k in part_classes[p])(x_group[u] in x_groups[k]))	(2)
forall(p in P, k1, k2 in part_classes[p] where k1<k2) (x_groups[k1] intersect x_groups[k2] = {})	(3)
forall(k1 in K, k2 in class_parents(k1))(x_groups[k1] subset x_groups[k2])	(4)
forall(k in K)(maxsize[k]<=sum(g in G) (bool2int(g in x_groups[k]) * sum(u in U)(bool2int(x_group[u] = g)))	(5)
forall(s in S)(x_rooms[s] subset part_rooms[session_part[s]])	(6)
forall(s in S)(x_lecturers[s] subset part_lecturers[session_part[s]])	(7)
forall(s in S, p in P where p = session_part[s])((part_room_use[p] = none -> x_rooms[s] = {}) /\ (part_room_use[p] = single -> card(x_rooms[s]) = 1) /\ (part_room_use[p] = multiple -> card(x_rooms[s])>=1))	(8)
forall(s in S)(card(x_lecturers[s]) = team[session_part[s]])	(9)
forall(p in P, l in part_lecturers[p]) (sum(s in part_sessions(p))(bool2int(l in x_lecturers[s]) = service[l, p]))	(10)
forall(p in P, s in part_sessions(p)) (week(x_slot[s]) in weeks[p] /\ weekday(x_slot[s]) in weekdays[p] /\ dailyslot(x_slot[s]) in dailyslots[p])	(11)
forall(s in S) ((x_slot[s] − 1) div nr_slots_per_day = (x_slot[s] + length[s] − 1) div nr_slots_per_day)	(12)
forall(k in K, s1, s2 in class_sessions[k] where rank(s1)<rank(s2)) (x_slot[s1] + length[s]>=x_slot[s2])	(13)
forall(p in P, s1 in part_sessions[p], r in part_rooms[p], s2 in room_sessions[r] where is_multi_rooms[p] /\ s1!=s2) (disjunctive([x_slot[s1], x_slot[s2]], [bool2int(r in x_rooms[s1]) * length[s1], bool2int(r in x_rooms[s2]) * length[s2]]))	(14)
forall(p in P, s in part_sessions[p] where is_multi_rooms[p]) (sum(r in part_rooms[p])(bool2int(r in x_rooms[s]) * capacity[r]) <=sum(g in class_groups[session_class[s]])(card(group_students[g])))	(15)
forall(p in P, s in part_sessions[p])(mandatory_rooms[p] subset x_rooms[s])	(16)
forall(r in R where not(virtual[r]))(let {set of S: RS= room_sessions[r] intersect single_room_sessions;} in (cumulative([x_slot[s]|s in RS], [bool2int(r in x_rooms[s]) * length[s]|s in RS], [sum(g in G)(bool2int(g in x_groups[session_class[s]])) * sum(u in U)(bool2int(g = x_group[u]))|s in RS], capacity[r]))	(17)
forbidden_period((r, S′), h1, h2) = forall(i in S′)(r in x_rooms[i] -> (x_slot[i] + length[i]<=h_1 \/x_slot[i]>h_2))	(18)
same_weekday((r, S′)) = forall(i, j in S′ where i<j)((r in x_rooms[i] intersect x_rooms[j]) -> (x_slot[i] div nr_weekly_slots = x_slot[j] div nr_weekly_slots))	(19)
same_rooms((r, S′)) = forall(i, j in S′ where i<j)((r in x_rooms[i] intersect x_rooms[j]) -> x_rooms[i] = x_rooms[j])	(20)
sequenced((r1, S1), (r2, S2)) = forall(i in S1, j in S2)((r1 in x_rooms[i] /\ r2 in x_rooms[j]) -> x_slot[i]+length[i]<=x_slot[j])	(21)
no_overlap((r, S′)) = disjunctive([x_slot[i]|i in S′], [length[i]*bool2int(r in x_rooms[i])|i in S′])	(22)

Table 5 Constraints and predicates of the MiniZinc model.

4.2 CHR model

CHR (for *Constraint Handling Rules*) [18,21,19,20] are a committed-choice language consisting of multiple-heads guarded rules that replace constraints by more simple constraints until they are solved. CHR are a special-purpose language concerned with defining declarative constraints in the sense of *Constraint logic programming* [23,25]. CHR are a language extension that allows to introduce *user-defined* constraints, i.e. first-order predicates, into a given host language as Prolog, Lisp, Java, or C/C++. CHR have been extended to $\text{CHR}^{\vee}$ [5] that introduces the *don't know* nondeterminism in CHR [11]. This nondeterminism is freely offered when the host language is Prolog and allows to specify easily problems from the NP complexity class.

To model and solve UTP instances with the CHR language, we use the CHR++ solver [7] (for Constraint Handling Rules in C++), which is an efficient integration of CHR in the programming language C++.

The full model for CHR++ is too long to be detailed here[4]. We give in Table 7 the list of constraints taken into account by the solver. The decision variables to be instantiated are given in Table 6. They are similar to those of the MiniZinc model, only the end-of-session variables are added.

$\forall s \in S : \text{x\_rooms}[s] \subseteq R$	set of rooms allocated to a session
$\forall s \in S : \text{x\_lecturers}[s] \subseteq L$	set of lecturers allocated to a session
$\forall s \in S : \text{x\_slot\_start}[s] \in H$	starting slot allocated to a session
$\forall s \in S : \text{x\_slot\_end}[s] \in H$	ending slot allocated to a session

Table 6 Decision variables (CHR).

To simplify its implementation, the model is partly non-cumulative and some resources such as lecturers cannot be shared. It also considers that the sectioning and allocation of students to groups is done beforehand. Thus, computing a solution amounts to finding a consistent resource allocation while placing the schedules for all sessions.

Several constraints can be set at the instance analysis stage. This is the case for constraints (1) to (9) of Table 7. Constraints (2), (3) and (4) filter the domains by removing the rooms, lecturers or time slots which are impossible by construction of the instance. Constraint (5) ensures that a session starts and ends on the same day by removing from the domain values that contradict it.

Other constraints are set and managed by rules which monitor modifications to the domains of variables. This is the case for Constraint (1) which ensures the integrity of the start and end of session variables. The same is true for (6) which ensures that the number of lecturers teaching a session is

[4] The interested reader can download the sources of the model [1].

Integrity constraint :	
$\forall s \in S\ :$ x_slot_end$[s] =$ x_slot_start$[s] +$ length(s)	(1)
Static constraints (instance input filtering)) :	
$\forall s \in S\ :$ x_rooms$[s] \subseteq$ part_rooms[session_part(s)]	(2)
$\forall s \in S\ :$ x_lecturers$[s] \subseteq$ part_lecturers[session_part(s)]	(3)
$\forall p \in P, \forall s \in$ part_sessions(p) : $\big($week(x_slot_start$[s]$) $\in$ weeks$[p]\big)$ $\wedge\big($weekday(x_slot_start$[s]$) $\in$ days$[p]\big)$ $\wedge\big($dailyslot(x_slot_start$[s]$) $\in$ dailyslots$[p]\big)$	(4)
$\forall s \in S\ :$ x_slot_start$[s]/nr_slots_per_day =$ x_slot_end$[s]/nr_slots_per_day$	(5)
$\forall s \in S\ :$ card(x_lecturers$[s]$) $=$ team$[session_part[s]]$	(6)
$\forall k \in K, \forall s \in$ class_sessions$[k]$: If $\big($part_room_use[class_part(k)] $=$ none$\big)$ then card(x_rooms$[s]$) $= 0$; If $\big($part_room_use[class_part(k)] $=$ single$\big)$ then card(x_rooms$[s]$) $= 1$; If $\big($part_room_use[class_part(k)] $=$ multiple$\big)$ then card(x_rooms$[s]$) ≥ 1	(7)
$\forall k \in K, \forall s, s' \in$ class_sessions$[k]$, $s.t.$ rank$(s) <$ rank(s') : `before`(s, s')	(8)
$\forall k_1, k_2 \in K, s.t.\ \exists g_1 \in$ class_groups$[k_1], \exists g_2 \in$ class_groups$[k_2]$, avec $g_1 = g_2$: $\forall s_1 \in$ class_sessions$(k_1), s_2 \in$ class_sessions(k_2) : `disjunct`(s_1, s_2)	(9)
Static predicates :	
$forbidden\_period((e, S'), h, h') = \forall i \in S'$: (x_slot_start$[i]$ + length$(i) \leq h) \vee$ (x_slot_start$[i] > h'$)	(10)
$sequenced((e_1, S_1), (e_2, S_2)) = \forall i_1 \in S_1, \forall i_2 \in S_2$: `before`(i_1, i_2)	(11)
$same\_rooms((e, S')) = \forall s_1, s_2 \in S', s.t.\ s_1 < s_2$: x_rooms$[s_1] \sim$ x_rooms$[s_2]$	(12)
Dynamic constraints :	
$\forall p \in P, \forall l \in$ part_lecturers$[p]$: $\big\|\{x \mid x \in$ part_sessions$(p), l \in$ x_lecturers$[x]\}\big\| =$ service$[l, p]$	(13)
$\forall s \in S, \forall r \in$ session_rooms(s) : $\sum\{$group_students$[g] \mid g \in$ session_room_group$(s, r), r \in$ x_rooms$[s]\} \leq$ capacity$[r]$	(14)
$\forall s \in S, s.t.\ has\_mandatory\_room(s)$: session_mandatory$[s] \subseteq$ x_rooms$[s]$	(15)
Dynamic predicate :	
$same\_weekday((e, S')) =$ $\forall s_1, s_2 \in S', s.t.\ s_1 < s_2$: x_slot_start$[s_1]/nr_weekly_slots =$ x_slot_start$[s_2]/nr_weekly_slots$	(16)
Introspective constraints :	
$\forall k_1, k_2 \in K, \forall s_1 \in$ class_sessions$[k_1], \forall s_2 \in$ class_sessions$[k_2]$, s.t. $s_1 \neq s_2$: x_lecturers$[s_1] \cap$ x_lecturers$[s_2] \neq \emptyset \Rightarrow$ `disjunct`(s_1, s_2)	(17)
$\forall k_1, k_2 \in K, \forall s_1 \in$ class_sessions$[k_1], \forall s_2 \in$ class_sessions$[k_2]$ s.t. $s_1 \neq s_2$: x_rooms$[s_1] \cap$ x_rooms$[s_2] \neq \emptyset \Rightarrow$ `disjunct`(s_1, s_2)	(18)

Table 7 Constraints and predicates of the CHR model.

valid and (7) which checks that the number of rooms allocated to a session corresponds to what is required in the instance.

We give as an example the CHR++ rule which checks the integrity of the variables of beginning and end of session. The rule uses a `plus` propagator to ensure consistency of the constraint. This is triggered as soon as a domain of a variable is updated:

```
session_slot(_, S_Start, S_End, S_Length)
  =>> CP::Int::plus(S_Start, (*S_Length)-1, S_End);;
```

We use CHR++ which allows us to manipulate values associated with logical variables and to wake up the corresponding rules as soon as a modification of the value occurs. This mechanism combined with the forward chaining of

CHR allows us to implement an efficient rule wake-up and domain propagation mechanism in the manner of a CSP solver.

Constraints (8) and (9) add new CHR constraints to the model. Indeed, constraints before and disjunct are constraints ensuring the precedence and non-overlapping of two sessions. They are accompanied by rules verifying the coherence of the disjunctive graph created implicitly by the addition of all these constraints. The static predicates correspond to those read from the instance. They are processed and some new constraints (filtering constraints, CHR constraints or unification of variables) are added.

Dynamic constraints ranging from (13) to (18) are only triggered under certain conditions. CHR guarded rules are used for this purpose. (13) checks that a lecturer teaches the expected number of sessions in each course part. (14) ensures that the capacity of the rooms is respected and (15) verifies that the rooms marked as mandatory are indeed found in the solution. Predicate (16) ensures that sessions subject to the same constraint *same_weekday* are set on the same day of the week.

Constraints (17) and (18) add constraints when certain conditions are verified. Thus, (17) adds a disjunct between two sessions when the same lecturer participates. (18) adds a constraint between two sessions if they take place in the same room. These constraints enrich the disjunctive graph representing the sequencing of all the sessions.

It should be noted that the CHR model performs domain filtering but also analyses the disjunctive graph in order to eliminate non-solutions. The edges of the disjunctive graph are oriented as the resolution progresses and the decision variables are instantiated.

4.3 Experimentations

We carried out preliminary experiments on a real-life instance modeling the second semester of the last year of Bachelor in Computer Sciences at Université d'Angers (available at [1]). The main objective was to validate the solvers and assess their ability to generate solutions in a reasonable time.

The instance contains 5 mandatory courses and 2 courses to choose among 4 additional courses. The instance thus consists of 9 courses decomposed into 24 parts, 45 classes and 241 sessions. Courses are taught during 12 weeks, 5 days a week (Monday to Friday), where each day is divided into 1440 slots. At the Faculty of Sciences of Université d'Angers, course sessions last 1h and 20 minutes or 2 hours and start at regular intervals every 90 minutes starting at 8h00 and finishing at 19h50. The 90 minutes interval includes a 10 minutes break allowing students and lecturers to change rooms.

The instance contains 8 rooms, 12 lecturers and 67 students. In our case, student sectioning was performed in advance and prepartitioned the students into 4 groups. Lecturers are either course owners involved in all the parts of a course (lecture, tutorial and lab) or tutors that are involved in labs of different courses. There are 47 rules defined in the instance: 13 weekly, 17 sequenced, 2 `same_slot`, 5 `same_week`, 5 `same_rooms` and 5 `same_lecturers`. The 47 rules were flattened into 216 constraints and the order of 1000 decision variables.

The `MiniZinc` and `CHR` solvers presented in Section 3 were used to solve the instance with an Intel Core i7-10875H 2.30GHz. Both solvers generate a valid solution in less than 5 seconds. The solutions are different due to the two resolution strategies but compliant with each solver which shows the convergence of both models and solvers.

5 Conclusion and Perspectives

We introduced in this paper a domain-specific language for university course timetabling. The language allows to model a wide variety of course timetabling problems such as those encountered in French universities. It provides support for typical timetabling entities (students, sessions, lecturers, rooms, groups) and features (student sectioning, resource distribution, session scheduling, resource allocation) and includes a rules language to easily express constraints (sequencing, periodicity, etc.). Rules allow to target any subset of domain entities and sessions and enforce timetabling-specific predicates.

We used the language to encode a real instance (Bachelor courses of a French university) and implemented a tool chain to convert the `XML` instance files into solver-compatible formats. In order to validate our approach, we implemented a `CSP` model in `MiniZinc` and `CHR` and produced solutions for the considered instance.

We are currently working on different extensions of the language and the back-end solvers. First, we intend to represent preferences and priorities in order to support timetable optimization and repair tasks. Second, the current `CP` models may be improved using dedicated scheduling constraints, search strategies and heuristics and take advantage of model simplication and reformulation techniques. Another objective is to improve scalability by testing our solvers on large-scale instances aggregating different curriculae or converted from academic benchmarks. Lastly, we intend to investigate the revision of timetables to manage unexpected events (e.g. unavailability of a lecturer, late registration of students) or to support incremental solution construction.

References

1. University Service Planning. URL https://ua-usp.github.io/timetabling/
2. Generic Constraint Development Environment (2022). URL https://www.gecode.org/
3. Minizinc (2022). URL https://www.minizinc.org/
4. Specification of Flatzinc. Version 1.6 (2022). URL https://www.minizinc.org/downloads/doc-1.6/flatzinc-spec.pdf
5. Abdennadher, S., Schütz, H.: CHR: A Flexible Query Language. In: Proceedings of the 3^{rd} International Conference on Flexible Query Answering Systems, pp. 1–14 (1998)
6. Aziz, N.L.A., Aizam, N.A.H.: University course timetabling and the requirements: Survey in several universities in the east-coast of Malaysia. p. 040013. Kuala Terengganu, Malaysia (2017). DOI 10.1063/1.4995845. URL http://aip.scitation.org/doi/abs/10.1063/1.4995845

7. Barichard, V., Stéphan, I.: Quantified constraint handling rules. In: ICLP 2019, vol. 306, pp. 210–223. Las Cruces (2019). DOI 10.4204/EPTCS.306.25. URL http://okina.univ-angers.fr/publications/ua20272
8. 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: E. Hebrard, N. Musliu (eds.) Integration of Constraint Programming, Artificial Intelligence, and Operations Research, pp. 69–81. Springer International Publishing, Cham (2020)
9. Beldiceanu, N., Carlsson, M.: A new multi-resource cumulatives constraint with negative heights. In: CP 2002, pp. 63–79 (2002)
10. Bettinelli, A., Cacchiani, V., Roberti, R., toth, P.: An overview of curriculum-based course timetabling. TOP **23**, 313–349 (2015). DOI https://doi.org/10.1007/s11750-015-0366-z
11. Betz, H., Frühwirth, T.: Linear-logic based analysis of constraint handling rules with disjunction. ACM Transactions on Computational Logic **14**(1) (2013)
12. 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). DOI 10.1007/s10479-010-0737-7
13. 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). URL http://www.jstor.org/stable/3010580
14. 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). DOI https://doi.org/10.1016/j.eswa.2021.115967. URL https://www.sciencedirect.com/science/article/pii/S095741742101318X
15. Castro, C., Manzano, S.: Variable and Value Ordering When Solving Balanced Academic Curriculum Problems. ARXIV (2001)
16. 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 (2021). DOI 10.1109/ACCESS.2021.3100613
17. Chiarandini, M., Di Gaspero, L., Gualandi, S., Schaerf, A.: The balanced academic curriculum problem revisited. Journal of Heuristics **18**(1), 119–148 (2012). DOI 10.1007/s10732-011-9158-2. URL https://doi.org/10.1007/s10732-011-9158-2
18. Frühwirth, T.: Constraint Handling Rules. In: Constraint Programming: Basics and Trends, pp. 90–107 (1994)
19. Frühwirth, T.: Constraint Handling Rules. Cambridge University Press (2009)
20. Frühwirth, T., Raiser, F. (eds.): Constraint Handling Rules: Compilation, Execution, and Analysis. Cambridge University Press (2011)
21. Frühwirth, T.: Theory and practice of constraint handling rules. Journal of Logic Programming **37**(1-3), 95–138 (1998)
22. 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). DOI https://doi.org/10.1016/j.ejor.2017.01.040. URL https://www.sciencedirect.com/science/article/pii/S0377221717300759
23. Hentenryck, P.V.: Constraint logic programming. Knowledge Engineering Review **6**(3), 151–194 (1991)
24. ITC19: International Timetabling Competition (2019). URL https://www.itc2019.org/
25. Jaffar, J., Maher, M.: Constraint logic programming: A survey. Journal of Logic Programming **19/20**, 503–581 (1994)
26. Lemos, A., Monteiro, P., Lynce, I.: Disruptions in timetables: A case study at universidade de lisboa. Journal of Scheduling (2021). DOI 10.1007/s10951-020-00666-3
27. 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 (2007)
28. Lindahl, M., Stidsen, T., Sørensen, M.: Quality recovering of university timetables. European Journal of Operational Research **276**(2), 422 – 435 (2019). DOI https://doi.org/10.1016/j.ejor.2019.01.026. URL http://www.sciencedirect.com/science/article/pii/S0377221719300451

29. 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 (2010). DOI 10.1287/ijoc.1090.0320
30. Müller, T., Rudová, H., Müllerová, Z.: University course timetabling and International Timetabling Competition 2019. In: E.K. Burke, L. Di Gaspero, B. McCollum, N. Musliu, E. Özcan (eds.) Proceedings of the 12th International Conference on the Practice and Theory of Automated Timetabling (PATAT-2018), pp. 5–31 (2018)
31. Müller, T., Murray, K.: Comprehensive approach to student sectioning. Annals of Operations Research **181**, 249–269 (2010). DOI 10.1007/s10479-010-0735-9
32. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: Minizinc: Towards a standard cp modelling language. In: C. Bessière (ed.) Principles and Practice of Constraint Programming – CP 2007, pp. 529–543. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
33. 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 (2013). DOI 10.1155/2013/793671. URL https://www.hindawi.com/journals/mpe/2013/793671/. Publisher: Hindawi
34. Schindl, D.: Optimal student sectioning on mandatory courses with various sections numbers. Annals of Operations Research **275** (2019). DOI 10.1007/s10479-017-2621-1
35. Vrielink, R.A.O., Jansen, E.A., Hans, E.W., van Hillegersberg, J.: Practices in timetabling in higher education institutions: a systematic review. Ann. Oper. Res. **275**(1), 145–160 (2019). DOI 10.1007/s10479-017-2688-8

Three-phase Curriculum-Based University Course Timetabling with Student Assignment

Elmar Steiner[1], Ulrich Pferschy[1], and Andrea Schaerf[2]

[1] Department of Operations and Information Systems, University of Graz, Austria
{elmar.steiner, ulrich.pferschy}@uni-graz.at
[2] Dipartimento Politecnico di Ingegneria e Architettura, University of Udine, Italy
andrea.schaerf@uniud.it

Abstract. We consider a complex university timetabling problem arising in a four-year study program of teacher education where every student has to choose two subjects. Since any combination of two subjects is feasible, the goal of designing a collision-free timetable for every student seems to be unreachable. However, the task becomes more tractable because for most courses several parallel groups are offered, i.e. sectioning of students is possible. Further difficulties arise from the highly individual progress of students who often follow neither the prescribed term of each course nor the prescribed ordering of courses. Under these and other conditions an optimized timetable should be determined and adjusted to the estimated student numbers and their past achievements. After moving main lectures into a regular time grid with minimal changes concerning the previously existing plan, the task of finding a timetable for all lectures with parallel groups is modeled as an integer linear program (ILP). Later, students with their actual demands are allocated a non-overlapping set of courses that is relevant and feasible for their individual study situation. This part can be handled by an assignment-type model followed by a round-robin allocation of remaining capacities.

Keywords: Course Timetabling · Student Sectioning · ILP Model

1 Introduction

A timetabling problem generally consists of assigning a set of activities to resources such that a set of complex constraints is fulfilled, which varies depending on the given problem. Whereas these constraints are usually considered to be *hard*, desirable characteristics of the timetable are introduced as *soft constraints* into the objective function. The goal is to find a feasible assignment while minimizing the weighted sum of the penalties representing these violations.
There is a wide range of real-world applications, including *university timetabling*, where different categories of specific problems are distinguished: *Examination timetabling* (ETT), *post-enrollment course timetabling* (PE-CTT) and *curriculum-based course timetabling* (CB-CTT). Both of the latter two deal with the assignment of courses to time periods and usually rooms, however, there are certain

differences. In PE-CTT the timetable is established after the enrolment of students, thus taking into account that students are enrolled in various combinations of events and somehow incorporating these selections. CB-CTT determines a timetable based on a curriculum of study programs, that is, a list of courses to be taken by a group of students. A recent survey on all the available formulations of educational timetabling is provided by Ceschia et al. [4].
These timetabling problems form only one side of the issues that operations research has to address in education, see Johnes [8] for an overview. Real-world educational planning scenarios often simultaneously comprise components of various problems, depending on the stage of planning and area of application (such as elementary or tertiary education). Therefore the correct choice of methods depends on the planning characteristics (such as information availability or choice of planning entities) and combined approaches seem appropriate.
One of the additional issues is the group of *student sectioning problems* (see e.g. [12]), where students are assigned to particular sections of a course satisfying constraints such as room or section capacity and avoiding conflicts in students' timetables due to overlaps. Quite often this is considered a *sub-problem* of course timetabling. That is, after deriving an adequate timetable, one seeks an optimal assignment of students to classes avoiding conflicts and taking into account students' needs/requests and other soft constraints such as preferences or daily workload.
The timetabling problem we are dealing with is a complex scenario involving several non-standard properties. Its main decision problem can be categorized as a variant of a sectioning problem.

After giving a general description of the problem in Section 2 we point to some related literature in Section 3. The mathematical models introduced for solving our timetabling problem will be presented in Section 4. Our approach consists of three phases: Phase 1 shifts important "main lectures" from their historical starting times into the regular time grid which serves as a basis of all our plans. In Phase 2 a complex ILP model is set up which determines in one optimization step the time periods of all courses (many of them consisting of several parallel groups) and also assigns individual sets of relevant courses in collision-free time periods to groups of students with identical properties. Since this step has to be carried out many months before the start of an academic year, these groups of students are only estimations of future student demand. The final course assignment of students is done at a later time in Phase 3 by matching actual students to estimated groups. Both Phases 1 and 3 employ generalizations of the linear assignment problem with additional conflict constraints. Computational experiments in Section 5 illustrate the potential and limitations of the large ILP model.

2 Problem description

We were asked by the central administration of the University of Graz, Austria, to develop an automated solution approach for a complex timetabling task arising in the teacher education study program which involves roughly 4,200 students. From an educational planning perspective, it can be considered a *multi-phase scheduling problem*. More precisely it consists primarily of a *timetabling* step producing as output the day of the week and starting time of every course (which stays constant over the whole term). This step consists of two phases: In Phase 1, starting times of courses with a larger audience (and no parallel groups) are moved from their historical starting times into the regular time grid prescribed by the university. Since these moves cause major disturbances and negative side effects, their total time deviation will be minimized under non-collision constraints. In the second phase the starting times for all courses with parallel groups are computed from scratch (also obeying the regular time grid). To assure feasible timetables for individual students this phase is coupled with *the sectioning* of *projected students*. Later, in the third phase, the planning problem asks for an *assignment* of pre-registered actual students which assigns to each student a set of courses that are feasible, relevant, and non-overlapping for the respective student.

The reason for the coupling of methods as well as separation into distinct phases is the structure of the given planning process at our university. This procedure essentially covers the capacity planning of the number of sections and the establishment of yearly timetables for the teacher education study program. The study program of teacher education requires the choice of *two subjects* (such as English and chemistry), thus the planning entails the coordination of all involved departments and those of their provided courses, which are part of the program's curriculum.

While the definition of the number of sections is based on enrollment predictions and the establishment of the timetable (first and second phase) needs to be carried out in March for the two terms of the academic year to come (starting in October), the actual enrollments (required for the third phase) only come to be known in September (for the winter term) resp. February (for the summer term), when an adaption and especially a re-scheduling of courses is not permitted anymore. So there is a crucial *temporal* delay between planning and information receipt. Moreover, since departments currently schedule courses autonomously and only distinctive overlapping time conflicts (e.g. of prominent courses) are resolved bilaterally, the majority of students of any combination of subjects will face time conflicts in their weekly timetable. Nevertheless, the final schedule has been ascertained to be of major influence in regards to the study conditions and therefore students' performance in completing their studies.

To improve the situation of studying, we seek to support both planning processes by deriving models to optimize the weekly timetable using the respective information given during phases, resulting - opposed to conventional approaches - in a *combination* of sectioning and timetabling. While the true target is being free of conflicts, we also seek a compact timetable for the individual student and didactic practicability. The target groups for facilitation are especially those students who exhibit non-standard study progresses, such that courses are not completed in the term that is recommended in the curriculum. For that reason and as opposed to many curriculum-based models the method needs to comprise the capability to avoid time conflicts between courses that are not nominally taken in the same year. The resulting term-overlapping constraints are determined from given historical data that describes to which extent courses are completed earlier or (more likely) later than recommended. Courses that are prone to be taken later than in the prescribed term will be called *displaced courses*.

The input structure of the problem primarily consists of a set of courses. These consist on one hand of so-called *main courses* (lectures) C^m, which usually cater to a larger audience and do not have parallel groups. However, many of them have a historically established starting time, possibly outside the regular time grid. Then there are standard *courses* (exercise classes, seminars, etc.) C, where each course $c \in C$ has a limited capacity $cap(c)$ and therefore a certain number of *sections* (parallel groups) are offered. As stated the number of these groups is determined by a separate planning process (involving also financial considerations) and is provided as an input to the timetabling task. For each course resp. parallel-group a lecturer is given. Equivalently, for each lecturer $l \in L$, its set of courses $C(l)$ is known. Following the most common teaching mode, we assume that each course or group is given by exactly one lecturer, although team teaching or shared courses may well occur in practice. However, our model could be easily extended to accommodate more than one lecturer per course.

Each course is part of the curriculum of exactly one subject and is prescribed for exactly one term. Besides that, there are also some general main courses which are part of every curriculum (any combination of subjects) and have to be taken by all students.

Note that – different from many existing timetabling applications – rooms are not considered in our planning task. This is because rooms are shared with the programs of the other ca. 30,000 students of the university. Therefore, an automated allocation of *rooms* would have far-reaching consequences for the decentral planning process of the whole university. However, rooms currently do not pose a major problem to the planers because the majority of courses have either very specific requirements with regards to rooms (such as chemistry labs) or none at all. While the former use a room that is tightly coupled to the department and usually shared in a limited and well-practiced manner, the former can use any room on the campus.

3 Related work

In the literature, there exist several strategies of section management, depending on the concrete problem where they are applied. The problem has been tackled either as a separate problem or integrated into the timetabling procedure. Aubin and Ferland [1] for example iteratively adjusted both timetable and section assignment given an initial timetable. Banks et al. [3] propose a rather simultaneous approach, where they assign sections of courses to time periods

and iteratively add constraints, each representing a student course selection, to satisfy as many choices as possible.
A very comprehensive approach is the one by Müller and Murray [12], who propose a multi-phase sectioning strategy, considering various information stages in educational planning and using different (heuristic) algorithms for each phase. In detail, they identify three different approaches to sectioning that they call *Initial sectioning*, *Batch sectioning*, and *Online sectioning*, which differ based on the time when it takes place and the information available at that time. The same approach has been further extended and specifically applied to a Faculty of Education by Müller and Rudová [13].
An integrated timetabling and sectioning problem has been recently proposed for the fourth International Timetabling Competition (ITC-2019) [14]. The ITC-2019 problem consists of sectioning students into classes based on course enrollments and then assigning classes to available periods and rooms. Courses may have a complex structure of classes, with one or more configurations, further divided into subparts and the parent-child relationship between classes. The other remarkable feature is that the timetable may differ from week to week, instead of replicating the same weekly timetable for the whole semester.
It is worth mentioning that for such a complex timetabling/sectioning problem as the ITC-2019 one, the solution techniques based on MIP solvers turned out to be very competitive. Indeed, the MIP formulation by Holm et al. [7] won the competition and produced the best solution for the majority of the instances. Most of the remaining best solutions have been obtained by a local search approach, which did not enter the competition as it was proposed by one of the organizers (i.e., Müller [11]).
Other successful applications of MIP models to timetabling problems are the works by Lach and Lübbecke [10] and by Bagger et al. [2], that worked on the CB-CTT problem obtaining both good solutions and tight lower bounds. Another complex, real-world sectioning problem has been proposed by Esmaeilbeigi et al. [6] for a military school. In their problem, a lesson has a multiphase structure, such that each phase may require different resources and is taken by different students.
Finally, complexity analysis of the student sectioning problem has been carried out by Dostert et al. [5] and Schindt [15], identifying the cases in which the problem is polynomial and those in which the problem is NP-hard.

4 The Mathematical Models

As pointed out in Section 2 our planning problem consists of three separate phases. In the following, we describe our solution approach for each of them. The aim of Phases 1 and 2 is a complete timetable for all courses.
Each course $a \in C^m \cup C$ belongs to exactly one subject $f(a)$ and is prescribed for a certain term $n(a)$ with $n(a) \in \{1, 2, \ldots, 8\}$, corresponding to winter and summer semesters of a four year program. As an exception, there is a small subset of main courses which is prescribed for all subjects (educational theory, etc.).

Furthermore, each regular course $a \in C$ comprises $g(a) \geq 1$ sections (parallel groups). Thus, the timetable consists of a starting time for each main course $a \in C^m$ and for each section of every course $a \in C$.

4.1 Phase 1: Alignment of main lectures

The feasible time periods of the timetable consist of a day of the week and a starting time. The latter is set by university rules to a fixed time grid starting at 8:15 and continuing with a sequence of 90 minutes of lecture time and 15 minutes breaks. This allows for seven time slots per day, i.e. a set P consisting of 35 time periods per week as shown below. We only assign lectures of 90 minutes and exclude from consideration the small number of lectures with deviating duration.

start	Mon	Tue	Wed	Thu	Fri
08:15					
10:00					
11:45					
13:30					
15:15					
17:00					
18:45					

In Phase 1 the *main lectures* C^m (which do not have parallel sections) will be aligned to the given time grid. A majority of main lectures already follow this prescribed time frame, but a non-negligible minority deviates from the time grid. Since the main lectures in general hardly change their time and room over the years and some of them are also part of other curricula outside our planning task, it makes sense to change their starting times as little as possible. Therefore, we set their starting times by solving a version of the linear assignment problem with additional conflict constraints. Thereby we match main lectures to time periods $p \in P$ of the given time grid with the additional restrictions that main courses belonging to the same term $t \in T$ must not overlap. This non-collision condition is imposed independently from the subject since it should be possible to study any combination of subjects without overlaps in the main courses as there exists no alternative for them. The special main courses which are prescribed for all subjects cannot overlap with any other main course of the same term. Function $C^m(t)$ returns the set of courses that belong to term $t \in T$. Additionally, lecturers $l \in L$ cannot be assigned to more than one course at the same time, both in the winter and the summer semester. Function $C^m(l)$ then returns the set of courses that are given by lecturer l. The single binary decision variable $y_{cp} = 1$ if main course c is assigned to time period p, and 0 otherwise.
As a linear objective function, we consider the *distance* $\Delta(c,p)$ between the current time slot of course c (i.e. as in the previous year) and the new time period $p \in P$. If both times are on the same day, $\Delta(c,p)$ describes the absolute difference in minutes between the current starting time and the beginning of period p. Otherwise, i.e. if the course is moved to a different day, we assume a penalty

value ρ equal to three times the maximum intra-day distance (independently from the new day). The resulting assignment-type integer linear program is as follows:

$$\min_{y} \quad \sum_{c\in C^m}\sum_{p\in P} y_{cp}\cdot \Delta(c,p) \tag{1a}$$

$$\text{s.t.} \quad \sum_{p\in P} y_{cp} = 1, \quad \forall c \in C^m, \tag{1b}$$

$$\sum_{c\in C^m(t)} y_{cp} \leq 1, \quad \forall p \in P,\ t \in T, \tag{1c}$$

$$\sum_{c\in C^m(l)} y_{cp} \leq 1, \quad \forall p \in P,\ l \in L, \tag{1d}$$

$$y_{cp} \in \{0,1\} \tag{1e}$$

The results are listed in Table 1, together with the computation time. It turned out that in the winter and summer semester 69% and 47.4%, respectively, of all main lectures had to be adjusted. The assignment of the remaining courses in C^m already followed the time grid and was not changed. Notably, the solutions do not comprise any alignment to another working day. Furthermore, the amount of rescheduled lectures varies substantially for different terms, such that earlier terms show more displacements.

Table 1: Results of main lecture adjustments.

semester	$\lvert C^m\rvert$	adjusted courses	av. adj.	max. adj.
winter	71	49	1h48m	7h
summer	38	18	2h51m	7h

It should be noted that Phase 1 is relevant mostly for the introductory year of the new planning tool, or when additional subjects are integrated into the planning process. Once all main courses are aligned with the time grid, Phase 1 will be used only for assigning new main courses and for handling exceptions such as enforced changes.

4.2 Phase 2: Timetabling-Sectioning

Phase 2 is the most complex part. It considers the computation of starting times for all freely assignable courses, many of them being offered with parallel groups, which necessitates the sectioning of the estimated student cohorts.
The demand structure of the planning task is captured by sets of students each of them enrolled in two subjects $\{f_1, f_2\}$ (of equal importance), where each f_i is chosen arbitrarily from a set F of 28 offered subjects. In our Central European

setting the progress of a student does not follow a strict yearly pattern, e.g. a student cannot be immediately identified as a third-year student, which is quite different from many international university systems. To place a student s in a certain term, we count the sum $ECTS_s$ of ECTS credits reached and assign the student to term $t_s = \lceil \frac{ECTS_s}{30} \rceil$, since 30 ECTS are the usual workload assigned for one term. Our experience tells us that most students proceed equally fast in their two subjects. Thus, we do not distinguish the progress in the two subjects. To scale the optimization problem of Phase 2 to a more tractable size, we combine sets of $\varDelta$ (e.g. $\varDelta = 5$) identical students, i.e. students with the same pair of subjects and in the same assumed term, to a so-called *student quantum*. The set of all quanta is denoted by I. Each quantum $i \in I$ is placed in the same term t_i as the corresponding students. Although the $\varDelta$ students represented by a single quantum may well differ in the precise set of courses they have passed already, this simplification should be acceptable because the students will have another one or two terms to increase their credits before the next assignment phase. Thus, even students with a currently identical track record may well differ in their state at the beginning of the next term. For this reason, it is also pointless to include a full precedence check in the selection of courses for a quantum.

