Machine-personnel assignment with training and interim worker requirements

Hatice Çalık · Pieter Smet · Everton Fernandes Silva

Received: date / Accepted: date

Abstract We introduce a machine-personnel assignment problem where personnel must be trained to operate certain groups of machines, whereas others can also be operated by interim workers hired at an additional cost. In order to achieve a feasible or a minimum-cost assignment for long-term planning, it may be necessary to cross-train employees on different machines. However, this requires switching to different machines frequently, which is not desirable for personnel. Therefore, each switch incurs a penalty cost. This specific characteristic makes the problem unique and complex when compared to well-known related problems in the literature. The problem requires assigning each machine to a minimum number of operators for each day of a given planning horizon while minimizing the total cost of hiring interim workers and switching machines. We provide integer programming formulations of the problem and develop an iterated local search method to solve instances with longer planning horizons. A comparison of the introduced methods on randomly generated instances indicate that the iterated local search algorithm is capable of finding high quality solutions within a reasonable time limit.

Keywords Cross-training · Staff assignment · Integer programming · Iterated local search

1 Introduction

The new problem studied in this paper, which we refer to as the *machine-personnel assignment problem with training and interim worker requirements* (MPATI), is motivated by a real-world application from the food and drink industry. Although we were introduced to this problem by a company operating in this specific sector, it is likely that similar applications exist in other sectors involving production or process optimization with possibilities of personnel training and hiring interim workers.

The MPATI deals with a division within the company which contains a fixed number of machines to be operated on a daily basis and permanent workers (personnel) whose shifts and working days in the planning horizon are predetermined. Although the majority of the machines must be operated only by personnel and under the supervision of a qualified worker, it is possible, and sometimes necessary, to hire interim workers to operate the others. Each machine requires a minimum number of operators (personnel or interim workers) assigned to every day, while an operator can only work on exactly one machine within a working day.

At the beginning of the planning horizon, only a subset of the workers is qualified to supervise or operate each machine independently. The set of qualified workers is not necessarily identical for different machines and, therefore, a worker qualified on one machine might be unqualified for another. Certain groups of workers may obtain the necessary qualification to operate certain machines after a training period. Training is possible only under the supervision of a qualified employee dedicated to work on those machines during the training days. In order to preserve their qualification, workers must avoid not operating the same machine for too many consecutive days. Otherwise, they lose their qualification and must undertake a short training period

H. Çalık - Corresponding author
KU Leuven, Department of Computer Science, CODeS; Leuven.AI, Belgium
E-mail: hatice.calik@kuleuven.be

P. Smet
KU Leuven, Department of Computer Science, CODeS; Leuven.AI, Belgium
E-mail: pieter.smet@kuleuven.be

E. Fernandes Silva
KU Leuven, Department of Computer Science, CODeS; Leuven.AI, Belgium
E-mail: everton.fernandesdasilva@kuleuven.be

which requires working on that machine under the supervision of a qualified person. Assigning workers to the same machines every day implies having only single-skilled personnel, which can result in infeasibilities in the long term. On the other hand, workers generally prefer not to switch machines frequently, given its additional adjusment period required. Therefore, a penalty is incurred every day a worker changes their machine assignment.

Learning and forgetting effects have been well-studied in scheduling problems. In these problems, job processing times are affected by the number of times a resource has performed a certain job [1,2]. The impact of these effects has been analyzed on the performance of both the system as a whole and individual workers [3,8,9].

Given the aforementioned restrictions, the MPATI requires a schedule which assigns workers to machines for each day of the planning horizon such that the total cost of hiring interim workers and the penalty cost of switching machines is minimized.