In the following, we describe the generation of the courses C_i assigned to each quantum $i \in I$. All courses prescribed for term t will be denoted as *regular* courses $C_r(t)$. As stated, some of these regular courses shall be assigned to conflict-free time slots with courses of previous terms - so-called *displaced courses*. As described further below we will rate courses as displaced according to historical exam data. Depending on the proportion of students taking such a course $c_r \in C_r(t)$ late, the sections of c_r will be split in two parts: One part remains in c_r to be done in term t. The remaining sections comprise a newly generated displaced course $c_d \in C_d(t+2)$ to be assigned with delay for term $t+2$. The quantum capacity of a regular or displaced course c denoted by q_c is given by the number of sections times the capacity of a section (parallel-group) $cap(c)$ scaled by the quantum size $\varDelta$.
To connect the course supply with demand, every quantum $i \in I$ with term t_i is assigned a set of courses C_i taken from the *relevant* courses $\bar{C}_i \subset (C_r(t_i) \cup C_d(t_i))$ that are required to be completed in the upcoming term. This set $\bar{C}_i$ consists of all courses, which - given quantum i assumed term t_i and combination of subjects - need to be completed according to the curriculum.
The generation of the quanta's course sets is described in Algorithm 1. Starting with the highest term t (i.e. $t = 8$) for each student quantum i at first and as long as the quantum capacity of the course is not met, all *displaced courses* $c_d \in C_d(t)$ are assigned to C_i. Secondly, regular courses $c_r \in C_r(t)$ of term t are assigned, however only if C_i does not contain any displaced predecessor course of c_r. In both cases, the algorithm stops as soon as at least 20 ECTS are reached. Note that the prescribed workload for a student in a term amounts to 30 ECTS. The chosen discrepancy serves as slack for the matching of real enrolled students in the second phase when a difference between quanta's course lists and the actual requirements of real students seems inevitable. This also helps to reach

Algorithm 1 Generation of courses C_i for quantum $i \in I$

```
1: $n$ terms
2: displaced courses $C_d(t)$, regular courses $C_r(t)$, relevant courses $\bar{C}_i$, $C_i = \{\} \quad \forall i \in I$,
3: course capacities $q_c$, quantum size $\Delta$
4: term $t = n$
5: while $t \geq 1$ do
6:     for $i \in I$ do
7:         for $c_d \in C_d(t) \cap \bar{C}_i$ do
8:             if $q_c \geq \Delta$ then
9:                 $C_i \leftarrow C_i \cup \{c_d\}$
10:                $q_c \leftarrow q_c - \Delta$
11:                if $ECTS(C_i) \geq 20$ then
12:                    break
13:                end if
14:            end if
15:        end for
16:        for $c_r \in C_r(t) \cap \bar{C}_i$ do
17:            if $ECTS(C_i) \geq 20$ then
18:                break
19:            end if
20:            if $C_i$ does not contain precedence of $c_r$ then
21:                $C_i \leftarrow C_i \cup \{c_r\}$
22:            end if
23:        end for
24:    end for
25:    $t \leftarrow t - 1$
26: end while
```

feasibility. Moreover, the university administration would like to keep some degree of freedom for the students to select additional courses on their own thus making it easier to accept a centralized course assignment regime.

From a different angle, this slack also reflects the special situation of main courses C^m which are not considered in C_i. These main courses are seen as crucial parts of each subject and therefore should be available for every student without collisions in the respective term. Thus, we will not consider their ECTS in the workload of the current term.

As mentioned above, the definition of the number of displaced sections is based on historical examination data. The examination data provides the set of students $S(a)$ that have completed this course and for each $s \in S(a)$ we are given the term $p(a, s)$ in which student s has passed the course a. Note that under the flexible rules of our university $p(a, s)$ may well differ from the prescribed term $n(a)$.

Based on this data we compute a *lateness value* $L(a, s)$ which represents the delay of student s in passing course a relative to other courses. Therefore, we count the courses (weighted by their ECTS) prescribed for later terms which s has taken in the same term or earlier than a, and the course prescribed for the same term as a but taken in an earlier term than a. The total number of these "preponed" courses serves as a lateness value $L(a, s)$ and is compared to a predefined threshold T to label course a as *passed late* by student s. Formally, we have:

$$\begin{aligned} L1(a,s) &:= \sum_{b \in C} ECTS(b) \text{ with } n(b) > n(a) \text{ and } p(b,s) \leq p(a,s) \\ L2(a,s) &:= \sum_{b \in C} ECTS(b) \text{ with } n(b) = n(a) \text{ and } p(b,s) < p(a,s) \\ L(a,s) &:= L1(a,s) + L2(a,s) \end{aligned}$$

Note that this definition also yields meaningful values for "slowly progressing" students which pass *all* their courses later than prescribed. If $L(a, s) > T$ then a is passed late by s. Computing the lateness over all students $s \in S(a)$, we determine the delay factor of course a as

$$del(a) = \frac{|\{s \in S(a) \mid L(a,s) > T\}|}{|S(a)|}.$$

To align the timetable with the actual progress of students we split the $g(a)$ sections of the regular course a as follows. Defining $g_1(a) := \lfloor g(a) \cdot del(a) \rfloor$, we introduce a new "auxiliary" course a' with $g_1(a)$ sections and prescribed for the successive year, i.e. for term $n(a') := n(a) + 2$. The original course a remains at term $n(a)$ but its sections are reduced to $g(a) := g(a) - g_1(a)$. In this way, some course sections are offered in line with the study schedule of slower progressing students.

The ILP-Model The optimization step in essence seeks to assign *sections* $g \in G(c)$ of all courses $c \in C$ to time periods $p \in P$ of a recurring working week. As is the case for many timetabling problems, the main goal of the planning task is reaching a feasible solution, while the actual objective function is of secondary importance. In our planning problem, the university administration did not specify a particular goal or quality criterion for the timetable. However, our discussions with student representatives and teachers exhibited clear preferences not dissimilar from goals observed in classical university timetabling tasks. Our objective function consists of two parts: The first part aims at avoiding pairs of lectures with long breaks in between for a student quantum. Considering travel times and missing facilities for spending free time this represents the desire of having courses in a single time block. The second part takes into account pedagogical as well as group dynamic aspects. It considers each session of a course and aims at minimizing the number of different secondary subjects followed by the student quanta of this session. Indeed, it would often be preferred to have a more homogeneous student body in a lecture, possibly all enrolled in the same or only two different other subjects (besides the subject of the course).
The main decision variable $y_{cgp} = 1$ if section g of course c is assigned to $p \in P$, and 0 otherwise. Likewise a *student quantum* $i \in I$ is assigned to a section g

of one of it's compulsory courses $c \in C_i$ if variable $x_{icg} = 1$, analogously for $c \in C^m$ with $G(c) = 1$. In addition, we will introduce auxiliary variables v_{ip} for every quantum $i \in I$ and time period $p \in P$ expressing continuity of assigned time slots for quantum i, and d_{cgf} to measure the heterogeneity of the quanta assigned to a section g of course $c \in C$.

The *model's input* technically comprises:

- a set of student quanta I and quantum size Δ (students per quantum),
- overall available time periods P,
- first and last time periods of a day $FP,\ LP \subset P$,
- courses $c \in C$ with capacities $cap(c)$ and number of parallel sections $G(c)$,
- main courses C^m and for every $c_m \in C^m$ the time period $p(c_m)$
- for each quantum $i \in I$ the two subjects $f_1(i)$, $f_2(i)$, and the required courses $C_i \subseteq C$ and $C^m(i) \subseteq C^m$ [3]
- set of lecturers L and the courses $C(l) \subset C \cup C^m$ taught by each lecturer $l \in L$
- subjects F, subject $q(c) \in F$ of course c
- threshold Π limiting the number of sections taking place at the same period

The model is defined as follows:

$$\min_{d,\,v} \quad \alpha \cdot \sum_{p\in P}\sum_{i\in I} v_{ip} + \beta \cdot \sum_{f\in F}\sum_{c\in C}\sum_{g=1}^{G(c)} d_{cgf} \tag{2a}$$

$$\text{s.t.} \quad \sum_{g=1}^{G(c)} x_{icg} = 1, \quad \forall i \in I, c \in C_i \cup C^m(i), \tag{2b}$$

$$\sum_{p\in P} y_{cgp} \leq 1, \quad \forall c \in C \cup C^m,\ g \in \{1,\ldots,G(c)\}, \tag{2c}$$

$$y_{c_m 1 p(c_m)} = 1, \quad \forall c_m \in C^m, \tag{2d}$$

$$\sum_{c\in C(l)}\sum_{g=1}^{G(c)} y_{cgp} \leq 1, \quad \forall l \in L,\ p \in P, \tag{2e}$$

$$\sum_{i\in I} x_{icg} \cdot \Delta \leq cap(c),\ \forall c \in C,\ g \in \{1,\ldots,G(c)\}, \tag{2f}$$

$$x_{ic'g_1} + x_{ic''g_2} + y_{c'g_1p} + y_{c''g_2p} \leq 3 \quad \forall i \in I,\ c' \neq c'' \in C_i \cup C^m(i)\ , \quad p \in P,\ g_1 \in \{1,\ldots,G(c')\},\ g_2 \in \{1,\ldots,G(c'')\} \tag{2g}$$

$$\sum_{g=1}^{G(c)}\ \sum_{c\in C_i\cup C^m(i)} \left(y_{cgp} - y_{cg(p-1)} - y_{cg(p+1)}\right) \leq v_{ip} \quad \forall i \in I\ , \quad p \in P - \{FP \cup LP\} \tag{2h}$$

[3] More precisely, $C^m(i)$ contains the main lectures that belong to the term that student quantum i is assumed to be enrolled in according to her/his accomplished ECTS (for the two subjects chosen by i).

$$\sum_{g=1}^{G(c)} \sum_{c \in C_i \cup C^m(i)} (y_{cgp} - y_{cg(p+1)}) \leq v_{ip} \quad \forall i \in I, p \in FP, \tag{2i}$$

$$\sum_{g=1}^{G(c)} \sum_{c \in C_i \cup C^m(i)} (y_{cgp} - y_{cg(p-1)}) \leq v_{ip} \quad \forall i \in I, p \in LP, \tag{2j}$$

$$\sum_{g=1}^{G(c)} y_{cgp} \leq \Pi \quad \forall p \in P,\ c \in C \cup C^m, \tag{2k}$$

$$\sum_{\substack{\{i \in I: \\ f_1(i)=f \vee f_2(i)=f\}}} x_{icg} \leq d_{cgf} \left\lceil \frac{cap(c)}{\Delta} \right\rceil \ \forall c \in C, g \in \{1, \dots, G(c)\}, \ f \in F \setminus \{q(c)\} \tag{2l}$$

$$x_{icg}, y_{cgp} \in \{0,1\}, \tag{2m}$$

$$v_{ip}, d_{cgf} \in \mathbb{N} \tag{2n}$$

As described above the objective 2a is twofold: (A) minimizing individual timetable compactness via minimizing auxiliary variable v_{ip}, which counts the number of free periods in-between assigned lectures for each student quantum. (B) The auxiliary variable d_{cgf} represents the overall number of second subjects (curricula) in a section of a compulsory course c. Minimizing this variable results in a higher homogeneity of student quanta per section and is hoped to be didactically advantageous, since course contents can be brought into line with the second subject. The coefficients α and β allow a linear combination of the two parts and should be chosen in collaboration with the decision-makers. Constraint 2b ensures that student quantum i is assigned exactly once to a section of a mandatory course. The set of required courses for a student i, however, comprises some of the main courses C^m - that were assigned to periods in Phase 1 - and some smaller ones, such as labs or seminars C - therefore $C_i \cup C^m(i)$. Via constraint 2c sections can take place at most once (if the planning of the number of sections is fairly reliably, 2c can be written with equality). However, some slots are already taken by the main courses (with one section, $g = 1$) as assigned in the preceding Phase 1 (2d). Constraint 2e avoids that a section g of course c in the set of all courses $C(l)$ that a teacher l is giving is assigned to the same period. Given the quantum size Δ, constraint 2f ensures that the capacities of the regular courses C are not exceeded. Constraint 2g essentially avoids collisions: Whenever a student quantum i is assigned both to a section 1 and 2, these cannot take place in the same time period. If they do, the student cannot be assigned to both of them.
Constraint 2h is used to activate the auxiliary variable v_{ip}: For all time periods, except those at the beginning and the end of each working day (sets FP and LP), it is verified whether the preceding and following time slot is also taken by a section that student quantum i has to follow. If not, then there exists an isolated lecture for student i at time period p and v_{ip} is set to one. Constraints 2i and 2j account for isolated lectures at the end and the beginning of the day. Constraint 2k bounds by a *threshold* Π the number of sections that may take place at the same time period p. Although the collision avoidance can be expected to

imply a certain spread of sections over the weekly time grid, it appears necessary to impose an explicit threshold, since rooms are not explicitly considered in the model. The more evenly sections are distributed over the week, the easier it will be to find rooms, whose number of naturally restricted. Constraint 2l finally adjusts the auxiliary variable d_{cgf}, which is used in the objective function to reduce the number of different second subjects of quanta in the same section. For all sections, the enrolled subjects of assigned students are compared to all curricula (except the one the course belongs to) and counted.

4.3 Phase 3: Student assignment

Phase 3 takes place several months after Phases 1 and 2, shortly before the beginning of a new term. In this phase actual, enrolled students $s \in S$ with their updated records of courses passed are matched to *quanta* i (multiples of students) resulting from Phase 2. Although a student's s required or relevant courses C_s depend on her/his study record and curriculum-related prerequisites - that is, completion of specific courses to enroll in others - the course lists of quanta C_i are estimated based on ECTS of students one or two terms in the past. Consequently, discrepancies between the required courses of a student and the course list of the quantum the student is assigned to will be inevitable.

$$\max \quad \sum_{s\in S}\sum_{i\in I} w_{si}x_{si} \tag{3a}$$

$$\text{s.t.} \quad \sum_{i\in I} x_{si} = 1, \quad \forall s \in S, \tag{3b}$$

$$\sum_{s\in S} x_{si} \leq \Delta, \quad \forall i \in I, \tag{3c}$$

$$x_{si} \geq 0 \tag{3d}$$

As formulated above we seek a *maximum weight perfect matching* on a complete bipartite graph by solving a variant of the classical linear *assignment problem.* Associated to each 'assignment' is a variable x_{si} such that $x_{si} = 1$ iff student s is assigned to quantum i, and 0 otherwise. The weight w_{si} in the objective function 3a represents the *degree of fit* and is defined as the cardinality of the intersection of the quanta's and the actual students' course set, $w_{si} := |C_s \cap C_i|$. Naturally, students will be only allocated the courses in $C_s \cap C_i$. Constraint 3b ensures that each student s is assigned to exactly one quantum i. Via constraint 3c at most Δ students can be assigned to one quantum. It is well known that the above mathematical program can be solved as a linear program and the integrality of x_{si} is given by default.

For ease of computation, we will apply the optimization model repeatedly on smaller parts of the data, since only matching of students and quanta that belong to the *same subject combination* appears purposeful. Therefore input data $S_{\{f_1,f_2\}}$ and $I_{\{f_1,f_2\}}$ is divided accordingly as well as corresponding weight matrices $W_{\{f_1,f_2\}}$.

Allocating residual course places Recalling that the courses C_i assigned to a quantum i are usually just above 20 ECTS and that possibly not all these courses are relevant for a student s matched to quantum i, it can be expected that a sizable number of course places remain free after the above assignment phase. These places will be allocated by a *Round Robin* procedure to any students left with less than 20 ECTS assigned courses.
Students are sorted as follows: To facilitate timely graduation, at first, all students requiring at most 30 ECTS for completing their program (not counting the courses allocated in the assignment phase) are selected and sorted in increasing order of missing ECTS. All other students are appended to this sequence and sorted in increasing order of ECTS received in the above assignment phase, which reflects a max-min fairness criterion.
Considering students in this sequence, we take the first student and assign her/him a section of a course that is not fully booked, is feasible for the student w.r.t. the study program, and that does not overlap with any previously allocated courses. Among these, a course is randomly selected from those which are prescribed for the earliest term. If no allocation is possible, the student is removed from the sequence, otherwise, the student is reinserted according to the sorting criteria. One can also choose to remove students (except those close to graduation) from the sequence once their workload exceeds 20 ECTS or another bound set by the university administration.

5 Computational Insights

5.1 Data from Graz University

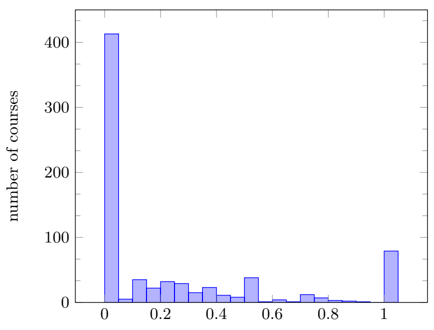
We applied our model to a subset of the courses at the University of Graz, specifically those which form the curriculum of the teacher education study program. As stated, the curriculum requires the choice of two out of 28 possible different subjects (e.g. English and German). Notice that the choice of subject pairs is not at all evenly distributed as depicted in Figure 1b. On the contrary, all combinations of the most prominent eight subjects account for more than 50% of the students, which have therefore been the focus of the study.

Table 2: Scope of the study.

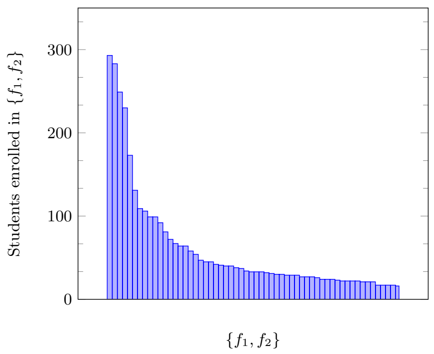
Subjects	8
Courses	767
Sections	1,454
Students	2,240
Quantum size Δ	5
Quanta	454

As summarized in Table 2, the supply side consists of 1454 different sections belonging to 767 courses and we seek to assign 454 quanta to them. The courses

belong to one of 8 subjects (English, German, History, Geography, Chemistry, Physics, Mathematics, Biology), constituting more than half of all students enrolled in teacher education. The quanta are established using historical anonymized examination data of all students enrolled in the study program, whereas courses and number of sections are taken from a different source representing the curricula.
The examination data further serves as the basis for the derivation of the *delay factor* of a course (as outlined in Section 4.2). As depicted in Figure 1a based on ECTS (not weekly hours) over 40% of the courses are passed late using a threshold of $T = 30$ and 79 courses are preponed entirely. Consequently, taking into account *delayed courses* C_d reflects actual student behavior in reality. The surprisingly high fraction of delayed courses indicates that (i) overlaps of courses may indeed be a reason for slowed-down progress (as often claimed by students, but sometimes questioned but other involved persons) and (ii) an optimized timetable should take the delay of courses into account for increasing study performance.



(a) Histogram of delay values.



(b) Distribution of chosen subject pairs.

Fig. 1: Information derived from examination data.

5.2 Preliminary Results

We conducted our tests on a PC with processor Intel Core i5-9500 with 3.00GHz and 32GB RAM. The data processing and preparatory computation steps as well as the mathematical models have been implemented in Python and solved using the Gurobi solver (version 9.0.0). The weighting factors α and β are both set to 1, though this setting will be the subject of further discussions with all stakeholders.
Initially, a *feasibility check* concerning the overall section capacities is carried out. Based on the provided data we compare for each course the number of available places with the required places resulting from the given number of students for

each term and subject combination. In case of a shortfall, additional sections are introduced such that at least a general coverage is possible, without considering conflict freeness.
At the time being, we can give results for Phase 1 and 2, while Phase 3 could not be carried out yet due to a lack of data. However, the first two phases are considered to be the essential part, since the final output, and the timetable's applicability respectively is largely based on the quality reached in Phase 2, building upon the output of Phase 1. Moreover, from a computational point of view, Phase 2 poses the major obstacle, while Phases 1 and 3 will not be a hurdle for practical solvability. Within the given scope we solve the current test case yielding 537,330 ILP variables (where 469,743 are binary) and reach preliminary results as follows.

Table 3: Results of the first two phases.

subjects	Π	time[s]	obj. part 1	obj. part 2	gap[%]
8	20	8.5	632	2,307	0.00
8	15	9.4	573	2,611	0.00
8	10	10.0	651	2,778	0.00
8	5	16.8	733	2,941	0.00

Examining the results of the case and analyzing different parameter settings we can make the following observations:

- The test case concerning the scope in Table 2 can be solved to optimality in a surprisingly short running time despite the considerable size of the ILP model.
- Reducing the parameter Π, which is limiting the number of sections at the same time period, results in decreasing solution quality and at some point, the instance is not feasible anymore (e.g. $\Pi = 4$ for the combinations of 8 subjects). Table 3 shows for different Π the number of integer and binary variables the ILP-model of the test case comprises, the objective function value, and the elapsed computation time (in seconds).
- Extending the scope, however, to a larger set of subject combinations may result in infeasibility of the problem - even with the addition of just one single subject (i.e. with all the associated pairs of subjects). If the problem remains feasible, different numbers of variables are generated for different added subjects. Also, the new solution values vary a lot as well as the computation times (see examples in Table 4).
  An obvious reason for this behavior is that the amount of enrolled students and the course structure is subject-related. A thorough examination of this relationship is subject to further investigation.
- In general it is a surprise, that computation time is not an issue, while feasibility is.

Table 4

added subject	time[s]	obj.	gap[%]	int. variables (binary)
Informatics	9.0	4,003	0.00	589,515 (485,538)
French	34.85	3,924	0.00	580,658 (477,011)
Nutrition	51.81	3,978	0.00	581,493 (477,696)
Sports	-	infeasible	-	635,892 (530,025)

For the practical application of our solution approach to the full planning problem with 28 subjects we currently see five measures for dealing with the inherent infeasibility issue:

(i) Increase the number of options by adding more sections.
(ii) Increase the capacity $cap(c)$ of each section of course c.
(iii) Restrict the optimization model to a subset of the most frequently chosen subjects (up to 10 or 12) and add less popular subjects by a manual process (basically as it is done now).
(iv) Omit pairs of subjects from consideration which are chosen by a very small number of students.
(v) Reduce the number of ECTS for which collision-free courses are provided by the planning system for every student.

While measures (iii) and (iv) will be unavoidable and easily accepted, there is an interesting cost/quality trade-off involved in the decision for (i) and (ii): The former causes additional costs (assuming that external teachers are available) whereas the latter comes for free but diminishes teaching quality. Thus, it will be very interesting for the decision-makers to be informed about the effect of employing certain levels of measures (i) and (ii). In particular, it will be interesting to identify a suitable subset of crucial courses for which these measures should be applied to reach feasibility. Measure (v) is easy to implement and does not incur any direct cost, but it compromises the original goal of this project.

6 Conclusions

In this paper, we developed a solution approach for a complex university timetabling task arising at the University of Graz, Austria. The main features which make our problem different from standard university timetabling instances are the following:

1. Each student is enrolled in two different subjects selected arbitrarily from a wide range of available subjects.
2. Student's progress does not follow a strict term pattern but may exhibit highly irregular behavior, including gaps and deviations from the prescribed ordering of courses.

3. Timetable planning is done at an early stage when the data of actual student numbers and their progress status, as it is required for finally assigning courses to students, is still subject to major changes.
4. Most courses are offered in several parallel groups, which requires the sectioning of students to reach a conflict-free timetable for a highly heterogeneous set of students.

Our solution approach consists of three phases. Two of them solve variants of the linear assignment problem, extended by conflict constraints. The main planning task (Phase 2) is performed by an intricate integer linear program (ILP). In this way, we managed to determine feasible timetables offering conflict-free course allocations for the projected student body. The data for the final allocation of individual students are still missing, but the handling of the main computational hurdle, namely the solution of a complex ILP, can be illustrated by our computational results. These reveal in particular that the choice of subjects given their course structure and amount of enrolled students per term has a non-trivial impact on solvability, computation time, and solution quality and is of interest for further investigation.
We expect to employ the full solution approach in practice for the planning task in the next year. It will also be interesting to investigate additional options for the objective function since different stakeholders have different ideas about the appropriate quality measure of a timetable.
In the future, we could also try to rate the difficulty of the problem according to the 'Complexity' as introduced in [9].

Acknowledgments This study was partially funded by the University of Graz under the Field of Excellence "COLIBRI".

References

1. Aubin, J., Ferland, J.A.: A large scale timetabling problem. Computers & Operations Research **16**(1), 67–77 (1989)
2. Bagger, N.C.F., Sørensen, M., Stidsen, T.R.: Dantzig–Wolfe decomposition of the daily course pattern formulation for curriculum-based course timetabling. European Journal of Operational Research **272**(2), 430–446 (2019)
3. Banks, D., van Beek, P., Meisels, A.: A heuristic incremental modeling approach to course timetabling. In: Carbonell, J.G., Siekmann, J., Goos, G., Hartmanis, J., van Leeuwen, J., Mercer, R.E., Neufeld, E. (eds.) Advances in Artificial Intelligence, Lecture Notes in Computer Science, vol. 1418, pp. 16–29. Springer (1998)
4. Ceschia, S., Di Gaspero, L., Schaerf, A.: Educational timetabling: Problems, benchmarks, and state-of-the-art results. arXiv preprint arXiv:2201.07525 (2022)
5. Dostert, M., Politz, A., Schmitz, H.: A complexity analysis and an algorithmic approach to student sectioning in existing timetables. Journal of Scheduling **19**(3), 285–293 (2016)
6. Esmaeilbeigi, R., Mak-Hau, V., Yearwood, J., Nguyen, V.: The multiphase course timetabling problem. European Journal of Operational Research **300**(3), 1098–1119 (2021)

7. Holm, D.S., Mikkelsen, R.Ø., Sørensen, M., Stidsen, T.J.: A graph-based MIP formulation of the international timetabling competition 2019. Journal of Scheduling pp. 1–24 (2022)
8. Johnes, J.: Operational research in education. European Journal of Operational Research **243**(3), 683–696 (2015)
9. de La Rosa-Rivera, F., Nunez-Varela, J.I., Puente-Montejano, C.A., Nava-Muñoz, S.E.: Measuring the complexity of university timetabling instances. Journal of Scheduling **24**(1), 103–121 (2021)
10. Lach, G., Lübbecke, M.E.: Curriculum based course timetabling: new solutions to Udine benchmark instances. Annals of Operations Research **194**(1), 255–272 (2012)
11. Müller, T.: ITC 2019: Preliminary results using the UniTime solver. In: Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling (PATAT), Volume III (2022)
12. Müller, T., Murray, K.: Comprehensive approach to student sectioning. Annals of Operations Research **181**(1), 249–269 (2010)
13. Müller, T., Rudová, H.: Real-life curriculum-based timetabling with elective courses and course sections. Annals of Operations Research **239**(1), 153–170 (2016)
14. Müller, T., Rudová, H., Müllerová, Z.: 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). pp. 5–31 (2018)
15. Schindl, D.: Optimal student sectioning on mandatory courses with various sections numbers. Annals of Operations Research **275**(1), 209–221 (2019)

Planning for high-speed railways in the Czech Republic

Pavel Dostál[1], Hana Rudová[1], and Vilém Pařil[2]

[1] Faculty of Informatics, Masaryk University
Brno, Czech Republic
hanka@fi.muni.cz

[2] Institute for Transport Economics, Geography and Policy
Faculty of Economics and Administration
Brno, Czech Republic
vilem@mail.muni.cz

1 Introduction

Strategic capacity planning [9,1] for trains in high-speed railways introduces an important problem that is of high interest for the planning of future railway infrastructure in the Czech Republic. We are working on optimization methods [3,4,2] for the planning of train capacities to assess and discuss additional connection sites between large cities and to devise suitable train timetables based on a fixed clock timetabling. We propose an integer linear programming formulation based on an arc-based multi-commodity network flow model and a space-time graph [12].

The demand of passengers represents the crucial input part of our model. The future demand will allow us to compute future supply for train capacities well. Prediction of the correct demand is one of the critical components in transport planning [10], which is essential for the efficiency of transport infrastructures [13]. To handle this problem, we are using new big data from mobile operators in the Czech Republic [5], which were collected to study the behavior of passengers along with the planned high-speed infrastructure (Praha-Brno-Ostrava). Consideration of data from mobile operators is a relatively new phenomen [6]. We are not aware that mobile operators' big data was applied for long-term capacity planning in high-speed railways. Given that mobile operators provided us with the data with a rich set of characteristics, we have applied them for the initial demand estimate to be included in our mathematical programming models. Currently, our approach uses the data provided by mobile operators directly. It has various drawbacks, such as insufficient data coverage at border connections[3], multimodal transportation at some edges[4], or missing considerations of future demand changes. To get more accurate demand prediction, we plan to combine

[3] The data from mobile operators were required to include passenger trips containing one of the edges Praha-Brno, Brno-Ostrava, or Praha-Ostrava only.

[4] Train track and highway are too close at the edge between Brno and Ostrava, which results in inaccurate data by mobile operators.

big data from mobile operators with the small data from questionnaires and statistics and device models of future passenger demands.

Our proposed model concentrates on rolling stock management and allocation while considering preliminary timetables and demands available from mobile operators' big data. This ongoing work will discuss the results of our current approach implemented in CPLEX Optimization Studio, which allows us to compute the optimal solution to the problem.

2 Problem description

Let us discuss the problem we are considering in our current work. We want to decide proper types and the number of trainsets, which is the typical *rolling stock management* [3] task. So, the first rolling stock management part of our problem lies in computing two types of trainsets used for the entire network and their number. Two trainset types are required to achieve a better investment cost due to a larger number of pieces of each type given by its capacity.

To decide so, we must know how much this particular set would cost if we run high-speed transportation with chosen trainset types. Therefore, we will consider *rolling stock allocation* [7] as well. Our rolling stock allocation problem consists of assigning trainsets to connections based on predefined train timetables.

Terminology The route connecting two terminal stations is called a *line*. The line is divided into several *segments*, which are defined as the route between two adjacent stations. An example may be the line Wien → Prague, divided into segments Wien → Břeclav, Břeclav → Brno and Brno → Prague. Each line is periodically served with a specific frequency, usually a day or a week-long. One instance of the line at a given time is called *connection*. For example, for the line Berlin → Wien, we may have 16 connections per day, the first connection starting at 5:10 in Berlin and arriving at Wien at 10:00. A segment served by a particular rolling stock piece at a particular time is called a *subconnection*, e.g., Břeclav → Wien at 9:15. An individual *trainset* can serve a connection, or more trainsets can be jointed to increase the overall capacity. There are trainsets of different *type*, which are currently distinguished by their capacities.

Objective The current problem is to minimize the total cost that transportation companies would have to pay to purchase and run high-speed transportation taking into account costs linked only to trainsets. The total cost is defined by investment cost including modernization, the variable cost depending on the traveled kilometers of each used trainset, and gain from operating trainsets abroad for each seat and each kilometer run abroad.

Constraints Based on our specification, the constraint relating to rolling stock management is only one, specifying that it is possible to choose only two trainset types.

On the other hand, several constraints are related to rolling stock allocation. First and foremost, passenger demand is considered a hard constraint, with each subconnection having predefined passenger demand (minimal capacity). Each subconnection must be served by at least one trainset, even if there would not be any predefined passengers from a dataset.

Maximally two trainsets can be joined into one high-speed train with an overall capacity not exceeding 1,000 seats. Trainsets may be joined and disjoined only in certain stations specified in the dataset. For each station at the end of each considered period, there must be the same number of trainsets at the beginning of the same period. Additional constraints are related to passenger comfort.

3 Model in example

We propose to use a multi-commodity network flow model where each trainset type appears as a separate commodity. The model is based on the paper by Schrijver et al. [11] where they used a multi-commodity network flow graph to minimize the total number of rolling stock units used. They did not consider any price calculation or restriction on trains' capacity or length.

We will use Figure 1 for demonstration of the multi-commodity network graph and our model. Each node is represented by a station in a given time. Blue and red edges represent two connections. Labels of each edge refer to the number of trainsets of each type (we have two types in our example). The source represents the beginning of the scheduling interval where all trainsets start, and the sink is the opposite as a terminal node for all trainsets.

There are two types of variables. The integer variable is defined for each edge and trainset type and specifies the number of trainsets. The boolean variable for each trainset type defines if a particular trainset type is used. We have proposed a linear integer programming model based on described multi-commodity network flow and implemented it using CPLEX Optimization Studio 12.8 [8]. In the full version of this paper, we will discuss all constraints and the objective with their integer linear programming model.

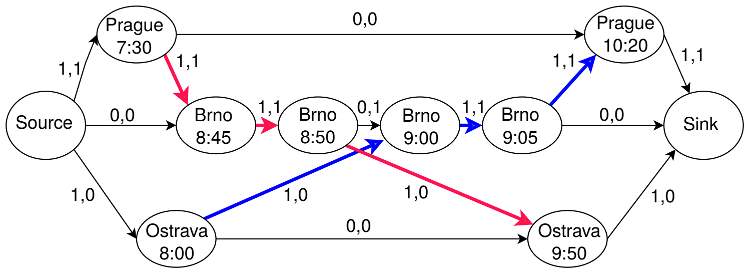


Fig. 1. Simple example of multi-commodity network flow.

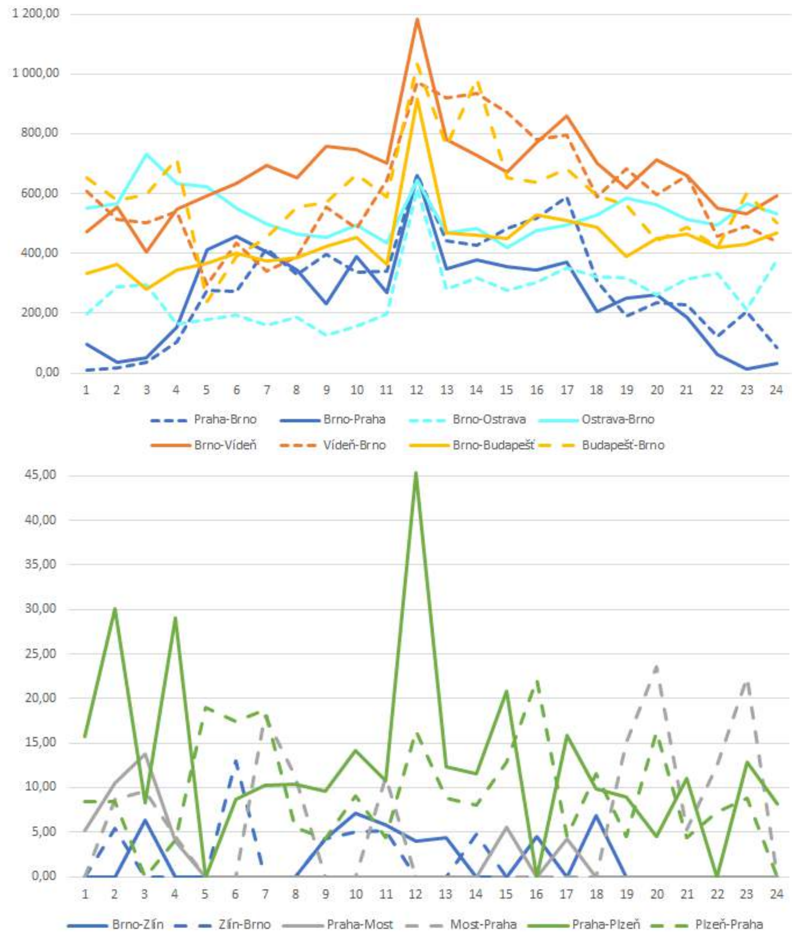


Fig. 2. Number of passengers for backbone (top) and border (bottom) connections during the day.

4 Data set and preliminary experiments

We have demand data available from mobile operators and preliminary train timetables provided by the Czech national railway company České dráhy, a.s.. We consider 15 stations in our problem. It includes terminal stations and stations where the exchange of trainsets can happen. There are 12 lines, 344 connections, and 721 edges between stations in time. Currently, we compute a solution for one day only. Figure 2 demonstrates the number of passengers in backbone and border connections[5] based on the data from mobile operators. We can see

[5] Backbone connections represent the critical railway infrastructure for high-speed trains, and border connections represent part of the infrastructure in border regions.

that the backbone data have reasonable demands. Still, it would be desirable to enhance them for border connections where data from mobile operators are insufficient because the passenger data were collected when including backbone connections only.

In Table 1, we can see characteristics of the optimal solution, which can be computed in approximately 1 minute. We would use 87 trainsets with 200 seats and 31 trainsets with 500 seats. We need to say that the rolling stock part of the problem is not very demanding because trainsets of the smallest capacity are necessary to cover border connections, and trainsets with 500 seats cover the maximal allowed capacity of 1,000 passengers.

We experimented with two other models based on boolean variables for each trainset type, individual trainset and subconnections, and a path-based model [12] with integer variables for each path and trainset type. First, we have used a toy network with two lines. The first model could not find any solution within 90 minutes, while our model provided a solution within 0.14 seconds. The path-based model succeeded in finding an optimal solution within 1.98 seconds. For the complete network, it was impossible to run the path-based model because the number of paths increased drastically in preprocessing, and it did not fit into the memory. To conclude, both other models were shown insufficient.

5 Conclusion

In this study, we aimed to solve resource stock management and allocation problems for high-speed railways in the Czech Republic based on the big data available from mobile operators. Our current results demonstrated to the Czech national railway company attained their high interest. However, it is necessary to enrich the current data with additional inputs corresponding with corrections of missing demand data that have weak parts, especially at border connections. For instance, we have now additional data about the sold tickets at particular border connections and moreover corresponding data from Transport Yearbooks on domestic connections. Also, we need to incorporate a forecast of future diverted and induced demand. Certainly, our current model would deserve extensions, for example, in terms of one-week cycles rather than the current one-day. A more

Table 1. Preliminary results.

Possible trainset capacities	200, 300, 350, 400, 450, 500, 700	
Selected trainset capacities	200	500
#trainsets	87	31
#served edges	607	175
Avg. distance in Czech Rep. (km)	1,036	590
Avg. distance abroad (km)	370	679
Avg. occupations (%)	21.4	47.6
#passengers	27,444	46,651
#trainset exchanges	109	

complex extension would be introduced by including the investment cost due to the high-speed line constructions. Finally, a full body of experiments will be presented in the final version of the paper.

Acknowledgements Data from mobile operators are provided with the support of the Ministry of Education, Youth, and Sports of the Czech Republic in the Operational Programme „Research, Development, and Education", grant New mobility – high-speed transport systems and transport behavior of the population ID muni 1312/2017, ID CZ.02.1.01/0.0/0.0/16_026/0008430.

References

1. Abril, M., Barber, F., Ingolotti, L., Salido, M., Tormos, P., Lova, A.: An assessment of railway capacity. Transportation Research Part E: Logistics and Transportation Review **44**(5), 774–806 (2008)
2. Borndörfer, R., Klug, T., Lamorgese, L., Mannino, C., Reuther, M., Schlechte, T.: Recent success stories on integrated optimization of railway systems. Transportation Research Part C: Emerging Technologies **74**, 196–211 (2017)
3. Caprara, A., Kroon, L., Monaci, M., Peeters, M., Toth, P.: Passenger railway optimization. In: Barnhart, C., Laporte, G. (eds.) Transportation, Handbooks in Operations Research and Management Science, vol. 14, pp. 129–187. Elsevier (2007)
4. Cats, O., Haverkamp, J.: Optimal infrastructure capacity of automated on-demand rail-bound transit systems. Transportation Research Part B: Methodological **117**, 378–392 (2018)
5. Ficek, M.: Handover documentation of big data public procerement within new mobility. In: Ministry of Education, Youth, and Sports of the Czech Republic in the Operational Programme "Research, Development and Education", grant New mobility – high-speed transport systems and transport behaviour of the population, ID CZ.02.1.01/0.0/0.0/16_026/0008430. Praha:CE Traffic (2020)
6. Gundlegård, D., Rydergren, C., Breyer, N., Rajna, B.: Travel demand estimation and network assignment based on cellular network data. Computer Communications **95**, 29–42 (2016), mobile Traffic Analytics
7. Huisman, D., Kroon, L.G., Lentink, R.M., Vromans, M.J.: Operations research in passenger railway transportation. Statistica Neerlandica **59**(4), 467–497 (2005)
8. IBM ILOG CPLEX Optimization Studio, CPLEX Language User's Manual 12.8 (2017)
9. Jensen, L.W., Landex, A., Nielsen, O.A., Kroon, L.G., Schmidt, M.: Strategic assessment of capacity consumption in railway networks: Framework and model. Transportation Research Part C: Emerging Technologies **74**, 126–149 (2017)
10. Lundqvist, L., Mattsson, L.G. (eds.): National transport models: Recent developments and prospects. New York: Springer (2001)
11. Schrijver, A.: Minimum circulation of railway stock. Cwi Quarterly **6**(3), 205–217 (1993)
12. Thorlacius, P., Larsen, J., Laumanns, M.: An integrated rolling stock planning model for the copenhagen suburban passenger railway. Journal of Rail Transport Planning & Management **5**(4), 240–262 (2015)
13. Viturka, M., Pařil, V.: Regional assessment of the effectiveness of road infrastructure projects. International journal of transport economics **42**(4), 507–528 (2015)

Noname manuscript No.
(will be inserted by the editor)

An Iterative Approach for the Mobile Workforce Tactical Scheduling Problem with Frequency Constraints

Anne-Laurence Thoux · Stéphane Dauzère-Pérès · Chloé Desdouits · Dominique Feillet

Received: date / Accepted: date

Abstract Keywords Workforce scheduling and routing · Frequency constraints · Iterative approach

1 Introduction

Workforce planning and scheduling problems mainly focus on designing teams or on creating daily plans. However, recent studies have highlighted that management aspects should not be left out, and that models should better consider the complexity of real-life objectives and constraints, see e.g. the reviews [3], [5] and [7]. Those reviews raise a need for a broader view when building the daily schedules of employees, for example when different resources are required to perform some tasks. This, for instance, is the case for surgery operations, where balancing the workload among resources is required (for instance the amount of work or the type of the tasks assigned) and resources must be coordinated (see e.g. [6]). Scheduling the tasks to perform in bidding order, or depending on their deadline, is no longer enough to ensure that the resulting plan is optimal, or that every required task is performed on time. In such cases, studying personnel scheduling problems on a longer horizon than several days becomes mandatory. This need is reinforced in some contexts such as healthcare, where the number of beneficiaries is continuously increasing. Optimizing the distribution of the resources is there an opportunity to reduce the costs and the frenetic working pace, while still providing high-quality services (see e.g. [6]).

Anne-Laurence Thoux · Stéphane Dauzère-Pérès · Dominique Feillet
Mines Saint-Etienne, Univ. Clermont Auvergne, CNRS, UMR 6158 LIMOS
CMP, Department of Manufacturing Sciences and Logistics, Gardanne, France
E-mail: anne-laurence.thoux@emse.fr

Anne-Laurence Thoux · Chloé Desdouits
DecisionBrain, Paris, France
E-mail: anne-laurence.thoux@emse.fr

In the light of the above, we believe that studying an optimization problem at the tactical level to plan the tasks to be performed by a team of employees on several weeks helps to handle more complexity and to find even better solutions. By decoupling the decisions depending on the temporal horizon they impact, more realistic workforce planning and scheduling problems should become solvable.

In our problem, which comes from industry, we focus on the scheduling of tasks with frequency constraints for a mobile workforce. Scheduling and routing optimization problems under frequency constraints have not been studied much in the literature, and usually with restrictions since the assignment of the tasks to the employees is generally given as an input (see [2] and [8]).

In the context of a mobile workforce, where employees travel from one client to the next to perform cleaning tasks, we call our problem the *Mobile Workforce Tactical Scheduling Problem with Frequency Constraints*. The goal is to determine a plan on several weeks which defines who will perform which task in which day. All the required tasks must be scheduled and a trade-off must be ensured between the clients' and the company's interests. Tasks must be distributed over the horizon according to frequency constraints and the total working cost must be minimized. As the workforce is mobile, the tactical plan is the basis on which the daily routes of the employees are optimized at the operational level. The tactical plan has thus to take traveling distances into account to be consistent. To ensure this consistency, we adapt the two-phase iterative heuristics of [4] for the integrated production planning and scheduling problem and of [1] for the production routing problem.

2 Solution Approach and Results

Absi et al. [1] developed a two-stage heuristic, that iterates between a lot-sizing phase and a routing phase, to solve a production routing problem. The lot-sizing phase decides which vehicle will deliver which client in which day, and the routing phase optimizes the routes in each day. Depending on the quality of the routes determined in this second phase, some metrics are raised as a feedback to improve the lot-sizing plan in the following iteration.

We propose to adapt this iterative method by replacing the lot-sizing phase by the construction of the tactical plan with frequent tasks and each vehicle by an employee of the team. The tactical plan is optimized by solving an integer programming model with a standard solver. The routes are determined for each employee and each day by an assignment heuristic and improved by local search when needed. The routing heuristics were developed by DecisionBrain (www.decisionbrain.com).

Our industrial instances include up to 30 employees and 2100 tasks to perform each month. In order to decrease the size of those instances and be able to plan on several months, we propose to pre-process the data in a clustering step. Using a gaussian mixture model, we determine groups of employees that are assumed to be equivalent and use these groups in the heuristic. Furthermore, this approach improves the construction of the operational routes, since several routes can simultaneously be designed and improved in one cluster.

We compare the results obtained with our algorithm for real-size instances to the results of the current DecisionBrain's algorithms and of a one-shot integer programming model, which estimates the distances while solving the tactical problem. It appears that our iterative method allows the required tasks to be performed with a good respect of the required frequencies. By varying the value of the penalties induced by not respecting the frequency constraints, we can observe the trade-off between the satisfaction of the frequency constraints and the cost resulting from subcontracting and traveling.

3 Conclusion and Perspectives

Scheduling tasks on a horizon of several weeks before optimizing the daily routes allows more complex constraints to be taken into account. We showed here the interest of scheduling tasks with frequency constraints on a longer horizon before optimizing routes. In particular, more complexity can be handled while getting solutions faster than when directly optimizing the daily routes.

By varying the values of some parameters in the iterative approach, different solutions can be determined and thus Pareto optimal solutions. Depending on the definition of the quality of the routes, the tactical plan can find the best distribution of the tasks to satisfy frequency constraints while taking the constraints of operational routes into account. Planners could then interact with the approach by selecting one of several solutions and improving them. More constraints could also be added in the tactical plan, such as dependencies between tasks, assignment of the tasks to a specific employee or day, different skills for employees and workload balancing between the employees and over the planning horizon.

By solving the problem in two phases, our goal is to find the best way to embed scheduling and routing considerations in workforce planning and scheduling problems. As far as future research is concerned, our main perspective involves studying the robustness of the plan regarding small perturbations (e.g. duration of a task or delay of an employee) or the ease of re-building the plan when major disruptions occur (e.g. the absence of an employee).

References

1. N. Absi, C. Archetti, S. Dauzère-Pérès, and D. Feillet, A two-phase iterative heuristic approach for the production routing problem, Transportation Science, 49(4), 784–795 (2014)
2. Y.-J. An, Y.-D. Kim, B. Jeong, and S.-D. Kim, Scheduling health-care services in a home healthcare system, Journal of the Operational Research Society, 63(11), 1589–1599 (2012)
3. J. A. Castillo-Salazar, D. Landa-Silva, and R. Qu, Workforce scheduling and routing problems: literature survey and computational study, Annals of Operations Research, 239(1), 39–67 (2016)

4. S. Dauzère-Pérès and J.-B. Lasserre, Integration of lotsizing and scheduling decisions in a job-shop, European Journal of Operational Research, 75(2), 413–426 (1994)
5. P. De Bruecker, J. Van den Bergh, J. Beliën, and E. Demeulemeester, Workforce planning incorporating skills: State of the art. European Journal of Operational Research, 243(1), 1–16 (2015)
6. N. Dellaert and J. Jeunet, A variable neighborhood search algorithm for the surgery tactical planning problem, Computers and Operations Research, 84, 216–225 (2017)
7. C. Fikar and P. Hirsch, Home health care routing and scheduling: A review, Computers & Operations Research, 77, 86–95 (2017)
8. W. Jang, H. Lim, T. Crowe, G. Raskin, and T. Perkins, The missouri lottery optimizes its scheduling and routing to improve efficiency and balance, Interfaces, 36, 302–313 (2006)

Grouping and timetabling for multi-league sports competitions

Miao Li$^{1[0000-0002-6205-814X]}$ and Dries Goossens$^{1[0000-0003-0224-3412]}$

Faculty of Economics and Business Administration, Ghent University, Tweekerkenstraat 2, 9000 Ghent, Belgium
miao.li@ugent.be;dries.goossens@ugent.be

Abstract. This work presents a bi-objective grouping and timetabling problem for sports competitions that are played using multiple leagues. We propose a decision-making framework to uncover the trade-off between minimizing travel distance and venue capacity violations, in order to meet the preferences of different stakeholders.

Keywords: OR in sports · bi-objective optimization · sport team grouping · sports timetabling · multi-league scheduling

1 Introduction and problem description

Youth and amateur sports provide non-professional players with the opportunity to exercise and develop athletic skills. In such sports, the number of teams involved can reach the thousands, and considerable efforts are required to organize their competition. A first challenge is that teams with players of the same age, gender or strength need to be grouped into leagues. This problem is known as the sports team grouping problem (STGP [3]). The main objective with this problem is to minimize the total travel distance travelled by all teams, knowing that teams visit each other team in their league, but none of the teams from other leagues. Another issue is setting up a timetable for each of these leagues, i.e. deciding when each match is to be played. This problem is known as multi-league sports timetabling problem (MLSP [1]). Since teams of the same club share the same infrastructure (venue), whose capacity should be respected, the leagues are interdependent and cannot (optimally) be scheduled one by one. In particular, per time slot on which the number of home-playing teams from a club exceeds the number of home matches the club can host, a capacity violation arises. Hence, the main objective in the multi-league sports timetabling problem is to minimize the total venue capacity violations over all clubs.

In practice, these problems are handled sequentially: first solve the sports team grouping problem, and then, based on the resulting league composition, solve the multi-league sports timetabling problem. However, some team grouping may allow a timetable where few or even no clubs face a capacity issue, while another grouping may be more problematic, resulting more venue capacity violations. As its main novelty, this paper integrates both decision problems (i.e., STGP and MLSP) as the multi-league grouping and timetabling problem (MLGTP), with two objective functions: the minimization of total distance travelled by all teams and the minimization of total venue capacity violations over all clubs. Both objectives are demonstrated to be conflicting when leagues have different sizes. We therefore investigate their trade-off, and develop a method that allows us to approximate the Pareto front.

2 Proposed method

For instances with a somewhat realistic scale, an approach like the ϵ-constraint method (ECM) [2] is intractable within a reasonable computation time. Hence we develop a two-phase two-layer constructive algorithm (SLCM) to find an approximate Pareto front for our bi-objective problem in a reasonable time. The optimization process begins with an initial Pareto solution. Next, in the first phase, the problem is decomposed into STGP and MLSP sub-problems, which are solved by a two-layer method sequentially and iteratively. At each iteration, the outer layer is first used to minimize the total distance travelled, where the initial assignment of teams to leagues is further improved by simulated annealing. In the second phase, we enlarge the search space and improve incumbent candidate Pareto solutions. Then, given a list of potential Pareto efficient solutions of MLGTP, an approximate Pareto front is identified. The overall process is able to optimize two distinct objectives simultaneously.

3 Preliminary results and conclusion

Due to the fact that there are no MCGTP instances available in the literature, we created some sets of instances with various numbers of teams, leagues and clubs, as well as league and club sizes, as given in Table 1.

Table 1. Overview of instance types and their features

Instance type	Instance ID	No. teams	No. clubs	No. leagues	League size				
					16	10	8	6	4
Small-scale	S1	18	8	3	–	–	1	1	1
	S2	34	16	3	1	1	–	1	–
Large-scale	N1	80	17	8	2	–	6	–	–
	N2	112	18	11	3	–	8	–	–
	N3	144	20	13	5	–	8	–	–
	N4	176	20	16	6	–	10	–	–
	N5	208	25	19	7	–	12	–	–

Note: '–' indicates the instance type does not contain leagues of the corresponding size

After preliminary tests for parameter configurations, we evaluate the performance of the proposed bi-objective solution method. Besides the two-phase two-layer constructive method (SLCM), in order to study the contribution of the second phase, each problem instance is solved by the two-layer constructive method (LCM) which only includes the first phase of SLCM.

We first assess the ability of the methods on small-scale instances. The results show that SLCM is capable of producing optimal Pareto solutions for all instances, and that the approximate Pareto set obtained by the LCM lies close to the optimal set. Additionally, compared with the computationally demanding ECM, LCM and SLCM both operate far more efficiently.

We next turn our attention to large-scale instances. For these instances, ECM cannot obtain a feasible solution within the time limit (7200s). Four evaluation metrics, namely the number of Pareto efficient solutions (NPS), the diversification metric (DM), the mean ideal distance (MID) and the spacing metric (SM) are applied to compare different sets of Pareto solutions produced by SLCM and LCM. While with respect to DM, both algorithms display more or less the same performance, based on the metrics NPS, MID and SM, SLCM is clearly superior to LCM. Overall, this indicates that the second phase of the SLCM approach is a valuable addition.

With respect to computation time, small-scale instances were solved by SLCM in less than 405 seconds, while we needed no more than 1800 seconds for the large-scale instances.

In summary, our preliminary results suggest that the proposed method is able to offer a good approximation of the Pareto front, helping the league organizers to find a good compromise proposal to balance the travel distance of all teams and the venue capacity violations over all clubs.

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)
2. Haimes, Y.: On a bicriterion formulation of the problems of integrated system identification and system optimization. IEEE transactions on systems, man, and cybernetics **1**(3), 296–297 (1971)
3. 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)

A Pragmatic Approach for Solving the Sports Scheduling Problem

Angelos Dimitsas[1], Christos Gogos[1], Christos Valouxis[2], Alexandros Tzallas[1], and Panayiotis Alefragis[3]

[1] University of Ioannina, Dept. of Informatics and Telecommunications, Arta, Greece
{a.dimitsas, cgogos, tzallas}@uoi.gr
[2] University of Patras, Dept. of Electrical and Computer Engineering, Patras, Greece
cvalouxis@upatras.gr
[3] University of Peloponnese, Dept. of Electrical and Computer Engineering, Greece, Patras
alefrag@uop.gr

Abstract. Sports Scheduling is a problem with many variations, regarding the sport type, the various hard rules that have to be obeyed and the quality metrics that are expected to be met. Various stakeholders including organizers, teams, spectators and others have interest in acquiring high quality schedules that satisfy rules and constraints crucial from their point of view. In this work we propose an approach for solving the Sports Scheduling problem as defined in the International Timetabling Competition 2021 (ITC2021). We describe the analytical formulation of each constraint, as it can be modeled for a CP solver and five moves that can be used for altering a schedule by a metaheuristic. We also document the experience gained in trying to address the problem using heuristics, metaheuristics, Constraint Programming and the capable ORTools CP/SAT solver. Despite the computational hardness of the problem instances, our approach managed to achieve good results for most of them.

Keywords: Sports Scheduling· Constraint Programming · Simulated Annealing

1 Introduction

Sports Scheduling is the problem of constructing a tournament schedule consisting of matches among competing teams that form a league. The schedule should satisfy the constraints imposed by the tournament's rules and be 'invisible' in the sense that the various stakeholders such as organizers, teams, spectators, and others should not have legitimate reasons to question it.

Sports scheduling exists for as long as there are sports and teams willing to participate in tournaments with matches against each other. For some sports, like tennis, instead of teams, individual athletes compete. Furthermore, there are tournaments, like chess or other board games tournaments, where the actual matches would be hardly identified as sports, in the typical sense. Esports (electronic sports), is another example of a competition for which its events should be scheduled according to a carefully crafted plan. The same principles regarding scheduling apply to all previously identified cases of tournaments and are special instances of the sports scheduling problem.

Several variations of tournaments exist including single round tournaments, double round tournaments, tournaments with elimination games, compact tournaments (all teams have matches in every timeslot), etc. Some heuristics for constructing sport schedules are known for many years, like the circle [11] method and the Berger [3] method. But when constraints are added the problem quickly becomes very hard to solve. Such constraints might involve the avoidance of consecutive away games for all or some teams, the enforcement of minimum distances (number of time slots) between a match and the rematch, and many others. In this paper, an approach of generating high quality schedules for the compact, double round robin (2RR) type of tournament, is presented. This approach is based on several moves that keep the schedule feasible and a Constraint Programming formulation that results in a model capable of performing complex moves when no progress can be achieved otherwise. A move is eventually a series of changes involving teams participating in a set of matches. Some moves result in better schedules and some others may lay the foundation for performing subsequent moves that will lead to even better schedules.

2 Related work

Several real life tournaments have been addressed using automated techniques involving mathematical programming, constraint programming, metaheuristics and heuristics; e.g., the Belgian soccer league [7], the Brazilian soccer tournament [16], the Finnish national youth ice hockey league [14], the Chilean soccer leagues [1], the South American qualifiers for FIFA 2018 [6], the Icelandic football league [8].

Lewis and Thompson, in [12] present the association of the sports scheduling problem to the a graph coloring problem. Moreover, an edge coloring presentation of the problem is available at [10].

Regarding the exploration of the solution space in [4] it is established that the solution space is not connected by the usually used neighborhood structures, i.e. it's impossible starting from a feasible timetable to reach all other possible timetables just by performing the usual heuristic moves proposed in the bibliography, and [9] proposed a new neighborhood operator to handle this issue.

Since sports timetabling usually results in problems of big sizes, decomposition approaches can be advantageous. In [18] a first schedule then break approach was tried. First it was decided when teams would meet, and the home advantage is decided later. The opposite, first break then schedule approach can be seen at [17], first it is decided where each team plays at home and the teams are paired later. An effort on minimizing breaks is available at [13]. A research on feasible home-away patterns is presented at [2].

3 Problem Description

The problem description can be found at [19] and it refers to tournaments categorized as time-constrained double round robin. Time-constrained or compact means that the timetable uses the minimum number of time slots, i.e. in each time slot all teams play in matches.

3.1 The Base Constraints

The base constraints for each tournament are the format of the tournament. All tournaments are in double round robin format, i.e. each team has two matches against every other team, one at home and one away. Some of the tournaments contain the Phase rule; the timetable is divided in half (two phases), a match and its rematch must be in a different phase. All tournaments are compact.

3.2 The Hard and Soft Constraints of ITC2021

All type of constraints can be either hard or soft as of the type attribute. Hard constraints must be satisfied and soft constraints create deviations penalized in the objective function. There are 9 types of constraints in 5 different constraint categories.

Capacity Constraints Capacity constraints regulate the matches played by a team or a group of teams at home or away.

CA1 constraints regulate the number of matches a team plays at home or away in specific slots.

CA2 constraints regulate the number of matches a team plays at home or away in specific slots against specific teams.

CA3 constraints regulate the number of matches a team plays at home or away in a sequence of slots.

CA4 constraints regulate the number of matches a group of teams play at home or away in specific slots against specific teams.

Game Constraints Game constraints enforce or forbid specific matches in certain slots.

GA1 constraints deal with fixed or forbidden matches to slots assignments.

Break Constraints If a team plays a game with the same home-away status as its previous game, we say it has a break.

BR1 constraints limit the breaks a team has in specific slots.

BR2 constraints limit the breaks a group of teams has in specific slots.

Fairness Constraints Fairness constraints attempt to increase fairness and attractiveness of a tournament.

FA2 constraints limit the difference in played home games of set of teams.

Separation Constraints Separation constraints regulate the number of slots between matches involving the same pairs of teams.

SE1 limits the difference between matches and rematches of the same teams.

4 A Pragmatic Approach

4.1 Heuristic Moves

We have identified five different heuristic moves that create a new timetable from an existing one. The new timetable conforms to the base constraints. In Table 1 a timetable is presented, this timetable will be used as a starting timetable for the examples for all available moves.

Table 1. Double round robin tournament created with the circle method.

1	2	3	4	5	6
1-4	1-3	1-2	4-1	3-1	2-1
2-3	4-2	3-4	3-2	2-4	4-3

- SwapHomes. Two teams $t_1 \neq t_2$ are selected, we swap match $t_1 - t_2$ with $t_2 - t_1$. Two matches are affected. An example can be seen in Table 2.

Table 2. Timetable 1 after SwapHomes move for teams 1 and 4.

1	2	3	4	5	6
4-1	1-3	1-2	**1-4**	3-1	2-1
2-3	4-2	3-4	3-2	2-4	4-3

- SwapRounds. Two slots $s_1 \neq s_2$ are selected, we swap the matches of s_1 with those of s_2. For a tournament of T teams, T matches are affected. An example can be seen in Table 3. For tournaments with the Phase rule swapping slots is allowed only on slots of the same phase.
- Swap Teams. Two teams $t_1 \neq t_2$ are selected, we swap team t_1 with t_2 in all matches. For a tournament of T teams, $4(T-1)$ matches are affected. An example can be seen in Table 4.
- PartialSwapTeams. Two teams $t_1 \neq t_2$ are selected, we swap opponents of team t_1 with those of t_2 in all slots and also keep in mind that if a match is already scheduled to also schedule its rematch. For a tournament of T teams, $4(T-1) - 2$ matches are affected. An example can be seen in Table 5.

Table 3. Timetable 1 after SwapRounds move for slots 1 and 5.

1	2	3	4	5	6
3-1	1-3	1-2	4-1	**1-4**	2-1
2-4	4-2	3-4	3-2	**2-3**	4-3

Table 4. Timetable 1 after SwapTeams move for teams 1 and 2.

1	2	3	4	5	6
2-4	**2**-3	**2-1**	4-**2**	3-**2**	**1-2**
1-3	4-**1**	3-4	3-**1**	**1**-4	4-3

– PartialSwapRounds. One match m is selected and moved from slot A to slot B, matches involving teams from m are moved to slot A, an ejection sequence between slots A and B occurs until each team participates in one match per slot. For tournaments with the Phase rule the match m is allowed only to move to a slot in the same phase. The number of matches affected varies. An example can be seen in Table 6.

4.2 Simulated Annealing

Simulated Annealing [21] is a well-known optimization technique that manages to produce near optimal results for a variety of problems. It escapes local minima by accepting inferior solutions with high probability during early stages of the process. This probability diminishes as the process continues. In particular, the acceptance probability of an inferior solution at step k of the procedure is given by $\frac{e^{f(cur)-f(new)}}{T_k}$ where $f(cur)$ is the cost of the current best solution, $f(new)$ is the cost of the new solution and T_k is the temperature after k decreases from an initial temperature T. The temperature is decreased based on the cooling factor a, using the formula $T_k = aT_{k-1}$.

There seems to be some art in calibrating Simulated Annealing to get the best possible results [5]. In our approach, after trial and error the following values were chosen, $T = 5$, $a = 0.999$. The procedure restarts when the temperature of 0.1 is reached.

4.3 Constraint Programming Formulation

Decision Variables For the set of teams $\mathbb{T}$, the set of slots $\mathbb{S}$, with S as the number of available slots and T as the number of teams we define the following binary decision variables.

Table 5. Timetable 1 after PartialSwapTeams move for teams 1 and 3.

1	2	3	4	5	6
1-**2**	1-3	1-**4**	**2**-1	3-1	**4**-1
4-3	4-2	3-**2**	3-**4**	2-4	**2**-3

Table 6. Timetable 1 after placing match 1-2 in slot 5. Note that in small tournaments the effect is always a swap of the slots, but in larger tournaments only some of the matches will exchange slots.

1	2	3	4	5	6
1-4	1-3	**3-1**	4-1	**1-2**	2-1
2-3	4-2	**2-4**	3-2	**3-4**	4-3

$$x_{i,j,s} = \begin{cases} 1, \text{If team i plays against team j in slot s} \\ 0, \quad \text{Otherwise} \end{cases} \quad \forall i,j \in \mathbb{T}, i \neq j, \forall s \in \mathbb{S} \tag{1}$$

To monitor the home away pattern we define:

$$y_{i,s} = \begin{cases} 1, \text{If team i plays at home in slot s} \\ 0, \quad \text{Otherwise} \end{cases} \quad \forall i \in \mathbb{T}, \forall s \in \mathbb{S} \tag{2}$$

We enforce the home-away pattern to follow the timetable:

$$y_{i,s} = \sum_{j=1}^{T} x_{i,j,s} \quad \forall i \in \mathbb{T}, i \neq j, \forall s \in \mathbb{S} \tag{3}$$

In all instances, constraints regarding breaks do not take into consideration if the breaks occur at Home or Away, so we just have to keep track in which slots a general break occurs:

$$z_{i,s} = \begin{cases} 1, \text{If team i has a break in slot s} \\ 0, \quad \text{Otherwise} \end{cases} \quad \forall i \in \mathbb{T}, \forall s \in \mathbb{S} \tag{4}$$

We enforce the break pattern to follow the home-away pattern:

$$z_{i,s} = \begin{cases} 1, y_{i,s} = y_{i,s-1}, s > 1 \\ 0, \quad s = 1 \end{cases} \quad \forall i \in \mathbb{T}, \forall s \in \mathbb{S} \tag{5}$$

Base Constraints Each team must play exactly one match at home against each other team:

$$\sum_{s=1}^{S} x_{i,j,s} = 1 \quad \forall i,j \in \mathbb{T}, i \neq j \tag{6}$$

To satisfy the compactness rule each team plays one match in each slot:

$$\sum_{j=1}^{T}(x_{i,j,s} + x_{j,i,s}) = 1 \quad \forall i \in \mathbb{T}, i \neq j, \forall s \in \mathbb{S} \tag{7}$$

For instances with the phase rule a match and its rematch must be in different phases:

$$\sum_{s}^{S/2}(x_{i,j,s} + x_{j,i,s}) = 1 \quad \forall i, j \in \mathbb{T}, i < j, \forall s \in \mathbb{S} \tag{8}$$

CA1 Constraints Each CA1 constraint with team t_c in "teams" field, with $\mathbb{S}_c$ the set of teams in "slots" field and max_c in "max" field, triggers a d_c deviation.

CA1 with mode="H":

$$d_c = \sum_{s\in\mathbb{S}_c} y_{t_c,s} - max_c \tag{9}$$

CA1 with mode="A" and S_c the size of $\mathbb{S}_c$:

$$d_c = S_c - \sum_{s\in\mathbb{S}_c} y_{t_c,s} - max_c \tag{10}$$

CA2 Constraints Each CA2 with team t_1 in "teams1" field, with $\mathbb{S}_c$ the set of slot in "slots" field, with $\mathbb{T}_c$ the set of slots in "teams2" field, with max_c in "max" field triggers a deviation d_c.

CA2 with mode="H":

$$d_c = \sum_{t_2\in\mathbb{T}_c} \sum_{s\in\mathbb{S}_c} x_{t_1,t_2,s} - max_c \tag{11}$$

CA2 with mode="A":

$$d_c = \sum_{t_2\in\mathbb{T}_c} \sum_{s\in\mathbb{S}_c} x_{t_2,t_1,s} - max_c \tag{12}$$

CA2 with mode="HA":

$$d_c = \sum_{t_2\in\mathbb{T}_c} \sum_{s\in\mathbb{S}_c} (x_{t_1,t_2,s} + x_{t_2,t_1,s}) - max_c \tag{13}$$

CA3 Constraints Each CA3 with $\mathbb{T}_{c1}$ the set of teams in "teams1" field, with $\mathbb{S}_c$ as the slots in "slots" field, with $\mathbb{T}_{c2}$ the set of teams in "teams2" field and max_c in "max" field triggers deviations d_c for each team in $\mathbb{T}_{c1}$ and for all slot sequences $\mathbb{S}_c$ of size $intp$ in "intp" field.

CA3 with mode="H":

$$d_c = \sum_{t_2\in\mathbb{T}_{c2}} \sum_{s=k}^{k+intp} x_{t_1,t_2,s} - max_c \quad \forall t_1 \in \mathbb{T}_{c1}, t_1 \neq t_2, 1 \leq k \leq S_c - intp \tag{14}$$

CA3 with mode="A":

$$d_c = \sum_{t_2 \in \mathbb{T}_{c2}} \sum_{s=k}^{k+intp} x_{t_2,t_1,s} - max_c \quad \forall t_1 \in \mathbb{T}_{c1}, t_1 \neq t_2, 1 \leq k \leq S_c - intp \tag{15}$$

Special case: In all instances there are at most two Hard CA3 constraints, one with mode="H" and the other with mode="A", $\mathbb{T}_{c1} = \mathbb{T}_{c2} = \mathbb{T}$, $\mathbb{S}_c = \mathbb{S}$, max_c is always 2 and $intp$ is always 3. If both rules exist then the home-away patterns "HHH" and "AAA" cannot appear, so for those instances a team cannot have two breaks in a row:

$$z_{i,s} + z_{i,s-1} \leq 1 \quad \forall i \in \mathbb{T}, \forall s \in \mathbb{S}, s > 2 \tag{16}$$

CA4 Constraints Each CA4 with mode2="GLOBAL" triggers a deviation d_c equal to the sum of the matches between the set of teams $\mathbb{T}_{c1}$ in "teams1" field and the set of teams $\mathbb{T}_{c2}$ in "teams2" field in all slots of the set $\mathbb{S}_c$ in "slots" field over max_c in "max" field.

CA4 with mode2="GLOBAL" and mode1="H":

$$d_c = \sum_{s \in \mathbb{S}_c} \sum_{t_1 \in \mathbb{T}_{c1}} \sum_{t_2 \in \mathbb{T}_{c2}} x_{t_1,t_2,s} - max_c \quad t_1 \neq t_2 \tag{17}$$

CA4 with mode2="GLOBAL" and mode1="A":

$$d_c = \sum_{s \in \mathbb{S}_c} \sum_{t_1 \in \mathbb{T}_{c1}} \sum_{t_2 \in \mathbb{T}_{c2}} x_{t_2,t_1,s} - max_c \quad t_1 \neq t_2 \tag{18}$$

CA4 with mode2="GLOBAL" and mode1="HA":

$$d_c = \sum_{s \in \mathbb{S}_c} \sum_{t_1 \in \mathbb{T}_{c1}} \sum_{t_2 \in \mathbb{T}_{c2}} (x_{t_1,t_2,s} + x_{t_2,t_1,s}) - max_c \quad t_1 \neq t_2 \tag{19}$$

Each CA4 with mode2="EVERY" triggers a deviation d_c for each slot of the slots set $\mathbb{S}_c$ in "slots" field equal to the sum of the matches between the set of teams $\mathbb{T}_{c1}$ in "teams1" field and the set of teams $\mathbb{T}_{c2}$ in "teams2" field over max_c in "max" field.

CA4 with mode2="EVERY" and mode1="H":

$$d_c = \sum_{t_1 \in \mathbb{T}_{c1}} \sum_{t_2 \in \mathbb{T}_{c2}} x_{t_1,t_2,s} - max_c \quad t_1 \neq t_2, \forall s \in \mathbb{S}_c \tag{20}$$

CA4 with mode2="EVERY" and mode1="A":

$$d_c = \sum_{t_1 \in \mathbb{T}_{c1}} \sum_{t_2 \in \mathbb{T}_{c2}} x_{t_2,t_1,s} - max_c \quad t_1 \neq t_2, \forall s \in \mathbb{S}_c \tag{21}$$

CA4 with mode2="EVERY" and mode1="HA":

$$d_c = \sum_{t_1 \in \mathbb{T}_{c1}} \sum_{t_2 \in \mathbb{T}_{c2}} (x_{t_1,t_2,s} + x_{t_2,t_1,s}) - max_c \quad t_1 \neq t_2, \forall s \in \mathbb{S}_c \tag{22}$$

GA1 Constraints Each GA1 triggers a deviation d_c calculated as the sum of matches of the set $\mathbb{M}_c$ in field "meetings" which occur in set of slots S_c in field "slots" under min_c in field "min" or over max_c in field "max".

$$d_c = \sum_{s \in \mathbb{S}_c} \sum_{t_1, t_2 \in \mathbb{T}_c} x_{t_1,t_2,s} - max_c \tag{23}$$

$$d_c = \sum_{s \in \mathbb{S}_c} \sum_{t_1, t_2 \in \mathbb{T}_c} x_{t_1,t_2,s} + min_c \tag{24}$$

BR1 Constraints Each BR1 with t_c the team in "teams" field, triggers a deviation d_c equal to the sum of teams t_c breaks in set of slots $\mathbb{S}_c$ in "slots" field over max_c in "max" field.

$$d_c = \sum_{s \in \mathbb{S}_c} z_{t_c,s} - max_c \tag{25}$$

BR2 Constraints In all instances where a BR2 constraint exists field "teams" contains all teams and field "slots" contains all slots except the first slot (as a team cannot have a break in the first slot). As such, a BR2 constraint triggers a deviation d_c equal to the sum of all breaks of all teams over max in "max" field.

$$d_c = \sum_{t \in \mathbb{T}} \sum_{s \in \mathbb{S}} z_{t,s} - max_c \tag{26}$$

FA2 Constraints In all instances where an FA2 constraint exists field "teams" contains all teams and field "slots" contains all slots. As such, an FA2 constraint triggers deviations d_c for each pair of teams equal to the largest difference in played home games over all slots more than $intp$ in "intp" field.

$$d_c = \underset{s \in \mathbb{S}}{max} (\sum_{m=1}^{s} y_{i,m} - \sum_{m=1}^{s} y_{j,m} - intp; 0) \quad \forall i, j \in \mathbb{T}, i < j \tag{27}$$

SE1 Constraints For SE1 we need to keep track of the distance between matches and rematches for all combinations of the set of teams $\mathbb{T}_c$ in field "teams". Each combination of teams triggers a deviation d_c equal to the sum of the number of time slots less than min in "min" field between the match and the rematch.

$$d_c = \left| \sum_{s \in \mathbb{S}} s * x_{t_1,t_2,s} - \sum_{s \in \mathbb{S}} s * x_{t_2,t_1,s} \right| - min_c \quad t1 \neq t2, \forall t_1, t_2 \in \mathbb{T}_c \tag{28}$$

Objective Function Hard constraints must not generate any deviation. Soft constraints' deviations are multiplied by p_c denoted by the field "penalty" and summed. Deviations under zero are ignored.

$$min \quad \sum_{c \in \mathbb{C}} d_c * p_c \tag{29}$$

Employing the CP/SAT Solver of ORTools The overly constrained nature of sports scheduling made it difficult for traditional Constraint Programming solvers to even reach a feasible solution let alone a good one. In our approach we used the ORTools [15] CP/SAT solver that allowed us to formulate the problem using CP terms. The distinctiveness of the CP/SAT solver is that it reformulates internally the CP model into a SAT (satisfiability) model that seems to be better adapted to the nature of the sports scheduling problem.

4.4 A Hybrid Approach

An initial solution satisfying the base constraints is constructed using CP/SAT and Hard constraints are perceived as soft and Soft constraints are ignored. The Simulated Annealing process tries to bring the solution in the feasible space. If a feasible solution is found then Hard constraints become mandatory and Soft constraints are activated. Each time the simulated annealing process terminates an improvement process using CP/SAT attempts to improve the current solution. In order to achieve this we randomly select a number of teams, or a number of slots, or a number of games, or some combination of the above and keep them fixed while the rest of the current solution is allowed to change. In Figure 1 a flowchart of the process is presented.

5 Experimental Results

5.1 Datasets

The problem instances of ITC2021[4] are formatted with the RobinX XML data format [20]. The instances were released in three phases (Early, Middle, Late) and each set contains 15 instances. All instances are in double round robin format of 16, 18 or 20 teams. Some instances contain the Phase rule. Not all constraints make an appearance in every tournament.

5.2 Results

The hybrid process was able to produce solutions for 37 out of 45 instances. The objective of the solutions can be seen in Table 7. Solution files are available at our github[5] repository.

6 Conclusions

Sports scheduling has several facets that make it an interesting and difficult problem. Sports scheduling problems are proved to be, in practice (and in theory), hard to solve. Sometimes even finding a feasible solution or proving that

[4] https://www.sportscheduling.ugent.be
[5] https://bit.ly/3wtrW4i

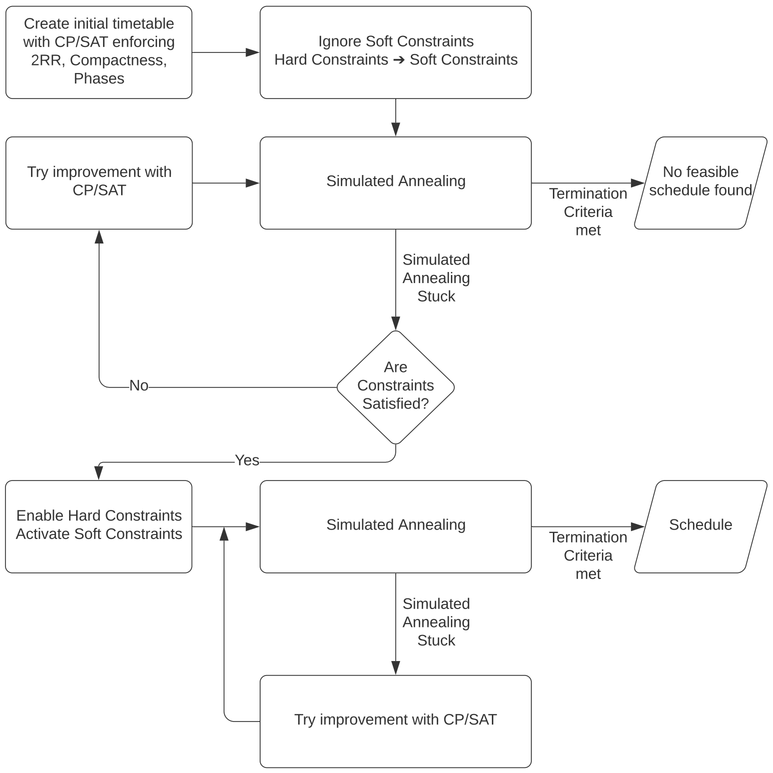

Create initial timetable with CP/SAT enforcing 2RR, Compactness, Phases
Ignore Soft Constraints Hard Constraints ➔ Soft Constraints
Try improvement with CP/SAT
Simulated Annealing
Termination Criteria met
No feasible schedule found
Simulated Annealing Stuck
Are Constraints Satisfied?
No
Yes
Enable Hard Constraints Activate Soft Constraints
Simulated Annealing
Termination Criteria met
Schedule
Simulated Annealing Stuck
Try improvement with CP/SAT

Fig. 1. Flowchart of the hybrid process.

such a solution does not exist is extremely challenging. We had the opportunity to assert this during our participation in the ITC2021 competition.

In this paper, an approach to solving the problem was presented that involved modeling of the problem using Constraint Programming. A Simulated Annealing solver employing small and large moves was implemented. Small moves, are often inspired by the perspective as a graph of the problem, make schedule changes that keep the schedule feasible, but are rather local. Large moves fix randomly selected teams, matches or slots and let the other ones free to move. We managed to receive good results and we firmly believe that our approach can be even more successful in addressing sports scheduling problems by using more processing power and more sophisticated strategies than the random selection for fixed objects.

Acknowledgements We acknowledge support of this work by the project "Dioni: Computing Infrastructure for Big-Data Processing and Analysis." (MIS

Table 7. Results after three hours of execution time for each instance using the hybrid process. Objective is presented as the tuple (deviation of hard constraints, penalty of soft constraints).

Instance	Objective	Instance	Objective	Instance	Objective
Early 1	0, 512	Middle 1	17, -	Late 1	0, 2234
Early 2	0, 266	Middle 2	48, -	Late 2	0, 5680
Early 3	0, 1354	Middle 3	0, 12170	Late 3	0, 3004
Early 4	6, -	Middle 4	0, 7	Late 4	0, 0
Early 5	5, -	Middle 5	0, 732	Late 5	39, -
Early 6	0, 3957	Middle 6	0, 1900	Late 6	0, 1440
Early 7	0, 9644	Middle 7	0, 2792	Late 7	0, 3009
Early 8	0, 1614	Middle 8	0, 301	Late 8	0, 1375
Early 9	0, 448	Middle 9	0, 1015	Late 9	0, 1108
Early 10	32, -	Middle 10	1, -	Late 10	6, -
Early 11	0, 8189	Middle 11	0, 2956	Late 11	0, 511
Early 12	0, 1025	Middle 12	0, 1596	Late 12	0, 7218
Early 13	0, 380	Middle 13	0, 780	Late 13	0, 3576
Early 14	0, 63	Middle 14	0, 1619	Late 14	0, 1650
Early 15	0, 4470	Middle 15	0, 1833	Late 15	0, 80

No. 5047222) which is implemented under the Action "Reinforcement of the Research and Innovation Infrastructure", funded by the Operational Programme "Competitiveness, Entrepreneurship and Innovation" (NSRF 2014-2020) and co-financed by Greece and the European Union (European Regional Development Fund).

References

1. Alarcón, F., Durán, G., Guajardo, M., Miranda, J., Muñoz, H., Ramírez, L., Ramírez, M., Sauré, D., Siebert, M., Souyris, S., Andrés, W., Rodrigo, W.Y., Gonzalo, Z.: Operations research transforms the scheduling of Chilean soccer leagues and South American world cup qualifiers. Interfaces **47**(1), 52–69 (2017)
2. Briskorn, D.: Feasibility of home–away-pattern sets for round robin tournaments. Operations Research Letters **36**(3), 283–284 (2008)
3. Chen, J., Dong, D.: Research on the general method of round robin scheduling. In: Advances in Multimedia, Software Engineering and Computing Vol. 2, pp. 393–399. Springer (2011)
4. Costa, F.N., Urrutia, S., Ribeiro, C.C.: An ILS heuristic for the traveling tournament problem with predefined venues. Annals of Operations Research **194**(1), 137–150 (2012)
5. Delahaye, D., Chaimatanan, S., Mongeau, M.: Simulated annealing: From basics to applications. In: Handbook of metaheuristics, pp. 1–35. Springer (2019)
6. Durán, G., Guajardo, M., Sauré, D.: Scheduling the South American Qualifiers to the 2018 FIFA World Cup by Integer Programming. European Journal of Operational Research **262**(3), 1109–1115 (2017)

7. Goossens, D., Spieksma, F.: Scheduling the Belgian soccer league. Interfaces **39**(2), 109–118 (2009)
8. Gunnarsdóttir, E.L.: An integer programming formulation for scheduling of the Icelandic football league. Ph.D. thesis, Reykjavík University (2019)
9. Januario, T., Urrutia, S.: A new neighborhood structure for round robin scheduling problems. Computers & Operations Research **70**, 127–139 (2016)
10. Januario, T., Urrutia, S., Ribeiro, C.C., De Werra, D.: Edge coloring: A natural model for sports scheduling. European Journal of Operational Research **254**(1), 1–8 (2016)
11. Lambrechts, E., Ficker, A., Goossens, D.R., Spieksma, F.C.: Round-robin tournaments generated by the circle method have maximum carry-over. Mathematical Programming **172**(1), 277–302 (2018)
12. Lewis, R., Thompson, J.: On the application of graph colouring techniques in round-robin sports scheduling. Computers & Operations Research **38**(1), 190–204 (2011)
13. Miyashiro, R., Matsui, T.: Round-robin tournaments with a small number of breaks. Department of Mathematical Informatics, The University of Tokyo, Mathematical Engineering Technical Reports METR **29**, 2003 (2003)
14. Nurmi, K., Goossens, D., Kyngäs, J.: Scheduling a triple round robin tournament with minitournaments for the Finnish national youth ice hockey league. Journal of the Operational Research Society **65**(11), 1770–1779 (2014)
15. Perron, L., Furnon, V.: OR-tools, https://developers.google.com/optimization/
16. Ribeiro, C.C.: Sports scheduling: Problems and applications. International Transactions in Operational Research **19**(1-2), 201–226 (2012)
17. Ribeiro, C.C., Urrutia, S.: Scheduling the Brazilian soccer tournament: Solution approach and practice. Interfaces **42**(3), 260–272 (2012)
18. Trick, M.A.: A schedule-then-break approach to sports timetabling. In: International conference on the practice and theory of automated timetabling. pp. 242–253. Springer (2000)
19. Van Bulck, D., Goossens, D., Belien, J., Davari, M.: The fifth international timetabling competition (ITC 2021): Sports timetabling. In: MathSport International 2021. pp. 117–122. University of Reading (2021)
20. 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)
21. Van Laarhoven, P., Aarts, E.: Simulated annealing: theory and applications. Dordrecht. Reidel Pub. Comp., Netherlands (1987)

Integer Programming Formulations for Compact Single Round Robin Tournaments

Jasper van Doornmalen[1][0000−0002−2494−0705], Christopher Hojny[1][0000−0002−5324−8996], Roel Lambers[1][0000−0002−0314−6094], and Frits Spieksma[1][0000−0002−2547−3782]

Eindhoven University of Technology Combinatorial Optimization Group
PO Box 513, 5600 MB Eindhoven, the Netherlands
{m.j.v.doornmalen,c.hojny,r.lambers,f.c.r.spieksma}@tue.nl

Abstract. We consider the problem of finding an optimal schedule for compact single round robin tournaments. To this end, we discuss one polynomial size and two exponential size integer programming formulations of this problem. We compare the strength of the linear programming relaxations of these three models. Moreover, we show that the pricing problems of both exponential size formulations can be solved in polynomial time.

Keywords: Sport scheduling · Round Robin · Mixed Integer Programming · Branch and Price

Integer programming continues to be a very popular way to obtain a schedule for a round robin tournament. It does not only allow to automatically generate schedules, but also to easily incorporate different kinds of constraints to find a schedule addressing needs of a specific tournament. To substantiate this claim of widespread use of integer programming, it is a fact that the literature contains lots of papers demonstrating the use of integer programming for finding schedules in sports timetabling. Well-known surveys are by Rasmussen and Trick [10] and Kendall et al. [8]. Complexity results regarding round robin tournaments are provided by Easton [6], Briskorn et al. [3], and Van Bulck and Goossens [12]. Integer programming formulations have also been studied, among others, by Trick [11] and Briskorn and Drexl [2]. More recently, the international timetabling competition [4] featured a round robin sports timetabling problem, and most of the submissions used integer programming in some way to obtain a good schedule. Other recent contributions using integer programming for sports timetabling include Durán et al. [5] and Bouzarth et al. [1]. For more papers in the field of sports scheduling, we refer to Knust [9], who maintains an elaborate classification of literature on sports scheduling.

We aim to take a fresh look at the problem of finding an optimal schedule for compact single round robin tournaments using integer programming techniques. Given a set of n teams, a *compact single round robin tournament* consists of $n-1$ rounds of matches such that each team plays against exactly one other team per round and each pair of opponents meets exactly once. The FIFA World Cup group stage is an example of a compact single round robin tournament. This type is also common for chess tournaments, for instance for all important tournaments between London 1862 and Curaçao Candidates tournaments of 1962 [7]. In the following, we assume that each hypothetical match at a specific round has an associated cost. Our goal is then to find a schedule that minimizes total cost. By finding cost-minimizing optimal schedules for compact single round robin tournaments, one can take practical considerations and wishes into account.

We discuss three different integer programming formulations for finding an optimal schedule. First, we consider a *traditional formulation* using $O(n^3)$ many variables and constraints that is also studied by Trick [11] and Briskorn and Drexl [2]. This model introduces, for every hypothetical match between two teams and round, a binary decision variable that indicates whether this match is scheduled on that round. We compare this formulation with two alternative novel formulations that are based on different encodings of a schedule: a matching and a permutation formulation.

The *matching formulation* introduces a binary variable for each pair (M, r), where M is a perfect matching of the n teams and r is the index of a round. These variables encode the entire schedule of round r. Instead of fixing the schedule of a round, the *permutation formulation* fixes, for a given team, the order of matches that it plays against all other teams. That is, it introduces a variable for each team t and each permutation of $\{1, \dots, n\} \setminus \{t\}$. Note that, in contrast to the traditional formulation, both the matching and permutation formulation have exponentially many variables.

Our main contributions are twofold:

1. Despite the exponential number of variables, we show that the linear programming relaxations of both the matching and permutation formulation can be solved in polynomial time. To this end, we show that the pricing problems of both formulations reduce to finding perfect matchings in suitably defined auxiliary graphs.
2. We provide a comparison of the optimal values of linear programming relaxations of the three different models. We show that the traditional and permutation formulation both provide the same optimal value of their linear programming relaxations, whereas the matching formulation is at least as strong as the other two formulations. In particular, we show that it can be strictly stronger.

Moreover, we discuss how the matching and permutation formulation can be solved within a branch-and-price framework.

References

1. Bouzarth, E.L., Grannan, B.C., Harris, J.M., Hutson, K.R.: Scheduling the valley baseball league. INFORMS Journal on Applied Analytics (2021). https://doi.org/10.1287/inte.2021.1076
2. Briskorn, D., Drexl, A.: IP models for round robin tournaments. Computers & Operations Research **36**(3), 837–852 (2009)

3. Briskorn, D., Drexl, A., Spieksma, F.C.R.: Round robin tournaments and three index assignments. 4OR **8**(4), 365–374 (2010)
4. van Bulck, D., Goossens, D., Beliën, J., Davari, M.: The fifth international timetabling competition (ITC 2021): Sports timetabling. In: Proceedings of MathSport International 2021 Conference, MathSport. pp. 117–122 (2021)
5. Durán, G., Guajardo, M., Gutiérrez, F., Marenco, J., Sauré, D., Zamorano, G.: Scheduling the main professional football league of Argentina. INFORMS Journal on Applied Analytics **51**(5), 361–372 (2021)
6. Easton, K.K.: Using integer programming and constraint programming to solve sports scheduling problems. Ph.D. thesis, Georgia Institute of Technology (2003)
7. Hooper, D., Whyld, K.: The Oxford Companion to Chess. Oxford Companions Series, Oxford University Press (1984)
8. Kendall, G., Knust, S., Ribeiro, C.C., Urrutia, S.: Scheduling in sports: An annotated bibliography. Computers & Operations Research **37**(1), 1–19 (2010)
9. Knust, S.: Classification of literature on sports scheduling. http://www.inf.uos.de/knust/sportssched/sportlit_class/, accessed: March 2022
10. Rasmussen, R.V., Trick, M.A.: Round robin scheduling–a survey. European Journal of Operational Research **188**(3), 617–636 (2008)
11. Trick, M.A.: Integer and constraint programming approaches for round-robin tournament scheduling. In: International Conference on the Practice and Theory of Automated Timetabling. pp. 63–77. Springer (2002)
12. Van Bulck, D., Goossens, D.: On the complexity of pattern feasibility problems in time-relaxed sports timetabling. Operations Research Letters **48**(4), 452–459 (2020)

Timetabling Research: A Progress Report

Jeffrey H. Kingston

School of Information Technologies, The University of Sydney, Australia
jeff@it.usyd.edu.au
http://jeffreykingston.id.au

Abstract. As the PATAT conference series passes its 25th year, this paper describes how the discipline of automated timetabling has changed in that time. It examines the sub-disciplines studied and the solvers used, and considers the effect of data sets, data formats, and competitions. The paper concludes by asking whether insight into the timetabling problem has deepened since 1995, and where the discipline should go from here.

Keywords: Automated Timetabling · History · Future

1 Introduction

As the PATAT conference series [13] passes its 25th year, this paper examines how the discipline of automated timetabling has changed since 1995, when the first PATAT conference was held.

Section 2 measures how timetabling's sub-disciplines (course timetabling, nurse rostering, and so on) have changed, and how its solvers have developed. Section 3 discusses progress within the sub-disciplines. Section 4 asks whether insight into timetabling has deepened, and Section 5 discusses the goals of our discipline and where it should go from here.

2 Progress since 1995

The first PATAT conference was held in 1995 [13]. Before then, although some significant work had been done, there was no forum devoted to automated timetabling, and the field was very fragmented [17]. From the start, PATAT was international in outlook and welcoming of any interesting contribution, and it immediately became the centre of the discipline, as it is today.

This section examines how automated timetabling has changed since 1995. To do this objectively, the author has classified the papers from three pairs of PATAT conferences. The chosen conferences were the first two (1995 and 1997), with 91 papers in total; the middle two (2006 and 2008), with 156 papers; and the most recent two (2016 and 2018), with 114 papers.

Each paper has been classified by sub-discipline, by kind (explained below), and by solver method. All papers in the proceedings of the chosen conferences have been included (plenary papers, full papers, and extended abstracts, as well as system demonstrations), and given equal weight.

Of course, the PATAT proceedings contain only a subset of the literature. But there is no reason to believe that they are unrepresentative: PATAT has always been open to any kind of timetabling paper.

Figure 1 shows how the relative number of papers from each sub-discipline has changed over time. In general the space given to the various sub-disciplines has become more balanced, except that high school timetabling has virtually disappeared for the moment. Personnel scheduling (excluding nurse rostering) covers many problems, including physician scheduling, call centers, and so on, so its growth is a healthy development.

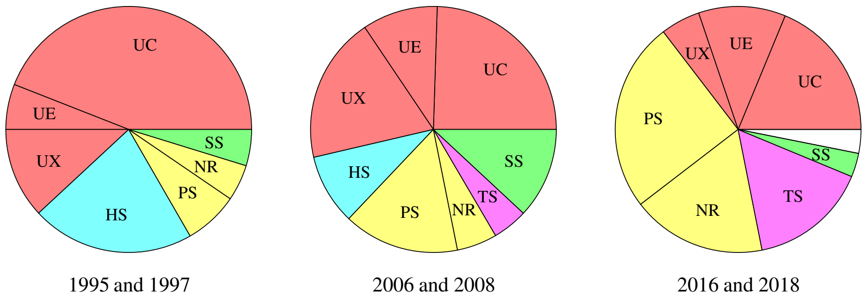


Fig. 1. The relative number of papers from each sub-discipline, over three pairs of PATAT conferences: 1995 and 1997; 2006 and 2008; and 2016 and 2018. The sub-disciplines are: university curriculum-based course timetabling (UC); university post-enrolment course timetabling (UE); university examination timetabling (UX); high school timetabling (HS); personnel scheduling excluding nurse rostering (PS); nurse rostering (NR); transport scheduling (TS); sports scheduling (SS); white means other. Only papers that study specific sub-disciplines are included. Those few papers that study several sub-disciplines are counted once for each sub-discipline.

For our next figure we need to define two kinds of papers.

A *case study paper* defines some problem, presents one or a few instances of that problem, and solves those instances. Case study papers are valuable for uncovering new sub-disciplines and new requirements within sub-disciplines. The solving in case study papers is usually less valuable, because it is done on new instances, and so is hard to evaluate objectively.

A *solver paper* takes a previously defined problem and presents one or more solvers for it. It compares them with previous solvers by testing them on standard data sets. (In this paper, a *data set* is a set of instances of a timetabling problem, stored together in a common format.) Solver papers are important for establishing objective standards of performance, helping to make automated timetabling into a truly scientific discipline [16].

Figure 2 shows how the relative number of case study and solver papers has changed. These two kinds cover all papers that solve instances, since such papers must either introduce their own instances or take them from elsewhere. In 1995–

97, with just one pioneering exception, all papers that solved instances were case studies. But now the two kinds are equally common.

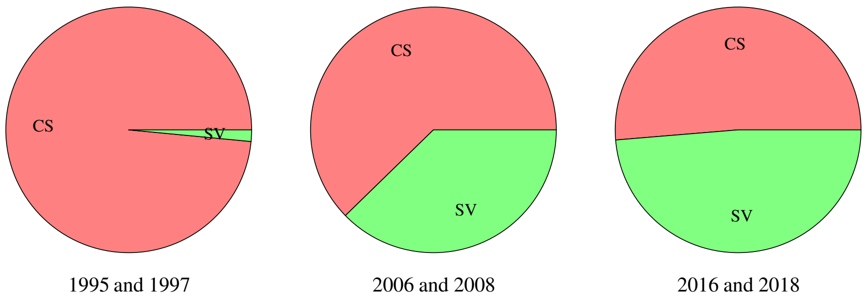


Fig. 2. The relative number of case study papers (CS) and solver papers (SV), over three pairs of PATAT conferences: 1995 and 1997; 2006 and 2008; and 2016 and 2018. Only papers that solve instances are included.

Figure 3 shows how the relative number of papers devoted to each type of solver has changed. The growth in integer programming is very clear, and has come at the expense of genetic algorithms, tabu search, and constraint programming. Integer programming is also frequently used in VLSN search, to optimally reassign the unassigned variables.

3 Progress within sub-disciplines

At any given moment, different sub-disciplines will be at different stages of development. We distinguish four stages here; their boundaries are not sharp.

A *Stage 1 sub-discipline* is one which can be met with in the literature, but only in a few case study papers. Its scope is far from clear.

A *Stage 2 sub-discipline* is one which is often met with in the literature, again in case study papers. Its scope is fairly clear.

A *Stage 3 sub-discipline* is also often met with in the literature. Apart from minor issues, its scope is clear, and expressed in standard data sets.

A *Stage 4 sub-discipline* is one whose research agenda has been exhausted. Activity declines, and there is no feeling of progress being made.

What constitutes progress in a sub-discipline depends on its stage. A Stage 1 sub-discipline needs case studies which help to elucidate its scope. A Stage 2 sub-discipline may need more case studies, or it may need to transition to Stage 3. What constitutes progress in Stage 3 sub-disciplines will be considered in Section 5; it includes improving the quality of solutions to near-optimality, and ensuring that data sets are real-world.

Major progress occurs when a sub-discipline moves from one stage to the next. Moving from Stage 1 to Stage 2 is relatively easy; all it takes is for interest to be sufficient to stimulate a number of case studies. Moving from Stage 2 to Stage 3

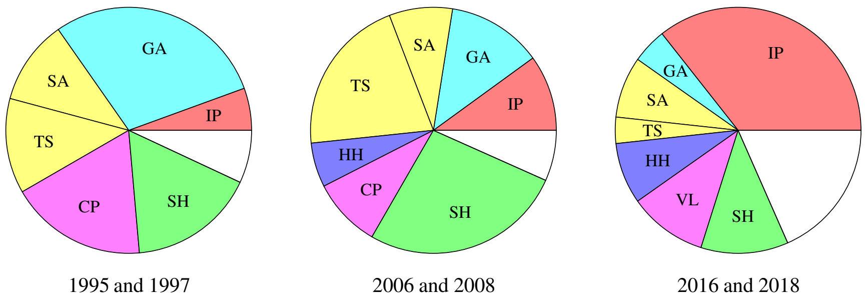


Fig. 3. The relative number of papers for each solver type, over three pairs of PATAT conferences: 1995 and 1997; 2006 and 2008; and 2016 and 2018. The solver types are: integer programming (IP); genetic and other evolutionary algorithms (GA); simulated annealing (SA); tabu search (TS); hyper-heuristics (HH); constraint programming and logic progamming (CP); VLSN search (VL); simple heuristics (SH); and white means other (many types, e.g. satisfiability solvers, dynamic programming, and flows and matchings). Only papers that solve instances are included. Papers that use several solver types are counted once for each type, except that simple heuristics are not counted when other solver types are used.

is harder, because it requires agreement on the scope of the sub-discipline, and the expression of that agreement in standard data sets. In practice, this difficult transition has usually been driven by competitions.

Let us consider now the stages reached by the various sub-disciplines.

University course timetabling is a clear Stage 3 sub-discipline, with three competitions to its credit, including the first ever timetabling competition [13], organized by Ben Paechter in 2003. The most recent competition, ITC 2019 [5], was organized by leading practitioners, and its data format is a step forward which brings this sub-discipline very close to the real world.

University examination timetabling boasts the first ever standard data set (the Toronto data set [18], assembled by Mike Carter ca. 1997). The Toronto instances now have very good solutions that are unlikely to be significantly improved on. Until recently university examination timetabling appeared to be the closest thing in timetabling to a Stage 4 sub-discipline. However, recently there has been a resurgence of interest, including new and more real-world models.

High school timetabling transitioned to Stage 3 about ten years ago, driven as usual by a competition. It was active for some years after that, but recently the number of committed researchers seems to have declined (Figure 1).

Personnel scheduling (excluding nurse rostering) is a Stage 2 sub-discipline whose transition to Stage 3 is arguably overdue. It encompasses many different problems, whose interrelationships remain to be elucidated.

Nurse rostering is a Stage 3 sub-discipline, with two competitions and at least four standard data sets. The most recent competition [1, 2] focused on how a nurse roster for one week interacts with the rosters for preceding and following weeks, taking a big step towards modelling the real world.

Transport scheduling is at Stage 2. There have been transport scheduling papers for decades, and there are well-established problems, such as vehicle routing and air-crew scheduling; but judging from the PATAT offerings the sub-discipline is fragmented over many problems and is not ready for Stage 3.

This author does not know whether there are other forums for presenting research on transport scheduling. There are many conferences devoted to many aspects of transportation [19], but examination of one recent vehicle routing paper [11] and one recent air-crew scheduling paper [6] revealed an extensive journal literature but no conferences and no evidence of data exchange. A vehicle routing competition (using generated data) was held recently [9].

Sports scheduling is also at Stage 2. The *travelling tournament problem*, a simplified problem, was formulated two decades ago [4]. A real-world data format, RobinX [20], has appeared recently, and a competition using RobinX is underway. Whether this will drive a transition to Stage 3 remains to be seen; a critical mass of committed researchers will be needed.

4 Insight into the timetabling problem

One would like to think that recent papers show more insight into automated timetabling than older papers. But what does that mean? And is it true?

Timetabling has several aspects for which insight would be desirable. The fundamental one must surely be how best to solve the problems. The NP-hardness of real-world timetabling problems has been known since well before 1995. It prevents the kind of deep insight that a polynomal-time solver would give proof of. Over the years attempts have been made to match problem types with solver types, but they have never produced anything that could be called an established body of theory. Questions such as why one simulated annealing cooling schedule should be better than another, or why one tabu list length should be better than another, have not been answered.

A less intractable aspect is specification: insight into what timetabling is. For example, the new sports scheduling format [20] could be said to offer insight into that sub-discipline. The scope of the timetabling problem is clearer to the researcher of today than it was to the attendees at the first PATAT conference in 1995, where a seminar (not documented in the proceedings) addressing the specification issue ended with nothing resolved.

One particular point that has become clear is that real-world specification is not hopelessly open-ended. Those who take on the hard work of collecting constraints do eventually reach the end of them, even when they work across multiple institutions. The researcher of 1995 did not know this.

If the specifications of the various sub-disciplines could be unified into one specification that was significantly smaller than the sum of the specifications of the separate problems, then that could be considered a step forward in insight. At present all that can be said is that all timetabling problems have events containing times and resources, some preassigned, and some left open for a solver to assign, subject to constraints. But even that may not be true of transport scheduling, and bringing together the disparate constraints found in different sub-disciplines might well produce nothing but chaos.

Another way to approach the insight question is simply to look through the literature for results that seem insightful. One such is the realization that curriculum-based university course timetabling and high school timetabling are closely related [12]. This author has published a method of specifying minimal perturbation problems that works for any timetabling problem and any kind of perturbation [7]. But results of this kind are few and scattered. Insight has deepened, but only very slowly.

5 Moving forward

Looking back across the decades, there does seem to be an element of fashion in the choice of solvers. For example, genetic algorithms were very popular during the early PATAT years, but have declined since. One wonders which kinds of solvers will survive the next 25 years. Will integer programming continue to grow, or will its undoubted recent gains plateau off, and its lack of robustness as instance size increases become increasingly seen as a liability?

The author considers such questions to be futile: most forecasts turn out to be wrong. Instead, this section examines the papers being written today, and asks which of them are moving the field forward. Although the answer will be subjective, any honest appraisal of our discipline must address this question.

First, we need to agree on the direction in which we should be moving. Inevitably, that is a matter of opinion. In the author's opinion, then, our discipline is a practical one that has always had one simple goal:

> *Automated timetabling seeks to help people find high-quality timetables quickly and reliably wherever they are needed.*

If this is accepted, then anything that helps to remove any significant obstacle to its achievement is forward progress.

Case study papers, which introduce a problem and solve it on new data, are generally forward-looking in Stage 1 and Stage 2 sub-disciplines, although their value decreases as their number increases. Case study papers in Stage 3 sub-disciplines are unlikely to offer anything new: they are backward-looking.

Solver papers, which introduce solvers and apply them to existing data, are characteristic of Stage 3 sub-disciplines. They are essential to the scientific advance of our discipline. But they suffer from diminishing returns: they are all about finding better solutions, but that becomes harder and harder as time passes. Some data sets have now been solved to optimality, or so close to it that *significant* further improvement is impossible (Figure 4). So we regard solver papers in sub-disciplines that reached Stage 3 some years ago as backward-looking, except when the instances they solve become more real-world, as in the recent nurse rostering [1, 2] and university course timetabling [5] competitions.

A classification of the papers from the two most recent PATAT conferences into forward-looking and backward-looking, based on these ideas, appears

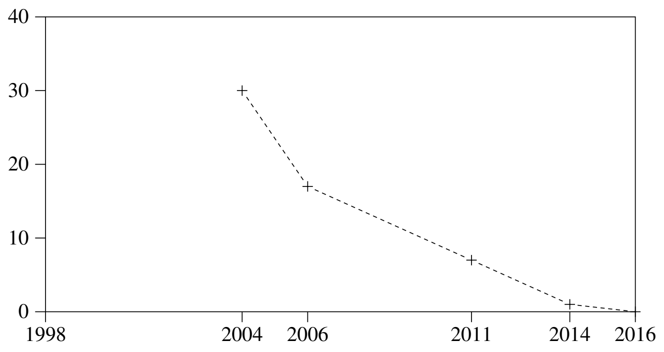


Fig. 4. Example of how results improve as the years go on. The number of hard constraint violations in the best known solution to high school instance BGHS98, collected by the author ca. 1997. The first two results are debatable, but by 2011 the instance was expressed in its current form, using the XHSTT data format, and the results were archived [15].

in Figure 5. About one-third of the papers are forward-looking, one-third are backward-looking, and one-third are case studies in Stage 2 sub-disciplines.

To conclude this section, here are some suggestions for papers that would move the discipline forward, even in Stage 3 sub-disciplines.

Large case studies. In Stage 3 sub-disciplines, ordinary case studies are no longer useful, but large case studies would be very useful. Many university course timetabling instances are for one department or faculty, despite the presence of students who take courses from several departments and indeed several faculties, and the fact that many of the challenging aspects of the problem are practical ones that arise from its large scale [10]. Several hospital scheduling problems are known beyond nurse rostering, but scheduling an entire hospital is virgin territory. And so on.

Faster and more robust solvers. Solution quality is one of three criteria by which solvers should be judged. The other two are running time and *robustness*: the ability to perform creditably on any real-world instance. Giving these other criteria more prominence would be a forward step. All solver papers should show running times, and all data formats should have running time attributes. Robustness can be encouraged by assembling and using data sets that contain real-world instances from a variety of sources. It is disturbing that what seems to be the most varied and real-world nurse rostering data set, Curtois' 'original instances' [3], is also the least used. (See also the Appendix to this paper.)

Minimal perturbation problems. For every timetabling problem there is a corresponding *minimal perturbation problem.* It takes an instance and solution (assumed to be already published), and a few changes to the instance, and asks for a revised solution incorporating the changes while altering the solution as little as possible. These very practical problems have been known for decades, yet their literature is still tiny [7].

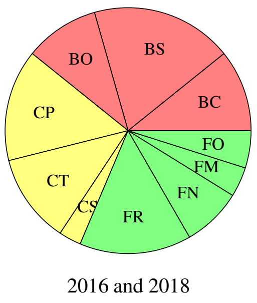


Fig. 5. Forward-looking and backward-looking papers, over one pair of PATAT conferences: 2016 and 2018. Backward-looking categories (shown in red) are: case studies in Stage 3 sub-disciplines (BC); solver papers in Stage 3 sub-disciplines (BS); other backward-looking (BO). Case study papers in Stage 2 sub-disciplines (shown in yellow) are: personnel scheduling excluding nurse rostering (CP); transport scheduling (CT); sports scheduling (CS). Forward-looking papers (shown in green) are: real-world oriented (FR); new application area (FN); minimal perturbation problem (FM); other forward-looking (FO). The assignment of papers to categories is mostly objective; the interpretation of the categories as backward-looking or forward-looking is subjective.

Infrastructure papers. Research infrastructure—mainly data formats, data sets, and competitions—often drives a discipline forward. For example, two recent competitions, for nurse rostering [1, 2] and university course timetabling [5], both made significant steps towards fidelity to the real world.

Dissemination of timetabling expertise. If automated timetabling is ever to become routine, then instances cannot be assembled only by researchers. Instead, people with administrative expertise (ward managers, departmental coordinators, and so on) must be trained in the use of software fit for their use. Today's literature is all but silent on this.

6 Conclusion

This paper has examined the progress of automated timetabling since 1995, when the first PATAT conference was held. There have been many positive developments: a better balance between sub-disciplines; a steady growth of data sets, data formats, and competitions; and improved solution quality, to a point that in some cases approaches optimality.

The newer sub-disciplines can follow the old track for some time yet: for them, case studies are the immediate need, and then data sets, data formats, competitions, and solver papers. But in well-established sub-disciplines, case studies are now contributing nothing useful, and solver papers are experiencing diminishing returns. There is a danger that these sub-disciplines could wither without yielding any benefit to society. The way forward for them, we have suggested, is to recommit to practice and orient our research accordingly.

7 Appendix: Success in practice

Timetabling research whose aim is success in practice is at a disadvantage in the academic world. Work leading to solvers that find new best solutions is virtually guaranteed publication, even if the solvers are highly tuned for one data set and run slowly. That is as it should be. But work leading to solvers that find good solutions on several data sets and run quickly, but do not find new best solutions, is likely to be denied publication, as this author can attest from personal experience. That is a problem.

One advantage of expecting solvers to produce new best solutions is that it provides a clear criterion for rejecting inferior work. We do not want ‘success in practice’ to be a loophole through which inferior work comes to be published. So we need a challenging, objective definition of success in practice.

Here is a proposal for such a definition:

> *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.*

We are not saying that a practical solver must reach this standard, any more than a theoretical solver must find a new best solution for every instance it is tested on. Rather, we are defining what a practical solver should aspire to.

A prerequisite for applying this definition is the availability of data sets that bring together real-world instances from a variety of sources. Some exist now, but we need more, and we need to value the work of making them.

Of course, the numbers chosen above are open to argument; they represent the author’s idea of a practitioner’s needs. A 5-minute run seems reasonable for exploring an alternative scenario. A 60-minute run seems reasonable for finding a timetable that will be used. If that timetable is within 5% of best known, then the difference will be barely noticeable: where the best known solution has 20 defects, the practical solution might have 21.

A definition of this kind could conceivably vary between sub-disciplines. But real-world time limits seem fairly uniform across sub-disciplines, perhaps because someone is waiting for the result, whatever the sub-discipline. Also, a definition could vary with instance size. But restricting to practical instances rules out unrealistically large sizes, and the given time limits seem reasonable for the rest. A practical solver might run much faster on small instances.

When questions arise about the detailed interpretation of the definition, they should be resolved in a way that reflects what is feasible in practice. Running times are wall clock times on widely available desktop hardware. Multiple cores are widely available, so multi-threading is allowed. Arbitrary tuning of parameters is permitted before the solver is released, but all other tuning of parameters is only permitted if it is done without human intervention and the time it takes is included in the running time.

The solver knows whether a 5-minute or 60-minute run is wanted, and may adapt itself accordingly. Indeed, a pair of unrelated solvers packaged together, one for 5-minute runs and one for 60-minute runs, is acceptable.

This definition is challenging even though it does not require solutions to be new bests. The challenge is spread across the three criteria for success in practice: good solution quality, moderate running time, and robustness.

References

1. Ceschia, S., Nguyen, T. T. D., De Causmaecker, P., Haspeslagh, S., and Schaerf, A.: Second international nurse rostering competition (INRC-II), problem description and rules. oRR abs/1501.04177 (2015). URL http://arxiv.org/abs/1501.04177
2. Ceschia, S., Nguyen, T. T. D., Causmaecker, P., Haspeslagh, S., and Schaerf, S.: Second international nurse rostering competition (INRC-II) web site, http://mobiz.vives.be/inrc2/
3. Curtois, T.: Employee Shift Scheduling Benchmark Data Sets, http://www.schedulingbenchmarks.org/ (2019)
4. Easton, K., Nemhauser, G., and Trick, M.: Solving the travelling tournament problem: a combined integer programming and constraint programming approach, In: PATAT 2002 (Fourth International Conference on the Practice and Theory of Automated Timetabling, Gent, Belgium, August 2002), Selected Papers, Springer Lecture Notes in Computer Science 2740, 100–109 (2003)
5. The Fourth International Timetabling Competition (ITC 2019), https://www.itc2019.org/home (2019)
6. Kasirzadeh, A., Saddoune, M., and Soumis, F.: Airline crew scheduling: models, algorithms, and data sets. EURO Journal on Transportation and Logistics (2015). https://doi.org/10.1007/s13676-015-0080-x
7. Kingston, J. H.: Specifying and solving minimal perturbation problems in timetabling. In: PATAT 2016 (Eleventh International Conference on the Practice and Theory of Automated Timetabling, Udine, Italy, August 2016), 207–210
8. Kingston, J. H., Post, G., and Berghe, G. V.: A unified nurse rostering model based on XHSTT. In: PATAT 2018 (Twelfth International Conference on the Practice and Theory of Automated Timetabling, Vienna, August 2018), 81–96
9. Mavrovouniotis, M. et al.: CEC-12 Competition on electric vehicle routing problem, https://mavrovouniotis.github.io/EVRPcompetition2020/
10. McCollum, B.: University timetabling: bridging the gap between research and practice. In: PATAT 2006 (Sixth International Conference on the Practice and Theory of Automated Timetabling, Brno, Czech Republic, August 2006), 15–35
11. Munari, P., Dollevoet, T., and Spliet, R.: A generalized formulation for vehicle routing problems. Working paper (2017)
12. Nurmi K., Kyngs̈, J.: A conversion Scheme for turning a curriculum-based timetabling problem into a school timetabling problem. In: PATAT 2008 (Seventh International Conference on the Practice and Theory of Automated Timetabling, Montreal, August 2008)
13. The PATAT conference series, https://patatconference.org/ (2020)
14. Ahmadi, S., Daskalaki, S., Kingston, J. H., Kyngäs, J., Nurmi, C., Post, G., Ranson, D., Ruizenaar, H.: An XML format for benchmarks in high school timetabling. In: PATAT 2008 (Seventh International Conference on the Practice and Theory of Automated Timetabling, Montreal, August 2008)

15. Post, G.: Benchmarking project for high school timetabling, https://www.utwente.nl/en/eemcs/dmmp/hstt/ (2020)
16. Schaerf, A., Measurability and reproducibility in university timetabling research: discussion and proposals. In: PATAT 2006 (Sixth International Conference on the Practice and Theory of Automated Timetabling, Brno, Czech Republic, August 2006), Selected Papers, Springer Lecture Notes in Computer Science 3867, 40–49 (2007)
17. Schmidt G. and Ströhlein, T.: Timetable construction—an annotated bibliography. The Computer Journal **23**, 307–316 (1980)
18. Toronto examination timetabling dataset, http://www.cs.nott.ac.uk/˜pszrq/data.htm
19. Transportation Conferences 2020–21, https://waset.org/transportation-conferences
20. Van Bulck, D., Goossens, D., Schönberger, J., and Guajardo, M.: RobinX: an XML-driven classification for round-robin sports timetabling. In: PATAT 2018 (Twelfth International Conference on the Practice and Theory of Automated Timetabling, Vienna, August 2018), 481–484

Design of an Exact Approach for Timetabling at Project-Oriented Schools

Michael Hölscher

Technische Universität Dresden, Fakultät Wirtschaftswissenschaften, Lehrstuhl für BWL, insb. Industrielles Management, 01062 Dresden
michael.hoelscher@tu-dresden.de

Keywords: Timetabling · Multi-Objective Optimization · Project-oriented Schools

1 Introduction

Automatic timetabling in an educational context is a complex planning task that can help to provide decision support for school administrations and replace manual planning through higher solution quality and speed. Depending on the application, a distinction can be made between the high school timetabling problem (HSTP) and the university course timetabling problem (UCTP) [1]. In the school context, planning is done for disjunct classes and an assignment to rooms, teachers and events is required that is free of conflicts [2]. This contrasts with scheduling at universities, where students often have a great freedom of choice, which is why it is more a matter of minimizing conflicts in timetabling approaches due to the many overlapping possibilities. A distinction is made in this class between curriculum-based course timetabling (CB-CTT) and post-enrolment course timetabling (PE-CTT). The CB-CTT formulation [3] takes into account a curriculum that reflects the courses to be fulfilled by the students and to which the constraints are oriented. In contrast, the PE-CTT formulation [4] requires students to enroll in courses before the actual timetabling is done. We want to focus on project-oriented schools that combine or recombine features from the context of classical school and university timetabling. The experimental school Universitätsschule Dresden (USD) serves as a real use case for such a school, which should be supported in the timetabling process. Since 2019, a learning concept has been tested there that focuses on project work and does not have a typical class structure. Students stay together in small project groups for a limited period of time and work on a specific topic. The planning horizon is much shorter than for regular schools, so one of the project cycles usually lasts six weeks and followed by a new planning iteration instead of one isolated timetabling procedure at the beginning of a school year. In principle, each student receives his or her individual timetable in the USD instead of a common timetable being generated for a whole class where everyone follows the same subject-related schedule. This distinction is shown again in Figure 1.

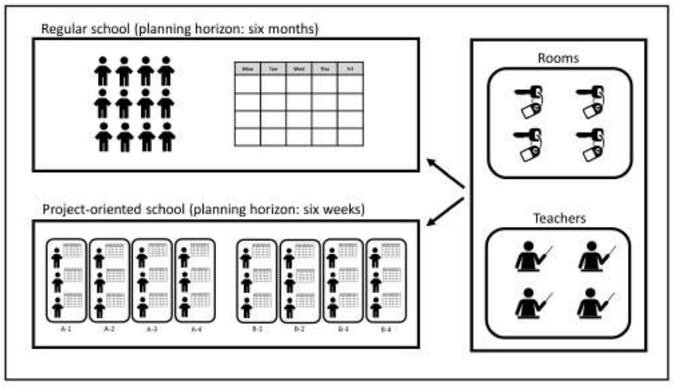

Regular school (planning horizon: six months)
Project-oriented school (planning horizon: six weeks)
Rooms
Teachers
A-1
A-2
A-3
A-4
B-1
B-2
B-3
B-4

Fig. 1. Comparison between regular school and project-oriented school

2 Problem Description

At USD special restrictions apply that must be taken into account during the solution process. The projects have a certain capacity, so there can be four to six students in a project group. In addition, each student is expected to work on several projects from different categories within a cycle. There is also the possibility for students to come up with a topic on their own and form a fixed project group. At the school, a distinction is made between group rooms and special rooms that have also a specific capacity that needs to be respected. The special rooms are for example a science lab, a music/art room or a sports hall. For each project, it is determined before the cycle begins whether and how many appointments are to take place in these special rooms. Regarding teachers, their availability must be taken into account. In addition, it is important to have the appropriate professional qualifications for the supervision of the specialized rooms. Each project also has a teacher that is directly responsible. It is not necessary to be supervised by this teacher on all dates, but there must be a minimum number of meetings per week to be able to discuss the progess of the project. The generated timetables should not have any overlapping conflicts in relation to the students and teachers. Each student must therefore be assigned to exactly one project and one room at each time slot, and each teacher can only supervise one room at a time.
The objective criteria that play a role in the underlying problem are, on the one hand, the highest possible satisfaction of the students in the assignment to projects. For this purpose, preferences are requested from each student regarding the projects, which are then to be fulfilled as best as possible in the solution process. In addition, there should be a fair distribution of the teachers' workload and a roughly equal number of supervised rooms for each teacher. And the use of rooms should be efficient, so the goal is to have as few rooms in use as possible.

3 Methodical Approach and Preliminary Results

In order to solve the timetabling problem at USD, we will try to use an exact procedure. For this purpose, a mathematical optimization model is to be set up that represents the above-mentioned conditions. All hard constraints must be met in order to achieve a feasible solution. The three mentioned objective criteria are implemented as soft constraints and give through their optimization an indication of the quality of the generated timetables. This results in a multi-objective perspective, which will be treated with the approach of a lexicographic optimization. Most important in the solution process is the consideration of student preferences, prior to the fair distribution of the teacher's workload and the efficient use of space. The aim is to test variants in which not only a strict hiearchical optimization is used, but also a certain degradation is allowed in order to find other solutions in the efficient set and compare them with each other. First results are promising and give hope that the problem can be solved to optimality in an acceptable time with the help of a commercial solver such as Gurobi. A systematic computational study is to be performed, with randomly generated instances of different sizes, to analyze the impact on computation times and to determine the possible limitations of the approach. A sensitivity analysis is also conceivable in order to find out how the computing time reacts to different parameter constellations. In the end, the USD should be supported by a practicable approach to generate feasible and optimized timetables, which are necessary to successfully implement the developed concept of project work at the school.

References

1. Schaerf, A. (1999). A survey of automated timetabling. Artificial intelligence review, 13(2), 87-127.
2. Carter, M. W., Laporte, G. (1997). Recent developments in practical course timetabling. In international conference on the practice and theory of automated timetabling (pp. 3-19). Springer, Berlin, Heidelberg.
3. Di Gaspero, L., McCollum, B., Schaerf, A. (2007). The second international timetabling competition (ITC-2007): Curriculum-based course timetabling (track 3). Technical Report, Queen's University, Belfast, United Kingdom.
4. Lewis, R., Paechter, B., McCollum, B. (2007). Post enrolment based course timetabling: A description of the problem model used for track two of the second international timetabling competition. Technical Report, Cardiff University, Wales, United Kingdom.

Noname manuscript No.
(will be inserted by the editor)

Modeling and Methods in Untis, a Popular Software System for School Timetabling

Sebastian Knopp

Received: date / Accepted: date

Abstract Untis is a company focusing on school timetabling since 1970. Its software is used by more than 25 000 educational institutions worldwide and satisfies the needs of a diverse range of users (including elementary and secondary schools, vocational schools, or universities of applied sciences). The majority of the customers apply the integrated optimization solver. Enabling each school to solve its individual timetabling problem requires taking into account a large range of constraints that are arbitrarily combinable. Therefore, a quite general optimization model is necessary. The modeling in the Untis software includes all 28 constraints listed in the survey of [Pillay(2014)] plus further important aspects such as the consideration of multiple time grids or an integrated planning over a whole year. The modelling incorporates only few hard constraints and more than fifty soft constraints. The optimization uses sophisticated construction and improvement heuristics which are embedded in different meta-heuristic approaches. While optimizing in a batch-mode (see [Schaerf(1999)]) is the core feature of the Untis software, various interactive modes, supported by the underlying optimization model, are offered to ease the work of the timetabler.

We address three topics in this talk. First, we outline core modelling aspects of the main timetabling problem addressed in the software. Second, we discuss the integrated planning over a whole school year, where timetables might change from week to week. Finally, some features of the interactive planning modes are shown.

First, let us outline the modeling and its core assumptions. We consider a week divided into a grid of periods, specifying times during which lessons can take place. The main resources involved are classes, teachers, and rooms. The goal is to schedule a given a set of lessons. Each lesson is assigned to a subject, such as music or mathematics. Each lesson must take place for a given number

Sebastian Knopp
Untis GmbH, A-2000 Stockerau, Belvederegasse 11
E-mail: sebastian.knopp@untis.at

of times during the week and requires a given set of resources. Additionally, we are given a set of couplings. A coupling is a set of lessons which must take place simultaneously. Coupling constraints are always observed. This is ensured by the solution representation. All other constraints are included as soft constraints which cover a broad range of requirements, including didactic aspects, workload constraints, resource utilization constraints, and time preferences. Violations of soft constraints are rated using penalty scores. The objective function is to minimize a total weighted penalty score.

Second, we survey the temporal planning over the course of a year. A feature important for many schools which has found only little attention in the literature. Often, timetables can differ from week to week. For example, some lessons might take place only every second week instead of every week. Other lessons should be taught only once a year during a block of three consecutive weeks, which is a common case for vocational schools. Also, differences in the curriculum between the first and the second half-year must often be taken into account. To address such requirements in the problem formulation, we assign to each lesson a set of calendar days during which the lesson is allowed to take place. Additionally, consistency is required as follows: The same lesson must take place at the same times during all weeks (if it takes place). In combination with coupling constraints, this modeling allows, e.g., to enforce two weekly alternating subjects which take place during the same period every week. These constraints are included in the optimization algorithm in an integrated way.

Finally, we show how the modeling as an optimization problem is applied to assist the user with the creation of a timetable. Usually, the timetabling process starts with importing or entering data and constraints. After that, a first and quick optimization run allows the user to assess if major problems exist in the input data. Problems might be caused by an extensive use of coupling constraints or a combination of constraints which is too rigid. The user can spot such issues by using an analysis tool which searches for maximum cliques in a conflict graph. Once all the inputs are completed, the user can start the thorough optimization run which creates a high-quality timetable. After that, manual modifications to the timetable can be made. Their impact on the objective function is immediately visualized in the user interface. In case a user dislikes the placement of a particular lesson, the software can suggest complex moves which shift that lesson. The objective function is taken into account to find suitable displacements for the lessons involved.

Keywords School Timetabling · Heuristics · System Demonstration

References

[Pillay(2014)] Pillay N (2014) A survey of school timetabling research. Annals of Operations Research 218(1):261–293

[Schaerf(1999)] Schaerf A (1999) A survey of automated timetabling. Artificial intelligence review 13(2):87–127

Noname manuscript No.
(will be inserted by the editor)

International Timetabling Competition 2021: Sports Timetabling

Dries Goossens · Jeroen Beliën · Morteza Davari · David Van Bulck

Received: date / Accepted: date

Abstract This extended abstract discusses the organization of the most recent International Timetabling Competition (ITC 2021). This competition focused on sports timetabling, where the problem is to decide on a suitable date for each of the matches to be played in the tournament. This is a complex and challenging problem, even for tournaments with few contestants. As a consequence, state-of-the-art typically focuses on a particular season of a sports competition for which a tailored algorithm is developed which is then compared to a manual solution. The aim of this competition was therefore to promote and provide insights in the development of more generally applicable sports timetabling solvers. To this purpose, participants required to solve a rich and diverse set sports timetabling instances involving various constraints that are common in real life. We introduce the contours of the problem instances, as well as the data format. We give an overview of the competition rules and timeline, and conclude with an overview of the finalists.

Keywords Sport timetabling · Competition · File format · Rules

D. Goossens and D. Van Bulck
Ghent University, Faculty of Economics and Business Administration
Department of Business Informatics and Operations Management
E-mail: dries.goossens@ugent.be; david.vanbulck@ugent.be

J. Beliën
KU Leuven, Campus Brussels
Research Centre for Operations Management
E-mail: jeroen.belien@kuleuven.be

M. Davari
SKEMA Business School
E-mail: morteza.davari@skema.edu

1 Introduction

Creating timetables for sports competitions has been a topic of research since the 1970s (e.g., [1]). Ever since, academic papers about sports timetabling have increased considerably in numbers and sports timetabling has become a specialized field [10], which has been discusssed at most of the PATAT meetings. Sports timetabling is often complex and challenging, even for a small number of teams. While generating a schedule where each team plays against each other team once and no team is involved in simultaneous matches is easy [6], some rather basic sports timetabling problems are already NP-hard. For instance, Briskorn et al. [2] show that there is no constant-factor approximation (unless P=NP) for a sports timetabling problem where certain matches cannot be played on certain rounds. Furthermore, real-life sports timetabling problems are characterized by a wide diversity of constraints, and conflicting interests of many stakeholders. At the same time, in professional sports, the timetable has an impact on commercial interests and revenues of the clubs, broadcasters, sponsors, as well as an impact on society through resulting traffic and policing costs.

Since 2002, there have been frequent timetabling competitions, which have been benificial for the research community. The first international timetabling competition (ITC) was organized in 2002 and focused on (a simplified version of) the university course timetabling problem [11]. The next ITC competition (2007) aimed to further develop interest in the general area of educational timetabling and involved three problems: curriculum-based timetabling, examination timetabling, and post-enrollment timetabling [12,13]. With high-school timetabling, the ITC placed yet another educational timetabling problem in the spotlights in 2011 [16,17]. The fourth ITC is again devoted to university course timetabling: it introduces the combination of student sectioning together with time and room assignment of events in courses [14,15]. In between, PATAT has supported two international nurse rostering competitions in 2010 [9] and 2014 [4], as well as a cross-domain heuristic search challenge (CHeSC 2011), where the challenge was to design a high-level search strategy that controls a set of problem-specific low-level heuristics, which would be applicable to different problem domains [3].

Many of the sports timetabling contributions in the literature read as a case study, describing a single instance for which a tailored algorithm is developed (which is then compared to a manual solution). Moreover, the state-of-the-art does not offer a general solution method, or even much insight in which type of algorithm would work well for which type of problem [18]. One notable exception is the travelling tournament problem [7], which minimizes the total team travel in a timetable. For this problem, substantial algorithmic progress has been reported after Easton et al. [7] made a set of artificial benchmark instances publicly available, and for which best results can be submitted to a website maintained by professor Michael Trick (see http://mat.tepper.cmu.edu/TOURN/). Hence, an international timetabling competition could make a valuable contribution to the field of sports timetabling, and given the efforts done by Van Bulck et al. [18,19] with respect to the development of an XML-based file format for problem instances and solutions, we believe the time was right for a timetabling competition on sports.

2 Problem description and file format

The input of a sports timetabling problem consists of a set of rounds R, a set of teams T, and a set of games G. The set of games consists of ordered pairs (i, j) in which $i \in T$ is the home team providing the venue where the game is played, and $j \in T$ is the away team. Although many tournament formats are conceivable, in this competition we focus on so-called double round-robin tournaments (2RR), which are very common in practice [8]. In a double round-robin tournament, each team plays against each other team twice, typically once at home and once away. Although there is a line of research that focuses on the simultaneous scheduling of multiple leagues with dependencies [5], we focus on a single league. No team can play more than one game per round. In practice, rounds typically correspond to weekends, which may consist of several time slots (e.g., Saturday evening, or Sunday afternoon), each with their capacity. We focus on so-called time-constrained tournaments, i.e., tournaments that use the minimum number of rounds required to play all matches. In a 2RR with n teams, n even, the minimum number of rounds to play all games equals $2(n-1)$; if n is odd, the minimum number of rounds is $2n$.

A timetable maps each game in G to a round in R such that no team plays more than one game per round. An example of a timetable for a double round-robin tournament with 6 teams is given in Table 1.

Table 1 A compact double round-robin timetable for a league with 6 teams.

r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}
(1,2)	(2,5)	(2,4)	(2,3)	(6,2)	(2,1)	(5,2)	(4,2)	(3,2)	(2,6)
(3,4)	(4,1)	(1,6)	(5,1)	(4,5)	(4,3)	(1,4)	(6,1)	(1,5)	(5,4)
(5,6)	(6,3)	(5,3)	(6,4)	(1,3)	(6,5)	(3,6)	(3,5)	(4,6)	(3,1)

Sports timetables need to satisfy a usually large set of constraints, which is partitioned into hard constraints and soft constraints. Hard constraints represent fundamental properties of the timetable that can never be violated. Soft constraints, in contrast, rather represent preferences that should be satisfied whenever possible. The validation of each soft constraint c results in a vector D_c of n_c integral numbers, called the deviation vector $D_c = [d_1\ d_2\ \dots\ d_{n_c}]$. If a constraint is satisfied, all elements of its deviation vector are equal to zero. Contrarily, the deviation vector of a violated constraint contains one or more strictly positive elements. For hard constraints, any deviation renders the schedule infeasible. Each soft constraint features a cost function f_c and weight w_c. A violated soft constraint triggers a penalty $p_c = w_c f_c(D_c)$, equal to a weighted mapping of its deviation vector by its cost function. The objective we use for the competition instances sums over all violated soft constraint penalties.

The instances feature a variety of constraints from the classicification developed by Van Bulck et al. [18]. The authors distinguish capacity constraints, game constraints, break constraints, fairness/attractiveness constraints, and separation constraints. Capacity constraints force a team to play home or away and regulate the total number of games played by a team or group of teams. Game constraints enforce or forbid specific assignments of a game to rounds. Constraints to increase the fairness or attractiveness involve balancedness of, e.g., home advantage, travel distances, etc. Break constraints regulate the frequency and timing of breaks in a competition; we say that a team has a break if it has two consecutive home games, or two consecutive away games. Separation constraints regulate the number of rounds between consecutive games involving the same teams.

The problem instances are expressed using the standardized XML data format developed by Van Bulck et al. [18]. The main intention of this data format is to promote problem instance data sharing and reuse among different users and software applications, and this is exactly what the timetabling competition envisions. The XML data format is open, human readable (i.e., no binary format), software and platform independent, and flexible enough to store the problem instances.

Most of the sports timetabling constraints are easy to express in words but are hard to enforce within specific algorithms such as mathematical programming or metaheuristics. We believe this format minimizes the specification burden and maximizes the accessibility. The main advantage of XML over plain text-only file formats lies in the structured way of data storage. Indeed, an important motivation behind XML is to separate data representation from data content.

A detailed description of the the file format is available on the competition website (`http://itc2021.ugent.be`). The website also provides access to a validator, allowing participants to verify whether their solution satisfies all hard constraints and to determine its score on the objective function.

3 Competition rules

We are much indebted to the various organizers of the previous international timetabling competitions. Their experience has crystallized into the rules that were used for the ITC 2019 competition [15], and to which we will largely adhere for this competition. In particular, we enforce no bound on the computation time. In fact, the objective function value of the solution is the only criterion that matters. While computation time is obviously not unimportant, a fair comparison in terms of computation time is quite challenging, and it could easily lead to disputes that we as organizers prefer to avoid. Moreover, from a practical point of view, sports timetabling problems are often not so

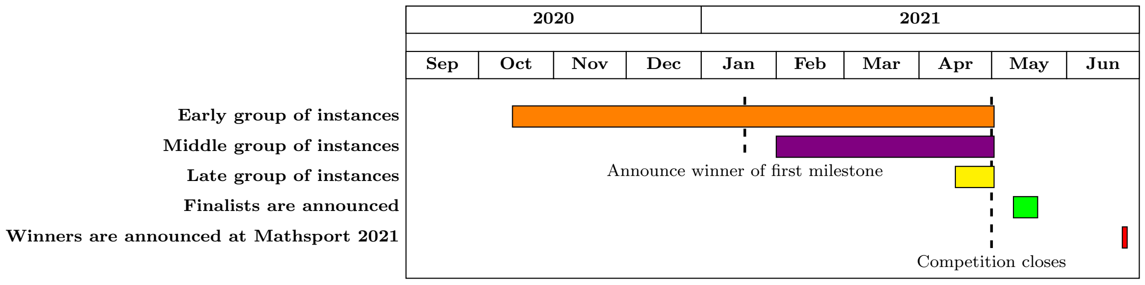


Fig. 1 Timeline for the International Timetabling Competition 2021

time-critical, as there are often several days or even weeks available to obtain a good solution.

We also allow to make use of any commercial solver. In this way, we would like to lower the threshold to participate, and reach out to the largest possible research community. Obviously, to keep it interesting, the instances for the competition will not solve to optimality with any straightforward formulations on e.g., state-of-the-art IP solvers.

Although we allow parameter tuning, we require that the same version of the algorithm is used for all instances. In other words, the algorithm should not "know" which instance it is solving. While the algorithm may analyze the problem instance and set parameters accordingly, it should apply this same procedure for all instances. The programmer should not set different parameters for different instances, however, if the program is doing this automatically, then this is acceptable. We will be asking for the source code of the finalists, in order to check whether the participants comply with this rule.

We believe these rules are efficient (in the sense that they do not require the organizer to run the participant's code) and fair/simple (in the sense that the only thing that matters is the obtained objective value; it avoids all discussion about measuring, e.g., computation time, the impact of random seeds, etc.).

4 Competition timeline and results

An overview of the competition timeline is given in Figure 1. In total, we released three groups of 15 artificially generated problem instances each: early, middle, and late instances. While all instances contributed to the final ranking of participants, instances that were released later in the competition had a higher weight. For instance, the overall best found solutions was respectively awarded 10, 15, and 25 instances for an early, middle, and late problem instance. The early group of instances were already available from our website at the time the competition was officially announced (mid October 2020), while the middle group of instances were only released in February 2021. The late instances followed half April 2021, which gave the participants two weeks to come up with solutions.

Team name	Research institute	Participants
TU/e	Eindhoven University of Technology	F. Spieksma, H. Christopher, R. Lambers, and J. van Doornmalen
Saturn	HSE University	S. Daniil and R. Ivan
MODAL	Zuse Institute Berlin	T. Koch, T. Berthold, and Y. Shinano
GOAL	Federal University of Ouro Preto	G. H. G. Fonseca and T. A. M. Toffolo
UoS	University of Southampton	T. Martínez-Sykora, C. Potts, C. Lamas-Fernández
Udine	University of Udine	R. M. Rosati, M. Petris, L. Di Gaspero, and A. Schaerf

Table 2 Overview of the 6 finalists (randomly ordered)

Around half January 2021, we organized a first milestone event where participants had the possibility to submit their best solutions found at that time. Although optional, participation in the first milestone was strongly encouraged as it provided participants with the feedback on where their algorithms ranked among their peers as well as a chance to win a small prize (free registration for Mathsport 2022). The first milestone was won by team UoS, followed by team Udine and TU/e (see Table 2).

At the time of the final submission deadline, 13 research teams from over 10 different countries successfully submitted solutions. As a comparison, the cross-domain heuristic search challenge attracted 17 teams, the two international nurse rostering competitions each attracted 15 teams, and the third and fourth international timetabling competition each attracted 5 teams that submitted one or more solutions by the final submission deadline.

Out of all 13 participating teams, the 6 finalists given in Table 2 were selected. The prize fund is 1,750 EUR to be split between the first, second, and third place competitors. Moreover, a discount on registration for the upcoming PATAT conference is awarded to the top three overall. We thank our sponsors OR in Sports and PATAT for their generous contribution to the rewards we could distribute over the winners. At the Mathsport International 2021 conference, team UoS (University of Southampton) was announced as the winner of the ITC 2021.

References

1. Ball, B. C., Webster, D. B. (1977). Optimal scheduling for even-numbered team athletic conferences. AIIE Transactions 9(2):161–169.
2. Briskorn, D., Drexl, A., Spieksma F.C.R. (2010). Round robin tournament and three index assignments. 4OR - A Quarterly Journal of Operations Research, 8:365–374.
3. Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., McCollum, B., Ochoa, G., Parkes, A.J., Petrovic, S. (2011). The Cross-Domain Heuristic Search Challenge – An International Research Competition. In: C.A.C. Coello (editor) Learning and Intelligent Optimization, pp. 631–634. Springer.
4. Ceschia, S., Dang, N., De Causmaecker, P., Haspeslagh, S., Schaerf, A. (2019). The second international nurse rostering competition. Annals of Operations Research, 274:171-–186.
5. Davari, M., Goossens, D., Beliën, Lambers, R., Spieksma, F.C.R. (2020). The multi-league sports scheduling problem, or how to schedule thousands of matches. Operations Research Letters, forthcoming.

6. de Werra, D. (1980). Geography, games and graphs. Discrete Applied Mathematics 2:327–337.
7. Easton, K., Nemhauser, G., Trick, M., (2001). The traveling tournament problem description and benchmarks. In: Walsh, T. (editor), Principles and Practice of Constraint Programming — CP 2001. Springer, Berlin, Heidelberg, pp. 580-–584.
8. Goossens, D., Spieksma, F.C.R. (2012). Soccer schedules in Europe: an overview. Journal of Scheduling 15(5): 641–651.
9. Haspeslagh, S., De Causmaecker, P., Schaerf, A., Stølevik, M. (2010). The first international nurse rostering competition 2010. Annals of Operations Research 218(1):221–236.
10. Kendall, G., Knust, S., Ribeiro, C.C., Urrutia, S. (2010). Scheduling in sports: An annotated bibliography. Computers & Operations Research 37(1):1–19.
11. Paechter, B., Gambardella, L.M., Rossi-Doria, O. (2002). International timetabling competition (ITC2002). http://sferics.idsia.ch/Files/ttcomp2002
12. McCollum, B., Schaerf, A., Paechter, B., McMullan, P., Lewis, R., Parkes, A.J., Di Gaspero, L., Qu, R., Burke, E.K. (2007). The second international timetabling competition (ITC2007). http://www.cs.qub.ac.uk/itc2007/index.htm
13. McCollum, B., Schaerf, A., Paechter, B., McMullan, P., Lewis, R., Parkes, A.J., Di Gaspero, L., Qu, R., Burke, E.K. (2010). Setting the research agenda in automated timetabling: The second international timetabling competition. INFORMS Journal on Computing 22(1):120–130.
14. Müller, T., Rudová, H. Müllerová, Z. (2018). University course timetabling and International Timetabling Competition 2019. In: E. K. Burke, L. Di Gaspero, B. McCollum, N. Musliu, and E. Özcan (editors), Proceedings of the 12th International Conference on the Practice and Theory of Automated Timetabling, Vienna, pp. 5–31. PATAT.
15. Müller, T., Rudová, H. Müllerová, Z. (2019). International Timetabling Competition (ITC2019). http://www.itc2019.org/home
16. Post, G., Di Gaspero L., Kingston, J.H., McCollum, B., Schaerf, A. (2011). International timetabling competition (ITC2011). http://www.utwente.nl/en/eemcs/dmmp/hstt/itc2011/
17. Post, G., Di Gaspero L., Kingston, J.H., McCollum, B., Schaerf, A. (2016). The third international timetabling competition. Annals of Operations Research, 239:69—75.
18. Van Bulck D., Goossens D., Schönberger J., Guajardo M. (2020a), RobinX: A three-field classification and unified data format for round-robin sports timetabling. European Journal of Operational Research, 280:568–580.
19. Van Bulck, D., Goossens, D., Schönberger, J., Guajardo, M. (2020b). An instance data repository for the round-robin sports timetabling problem. Management and Labour Studies, 45:184–200.

Real-world university course timetabling at the International Timetabling Competition 2019

Hana Rudová[1], Tomáš Müller[2], and Zuzana Müllerová[3]

[1] Faculty of Informatics, Masaryk University, Brno, Czech Republic
hanka@fi.muni.cz
[2] Purdue University, West Lafayette, Indiana, USA
muller@unitime.org
[3] UniTime, s.r.o., Zlín, Czech Republic
mullerova@unitime.org

1 Introduction

The International Timetabling Competition 2019 (ITC 2019) [12] introduced a variety of real-life university course timetabling problems coming from different parts of the world. A novel model of a complex course timetabling problem allows the specification of problems from many different universities. In the competition, representative problems from ten universities worldwide were considered. However, they represent a fraction of the institutions using UniTime [16], a non-commercial software, from which the instances for the competition were taken. Thirty benchmark problems together with six test instances are available at the competition website [6], which allows for solution validation and provides a repository of existing solutions. This paper will discuss the characteristics of the course timetabling problems considered in the competition. We will demonstrate that the model proposed for the competition allows encapsulating very different features.

The first International Timetabling Competition 2002 considered a simplified course timetabling where post-enrollment problems were solved. For these problems, course enrollments of students are defined, and courses must be assigned in timeslots and rooms without any overlap for students. All benchmark instances were randomly generated. The second competition in 2007 [9] has organized two course timetabling tracks. One of them slightly extended the post-enrollment problem from the first competition [8] and the other introduced curriculum-based timetabling based on problems from the University of Udine in Italy [4]. The curriculum contains a set of courses, which must be assigned into timeslots with no overlaps. All the competitions, including ours, were supported by the PATAT conference together with several other competitions from different domains [13]. A recent survey about educational timetabling benchmarks and competitions is available at [1].

2 Characteristics

Our problems use a complex course structure to model the presence of students in different parts of a course. A course may contain one or more course configurations, each with one or more classes that can be of different types and have an optional parent-child relationship between them. These classes are to be timetabled into rooms and time periods. A class may occupy multiple time periods, possibly spanning multiple days and weeks. This allows us to model classes with multiple meetings at the same time and room, and/or classes that are taught only during certain weeks of the semester. All benchmark problems have five-minutes time periods, which are going from midnight to midnight, have seven days a week, and are running for a given number of weeks (between 6 and 21). This permits a very flexible organization of time and supports various irregularities and other exceptions in class placements. Students are enrolled in courses and are to be assigned to classes based on the defined course structure. A student must get one class of each type from a single course configuration, following the parent-child relationship when defined. For example, each student must get a lecture and a seminar, where only some lecture-seminar combinations are allowed. Finally, there are soft and hard distribution constraints of nineteen types defined on subsets of classes. Most of the constraints such as `SameDays` or `NoOverlap` can be validated on pairs of classes, i.e., each pair that does not satisfy the constraint incurs a penalty. Four types of constraints such as `MaxDays` must be validated on the whole subset of classes.

There are four essential optimization criteria. The goal is to minimize penalties for time and room assignments of classes, penalties for unsatisfied soft distribution constraints, and the number of student conflicts. Minimizing the number of student conflicts is a fundamental part of the problem, which is crucial for university course timetabling. A student conflict exists if the student cannot attend a pair of his/her classes. The conflicts are not only between classes that overlap in time, but they are also between classes that students cannot attend due to travel distances between assigned rooms.

3 Problems from different universities

We will see that the proposed XML model allows specifying very different real-life university course timetabling problems. Timetabling problems may differ even within the same institution.

We have included three different problems from Masaryk University (Czech Republic). The timetable for the Faculty of Informatics can be generated based on pre-enrollments of students into courses. Otherwise, it is a relatively standard mid-size problem with about 500 classes each scheduled weekly or sometimes bi-weekly. There are two different types of problems for the Faculty of Education and Faculty of Sport Studies, representing (1) the common present form of study and (2) the lifelong together with the combined forms of study [11]. This second problem is very specific and complex. Here, a different timetable is needed each week, and each course is taught only several weeks during the semester. On top of that, the Faculty of Sport Studies timetables are significantly influenced by traveling to various sports facilities that are spread over the city. We can see specific curricula patterns for the Faculty of Education, typically composed of a pair of "sub-curricula", each representing one field of study such as Math, Physics, English, or Music. These pairs may result in many student conflicts because it is impossible to satisfy all the possible combinations.

Purdue University (USA) uses automated timetabling for all its departments together [15], which means that we can see the large-scale problem representing all courses of the large public university with about 40,000 students. The construction of the timetable starts with timetabling for the large lecture rooms, which the university shares. We have only a few courses for each student in this problem, but the room utilization is very high since large rooms for hundreds of students represent a scarce and expensive resource. At Purdue, we can also see a typical example of an American university where classes are taught several times a week at the same time and same room, for instance, Monday, Wednesday, Friday at the half-hour (7:30 am, 8:30 am, ... 4:30 pm). Also, courses may be taught using different patterns, e.g., either two times a week for an hour and a half or three times a week for one hour. In contrast to other problems discussed before, there are neither curricula nor pre-enrollments. The timetable is constructed based on last-like semester course enrollments (e.g., timetable construction for Fall 2019 used as an input real course enrollments for Fall 2018).

AGH University of Science and Technology from Poland builds course timetables separately for each faculty. Still, they share some resources, and some of the faculties provide a lot of courses for students outside of their faculty. For instance, in our data sets, the Faculty of Humanities has almost two-thirds of the classes for students of other faculties. The data are structured so that the courses for students from these faculties can be managed and timetabled separately. AGH uses pretty rigid curricula, only containing mandatory and elective courses. In the original (UniTime) problem, students of the same curriculum are kept together and attend the same classes. There are no student conflicts allowed to be created by the solver.

For several other universities from Asia, such as Turkish-German University, İstanbul Kültür University, and Bethlehem University, it may seem that there are no students involved because no students are present in the data set. This is because these universities decided to model student course requirements using the `SameAttendees` or `NoOverlap` distribution costraints.

Many other specifics of the competition problems will be described in the full version of this paper.

4 Conclusion

The competition problems introduce complex real-world problems with many different characteristics, which led to a relatively small number of approaches capable of solving them. Initially, five different teams submitted results to the competition, and two others, including one competition organizer, computed solutions as well. Thanks to the open-source prize, the source codes of three solvers are now publicly available. As of June 2022, thirteen teams have submitted some solutions published on the competition website. The winning team applied a parallelized matheuristic [10] based on the graph-based mixed integer programming formulation [5]. The second team applied mixed integer programming [14], and the third team's solution [3] is representative of a metaheuristic method, which is the modified version of simulated annealing. The fifth team opted for a solution using the MaxSAT solver combined with a local search [7]. The winning team also maintains a website [2], where their lower bounds for all competition instances are published. We can see that five instances are now solved optimally. For many instances, the gap is still significant, promising opportunities for future research. Also, there is a high potential for developing more efficient methods capable of solving the problems in a more reasonable time frame.

References

1. Ceschia, S., Gaspero, L.D., Schaerf, A.: Educational timetabling: Problems, benchmarks, and state-of-the-art results (2022), arXiv:2201.07525
2. DSUM data science for university management, ITC 2019, https://dsumsoftware.com/itc2019/
3. Gashi, E., Sylejmani, K., Ymeri, A.: Simulated annealing with penalization for university course timetabling. In: Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling PATAT 2021, Volume II (2021). pp. 361–366 (2021)
4. Gaspero, L.D., McCollum, B., Schaerf, A.: The second International Timetabling Competition (ITC-2007): Curriculum-based course timetabling (track 3). Tech. Rep. QUB/IEEE/Tech/ITC2007/CurriculumCTT/v1.0, Queen's University, Belfast (2007)
5. Holm, D.S., Mikkelsen, R.Ø., Sørensen, M., Stidsen, T.J.R.: A graph-based MIP formulation of the International Timetabling Competition 2019. Journal of Scheduling (2022), published: 11 March 2022
6. ITC 2019: International Timetabling Competition, https://itc2019.org
7. Lemos, A., Monteiro, P.T., Lynce, I.: ITC 2019: University course timetabling with MaxSAT. In: Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling PATAT 2021, Volume I (2020). pp. 105–128 (2021)
8. 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 Working Papers in Accounting and Finance A2007-3, Cardiff Business School, Cardiff University (2007)
9. 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 Journal on Computing **22**(1), 120–130 (2010)
10. Mikkelsen, R.Ø., Holm, D.S.: A parallelized matheuristic for the International Timetabling Competition 2019. Journal of Scheduling (2022), published: 3 May 2022

11. Müller, T., Rudová, H.: Real-life curriculum-based timetabling with elective courses and course sections. Annals of Operations Research **239**(1), 153–170 (2016)
12. Müller, T., Rudová, H., Müllerová, Z.: University course timetabling and International Timetabling Competition 2019. In: PATAT 2018 — Proceedings of the 12th International Conference on the Practice and Theory of Automated Timetabling. pp. 5–31 (2018)
13. PATAT conferences, http://patatconference.org/communityService.html
14. Rappos, E., Thiémard, E., Robert, S., Hêche, J.F.: A mixed-integer programming approach for solving university course timetabling problems. Journal of Scheduling (2022), published: 15 February 2022
15. Rudová, H., Müller, T., Murray, K.: Complex university course timetabling. Journal of Scheduling **14**(2), 187–207 (2011)
16. UniTime.org: University timetabling, comprehensive academic scheduling solutions, https://www.unitime.org/

Noname manuscript No.
(will be inserted by the editor)

A MIP based approach for International Timetabling Competation 2019

Dennis S. Holm* · **Rasmus Ø. Mikkelsen*** · **Matias Sørensen** · **Thomas R. Stidsen**

Received: date / Accepted: date

Keywords Mixed Integer Programming · Matheuristics · Fix and Optimize · Conflict graphs · University Timetabling · International Timetabling Competition 2019 · ITC 2019

1 Introduction

Top five finalist summary paper.

This summary paper was written as a part of the submission for the International Timetabling Competition 2019 (ITC2019). It aims to give an overview description of the algorithm used to solve the ITC2019 problem instances. Since the paper is limited to 4 pages, the description cannot be very comprehensive. The algorithm is divided into different parts. First part is a reduction algorithm where unnecessary information in the data is removed. It is followed up by two initial solution algorithms and a Fix-and-Optimize matheuristic. The initial solution algorithms and Fix-and-Optimize algorithms all depend on a Mixed Integer Programming (MIP) formulation, which will also be described briefly. Finally the computational setup is presented as it defines the resulting algorithm.

* First Author

Dennis S. Holm*
Akademivej
Building 358
2800 Kgs. Lyngby
E-mail: dsho@dtu.dk

Rasmus Ø. Mikkelsen*
Akademivej
Building 358
2800 Kgs. Lyngby
E-mail: rasmi@dtu.dk

Matias Sørensen
E-mail: sorensen.matias@gmail.com

Thomas R. Stidsen
Akademivej
Building 358
2800 Kgs. Lyngby
Tel.: +45 45254449
E-mail: thst@dtu.dk

2 Reducing the problems

The reduction of the problems concern two parts. One part considers the removal of times/rooms of classes that are never allowed in a feasible solution. The other part considers the removal of distribution constraints that are dominated by other distribution constraints.

2.1 Reducing times/rooms

This part is very useful for the MIP because it reduces the number of variables. Consider a graph where each vertex corresponds to a class-time pair. To get a valid class-time assignment in the problem, a vertex must be chosen for each of the classes. Now add an edge between two vertices if the two pairs cannot both be chosen in a valid solution. That could happen if a hard distribution constraint states, that the times of the two classes are not allowed simultaneously or if the two vertices represent the same class. Likewise a conflict graph of class-room pairs can be constructed.

Now consider a conflict graph G described as above and consider the sets of vertices $V(c_i)$ that represents class c_i. If $|V(c_i)| = 1$ for a c_i we denote the vertex as *fixed.* This means that any neighbour of a fixed vertex cannot be chosen in a valid solution, thus such vertices can be removed from G.

Consider a clique C in the graph. C describes that only one of the vertices $V(C)$ can be chosen. If $V(c_i) \subset V(C)$ for a specific c_i then the vertices of $V(C) \setminus V(c_i)$ can be removed from G. By reducing the graph G to G' with the above methods one might find vertices that are fixed in G' but not in G. It is therefore important to keep reducing G' until no more reductions can be made.

2.2 Reducing distribution constraints

Redundant distribution constraints: A constraint that consider at most one class, a soft constraint with 0 penalty, or cannot be violated by the classes it consider.

Dominated distribution constraints: A distribution constraint (hard or soft) d_1 is said to be dominated by a hard distribution constraint of equal type d_2 if the classes of d_1 is a subset of the classes of d_2.

Redundant and dominated distribution constraints are removed from the problem.

3 MIP

The Mixed Integer Programming formulation consider a binary decision variable $x_{c,t,r}$ that is equal to 1 if a class c is scheduled at time t in room r and 0 otherwise. If the problem includes student sectioning the MIP formulation also consider the binary variable $E_{s,c}$ which is equal to 1 if student s is attending class c and 0 otherwise.

The decision variable $x_{c,t,r}$ leads to auxiliary variables $y_{c,t}$, $z_{c,d}$ and $w_{c,r}$, which respectively consider the assignment of time t, day d or room r for a class c. Note that the variables $y_{c,t}$ and $w_{c,r}$ are represented by vertices in the conflict graphs presented in section 2. The distribution constraints used to define the edges of a conflict graph are modelled by a clique cover of the conflicts graphs. This is the modelling of most of the hard distribution constraints. Note that *SameAttendees* requires an additional conflict graph on $x_{c,t,r}$ when the times by themselves are not overlapping but the room assignments violate the constraint. Conflict graph are created for the soft distribution constraints as well. Here the edges have a cost corresponding to the distribution constraint(s) that created the edge. Each edge can be used as a constraint in the model. But to lower the amount of constraints

it is better to divide the graph into subgraphs where all edges have equal cost and then find a star cover of each graph. Each star is added as a constraint to the model.
The distribution constraints *MaxDays*, *MaxDayLoad*, *MaxBreaks* and *MaxBlock* are modelled in a more advanced way.
Student sectioning is performed like *SameAttendees* except that the cost of overlap between courses c_1 and c_2 is related to E_{s,c_1} and E_{s,c_2}.

4 Initial solution

An initial solution is constructed by two simple constructive matheuristics. The constructive heuristics split the problem into two or three parts respectively, where the parts are solved one followed by the other.

Algorithm 1 Two-Stage Constructive Algorithm (2SCA)

```
1: assign times and rooms to classes
2: assign students to classes
3: return assignments
```

For the 2SCA algorithm a MIP is defined with the only objective being the number of unassigned classes. When a feasible schedule is found, the schedule is given to the original MIP which is solved to assign students to classes.

Algorithm 2 Three-Stage Constructive Algorithm (3SCA)

```
1: assign times to classes
2: try to assign rooms to classes
3: while assignment of rooms was not possible do
4:     find a new assignment of times to classes
5:     try to assign rooms to classes
6: end while
7: assign students to classes
8: return assignments
```

In the 3SCA algorithm a MIP that considers only the assignment of times is solved first. The time-assignment is then given to another MIP that considers the assignment of rooms. If there is no feasible room-assignment to the given time-assignment another time-assignment will be found. When a feasible time and room assignment has been found the schedule is given to a MIP that considers student sectioning.

5 Fix-and-Optimize

To improve the solutions found in section 4 we use a Fix-and-Optimize matheuristic. The Fix-and-Optimize splits the decision variables into two sets F and U. We then consider the *subproblem* where the variables of F are fixed and we optimize the subproblem. The results are strongly dependent on the way the sets are chosen. If a large set U is chosen, the model will be too complex, on the other hand if U is too small there will be no improvement.

When choosing U we consider a neighbourhood of courses. That is the decision variable $x_{c,t,r}$ (and related auxiliary variables) for all classes that are part of a given set of courses. As the instances vary greatly in difficulty, we choose the size of U to be 25% of class assignments as a base line.

Algorithm 3 Pseudo code for Fix-and-Optimize

```
 1: MIP: Set solution $sol^*$
 2: MIP: Fix all variables to current value
 3: while time do
 4:     $U$ = GetVariablesToUnfix()
 5:     MIP: Unfix all $U$
 6:     $sol_{new}$ = Solve MIP
 7:     if $sol_{new}$ is improving then
 8:         MIP: Set soltuion $sol_{new}$
 9:     end if
10:     MIP: Fix all variables to current value
11: end while
```

The pseudo code of Fix-and-Optimize is shown in algorithm 3. The solution sol^* is the warm start solution, that could be the initial, best known or any other solution. The set of variables U is determined by function `GetVariablesToUnfix()`.

5.1 Dynamically updating parameters
The goal of Fix-and-Optimize is to find a balance between the MIP complexity and the availability and ease of finding improving solutions. On smaller and easier instances it is preferable to unfix in a more aggressive manner, while a more conservative strategy should be used for more difficult cases. The correct strategy is difficult to gauge a priori and therefore the parameters of Fix-and-Optimize are updated dynamically through the search.

6 Computational setup

When a data instance is received we start by reducing the file as described in section 2, this gives a reduced data instance that is used to construct the MIP described in section 3. The 3SCA algorithm described in section 4 is run without considering soft distribution constraints to find a pool of initial solutions. Additionally the 2SCA is also run. The MIP and a number of Fix-and-Optimize algorithms are run in parallel. The MIP has focus on improving the lower bound while the Fix-and-Optimize algorithms produce new solutions that are passed to the MIP to help reduce the branch and bound tree. The Fix-and-Optimize algorithms focus on separate neighbourhoods and regularly reset to the best known solution, such that none are “left behind”. If enough time passes with no improvement in best known solution, the Fix-and-Optimize algorithms begin to diversify; each search starts from a new initial solution (from the 3SCA algorithm), no longer resets to the best known solution and considers all available neighbourhoods. This continues until the best known solution is improved, where after the Fix-and-Optimize algorithms revert to their normal strategy.
For instances where the number of students exceeds 30.000, we start an additional process where a specialized MIP is defined that applies student sectioning to fixed timetables. The timetables are produced by the 3SCA algorithm and a variant of the Fix-and-Optimize algorithm that is set up to produce timetables without considering the students.

ITC 2019: Results Using the UniTime Solver

Tomáš Müller

Purdue University, West Lafayette, Indiana, USA
muller@unitime.org

Abstract. This abstract presents results on using the Uni-Time solver on the International Timetabling Competition 2019 late data sets. The results are compared with the best solutions that are published on the competition website.

Keywords: University course timetabling · ITC 2019 · UniTime

1 Introduction

Building on the success of the earlier timetabling competitions, the International Timetabling Competition 2019 (http://www.itc2019.org) is aimed to motivate further research on complex university course timetabling problems coming from practice. The competition data sets are based on real-world problems that have been collected using the UniTime application [11]. The individual timetabling problems are quite large with the largest problem having close to 9,000 classes and over 38,000 students. The data for the competition have been collected from 10 institutions around the world and there are a lot of differences between them. For example, some instances have no students (classes are spread in time using hundreds of non-overlapping constraints), some instances are based on student pre-registrations (aka post-enrollment course timetabling) and some instances are based on curricular data. There have been three sets of 10 instances published during the competition: early, middle and late.

The competition problem combines student sectioning together with standard time and room assignment of individual course events [8]. Classes are organized in a course structure defining the valid combinations of classes a student can take. For example, each student taking a Mathematics course needs to attend a lecture and a lab that is associated with the lecture. The problem also deals with travel times between individual rooms, classes that have different lengths and multiple meetings on a week, classes that are meeting only during certain weeks, and various additional distribution constraints, such as minimizing gaps between classes of an instructor or defining how many class hours an instructor can teach on a day.

UniTime [12] is a comprehensive educational scheduling system that supports developing course and exam timetables, managing changes to these timetables, sharing rooms with other events, and scheduling students to individual classes. It is a distributed system that allows multiple university and departmental schedule managers to coordinate efforts to build and modify a schedule that meets their diverse organizational needs while allowing for minimization of student course conflicts. The software is distributed free under an open-source license and the UniTime project is a part of the Apereo Foundation, a non-profit organization whose mission is to develop and sustain open-source software for higher education.

As the UniTime course timetabling problem is quite complex, with many additional aspects, some simplifications have been made in the competition problem as well as on the problems collected from UniTime. The aim was to reduce the modeling complexity without losing any of the hardness (or computational complexity) of the problems. For example, in UniTime, it is possible for a class to need two or more rooms, or in certain cases, for multiple classes to share a room. Also, some distribution constraints have been removed or reformulated in the competition problem. For example, instead of having a back-to-back constraint, the competition problem requires such classes to be placed in the same room, on the same day, and with limited time between the first and the last class. This makes for the same outcome when the constraint is satisfied, but the penalization of a partially violated soft constraint is a bit different. More details are discussed in [8].

The paper is organized as follows: in the next chapter, the competition solver is described. There is a short description of the UniTime solver and the code written to make the solver work on the competition problem. Results are presented in the following chapter and conclusions are presented at the end of the paper.

2 The Solver

In this work, the UniTime course timetabling solver is used as it is, even using the default configuration that ships with the UniTime application. New code has been only needed to load the competition problem into the UniTime solver and to save the solution in the competition format. Other than that, some of the penalizations of violated soft distribution constraints have been changed to follow the competition problem. The code is open-source (under the Apache license) and available in GitHub [6].

The UniTime solver is based on an iterative forward search (IFS) algorithm [11]. This algorithm is similar to local search methods; however, in contrast to classical local search techniques, it operates over feasible, though not necessarily complete, solutions. In these solutions, some classes may be left unassigned. All hard constraints on assigned classes must be satisfied. Such solutions are easier to visualize and more meaningful to human users than complete but infeasible solutions. Because of the iterative character of the algorithm, the solver can also easily start, stop, or continue from any feasible timetable, either complete or incomplete.

The algorithm makes use of Conflict-based Statistics (CBS) [9] to prevent itself from cycling. The IFS algorithm is used until a complete timetable is found. In the next phase, a local optimum is found using a Hill Climbing (HC) algorithm. Once a solution can no longer be improved using this method, the Great Deluge (GD) technique [1] is used. The GD algorithm is altered so that it allows some oscillations of the bound that is imposed on the overall solution value [7].

The solver splits the problem into two sub-problems: student sectioning and class assignment. In the beginning, students are assigned to individual classes following their course demands and course structure. Students with similar courses are kept together as much as possible, using a simple construction heuristics while sectioning one course at a time. This allows for the computation of potential student conflicts between individual classes, that is, the numbers of students assigned to pairs of classes that are overlapping in time or are one after the other in rooms that are too far apart. During the solver run, classes are assigned in times and rooms while the number of student conflicts is minimized, together with the other penalizations on assigned times, rooms, and violated soft distribution constraints. When the class assignment solver is finished, a local-search technique is used to move students between alternative classes or to swap two students between such classes. During the class assignment, student conflicts between two classes that have some alternatives are weighted less (0.2 of the weight defined in the problem) than the conflicts between classes with no alternatives (i.e., conflicts that cannot be removed by re-sectioning).

More details about the UniTime solver, including various improvements that have been done over the years, are presented in [7].

3 Results

The best and the average penalty from 10 independent runs are presented in the following table. The results were computed using a 2021 model of MacBook Pro with an Apple M1 Max processor, 64 GB memory, OS X 12.3 and Java 8. The solver uses only one CPU core, and the time limit was restricted to two hours. To make use of multiple processor cores, 8 independent runs were done in parallel. UniTime solver cpsolver-1.3.189 was used in the experiment. All the runs were done with the same parameters (using the UniTime's default solver configuration), without any parameter tuning or consideration of a particular instance. The results are compared with the best solutions available at the time of the experiment.

Table 1 shows the results from the experiment compared with the best solutions uploaded at the competition website as of June 27, 2022. The first two columns (named *UniTime*) show the results of this experiment. For each of the late instances, the penalty from the best solution of the 10 independent runs and the average penalty from all the 10 runs is listed respectively. These results are compared with the best solutions from the five competition finalists (columns *Holm* [4], *Rappos* [10], *Gashi* [3], *Er-rhaimini* [2], and *Lemos* [5]), which together with the solver of the author of this paper (column *Müller*) are the six best solvers available at the time of the writing.

Table 1. UniTime solver results compared with the best results on the late instances.

Late	UniTime		Best Result at ITC2019.org					
Instance	2h Best	Average	Holm	Rappos	Gashi	Er-rhaimini	Lemos	Müller
agh-fal17	130 635	133 754.9	140 194		184 030	153 236	142 687	**117 627**
bet-spr18	352 249	353 373.5	**348 524**	360 057	360 437	373 039	353 920	348 536
iku-spr18	40 765	43 082.0	**25 863**	36 711	85 969	70 932	45 537	35 783
lums-fal17	398	411.1	**349**	386	486	558	813	368
mary-fal18	4 924	5 101.4	**4 331**	5 637	7 199	6 944	44 097	4 805
muni-fi-fal17	3 506	3 789.5	**2 837**	3 794	4 712	4 820	4 161	3 180
muni-fspsx-fal17	12 455	15 639.9	12 390	33 001	41 933	104 625	101 317	**10 058**
muni-pdfx-fal17	117 382	125 200.6	**82 258**	151 464	159 203	191 887	151 461	97 449
pu-d9-fal19	46 067	47 441.5	**39 081**	134 009	82 757	70 450	47 543	44 603
tg-spr18	16 140	20 418.2	**12 704**	12 856	15 992	19 738	31 900	14 548

The best know solution of each instance is marked in **bold**. Solutions of the finalists that were improved after the competition has ended are underlined. This means that in these cases a better solution was uploaded on the ITC 2019 website after the competition.

Within the short period of time, the solver was consistently able to produce a solution that is better than the second best solver from the finalists in seven cases. This is indicated by the table colors. The second best results from the five finalists is indicated by blue color. All UniTime results from this experiment that are better than this result are marked with violet color. The remaining three instances (iku-spr18, lums-fal17, and tg-spr18) are the only three late instances that do not have any students.

Better results can be achieved with longer run times and some parameter tuning, which has only been done to some extent. These include some additional improvements, e.g., allowing students to be re-sectioned continuously during the search or removing some of the complexity of the solver (that is not needed for the competition). The best results achieved are listed in the last column (named *Müller*). With these changes, the UniTime solver has produced the best know solution for two late instances (agh-fal17 and muni-fspsx-fal17), and it was able to produce second best results for all but one instance (tg-spr18).

4 Conclusion

The presented solver did not compete in the competition as the author of this abstract is the technical lead and principal developer of the UniTime system and a co-organizer of the competition. Nonetheless, the presented results can provide a good reference of how the UniTime solver would do on the competition problems.

While the UniTime system benefits of almost two decades of research and development, it is good to see that the competitors are able to produce results that are in par or better than what UniTime would produce out of the box using a reasonable runtime.

References

1. Dueck, G.: New optimization heuristics: The great deluge algorithm and the record-to record travel. Journal of Computational Physics **104**, 86–92 (1993)
2. Er-rhaimini, K.: Forest growth optimization for solving timetabling problems. In: ITC 2019: International Timetabling Competition (2020)
3. Gashi, E., Sylejmani, K., Ymeri, A.: Simulated annealing with penalization for university course timetabling. In: Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling - PATAT 2021. vol. 2, pp. 361–366 (2021)
4. Holm, D.S., Mikkelsen, R.Ø., Sørensen, M., Stidsen, T.J.R.: A graph-based MIP formulation of the international timetabling competition 2019. Journal of Scheduling (2022), published: 11 March 2022
5. Lemos, A., Monteiro, P.T., Lynce, I.: ITC 2019: University course timetabling with MaxSAT. In: Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling - PATAT 2021. vol. 1, pp. 105–128 (2020)
6. Müller, T.: UniTime ITC 2019 solver source codes, https://github.com/tomas-muller/cpsolver-itc2019
7. Müller, T.: University course timetabling: Solver evolution. In: Practice and Theory of Automated Timetabling 2016 Proceedings. pp. 263—-282 (2016)
8. Müller, T., Rudová, H., Müllerová, Z.: University course timetabling and international timetabling competition 2019. In: Practice and Theory of Automated Timetabling 2018 Proceedings. pp. 5–31 (2018)
9. Müller, T., Barták, R., Rudová, H.: Conflict-based statistics. In: EU/ME Workshop on Design and Evaluation of Advanced Hybrid Meta-Heuristics (2004)
10. Rappos, E., Thiémard, E., Robert, S., Hêche, J.F.: A mixed-integer programming approach for solving university course timetabling problems. Journal of Scheduling (2022), published: 15 February 2022
11. Rudová, H., Müller, T., Murray, K.: Complex university course timetabling. Journal of Scheduling **14**(2), 187–207 (2011)
12. UniTime: University timetabling – Comprehensive academic scheduling solutions, https://www.unitime.org

Towards A Unified Timetabling Model

Jeffrey H. Kingston

School of Information Technologies, The University of Sydney, Australia
jeff@it.usyd.edu.au
http://jeffreykingston.id.au

Abstract. Timetabling has many sub-disciplines: nurse rostering, high school timetabling, examination timetabling, and so on. Their problems are usually considered to be separate, but there might be advantages in unifying them. This paper describes some advantages of unification, analyses a few design issues, and tentatively suggests some design ideas. The work is incomplete and is offered as a stimulus to discussion, not as a formal proposal.

Keywords: Timetabling · Modelling.

1 Introduction

Timetabling has many sub-disciplines: nurse rostering, high school timetabling, examination timetabling, and so on. Their problems are usually considered to be separate, but there might be advantages in unifying them.

Timetabling problems may be modelled using *events*, which are meetings at which some *resources* (teachers, rooms, and so on) are occupied together at some *times*, and *constraints*, which are rules that limit the time and resource assignments, with penalties (hard and soft) to impose if the rules are broken. This basic common structure is what makes unification possible.

This paper describes some advantages of unification, analyses a few design issues, and tentatively suggests some design ideas. The work is incomplete and is offered as a stimulus to discussion, not as a formal proposal.

2 Why unify?

Why unify at all? Isn't it sufficient if research supervisors keep abreast of developments in the various sub-disciplines?

There is no pressing need to unify. Good progress is being made without it, and solvers for the unified problem are unlikely to perform better than solvers for specific problems. But there are other reasons for unifying, as follows.

Even if solvers do not benefit, other kinds of software might: software for evaluating solutions like the HSEval web site [6], or solve platform software like KHE [7]. These two systems currently support high school timetabling and nurse rostering, but nothing else.

Specification work done in one sub-discipline may be valuable in another. The XESTT nurse rostering model [10] was based on the XHSTT high school timetabling model [15]. It is argued below that UniTime [20], the leading university course timetabling model, would benefit from some XHSTT ideas.

Unification might also shed light on the scientific question of how related the sub-disciplines are. Student sectioning, for example, as found in universities and some high schools, seems to be different from everything else [8]. But is that really so? Such questions have little practical importance, but surely there is room in our discipline for some intellectual curiosity about them.

Perhaps the strongest reason concerns problems that fall outside, or span across, the usual categories: hospital problems other than nurse rostering, high schools whose senior students follow a university-style curriculum, and so on. These problems are widespread, but they are marginalised at present. A unified model would help to specify them and bring them into the mainstream.

Our main concern in this paper is with what might be called 'internal models': models concerned with precisely specifying instances, ready for solving. There are also 'external models', which express instances in terms that end users are familiar with. Conversion between external and internal is a significant issue that we do not address here; it is in practice aided by software, such as instance translators and interactive user interfaces.

3 Existing models

The author knows of no recent models that try to cover all or most sub-disciplines of timetabling. So this paper is inspired by analyses of several sub-disciplines and their leading models. These are the subject of this section.

Many years ago several authors designed unified timetabling models built on mathematical foundations, for example [5, 13]. This work proved to be useful only in showing what not to do. Its excessive generality was avoided in the following generation of models, many of which are discussed below. Although our proposal could be described as general because it spans multiple sub-disciplines, it is actually just as concrete as the models we'll discuss now.

High school timetabling was very fragmented for many years. Virtually no data was exchanged, at least at the international level, before the advent of XHSTT [15]. It improved the situation greatly, opening the way to the Third International Timetabling Competition [16], and also to the XHSTT-2014 data set [14], which contains real-world instances from many countries.

To this author, XHSTT's major choices still seem right ten years later. There are opportunities for increasing generality and reducing verbosity, but XHSTT's four-part structure (times, resources, events, and constraints) has worked well, and several specific ideas deserve to be passed on to new models: the resource concept (the old idea [1] that all entities that attend events, including teachers, rooms, nurses, and so on, are basically the same); the ability to define arbitrary sets of times and resources and use them in constraints; and the way that cost calculations are specified transparently and uniformly over all constraints.

Examination timetabling was the first sub-discipline to have a widely used model, the one for the Toronto instances [17, 18]. Its constraints are mainly local workload limits. It has been criticised for incompleteness (for example, it omits rooms). The more complete models that have emerged more recently have added features already found in other models, such as university course timetabling.

Nurse rostering has a solid history of modelling and data sharing. In fact, it arguably has too many models rather than too few. These models were analysed during the design of the XESTT nurse rostering format [10], which turned out to be XHSTT with some enhancements, including two new constraints. Some recently suggested nurse rostering constraints [4] are expressible in XESTT.

University course timetabling has only one widely cited model: UniTime. Its creators have investigated the requirements of many universities (at least ten [12]), and their current model seems to be close to complete. This author has used the UniTime university course timetabling model, version 2.4 [20] as his main source for university course timetabling, along with the closely related International Timetabling Competition 2019 (ITC2019) model [12].

University course timetabling has several challenging features. Its instances can be very large, with thousands of students, perhaps, each of whom needs an individual timetable that may vary from week to week. Then there is student sectioning, which can be more than assigning each student to one section of a course, since a course may have multiple configurations, each with a variety of parts (lectures, tutorials, and so on), logically related in various ways. Finally there are room capacity and walking time constraints.

Generalization would improve UniTime. Introducing the resource concept, and expressing the common parts of constraints uniformly, are two obvious steps. Arbitrary sets of times would also help: several of UniTime's 'group constraints' are identical except for the sets of times they apply to.

Sports scheduling needs one event for each fixture, holding two preassigned resources (the teams) and possibly a third one (the venue). The well-known travelling tournament problem [3] is precisely specified, but not very general. The only general model is the RobinX round-robin tournament model [19, 21]. Its creators say that it is based on analyses of many real-world tournaments.

RobinX defines 21 constraints. Some are familiar from other models. For example, there is a travel distance constraint like the university constraint on walking time. Others are more familiar than they seem. For example, RobinX schedules have a particular large-scale structure, such as *mirrored round-robin*. These are enforced by constraint `SE2`, which says that if one event is assigned a certain time, a corresponding event must be assigned a certain other time. This is similar to the nurse rostering 'complete weekends' constraint, which says that nurses who work on Saturday or Sunday must work on both days.

However, there are also constraints that do not seem to fit well with other models. Some RobinX constraints aim to produce a *fair* schedule: one which spreads defects evenly among the teams. The XHSTT approach to fairness is to specify an acceptable maximum number of defects and use a quadratic cost function to strongly penalize larger deviations. RobinX tries to minimize the maximum, over all pairs of teams, of the difference in value. And some RobinX constraints, notably `CA5` and `GA2`, seem to be far removed from other models. Altogether, then, sports scheduling will seriously test a unification project—but it is important, for that very reason.

4 Issues

This section discusses some issues raised by unification. The discussion often takes XHSTT as its starting point, but only because it is familiar, precisely specified, and already supports much of what is needed.

4.1 Times

As mentioned, the various sub-disciplines have a basic similarity that makes unification possible. But they model time differently. High school timetabling has *periods*, which are disjoint time intervals. RobinX is much the same. Nurse rostering has *shifts*, which are longer intervals that may overlap. UniTime has hierarchically defined sets of intervals, such a '9-10am Monday, Wednesday, and Friday, starting in Week 2 of semester'. But still there is a basic similarity: all models deal with intervals, or finite sets of intervals, of real time.

What seems to be needed is a finite set of *times*, each representing a finite set of intervals of real time. Importantly, each event must choose its times from this finite set: no timetabling problem known to the author allows solutions to choose arbitrary time intervals. In this way, time is discretized.

There are implementation efficiency issues in calculating overlaps between times. UniTime addresses these issues by requiring the sets of intervals that can be times to have a regular hierarchical structure (same time each week, etc.). A general model should allow a time to represent an arbitrary finite set of intervals, but in a way that optimizes calculations involving regular structures. The author has designed a time model of this kind (unpublished). In this model, the hierarchy (semesters, weeks, days) is arbitrary and is defined within the instance, and one time is very like a finite set of UniTime times.

4.2 Resources

Resources are entities that attend events: teachers, rooms, nurses, and so on. XHSTT allows any number of resource types to be defined (`Teacher`, `Room`, and so on), along with any number of resources of each type. Arbitrary sets of resources, called *resource groups*, may be defined and used in constraints.

It seems to be necessary to add resource attributes, such as room capacity and location, since some constraints in models other than high school timetabling depend on them. XHSTT does not offer resource attributes, except membership in a resource group, which is equivalent to a Boolean attribute.

Some attributes can be simulated by constraints. For example, a teacher's workload limit is an attribute of the teacher, but it is expressed in XHSTT by the upper limit of a limit workload constraint that applies to the teacher. But even if this could be made to work in all cases, it is unnatural and can become very verbose. One would not want to add an entire constraint for each pair of rooms, for example, just to hold the walking time between them.

4.3 Events

An *event* is a meeting between some resources at some times. The times and resources may be preassigned, or left to a solver to assign. We think of an event as a variable which may be assigned a set of times, but which also contains other variables called *tasks*, each of which may be assigned a set of resources.

An XHSTT task may be assigned only one resource, but problems other than high school timetabling need a more flexible design: a task that may be assigned several resources. Their number may be constrained but need not be fixed. This is the norm in nurse rostering, for the nurses assigned to a shift, and in university course timetabling, for the students assigned to a lecture.

There is a big difference between, say, a university course meeting at 9am on Mondays, Wednesdays, and Fridays, and a high school course meeting at 9am on Mondays, 11am on Wednesdays, and 3pm on Fridays. The university course meets at rigidly coupled times, so it could be represented by an event assigned a single time containing three intervals. The high school course meets at separately chosen times, so it must be represented by an event assigned three times, each containing one interval. Care is needed over resource stability: a course must be able to meet in different rooms at different times, if desired.

4.4 Constraints

Similar constraints from different sub-disciplines should be merged into a single constraint (possibly generalized) in the unified model, where possible. Several familiar constraints are easily merged, including constraints that limit events and tasks to preferred times and resources, and constraints on resources that prohibit clashes, specify unavailable times, and impose workload limits.

Here is a deeper example. Consider a constraint that limits the length of sequences of consecutive free days, as is found in nurse rostering. This is based on a sequence of sets of tasks, one set for each day, containing the tasks that a given resource is assigned on that day. Maximal sequences of consecutive empty sets must be identified and the ‘length’ function applied to them.

Now consider a constraint that limits the number of idle times that a resource may have on one day (times when the resource is not busy, provided it is busy earlier and later in the day). Again, this is based on a sequence of sets of tasks, one for each time of the day, and again sequences of empty sets of tasks must be identified. The function is different (the total length of empty sequences not at the ends, rather than the length of each sequence taken separately), but the two constraints have the same underlying structure.

Or consider a nurse rostering constraint requiring at least one senior nurse to be on duty at 3pm. XESTT models this by a *limit resources constraint*, which takes some tasks and a set of resources, and imposes lower and upper limits on the number of assignments to the tasks of resources from the set. In the example, the tasks are all tasks running at 3pm (possibly from several shifts), the resources are all senior nurses, and there is a lower limit of 1.

This constraint can also be used to specify that a given university student should attend one of the laboratory sessions of a given course. The tasks are the student tasks of the course's laboratory events, the resources consist of just the given student, and there are lower and upper limits, both 1.

But however cleverly we find such connections between constraints, a serious problem remains. Several sub-disciplines have constraints that cannot be fitted to a general pattern, because they use functions, such as Pythagorean distance, or the RobinX fairness function, which are essentially arbitrary. When all of them meet in a unified model, the result might be chaos.

A possible way out is to provide a single, uniform method of selecting the tasks to which a constraint applies. This would select all tasks from a given set of tasks S which are assigned resources from a given set of resources R and lie in events assigned a time from a given set of times T. Then the only arbitrary aspect would be the function to apply to the selected tasks, which could be taken from a long fixed list: the number of tasks, their total duration in minutes, the walking time between their locations, and so on. The author has designed constraints of this kind (unpublished) which can express all of the constraints of all of the models described in Section 3. Arbitrarily complex *constraint trees* are included; they are needed for student sectioning.

4.5 Reducing verbosity

It seems inevitable that a unified model will ultimately be expressed in XML. XML-based models are notorious for leading to large, verbose files.

One way to reduce verbosity is to reduce repetition. For example, XESTT allows a constraint to be defined for one day (or week etc.), and then annotated with 'and repeat this every day (or every week)'. This could be generalized to nested iterators ('for each resource r in set R, for each time t in set T, ...').

Verbosity can also be reduced in a more local way. For example, XHSTT consumes a lot of space saying whether a constraint is hard or soft, what its weight is, and so on. All this could be replaced by a single string, for example

```
cost="count:2-5|s20"
```

which says (reading the bar as 'or else'), '`count` (the number of tasks) must lie between 2 and 5, or else a soft cost equal to the distance from these limits times 20 is incurred'.

Lower and upper limits, like 2 and 5, appear in many XHSTT constraints but not all. It seems to be a good idea to give them to all constraints: it is more general and more uniform and creates no problems. It was pointed out long ago that limits on the measures of sets are basic in timetabling [11].

5 Conclusion

This paper has offered some ideas for unifying the sub-disciplines of timetabling. Although the ideas are incomplete, the author hopes that they will stimulate interest in a field which does not really exist, but perhaps should.

References

1. Appleby, J. S., Blake, D. V., Newman, E. A.: Techniques for producing school timetables on a computer and their application to other scheduling problems. The Computer Journal **3**, 237–245 (1960)
2. Ceschia, S., Nguyen T. T. D., De Causmaecker, P., Haspeslagh, S., Schaerf, A.: Second international nurse rostering competition (INRC-II), problem description and rules. oRR abs/1501.04177 (2015). http://arxiv.org/abs/1501.04177.
3. Easton, K., Nemhauser, G., and Trick, M.: The traveling tournament problem description and benchmarks. In: Walsh, T. (ed.) Principles and Practice of Constraint Programming (CP 2001), 580–584, Springer, Berlin, Heidelberg (2001)
4. Gätner, J., Bohle, P., Arlinghaus, A., Schafhauser, W., Krennwallner, T., Widl, M.: Scheduling matters: some potential requirements for future rostering competitions from a practitionerâĂŹs view. In: PATAT 2018 (Twelfth international conference on the Practice and Theory of Automated Timetabling, Vienna, August 2018)
5. Kingston, J. H.: Modeling timetabling problems with STTL, Springer Lecture Notes in Computer Science **2079** 309 (2001)
6. Kingston, J. H.: The HSEval High School Timetable Evaluator, http://jeffreykingston.id.au/cgi-bin/hseval.cgi (2010)
7. Kingston, J. H.: KHE web site, http://jeffreykingston.id.au/khe (2014)
8. Kingston, J. H.: Integrated student sectioning. In: PATAT 2014 (Tenth international conference on the Practice and Theory of Automated Timetabling, York, UK, August 2014, 489–492 (2014)
9. Kingston, J. H.: XESTT web site, http://jeffreykingston.id.au/xestt (2018)
10. Kingston, J. H.: A unified nurse rostering model based on XHSTT. In: PATAT 2018 (Twelfth international conference on the Practice and Theory of Automated Timetabling, Vienna, August 2018)
11. Kitagawa F., Ikeda, H.: An existential problem of a weight-controlled subset and its application to school timetable construction. Discrete Mathematics **72**, 195–211 (1988)
12. Müller, T., Rudová, H., and Müllerová, Z.: University course timetabling and International Timetabling Competition 2019. In: PATAT 2018 (Twelfth international conference on the Practice and Theory of Automated Timetabling, Vienna, August 2018)
13. Ozcan, E.: Towards an XML based standard for Timetabling Problems: TTML. In: Multidisciplinary Scheduling: Theory and Applications, Springer Verlag 163 (24), May 2005
14. Post, G.: XHSTT web site, http://www.utwente.nl/ctit/hstt/ (2011)
15. Ahmadi, S., Daskalaki, S., Kingston, J. H., Kyngäs, J., Nurmi, C., Post, G., Ranson, D., Ruizenaar, H.: An XML format for benchmarks in high school timetabling. Annals of Operations Research **194**, 385–397 (2012)

16. Post, G., Di Gaspero, L., Kingston, J. H., McCollum, B., Schaerf, A.: The Third International Timetabling Competition. In: PATAT 2012 (Ninth international conference on the Practice and Theory of Automated Timetabling, Son, Norway, August 2012), 479–484 (2012)
17. Qu, R., Burke, E., McCollum, B., Merlot, L., Lee, S.: A survey of search methodologies and automated system development for examination timetabling. Journal of Scheduling **12** 55–89 (2009)
18. Qu, R.: Benchmark data sets in exam timetabling, http://www.cs.nott.ac.uk/˜rxq/data.htm. (2012)
19. RobinX: An XML-driven classification for round-robin sports timetabling, http://www.sportscheduling.ugent.be/RobinX/index.php (2019). Note: the constraints are listed at http://www.sportscheduling.ugent.be/RobinX/threeField.php
20. The Comprehensive University Timetabling System, www.unitime.org, especially www.unitime.org/uct_dataformat_v24.php (2018)
21. Van Bulck, D., Goossens, D., Schönberger, J., Guajardo, M.: RobinX: an XML-driven classification for round-robin sports timetabling. In: PATAT 2018 (Twelfth international conference on the Practice and Theory of Automated Timetabling, Vienna, August 2018)

Sustainable energy aware industrial production scheduling

Panayiotis Alefragis[1] [0000–0002–1313–1750], Konstantinos Plakas[2], Ioannis Karambinis[2], Christos Valouxis[2], Michael Birbas[2], Alexios Birbas[2], and Christos Gogos[2]

[1] Electrical & Computer Engineering Department, University of Peloponnese, Patras, Greece alefrag@uop.gr
[2] Electrical & Computer Engineering Department, University of Patras, Patras, Greece
{plakas,lkarabinis,mbirbas,birbas}@ece.upatras.gr, cvalouxis@upatras.gr
[3] Informatics & Telecommunications Department, University of Ioannina, Arta, Greece
cgogos@uoi.gr

Abstract. This paper describes a software component that was developed to solve the energy aware production scheduling problem. Firstly, day-ahead energy prices and energy production mix are forecast using publicly available data. Secondly, a Constraint Programming (CP) scheduling model was developed in order to minimize production cost and CO_2 emissions. The paper presents a hybrid methodology to forecast day ahead energy prices, a simplified CP model and preliminary results from the application of energy aware scheduling algorithms to a 3D additive printing industrial production use case.

Keywords: production scheduling · Constraint Programming · energy price forecasting

1 Introduction

Emerging industrial sustainability domain dictate new production efficiency interventions since manufacturing plants are facing increasing pressure to reduce their carbon footprint, driven by concerns related to energy costs and climate changes. To create an energy sustainable environment in the industrial production ecosystem multiple aspects should be taken into account and a hierarchical decision-making process should be implemented. Supply chain, production planning and scheduling and maintenance planning inter-wind with floor-shop energy monitoring, gas emissions tariffs tracking and energy market prices to create a sustainable manufacturing system. In this paper we focus on the production planning and scheduling aspect where day-ahead energy prices are forecast and used.

2 Related work

Customer Environmental Awareness (CEA) urge energy-intensive manufacturers into creating an energy saving strategy. In [13], several mathematical models have been developed to support enterprises that are facing choices of self-saving, shared savings and guaranteed savings to determine the optimal strategies of improving energy efficiency when CEA is considered. At production level, production scheduling is critical in decision making process while been computationally demanding and sensitive on data availability and credibility. Many decision support approaches has been proposed. During the FP7 ARTISAN project an energy-aware hierarchical optimization DSS that used an Iterated Local Search with application to the textile industry was implemented [14]. A rescheduling method is proposed to tackle the problem of reducing energy consumption when resolving dynamic flexible job-shop scheduling problem under machine breakdowns [11]. In [4], a hybrid mathematical model and an NGSA-II multi-objective genetic algorithm is used to address integrated production scheduling, maintenance planning and energy controlling for sustainable manufacturing systems. A recent trend is the collaboration between manufacturing enterprises and energy providers. In [12], a multi-agent architecture aimed at elaborating predictive and reactive energy-efficient scheduling through collaboration between cyber physical production and energy systems is proposed. A framework of data-driven sustainable intelligent/smart manufacturing based on demand response for energy-intensive industries is proposed in [8] where a framework is implemented to support multi-level demand response models that address machine, shop-floor and factory levels. A framework to allow collaboration between energy providers and manufacturing companies is proposed in [10]. 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.

3 Day Ahead Energy Price Forecasting

Electricity energy prices and source mix varies based on the time of the day and the period of the year. Synchronizing energy hungry production tasks with "green" energy availability is of outmost importance for sustainable production. To achieve sustainability, the variability of the energy production should be incorporated in the production scheduling process. Fig. 1 presents a typical intraday electricity production and demand variability [15].

To implement a forecast on the day ahead energy cost data from multiple sources have to be acquired. In addition, for every supported energy market, a different forecast model should be created as prices per market usually follow different patterns. The implemented algorithms use as input the following data:

1. Electric energy production data
   (a) Day ahead historical predictions
   (b) Realized historical production

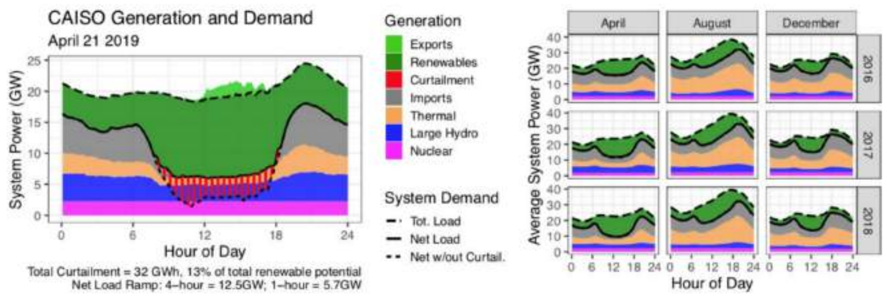


Fig. 1. Typical Electricity Production variability [15]

   (c) Connected energy markets predictions if available
   (d) Published energy production estimations
2. Electric energy demand data
   (a) Day ahead historical demand predictions
   (b) Realized historical consumption
   (c) Up and Down Reserves
   (d) Connected energy markets predictions if available
   (e) Published energy demand by energy market operator
3. System Marginal Prices per energy market supported
   (a) Day ahead historical predictions
   (b) Realized historical SMP
   (c) Connected energy markets SMP if available
4. Weather data
   (a) Temperature, wind speed, solar radiation etc.
   (b) Outside temperature, relative humidity, etc.
5. Miscellaneous data
   (a) Holidays, working days, weekdays, year period

All input was sourced from publicly available data sources. We performed a feature selection analysis [3] to determine the most important features from the available data sets. Fig. 2 present the features importance for the prediction of the energy prices. It can be observed that due to the intense variability in the behaviour of the energy market players the most important feature component is the mean price of the last 7 days, which was not the most significant component if the same analysis was performed some years ago when the energy market was more stable . Multiple forecasting algorithms were used to create a hybrid ensemble prediction model that exhibits a more robust performance compared to individual forecasting algorithms. The individual forecasting algorithms that were used are regression methods (OLS, Ridge, Lasso) [2], Tree based methods (Random Forest) [9] and RNN(LSTM) [7]. Fig. 3 shows a visual representation of the predicted System Marginal Price (SMP) for the day-ahead Greek Energy Market versus the actual realized values for the Greek energy market.

4 Energy-aware production scheduling problem

The Production Scheduling Component is part of the ENERMAN Intelligent Decision Support System and implements a number of energy aware production mapping and scheduling algorithms. The component provides as output the assignment and scheduling of jobs to machines, machine operational mode per task and produces the estimated total energy consumption, energy cost and the estimated total CO_2 emissions for the produced solution.

The Production Scheduling Component expects in the final version the following inputs:

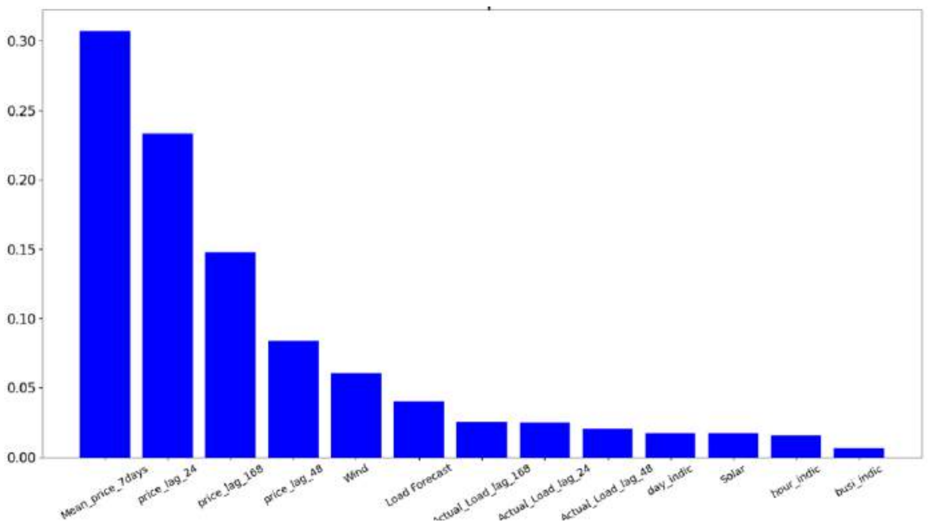
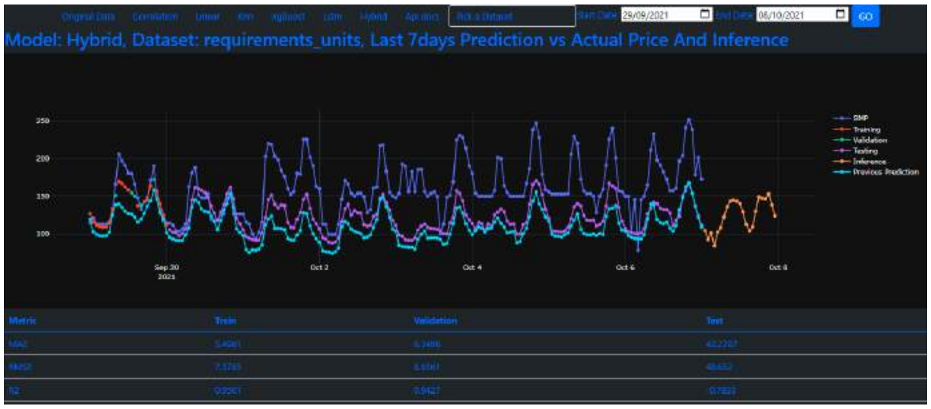


Fig. 2. Features Importance Analysis



Model: Hybrid, Dataset: requirements_units, Last 7days Prediction vs Actual Price And Inference

Metric	Train	Validation	Test
MAE	5.4081	6.3486	42.2207
RMSE	7.1785	8.6567	48.652
R2	0.9583	0.9427	0.7833

Fig. 3. Hybrid Energy Market Price Forecasting

1. Task related data
   (a) Energy consumption per task for each compatible machine
   (b) Execution time for each compatible machine and operation mode
   (c) If it is an obligatory or optional job
   (d) Job execution flexibility (availability, deadline, time windows)
2. Machine related data
   (a) Energy consumption function at different operational states
   (b) Machine availability (planned maintenance, operating personnel etc)
   (c) Machine pre-allocated capacity
3. Production description data
   (a) Feasibility of schedulable time periods (Available working days & hours)
   (b) Time horizon and related penalties if not a schedule meets makespan requirements
   (c) Dependencies between tasks
   (d) Intermediate products storage availability and feasible time windows
   (e) Transfer time and energy cost to move intermediate products between machines / sites
4. Energy prices per hour for the relative energy markets
5. Energy production mix per hour for the relative energy markets

Some of the implemented heuristics and ILP algorithms are based on ideas presented in [6], [1] and [5] but are outside the scope of the current paper. In the current paper, a reduced version of a Constraint Programming model to solve the problem is presented. The CP model only supports one operational mode per machine and 2b, 3b-3e inputs are ignored. The model will be extended in the future to support the full problem definition.

Let $T = \{1, 2, \ldots, t\}$ be a number of independent non-preemptive tasks and $M = \{1, 2, \ldots, m\}$ be a set of heterogeneous machines. The goal is to allocate and schedule all tasks to the machines while minimizing the total cost and/or the CO_2 emissions. Each task can only be executed on a subset $M_t \subseteq M$ of the available machines and due to the heterogeneity of the machines, the execution time D_{ij} and consumed energy C_{ij} of task t_i on machine m_j are not the same. Let $T_m \subseteq T$ be the tasks that can be executed on machine m. Let the variables s_i and e_i denote the start and end time of task t_i, while the variable x_{ij} is a binary decision variable that equals to 1 when task t_i is assigned to machine m_j, otherwise $x_{ij} = 0$. Using s_i, e_i, D_{ij} and x_{ij} an optional fixed size interval variable I_{ij} is introduced for each $t_i \in T_m$ and $m_j \in M_t$. In addition, each task $ti \in T$ is associated with a resource envelope type $r_l \in R$ and for each type pair $(r_l, r_n) \in R$ different energy consumption H_{ln} and execution time G_{ln} is defined to represent the setup process between tasks. Finally, for each machine $m \in M$ and for every pair $(t_i, t_k) \in T_m$ a pair of Boolean variables p_{mik}, q_{mik} is introduced that help us to identify that task t_i precedes t_k. The above problem can be formulated as a CP model and optimal solution for realistic problems can be achieved in minutes. A simplified version of the scheduling problem model is as follows:

$$\forall\, t_i \in\, T, \sum_{m \in M_t} x_{im} = 1 \tag{1}$$

$$\forall\, m \in\, M, \forall\, t_i, t_k \in\, T_m, i \neq\, k,\; p_{mik} = 1 \;\leftrightarrow\; x_{im} + x_{km} = 2 \tag{2}$$

$$\forall\, m \in\, M, \forall\, t_i, t_k \in\, T_m, i \neq\, k,\; p_{mik} = 0 \;\leftrightarrow\; x_{im} + x_{km} < 2 \tag{3}$$

$$\forall\, m \in\, M, \forall\, t_i, t_k \in\, T_m, i \neq\, k,\; q_{mik} = 1 \;\leftrightarrow\; s_i \leq\, s_k \tag{4}$$

$$\forall\, m \in\, M, \forall\, t_i, t_k \in\, T_m, i \neq\, k,\; q_{mik} = 0 \;\leftrightarrow\; s_i \geq\, s_k \tag{5}$$

$$\forall\, m \in\, M, \forall\, t_i, t_k \in\, T, i \neq\, k, p_{mik} = 1,\; q_{mik} = 1 \leftrightarrow\; s_i + D_{im} + G_{r_i r_k} \leq\, s_k \tag{6}$$

$$\forall\, m \in\, M, \forall\, t_i, t_k \in\, T, i \neq\, k,\; p_{mik} = 1,\; q_{mik} = 0 \leftrightarrow\; s_k + D_{km} + G_{r_k r_i} \leq\, s_i \tag{7}$$

Equation (1) ensures that each task is assigned to exactly one machine. A non-overlapped in time execution sequence between two tasks t_i, t_k is imposed by (6) and (7), when they are assigned to the same machine. The full version of the model includes additional interval variables that act like pre-scheduled tasks in the model and prohibit the real tasks to be scheduled during a machine unavailability periods, adaptation to constraints (6) and (7) are required to take into account these pre-scheduled tasks.

Multiple objectives are supported. For example, if we want to minimize the total energy consumption the objective is set to

$$min \sum_{i \in T, j \in M} x_{ij} * C_{ij} \tag{8}$$

If we want to minimize total CO_2 emissions or the total energy cost, an extension to the above model is required. Given a time horizon L where for each time period $[l_a, l_b]$ we have forecast the cost of energy S_{ab} and the renewable energy percentage P_{ab} in the available energy, for each machine $m_j \in M$ and for every task $t_i \in T_m$ an extra array of variables E_{ij} is introduced that for each point in time in the time horizon calculates the cost or the CO_2 emissions. For example, if we want to minimize the total energy cost the objective function can be written

$$min \sum_{i \in T, j \in M_t} x_{ij} * E_{ij}[s_i] \tag{9}$$

Given the solution of the optimization model is part of a decision process multiple objectives can be combined using weights introduced by the user and alternative solutions can be generated.

Figure 4 presents optimal solutions for the two aforementioned objectives for a small example, where grey areas are prohibited scheduling periods for each machine, blue tasks have $A0$ resource envelope while orange tasks have $A1$ resource envelope. To demonstrate the effect of energy prices we used a linear decreasing cost per minute. It can be observed in the right Gantt chart that the tasks are scheduled as right as possible while still satisfying where lower energy prices are realized.

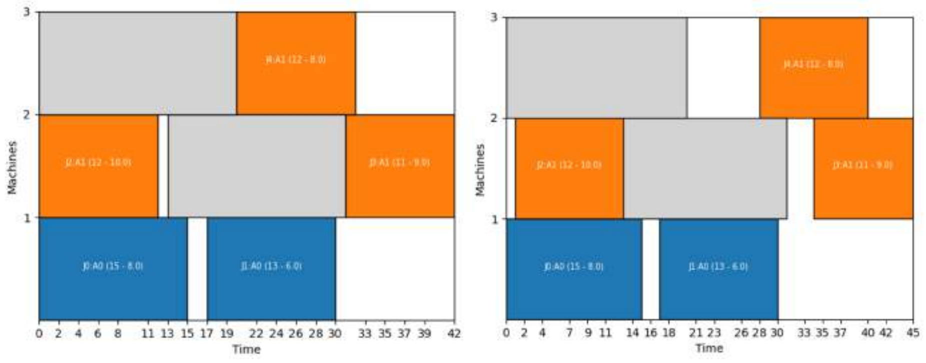


Fig. 4. Solutions using different objective functions example.

5 Use case description and preliminary results and conclusions

The Enerman project has multiple use cases that must be supported by the production scheduling algorithms. Problem data originate from industrial partners that have energy demanding production processes like automotive manufacturing, semiconductor production, steel and aluminum production, food processing and 3D additive components manufacturing. Preliminary results originate from a 3D metallic component printing process were machines with different laser technologies and variable performance capabilities are present in the production environment. Each task is independent but the setup time between tasks on the same machine depends on the powdered material used to manufacture the previous component. If the same material is used the setup time can be reduced but the setup time never reaches 0 as some cleaning between jobs is required. In addition, each machine has different operational points for the laser that allow more energy efficient production to be realized by prolonging the production time. Preliminary results that use historic production schedules and forecast energy prices for the specific energy market that the company is operating show that if the tasks have used the introduced optimization model to produce an alternative production schedule that aligned the more energy demanding tasks with low cost energy periods, a significant cost reduction could be achieved. To assess the effect of the forecast quality, we performed a lower bound optimal scheduling were the realized SMP values were used instead of using the forecast values. It was observed that using the forecast energy prices a reduction of about 7% in the production cost was achieved compared to a reduction of about 9% that was achieved if we have predicted exactly the realized SMP prices, while in both case a model optimal solution has been found.

6 Conclusions and future work

This short paper presents preliminary work over the problem of minimizing the cost of production scheduling in an industry setting. Industries that are heavily depended on the energy cost for their operation need an automated way of avoiding suboptimal schedules. Future prices are difficult to predict. Thus, it is very important to generate high quality forecasts to be used as input to the scheduling algorithm. In this context, we propose a CP model that is able to produce good schedules taking into account the forecast electricity prices. The CP optimization model will be extended to support dependencies between tasks, intermediate product storage capacity constraints, time windows for intermediate product storage and time and energy cost for the transfer of the intermediate product between machines.

Acknowledgements The work in this paper is partially financed by H2020-Enerman project, European Union's Horizon 2020 research and innovation programme (grant agreement No 958478)

References

1. Alefragis, P., Gogos, C., Valouxis, C., Goulas, G., Voros, N., Housos, E.: Assigning and scheduling hierarchical task graphs to heterogeneous resources. Proceedings of the 10th Practice and Theory of Automated Timetabling (PATAT) (2014)
2. Antonio, G.P., Maria, P.O., Sanchez-Monedero, J., et al.: Ordinal regression methods: Survey and experimental study. IEEE Transactions on Knowledge & Data Engineering **28**(1), 127–146 (2016)
3. Chandrashekar, G., Sahin, F.: A survey on feature selection methods. Computers & Electrical Engineering **40**(1), 16–28 (2014)
4. 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)
5. Emeretlis, A., Theodoridis, G., Alefragis, P., Voros, N.: Static mapping of applications on heterogeneous multi-core platforms combining logic-based benders decomposition with integer linear programming. ACM Transactions on Design Automation of Electronic Systems (TODAES) **23**(2), 1–24 (2017)
6. Gogos, C., Valouxis, C., Alefragis, P., Goulas, G., Voros, N., Housos, E.: Scheduling independent tasks on heterogeneous processors using heuristics and column pricing. Future Generation Computer Systems **60**, 48–66 (2016)

7. Li, W., Becker, D.M.: Day-ahead electricity price prediction applying hybrid models of lstm-based deep learning methods and feature selection algorithms under consideration of market coupling. Energy **237**, 121543 (2021)
8. Ma, S., Zhang, Y., Liu, Y., Yang, H., Lv, J., Ren, S.: Data-driven sustainable intelligent manufacturing based on demand response for energy-intensive industries. Journal of Cleaner Production **274**, 123155 (2020)
9. Mei, J., He, D., Harley, R., Habetler, T., Qu, G.: A random forest method for real-time price forecasting in new york electricity market. In: 2014 IEEE PES General Meeting| Conference & Exposition. pp. 1–5. IEEE (2014)
10. 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)
11. Nouiri, M., Bekrar, A., Trentesaux, D.: Towards energy efficient scheduling and rescheduling for dynamic flexible job shop problem. IFAC-PapersOnLine **51**(11), 1275–1280 (2018)
12. Nouiri, M., Trentesaux, D., Bekrar, A.: Towards energy efficient scheduling of manufacturing systems through collaboration between cyber physical production and energy systems. Energies **12**(23), 4448 (2019)
13. Ouyang, J., Fu, J.: Optimal strategies of improving energy efficiency for an energy-intensive manufacturer considering consumer environmental awareness. International Journal of Production Research **58**(4), 1017–1033 (2020)
14. Plitsos, S., Repoussis, P.P., Mourtos, I., Tarantilis, C.D.: Energy-aware decision support for production scheduling. Decision Support Systems **93**, 88–97 (2017)
15. Satchwell, A., Cappers, P., Deason, J., Forrester, S., Frick, N.M., Gerke, B.F., Piette, M.A.: A conceptual framework to describe energy efficiency and demand response interactions. Tech. rep. (07 2020)

Noname manuscript No.
(will be inserted by the editor)

On the call intake process in service planning

Gerhard Post · Stefan Mijsters

Received: date / Accepted: date

Abstract The call intake process for field service assigns a time window to a client's request for a service. What time window is chosen has influence on the quality of the total planning. The quality is measured by the number of planned requests, the lateness of planned requests and the total travel time.

We constructed challenging datasets, for 25 engineers and an average of 200 tasks per day. We use simulations on these datasets to study the effect of different strategies. Aspects that turn out to be beneficial are: ignore the severity of lateness, give preference to empty shifts, cluster tasks in a shift, and use intermediate optimization.

Keywords Vehicle routing, field service, clustering, call intake, scheduling

1 Introduction

There is a large body of literature on Vehicle Routing Problems (VRPs) with different types of settings and constraints. For a survey on rich VRPs, see for example [Caceres-Cruz et al (2014)]. Here we are interested in the area usually called 'field service', where in addition to the usual set-up the shifts (working hours) of the (field) engineers are fixed beforehand. Having fixed employee shifts might seem not very relevant, but it is: the main objective in VRP usually is to minimize the number of vehicles first, and secondary the total travel distance. In the case we consider here, this 'number of vehicles' is fixed: each shift of an employee is open for service tasks, and there is no gain

Gerhard Post
PCA, Klipperweg 19, 8102 HR Raalte, The Netherlands
and
Department of Applied Mathematics, University of Twente, The Netherlands
E-mail: g.f.post@utwente.nl

Stefan Mijsters
PCA, Klipperweg 19, 8102 HR Raalte, The Netherlands

in leaving a shift empty. On the contrary, spreading the work can be one of the objectives.

More in particular, we consider the situation in which a company (or 'service provider') provides services to its clients upon request. Such a request leads to a task that is executed by an engineer of the company at the address of the client. A task can be a small repair (in case the company is a housing corporation or an installation company), a damage assessment (in case of an insurance company), or some other service. We assume in our datasets that the tasks require between 30 minutes and 60 minutes of service time. Since tasks are executed at the addresses of the clients, the engineer has a daily shift, usually starting at the home address of the engineer, visiting approximately 10 clients, and driving to the end destination, which is either the home or the company address[1].

2 The routing phase

For the moment we assume that somehow all tasks are defined. In particular a task can require a skill, has an (expected) service time, and a time window defines the earliest start time and the latest start time. Usually, time windows originate from the so-called 'block times' that the company applies to all service requests. These block times usually are two to four hours long. If all tasks have time windows within one day, the routing problem essentially consists of daily problems. In particular, tomorrow is relevant; we need to finalize the schedules and inform the field engineers on the routes.

Quite similar to the Maintenance Personnel Scheduling Problem (MPSP) in [Misir et al (2015)], the tasks assigned to a shift of the engineer must obey the following constraints:

- Pre-assigned tasks and other appointments should be scheduled as given.
- The field engineer must be skilled for the task.
- A task should start within its time window.
- The expected travel times between the scheduled tasks should be respected.
- The start address and the finish address of a shift can be different; traveling from the start address to the first task, and traveling from the last task to the finish address can sometimes partly be done in private time.
- If the shift has a break, the duration of the task or travel is extended with the duration of the break. In particular, a task or travel cannot start during the break, but can start before and finish after the break.
- If a task is assigned to a region, the engineer should work in this region. This region can be a geographical region, but also an administrative region. An engineer can be assigned to different regions during the week and even during the day. Multiple regions at the same time are possible.

[1] When the routing aspect of a shift with tasks is being discussed, we might use the word *route*.

– A task can have a pre-assigned engineer, which must be respected, or a preferred engineer. Assigning a preferred engineer takes precedence over minimizing the travel time.

Since the shifts are fixed, the personnel rostering constraints mentioned in [Misir et al (2015)] are not relevant. The main objective is first: to assign as many tasks as possible, second: take preferences into account, and third: minimize the total travel time. Note that minimizing the travel time reduces the direct costs, but also might create space for an extra task in a shift. This is not relevant anymore when we optimize the planning for tomorrow, but for later days it might create space to assign additional tasks.

3 The call intake process

In the previous section we discussed the routing problem to be solved. As said there, the routing problem usually split in daily problems, because of the time windows that are attached to the requests. The assignment of the time windows is done in the phase we call the 'call intake process'. This process for the service planning differs quite of lot from home deliveries, as discussed in [Strauss et al (2021)] and [Visser (2019)]:

– Often home deliveries are for today or tomorrow. In service planning the time scale usually is in weeks.
– In home deliveries the requests do not require skills, and can be assigned to all resources.
– In home deliveries the service durations are short, maybe just 2 minutes. Hence a shift can contain over 100 tasks in an urban region.
– In home deliveries the client orders via internet, while in service planning the majority is done via a planner or the customer contact center.
– In service planning the client usually does not pay for the service, hence giving an incentive to efficient time windows is harder.

Summarizing, we might have more influence on the time window for a request, and, moreover, it might be important to use this to get a favorable time window for a request. We can give the company's planner insight in the differences in travel times for different time windows; if in a certain week there are no convenient time windows for both, the client and the service provider, the planner might switch to the next week, not mentioning (or even not having available) unfavorable time windows.

In this call intake process, there is a balance between using low travel times, and filling the shifts of service engineers for the upcoming days. How eager should we be filling these shifts? If a shift for tomorrow can accommodate a task, shouldn't we simply take it, to avoid that a part of the capacity of the shift is left unused?

Note that the call intake process can be viewed as the construction phase of a routing problem. Methods that improve this phase can be helpful in the call intake process. On the other hand, we know that the result of the construction phase is not directly related to the result after optimization. Hence, we want to study these differences as well.

4 Explanatory example

4.1 Set-up

We discuss a small 1-dimension example, to explain the way the call intake process works, and to show the effect of clustering, which we explain in detail in Section 7. The example can be analyzed completely.

The set-up is the following:

– There is one engineer at position 0. All incoming tasks can be executed by this engineer.
– Every day two tasks appear, with equal chance for the positions -1 or 1. The deadline is three days (tomorrow and the two days after that).
– The engineer can handle two tasks per day, which are either both at position 1 (travel time is 2 units), or both or position -1 (travel time again 2 units) or one at position -1, and 1 at position 1 (travel time 4 units).
– The time windows coincide with the days.
– The planning is two tasks behind. That means the following. Today we do the planning for tomorrow and the two days after tomorrow. However, there are already two tasks planned for tomorrow and the day after. Since postponing tasks has no benefits, we have 5 (reasonable) 'states':
  1. State $(1, 1) \odot (., .)$: tomorrow the engineer has two tasks at position 1, nothing planned yet for the day after.
  2. State $(-1, -1) \odot (., .)$: tomorrow the engineer has two tasks at position -1, nothing planned yet for the day after.
  3. State $(1, -1) \odot (., .)$: tomorrow the engineer has one task at position 1 and one at position -1, nothing planned yet for the day after.
  4. State $(1, .) \odot (-1, .)$: tomorrow the engineer has a task at position 1, and the day after at position -1.
  5. State $(-1, .) \odot (1, .)$: tomorrow the engineer has a task at position -1, and the day after at position 1.

4.2 Strategy: first possible day

Starting from the 5 states above, we apply the strategy 'first possible day'. That means that any task that appears is planned at the first possible day i.e. the first day with at most one planned task. Note that it not allowed to look ahead! From a meta point of view, we know that 2 tasks per day will appear (since that is our set-up), but for the simulation we do not know this. For example, if we are in State 4, and a task at position -1 appears as first task, we plan it for tomorrow, even though the next task might be at position 1.

With this strategy we have the following transition matrix between the states; matrix element (i, j) is the probability that State i moves to State j on the next day.

$$\begin{pmatrix} 0.25 & 0.25 & 0.5 & 0 & 0 \\ 0.25 & 0.25 & 0.5 & 0 & 0 \\ 0.25 & 0.25 & 0.5 & 0 & 0 \\ 0 & 0.5 & 0.5 & 0 & 0 \\ 0.5 & 0 & 0.5 & 0 & 0 \end{pmatrix}.$$

We see that in the steady state the States 4 and 5 disappear, because they do not satisfy the ‘first possible day’ strategy. Moreover, we can calculate that in the steady state, the States 1 and 2 have a probability $\frac{1}{4}$, and State 3 has probability $\frac{1}{2}$. From this we obtain the expected daily travel time:

$$\frac{1}{4} * 2 + \frac{1}{4} * 2 + \frac{1}{2} * 4 = 3.$$

4.3 Strategy: cluster

In the ‘first possible day’ strategy, we do not use the freedom to postpone a task. Doing this is an example of clustering. Our strategy for the next appearing task at position x is the following:

– If tomorrow has only one task planned, we plan it tomorrow.
– If tomorrow is full, plan it on a day we already visit position x.
– If after tomorrow we don’t visit x yet, plan it on the first empty day.

Again we can calculate the transition matrix, which is now

$$\begin{pmatrix} 0.25 & 0.25 & 0 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0 & 0.25 & 0.25 \\ 0 & 0.5 & 0 & 0 & 0.5 \\ 0.5 & 0 & 0 & 0.5 & 0 \end{pmatrix}.$$

In this case, State 3 disappears from the steady state, and the other states all have probability $\frac{1}{4}$. Analyzing the expected daily travel time for this asymptotic situation yields:

$$\frac{1}{4} * 2 + \frac{1}{4} * 2 + (\frac{1}{8} * 2 + \frac{1}{8} * 4) + (\frac{1}{8} * 2 + \frac{1}{8} * 4) = 2.5$$

Hence clustering yields a saving of 16.7% in travel time, while still planning all tasks within the deadline.

5 Simulation set-up

The example above is exceptional in the sense that it can be completely analyzed. It the situation we will describe below, this is impossible due to the huge number of tasks and the complicated 2-dimensional geometry, which is common in a real life. To study the effect of different strategies we will run simulations.

To have a realistic situation, we created five datasets for a company with 25 engineers. We generated data for 270 (working) days, with an average of 200 requests per day. These requests are turned into tasks during the call intake process. This means that a time window is assigned to it, and temporarily an engineer to assure feasibility of the schedule. The time and engineer of the assignment can be changed, as long as the time window is respected: the task should start in its associated time window. We assume an online process, by which we mean that the requests have to be processed in the order they appear, without knowledge of the requests later in the list. In more detail, the data is created in the following way.

5.1 Locations and travel times

All locations are picked randomly from a 100×100 grid, all grid points with equal chance $\frac{1}{10{,}000}$. The travel time is calculated by the Euclidean distance at a speed of 1 grid point per minute. The travel time is rounded to the nearest *second.* For example traveling from $(24, 67)$ to $(44, 50)$ gives a grid distance of $\sqrt{20^2 + 17^2} = 43.462...$ minutes, which gives as travel time 43 minutes and 28 seconds, or 2608 seconds.

5.2 Engineers

There are 25 engineers, generated randomly as described below.

- The shift starts and ends at the home location of the engineer. The home location is generated randomly from the grid.
- All shifts are 8 hours long, no break is considered. Private travel time is not allowed, i.e. the work starts at the shift's start time by traveling to the first task, and ends by traveling back home, where the arrival should be at or before the shift's end time.
- There are five skills, and the employees have one to four skills. The frequencies of the skills among the engineers are different; they are 10, 14, 15, 18, and 21, so in total there are 78 resource skills (an average of 3.12 skill per engineer).

5.3 Requests

The requests have the following properties.

- Each request has a intake day, the day on which it becomes available. On this day it must be planned to one of the following days (not on the intake day itself). As explained above, we consider the online situation, by which we mean this the request also has a sequence number, and requests should be assigned following the order by the sequence numbers, without knowledge of upcoming requests.
- Each request has a preferred deadline which lies between 5 and 10 days. This reflects the situation that a company has internal or external agreements on how many requests should be planned within the preferred deadline, for example 80% of the requests. If the preferred deadline is 7 days and the intake day is 102, then preferably the task should be scheduled not later than day 109. However, it is allowed to pass the preferred deadline by 50%, so by 3 days in this case. It is allowed, but not preferred, to schedule the task on day 110, 111, or 112. If this is not done, the request is registered as unplanned, and disappears from our simulation. From the company's point of view, an external resource is required to handle this request.
- Each request has a duration (in minutes), the service time needed for the task. The durations are taken between 30 minutes and 60 minutes, with an average of 45 minutes. The distribution is not taken uniformly, but triangular in the following way. Assume we have N different values. The extremes (here 30 and 60) have chance $m = \frac{1}{N^2}$ and the middle value (here 45) has chance $M = \frac{2}{N} + m$. The chance for the other values is found by linear interpolation. The obtained value is rounded to the nearest 5 minutes, so that requests have durations of $30, 35, 40, \ldots,$ or 60 minutes.
- Each request has a location, which is taken randomly from the grid.
- Each request requires a skill, which is chosen uniformly from the 5 available skills.
- Per day we generate between 180 and 220 requests. These are chosen from the triangular distribution, which is generated in the same way as for the request durations.

5.4 Time windows

We turn a request to a task by assigning a time window to it. This time window prescribes the start time of a task. In our simulations we use time windows of 2 hours. Since a shift is 8 hours long, it intersects with 4 time windows.

5.5 Validation

The data we generate represents more or less one year. We use the first 50 days as warm-up period. Our validation is over the next 200 days. We also leave out the last 20 days; since we can plan at most 15 days in advance, the end of period effects are not yet visible in the validation period. The results in validation period are uniform during the whole period; hence the warm-up period is sufficient. What we consider are the requests with intake day in the validation period. In particular we are interested in (the percentage of) unplanned requests, and (the percentage of) the requests that were planned outside the preferred deadline.

5.6 Strategies

We investigate the effect of different strategies, and the use of optimization. For the strategies we consider following ones:

– **First possible day.** We assign the request to the first possible day. If on the first possible day there are several possible shifts, we assign it to the shift and position in which the *extra* travel time is the lowest. The other stops remain in the same order. While inserting we have to consider the time window we try to assign to the request, and the time windows of the already assigned tasks in the shift.
– **Min travel time.** If there are several options for a request, we assign it to the shift in which the **extra** travel time is the lowest, in the same way as in **First possible day**. In case of ties we use the earliest possibility.
– **Add at end.** We only assign a task at the end of a route. Among all possibilities we choose the one with lowest travel time from the previous stop. The travel time back to home is **not** considered. In case of ties, we use the earliest possibility.

All strategies first consider shifts within the preferred deadline. If in this period an option is found, it is always used (though from a higher level, it might be inefficient).

If no option within the preferred deadline is found, we consider the 50% extension interval. It is not prescribed to minimize lateness, but again, if there are options, we will choose one of them.

6 Results for the basic strategies

In this section we present the results of the simulations we executed. We start with the basic strategies, as described in previous subsection. Based on the results we try to improve the simulations by different parameters.

We present the results for of all five datasets in one table. The results of the different datasets are very similar, so there is no added value in presenting five different tables. Per day there are on average 200 tasks, that means that each dataset has around 40,000 tasks to validate, in total there are 200,316 tasks. We present the following performance indicators:

– **Unplanned.** The percentage of the tasks that could not be planned in the 50% extended deadline interval.

- **Lates.** The percentage of tasks that was planned too late, i.e. not within the preferred deadline. The percentage is taken relative to all planned tasks.
- **Avg late.** The average number of days late, among all late tasks.
- **Travel.** The average travel time in minutes per task, calculated as follows: the total travel time on the validation days divided by the total number of tasks scheduled on the validation days; so it represents the travel time per executed task.

6.1 If late then minimize lateness

Here, we follow the basic strategy, but we try to minimize lateness. That means: if we cannot find an option within the preferred deadline, then first we consider the options with 1 day late, and if this returns no result, we look at 2 days late, etcetera. This leads to the following results.

Strategy	Unplanned (%)	Lates (%)	Avg late	Travel
First possible day	15.16	82.21	2.03	23.6
Min travel time	15.11	82.18	2.03	23.5
Add at end	14.07	81.60	2.02	23.0

Results with increasing penalty on late days

We see that the 'First possible day' strategy and the 'Min travel time' perform very similar; the simplest strategy 'Add at end' performs better in all respects. Note that the number of unplanned tasks is high, and the number of tasks that is planned within the preferred deadline is very low. That probably explains that the first two strategies are very similar. That the third strategy performs better is at least partly due to being optimistic about extra travel time, especially if the shift is more or less empty. Before turning to this point, we want to investigate the effect of minimizing lateness. If the planning is so tight, it seems reasonable not to care *per task* about minimizing lateness. We simply want to avoid unnecessary travel time. That is the subject of the next subsection.

6.2 If late then don't care how much

As always, our strategies first try to assign within the preferred deadline, but if no option is found there, we consider all options in the allowed days late, and pick the best one. This leads to the table below.

Strategy	Unplanned (%)	Lates (%)	Avg late	Travel
First possible day	15.16	82.21	2.03	23.6
Min travel time	13.05	79.57	2.50	22.7
Add at end	10.24	76.63	2.85	20.1

Results with equal penalty on all late days

First we note that for the 'First possible day' strategy nothing changes (it still prefers to minimize lateness). The other 2 strategies clearly benefit from this relaxation: all performance indicators drop significantly except the average days late. Still the 'Add at end' strategy is clearly better than the others. Based on this experience we will not penalize extra lateness in further experiments. For instances in which it is hard to assign all tasks, this is probably always a good idea. Let's see what making it easier to start in an empty shift will do.

6.3 In an empty shift ignore the travel time back to home

In construction algorithms it is well known that weighing different travels in different ways can make a difference [Solomon (1987)]. Since we are still in the range of construction, we expect to see this here as well. So we calculate the extra travel time in empty shifts as only the time traveling to the task, and not the way back. The results are presented in the table below.

Strategy	Unplanned (%)	Lates (%)	Avg late	Travel
First possible day	13.69	81.11	2.01	22.5
Min travel time	11.14	77.62	2.65	20.4
Add at end	10.24	76.63	2.85	20.1

Results with travel time reduction for empty shifts

For the strategy 'Add at end' there is no difference (of course). The other two strategies clearly benefit from this change. The 'Min travel time' strategy is closing in on the 'Add at end' strategy. Before discussing a clustering strategy, we do a final check if preferring the first days has some influence on the results. We will keep the empty shift travel time reduction.

6.4 Give travel time reduction to early days

In less challenging planning problems, it might be a good idea to fill the gaps in shifts of tomorrow, and maybe one or two days after that. However, in our cases the shifts will be full anyway. Nevertheless, we do a run with reductions on the travel times: 10 minutes for tomorrow, 6 minutes for the day after, and 2 minutes for the day after that.

Strategy	Unplanned (%)	Lates (%)	Avg late	Travel
First possible day	13.69	81.11	2.01	22.5
Min travel time	11.19	77.68	2.66	20.5
Add at end	10.27	76.57	2.85	20.1

Results with empty shift reduction and reduction for early days

As expected the results are almost the same as in Subsection 6.3. For the 'First possible day' strategy there is no difference (of course). The results for the other two strategies are slightly worse, what could be expected as the 'First possible day' strategy is worse; the planning is so tight that it is a waste of capacity to assign earlier at the expense of higher travel time.

7 Dynamic clustering

Before turning to results with optimization, we want to discuss 'dynamic clustering'. In VRP clustering is well-known; an early reference is [Beasley (1983)]. In particular in periodic VRP, see [Campbell & Wilson (2014)], clustering techniques are widely used.

Although the problem here is not periodic, it shares that there is freedom to decide on which day and time window we place a request. The benefits are comparable: if we have requests to the north and to the south, it would be nice to create tasks on one day for the north, and on the other day for the south, see also Section 4. In periodic VRP it might not be (fully) possible, because some locations in the north and south have to be visited each day. In our situation, we have the clients that have preferences. We ignored these, assuming that the client will accept the proposed time window; in practice the client will have choice from some, for example three, time windows. From the point of view of the service provider, it would be wise the limit the possibilities, especially when the service area is large. The company might do this by assigning its engineers to geographical regions, making sure that tasks for an engineer are in an acceptable range.

Nevertheless, the travel times can be rather high, especially if the engineer is a specialist, serving a large area. In this case it is inevitable that the engineer has to visit a location 'A' far from home in one of the shifts, but we would like that other tasks in this shift are not too far away from location A. This strategy we call 'dynamic clustering': once one or more tasks are assigned to a shift, the new task should be close (in travel time) to *all* already assigned tasks. This maximum travel time between tasks in a cluster, we call the *cluster diameter*. To avoid the risk of partly idle shifts, we do not enforce dynamic clustering for tomorrow and the day after tomorrow. For any time window where dynamic clustering is active, we require that a request is added to a cluster, if possible; only if no cluster is found, an empty shift can be used. In this way we hope to fill the clusters to its capacity.

It depends on the instance what is the best cluster diameter and what is the best day to abandon it. After some experiments on our datasets, it turned out that a diameter of 18 minutes works well. Note that this is around 10% lower than the average travel time per task in the experiments till now. If the cluster diameter is too small, we can expect that many requests will remain unplanned. If the diameter is too high, we can expect less effect from dynamic clustering.

We keep on using the travel time reductions of 10, 6, and 2 minutes, for the first three days; i.e. we use the same parameters as in Subsection 6.4. We can combine dynamic clustering with the strategies described there, because dynamic clustering only restricts the possible options. The simulations lead to the following results.

Strategy	Unplanned (%)	Lates (%)	Avg late	Travel
First possible day	0.68	30.32	1.58	13.7
Min travel time	0.16	18.19	1.93	12.6
Add at end	0.97	27.93	2.17	13.7

Results with dynamic clustering

The improvement is spectacular. In the previous experiments, it seemed that not nearly all tasks could be scheduled. However, when using dynamic clustering and the 'Min travel time' strategy, only 323 out of 200,361 tasks were not planned. The travel time per task drops by 40%; without clustering the travel time consumed around 30% of the shift time, but when using clusters this drops to 21.0%. The conclusion is, that if only construction is applied, dynamic clustering is the best to do. It remains to investigate the effect of optimization.

8 Optimization

We will study the effect of optimization, for the situation without or with dynamic clustering. In both cases we apply an optimization algorithm per day for 30 second. The algorithm we use is an ALS method [Pisinger & Ropke (2007)], in the way described in [Curtois et al (2018)]. For the experiments without clustering, we apply optimization on the first five days; for the experiments with dynamic clustering we only optimize the first two days; we leave the clusters as they are. The simulation without clustering gives the following results.

Strategy	Unplanned (%)	Lates (%)	Avg late	Travel
First possible day	0.73	28.87	1.67	14.4
Min travel time	0.61	32.35	2.35	13.7
Add at end	3.74	58.58	2.70	13.8

Results with optimization on 5 days, without dynamic clustering

Compared to Subsection 6.4, we see a large improvement. This is not unexpected, as optimization in VRPs usually largely improves the solution. Note, however, that these solutions are worse or at best comparable to the results in Section 7. That makes us curious about the result with optimization and dynamic clustering. These are presented in the table below.

Strategy	Unplanned (%)	Lates (%)	Avg late	Travel
First possible day	0.00	5.75	1.19	13.1
Min travel time	0.01	9.80	1.86	11.9
Add at end	0.23	16.36	2.07	12.0

Results with dynamic clustering and optimization on 2 days

The results improve, but not as much as without using clusters. We could have expected this, because all tasks in a route are already close together, and tasks in other routes probably are at some distance. In view on the number of tasks planned and the preferred deadline, the ‘First possible day’ strategy works the best; all 200,361 tasks are planned, and only 5.75% of the tasks is outside the preferred deadline. However, if reducing travel time is important, the strategy ‘Min travel time’ could be used. In this case only 23 tasks are not planned, the lateness is higher, but the travel time per task is lower, saving 4075 hours of travel time.

9 Conclusion

We discussed the call intake process for service planning. We described several methods on how to choose time windows for the requests. We ran a simulation on rather complex data. On this data we noted that giving preference to empty routes, dynamic clustering and intermediate optimization are important aspects; in our simulations the results improve considerably.

This simulation is a simplified model of reality. First there is the client: the best time window for the company might not be feasible for the client, which might worsen the results. However, some preliminary tests with random choices among the options available, show that this effect is limited. It will not change the validity of the discussions in the previous sections.

Another aspect is the homogeneity of the data. First, the geometry is very simple, as well as the expectations for the task properties. In a real situation the service area might be split in regions, and the engineers work only in some of the regions. This restricts the options for the requests, but ensures that if there is only one available engineer, the engineer will not drive two hours to the other side of the service area and two hours back. Experiments show that removing the region restriction during optimization can have a positive effect on the results.

Another aspect of the geometry is that population densities can vary in the regions we consider. In such cases it might be beneficial to decrease the cluster diameter for an engineer working in highly populated regions. It is difficult to predict what choice is the best, but in practice we can monitor the results, and adjust the parameters accordingly.

Finally, there is the planner’s acceptance. Especially looking at a *single* route, it is important that the route is optimal at all times. Since we usually there are not more than 10 tasks in a shift, we can guarantee optimality, by solving a dynamic program, see [Held & Karp (1962)]. We call this method Optimized Best Fit (OBF). Each (shift/time window)-combination can be solved in a few milliseconds, making it feasible to combine the strategies with optimization per shift. OBF did not improve the results in our experiments. For the same reason of planner’s acceptance, we always use the reductions on the first days. In times that the workload is low, for sure it is preferable to fill the upcoming days. The experiments above show that the‘First possible day’ strategy competitive with the ‘Min travel time’ strategy even when the planning is tight.

The method OBF with dynamic clustering supports several companies using PCA's product Marlin, see [PCA]. It helps them in easily providing good options to their clients, and reduces the effort of planning as well as the total travel time, thus improving the productivity.

References

[Beasley (1983)] J.E. Beasley, *Route first—cluster second methods for vehicle routing*, Omega **11**, 403–408, (1983).

[Caceres-Cruz et al (2014)] J. Caceres-Cruz, P. Arias, D. Guimarans, D. Riera, and A. A. Juan, *Rich vehicle routing problem: Survey*, ACM Computing Surveys **47**, 1–28, (2014).

[Campbell & Wilson (2014)] A.M. Campbell and J.H. Wilson, *Forty years of periodic vehicle routing*, Networks **63**, 2–15, (2014).

[Curtois et al (2018)] T. Curtois, D. Landa-Silva, Dario, Y. Qu, and W. Laesanklang, *Large neighbourhood search with adaptive guided ejection search for the pickup and delivery problem with time windows*, EURO Journal on Transportation and Logistics **7**, 151–192, (2018).

[Held & Karp (1962)] M. Held and R.M. Karp, *A dynamic programming approach to sequencing problems*, Journal of the Society for Industrial and Applied Mathematics **10**, 196–210, (1962).

[Misir et al (2015)] M. Misir, P. Smet, and G. Vanden Berghe, *An analysis of generalised heuristics for vehicle routing and personnel rostering problems*, Journal of the Operational Research Society **66**, 858–870, (2015).

[PCA] See https://pca.nl/en/functionalities/service-and-maintenance-planning/.

[Pisinger & Ropke (2007)] D. Pisinger and S. Ropke, *A general heuristic for vehicle routing problems*, Computers & Operations Research **34**, 2403–2435, (2007).

[Solomon (1987)] M. Solomon, *Algorithms for the vehicle routing and scheduling problems with time window constraints*, Operations Research **35**, 254–265, (1987).

[Strauss et al (2021)] Arne Strauss, Nalan Gülpınar, Yijun Zheng, *Dynamic pricing of flexible time slots for attended home delivery*, European Journal of Operational Research **294**, 1022–1041, (2021).

[Visser (2019)] Thomas Visser, *Vehicle Routing and Time Slot Management in Online Retailing*, EPS-2019-482-LIS, (2019).

Optimising Scheduling of Hybrid Learning using Mixed Integer Programming

Matthew Davison[1], Ahmed Kheiri[2], Konstantinos Zografos[2]

[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,k.zografos}@lancaster.ac.uk

Keywords: University Timetabling · Hybrid Learning · Mixed Integer Programming

1 Introduction

The COVID-19 pandemic artificially reduced the already limited capacity of physical spaces at universities. It forced a greater use of *hybrid learning*, which is a mode of teaching that combines online and in-person elements. Previous studies dealt with limited capacity by controlling the quantity and flow of students on campus [5] or on the areas surrounding campus [1]. Other studies investigated what policy changes allow a better use of resources [2].

Online classes are one way to reduce demand for physical space. Universities in the future will likely continue to offer a mix of online and in-person classes beyond the pandemic [3]. Since students and staff typically prefer to attend classes in-person, this motivates the need to limit the number of classes held online, whilst still taking advantage of their ability to reduce the demand for physical space. More specifically, the problem is to investigate how universities can maximise the number of courses that they offer, whilst simultaneously limiting the number of online classes that are used to achieve this. This timetabling problem differs to the classic timetabling problem in that the input is a list of classes, but not all classes need to be assigned. Solutions to this problem identify how many courses could be offered, which is useful information for universities planning semesters. The preliminary model presented in this paper illustrates one way of solving this particular problem.

2 Model Formulation

Timeslots are lengths of time that have a start and end. In this problem we assume these are five minutes long. Timesets are defined as a subset of the set of all timeslots. These are used to better model complicated arrangements. For example, a timeset could describe a two hour class meeting every other week. Table 1 provides the notation for the sets used within the formulation of the model.

We define the matrix A where the entry A_{r_1,r_2} is equal to the number of timeslots it takes to travel from room r_1 to room r_2. In particular, for d a non-negative integer, $A_{r_1,r_2} = d$ represents a travel time of $5d$ minutes.

Table 1. Key notation. The first six sets are primarily used to describe elements of the problem, the last seven sets are primarily used in the construction of constraints.

S	Set of timeslots
T	Set of timesets. Each $t \in T$ is a subset of S
R	Set of rooms
C	Set of classes
K	Set of courses
L_k	Set of sections for course k. Each $l \in L_k$ is a subset of C
R_r^u	Set of timeslots when room $r \in R$ is unavailable
R_c	Set of rooms suitable for class $c \in C$
T_c	Set of timesets suitable for class $c \in C$
R_G	Let $G \subseteq C$. $R_G := \cap_{c \in G} R_c$
C_G	Let $G \subseteq C$. $C_G := \{(c_1, c_2) \in G \times G : c_1 \neq c_2\}$
R_r^c	Let $r \in R$. $R_r^c := \{c \in C : r \in R_c\}$
O_s	Let $s \in S$. $O_s := \{a \in T : s \in a\}$

The set C contains all classes regardless of if they can be held online, in-person or both. The online space is modelled as a room that is always available and can host multiple classes at the same time. Let r^* represent this online space. A class, $c \in C$, can be held online if and only if $r^* \in R_c$. For r^* assume that $A_{r^*,r^*} = 0$ and $A_{r^*,r} = d^*$ for all $r \in R \setminus r^*$ where d^* is a fixed number of timeslots. For this paper, we assume any class can happen online and that $d^* = 0$.

2.1 Definition of variables

One set of variables used in this problem are binary variables indicating if a class is assigned to a particular room and timeset. They are defined as follows:

$$x_{c,r,t} = \begin{cases} 1 & \text{Class } c \in C \text{ is held in room } r \in R \text{ in timeset } t \in T, \\ 0 & \text{Otherwise.} \end{cases}$$

The second set of variables are binary variables that indicate if a class uses a room, or if a class uses a timeset. They are defined as follows:

$$y_{c,r} = \begin{cases} 1 & \text{Class } c \in C \text{ is held in room } r \in R, \\ 0 & \text{Otherwise.} \end{cases}$$

$$y_{c,t} = \begin{cases} 1 & \text{Class } c \in C \text{ is held in timeset } t \in T, \\ 0 & \text{Otherwise.} \end{cases}$$

These variables are related to each other by linking constraints:

$$y_{c,r} = \sum_{t \in T} x_{c,r,t}, \quad \forall r \in R, c \in C,$$

$$y_{c,t} = \sum_{r \in R} x_{c,r,t}, \quad \forall t \in T, c \in C.$$

We also define binary variables to indicate if courses are offered:

$$g_k = \begin{cases} 1 & \text{Course } k \in K \text{ is offered,} \\ 0 & \text{Otherwise.} \end{cases}$$

Parameter arrays Defined in Table 2, parameter arrays are fully determined by a given problem instance. They are used to indicate if a group of resources satisfy a particular condition, thus indicating what constraints to include in the model corresponding to that instance.

Table 2. Definition of each parameter array. t, t_1 and t_2 are in the set T. r, r_1 and r_2 are in the set R

Array	Description
D_0	A matrix where $D_0[r,t]$ is equal to one if room r is unavailable at some point during timeset t, zero otherwise
D_1	An array where $D_1[r_1, r_2, t_1, t_2]$ is equal to one if there is not enough time between t_1 and t_2 to travel between r_1 and r_2, zero otherwise
D_2	A matrix where $D_2[t_1, t_2]$ is equal to zero if the first meeting in t_1 concludes before the start of the first meeting in t_2, one otherwise
D_3	A matrix where $D_3[t_1, t_2]$ is zero if the meetings in t_1 and t_2 do not occur on overlapping weeks and days, if any meeting in t_1 and t_2 occurs on the same day and week then $D_3[t_1, t_2]$ is equal to the number of timeslots between the start of the earliest meeting and end of the latest
D_4	A matrix where $D_4[t_1, t_2]$ is equal to zero if t_1 and t_2 start at the same time of day, one otherwise
D_5	A matrix where $D_5[t_1, t_2]$ is equal to zero if t_1 completely overlaps t_2 in the times of day they meet or vice versa, one otherwise
D_6	A matrix where $D_6[t_1, t_2]$ is equal to zero if t_1 meets on a subset of days that t_2 does or vice versa, one otherwise
D_7	A matrix where $D_7[t_1, t_2]$ is equal to zero if t_1 and t_2 do not meet on any of the same days, one otherwise
D_8	A matrix where $D_8[t_1, t_2]$ is equal to one if t_1 overlaps t_2, zero otherwise.

2.2 Constraints

There are various constraints that need to be included within any university timetabling model. Some are more specialised so that the timetable adheres to university policy or allows students and staff to travel comfortably between classes. In this section, some constraints included within our model are described.

Most of these constraints have been proposed in the existing timetabling literature [4] and what is presented in this paper is our approach to modelling these constraints.

Classes can only be assigned at most a single room and a single timeset

$$\sum_{t\in T}\sum_{r\in R} x_{c,r,t} \leq 1, \quad \forall c \in C.$$

Classes can only be assigned compatible rooms and times To ensure only compatible rooms and times are used, for each $c \in C$ add the following constraints:

$$\sum_{r\in R} x_{c,r,t} = 0, \quad \forall t \in T \setminus T_c.$$

$$\sum_{t\in T} x_{c,r,t} = 0, \quad \forall r \in R \setminus R_c.$$

Classes should not happen in a room when that room is not available

$$\sum_{t\in T}\sum_{r\in R} D_0[r,t]x_{c,r,t} = 0, \quad \forall c \in C.$$

In-person classes should not use the same room at the same time

$$\sum_{c\in R_r^c}\sum_{t\in O_s} x_{c,r,t} \leq 1, \quad \forall r \in R \setminus r^*, s \in S.$$

Group of classes should occur in the same room Let G be a set of classes that must occur in the same room. For each $r \in R_G$ define a binary variable s_r^G that takes the value one if every class in G uses room r and zero otherwise. Add the following constraints:

$$\sum_{c\in G} y_{c,r} = |G|s_r^G, \quad \forall r \in R_G,$$

$$\sum_{r\in R_G} s_r^G \leq 1,$$

$$y_{c,r} = 0, \quad \forall r \notin R_G, c \in G.$$

Attending a group of classes Staff and students have collections of classes they must attend. Denote a collection of classes as G. For each $(c_1, c_2) \in C_G$ add the following constraints:

$$D_1[r_1,r_2,t_1,t_2](x_{c_1,r_1,t_1} + x_{c_2,r_2,t_2}) \leq 1, \quad \forall t_1 \in T_{c_1}, t_2 \in T_{c_2}, r_1 \in R_{c_1}, r_2 \in R_{c_2}.$$

Group of classes should occur in a certain order Let G be a sequence of classes that should occur in order, meaning that the first meeting of a class should completely finish before the start of the next class. Suppose $G = (c_1, c_2, \dots, c_k)$, then for each pair (c_i, c_{i+1}) where $i \in \{1, \dots, k-1\}$ add the following constraints:

$$D_2[t_1,t_2](y_{c_i,t_1} + y_{c_{i+1},t_2}) \leq 1, \quad \forall t_1 \in T_{c_i}, t_2 \in T_{c_{i+1}}.$$

Group of classes should be grouped within a period of time Let G be a set of classes that should all happen within H timeslots if they happen on the same day and week. For each $(c_1, c_2) \in C_G$ add the following constraints:

$$I(D_3[t_1, t_2] > H)(y_{c_1,t_1} + y_{c_2,t_2}) \leq 1, \quad \forall t_1 \in T_{c_1}, t_2 \in T_{c_2},$$

where I is an indicator function that takes the value one if $D_3[t_1, t_2] > H$ holds and zero otherwise.

Timing constraints All timing specific constraints have the same form. Suppose G is the set of classes the constraint applies to. For each $(c_1, c_2) \in C_G$ add the following constraints:

$$D[t_1, t_2](y_{c_1,t_1} + y_{c_2,t_2}) \leq 1, \quad \forall t_1 \in T_{c_1}, t_2 \in T_{c_2},$$

where D is the appropriate parameter array for that constraint. Table 3 describes a constraint on a group of classes and the associated parameter array.

Table 3. Constraints and associated parameter array

Constraint	Associated D
Classes should start at the same time	D_4
Classes should occur during the same time of day	D_5
Classes should occur on the same days of the week	D_6
Classes should occur on the different days of the week	D_7
Classes should not overlap in time	D_8

Course structure constraints Courses can be split into sections that teach identical content. Sections are a collection of classes and it is possible for two sections of the same course to share classes. We define binary variables to indicate if a course section is offered:

$$h_{k,l} = \begin{cases} 1 & \text{Section } l \in L_k \text{ of course } k \in K \text{ is offered,} \\ 0 & \text{Otherwise.} \end{cases}$$

For a given course, $k \in K$, a section $l \in L_k$ can be offered only if all of the classes in that section are offered. This is modelled by the following constraints:

$$h_{k,l} \leq \frac{1}{|l|} \sum_{c \in l} \sum_{t \in T} \sum_{r \in R} x_{c,r,t}, \quad \forall l \in L_k, k \in K,$$

where $|l|$ is the number of classes in a section. A course is only offered if there is at least one section being offered. This is modelled by the following constraints:

$$g_k \leq \sum_{l \in L_k} h_{k,l}, \quad \forall k \in K.$$

2.3 Objectives

There are many objectives in the timetabling literature. In this section some of the objectives that could be used within this model are presented.

Maximise number of courses offered

$$\max \ z_1 = \sum_{k \in K} g_k,$$

Maximise number of classes held

$$\max \ z_2 = \sum_{c \in C} \sum_{t \in T_c} \sum_{r \in R_c} x_{c,r,t}.$$

Whilst courses are important, offering as many classes as possible provides flexibility within these courses.

Minimise number of classes held online

$$\min \ z_3 = \sum_{c \in C} \sum_{t \in T_c} x_{c,r^*,t}.$$

Ideally, there would be no need for online classes but it cannot be ignored that they can help increase z_1 because they are not subject to physical space limitations, a common limit to teaching capacity.

Minimise cost of assignment

$$\min \ z_4 = \sum_{c \in C} \left(\sum_{r \in R_c} P_{c,r} y_{c,r} + \sum_{t \in T_c} P_{c,t} y_{c,t} \right),$$

where $P_{c,r}$ and $P_{c,t}$ are non-negative penalties for assigning class $c \in C$ to room $r \in R_c$ and timeset $t \in T_c$ respectively. The exact value of these penalties are subjective in practice. They could represent:

- Actual monetary cost of using the resource.
- Approximation of preference (low penalty indicating higher preference).
- Arbitrarily large penalties to deter solution from using a resource.

Weighted objective approach

$$\max \ z = w_1 z_1 + w_2 z_2 + w_3 z_3 + w_4 z_4,$$

where w_1 and w_2 are non-negative weights, and w_3 and w_4 are non-positive weights. By properly tuning these weights the model is able to determine an optimal mix of in-person and online teaching.

3 Results

To verify that this model produces feasible timetables, three instances from the 2019 International Timetabling Competition (ITC-2019) were used [4]. Described in Table 4 is the number of classes, timesets and rooms for each instance.

The problem defined in the ITC-2019 involves creating a complete timetable and allocating students to classes based on their course requests. Our model does not consider student allocation nor requires a complete timetable to be constructed. Therefore, in this experiment, we maximise z_2 only. Using this solution, it is possible to evaluate z_3 and z_4.

For our experiments we used an internal computing node running CentOS Linux with an Intel Xeon E5-2699 v3 CPU running at 2.30GHz and 528GB of RAM. The model was implemented in Python 3.5 and solutions were found using the commercial solver Gurobi 9.0, which successfully produced valid timetables. Table 4 provides information about each solution.

Table 4. The problem instances and quantities of key features. In the "Type" column, "T" indicates "test" instances and "C" indicates "competition" instances. The z_3 recorded is the number of classes that does not require a room, which we treat as "online-only" classes. The z_4 recorded here is the same value reported by the ITC-2019 validation tool [4]

Instance	Type	$\|K\|$	$\|C\|$	$\|T\|$	$\|R\|$	z_2	z_3	z_4	Time (s)
lums-sum17	T	19	20	93	62	20	0	73	0.003
bet-sum18	T	48	127	50	46	127	6	3502	0.011
tg-fal17	C	36	711	1645	23	711	15	9610	58757.559

As can be seen from Table 4 in all three instances $|C| = z_2$ meaning that our solutions are optimal for this objective. Since we are offering all possible classes, it is clear that it is possible to offer every course.

Acknowledgements We gratefully acknowledge the support of the EPSRC funded EP/S022252/1 STOR-i Centre for Doctoral Training.

References

1. Al-Yakoob, S.M., Sherali, H.D.: A mixed-integer programming approach to a class timetabling problem: A case study with gender policies and traffic considerations. European Journal of Operational Research **180**(3), 1028–1044 (2007). https://doi.org/https://doi.org/10.1016/j.ejor.2006.04.035
2. Barnhart, C., Bertsimas, D., Delarue, A., Yan, J.: Course Scheduling Under Sudden Scarcity: Applications to Pandemic Planning. Manufacturing & Service Operations Management **24**(2), 727–745 (2022). https://doi.org/10.1287/msom.2021.0996, https://doi.org/10.1287/msom.2021.0996
3. Lederman, D.: What's the future of the physical college campus? Inside Higher Ed (2021), https://www.insidehighered.com/news/2021/07/16/whats-future-physical-college-campus

4. Müller, T., Rudová, H., Müllerová, Z.: University course timetabling and International Timetabling Competition 2019. Proceedings of the 12th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2018) pp. 5–31 (2018)
5. Vermuyten, H., Lemmens, S., Marques, I., Beliën, J.: Developing compact course timetables with optimized student flows. European Journal of Operational Research **251**(2), 651–661 (2016). https://doi.org/https://doi.org/10.1016/j.ejor.2015.11.028