There are numerous examples of practical applications in the literature in which an accurate representation of employee skills is essential [5]. While there exist many papers on staff scheduling with fixed skills [10], the problem setting considered in this paper is rather uncommon due to the presence of decisions concerning employee cross-training. In one of the few related studies, Wirojanagud et al. [11] use an integer programming model to decide when to hire, fire or cross-train employees to minimize the costs incurred by production losses and training. In contrast to our problem setting, decisions are made at an aggregate level, without considering individual workers who may have varying preferences. De Bruecker et al. [4] propose a three-phase integer programming approach for optimizing the shift and training schedule of aircraft maintenance workers. Emphasis is placed on finding the optimal trade-off between low-cost schedules that require highly cross-trained workers and the training costs required to establish such a workforce. The impact of long-term staffing decisions has been studied by Komarudin et al. [7] who introduce a methodology for evaluating the effect on operational costs associated with staff allocation when exists multiple skills among the workforce being scheduled. For a comprehensive review of staff scheduling and personnel rostering literature, we refer to [6].

Our contribution in this paper is threefold. First, we introduce a new, relevant machine-personnel assignment problem combined with disaggregated cross-training and interim worker hiring decisions which have never been considered in the literature. Second, we provide mathematical formulations of this newly introduced problem with several valid inequalities. Third, we present a heuristic method which produces very high quality solutions within the imposed time limit.

The remainder of the paper is organized as follows: Section 2 formally describes the problem with an integer programming (IP) formulation. This section also introduces a mixed integer programming (MIP) variant of this IP and several valid inequalities obtained by exploiting practical properties of the problem. Section 3 provides an Iterated Local Search (ILS) algorithm for solving the MPATI. The performance of both mathematical models and the heuristic algorithm are then evaluated on randomly generated instances in Section 4. Finally, we conclude and offer directions for future research in Section 5.

2 Problem formulation

Given a set of workers P working the same shift of a process unit in a factory or a company, the problem consists of operating a set of machines M during a planning horizon denoted by a set of days D. While a subset M^N of machines M can only be operated by workers in P, the remaining set $M - M^N = M^I$ of machines can also be operated by interim workers who must be hired at additional cost f per person per day. Each machine $m \in M$ requires a minimum number of operators n_{md} on day $d \in D$.

At the beginning of the planning horizon, not every worker is qualified to independently operate every machine. This information is available via a skill matrix $S = [s_{pm}]$ where $s_{pm} = 1$ if worker $p \in P$ is qualified to operate machine $m \in M$ without supervision and $s_{pm} = 0$ otherwise. A worker $p \in P$ can undergo a training period of l_{pm} days (not necessarily consecutive) to become qualified to operate machine $m \in M$ without supervision. While trainees are considered as operating workers, they can only be trained on a machine if there is a qualified worker assigned to the same machine on the same day. A worker $p \in P$ must be assigned to exactly one machine on each working day $d \in D_p$ where $D_p \subseteq D$ is the set of days that worker p is available. Worker p who is not assigned to a machine for C_p consecutive working days of D_p must undergo a short retraining of R_p days (not necessarily consecutive) on that machine to reacquire their qualification. We refer to C_p as 'skill memory' throughout the remainder of the paper.

Although working on different machines on different days might be inevitable for workers, it is undesirable and therefore a penalty cost π is incurred per machine switch per worker. The objective is to find a machine-worker assignment for the entire planning horizon while minimizing the total cost of hiring interim workers and switching penalties.

In order to formulate this problem as an IP model, we define the following additional parameters:

e_{0p}: the index of the first working day of worker p.
e_{dpk}: the index of the k-th previous working day of worker p for day $d \in D_p$.
h_{pm}: the number of consecutive working days worker $p \in P$ has not been working on machine $m \in M$.
t_{pm}: the number of training days of worker $p \in P$ on machine $m \in M$.

Moreover, we introduce the following decision variables:

$x_{pmd} = 1$ if worker $p \in P$ is assigned to machine $m \in M$ on day $d \in D$, 0 otherwise.
$\gamma_{pmd} = 1$ if worker $p \in P$ is qualified to work on machine $m \in M$ on day $d \in D$, 0 otherwise.
$w_{pmd} = 1$ if worker $p \in P$ is qualified and working on machine $m \in M$ on day $d \in D$, 0 otherwise.
$\alpha_{pmd} = 1$ if worker $p \in P$ lost their qualification for machine $m \in M$ on day $d \in D$, 0 otherwise.
$\tau_{pmd} = 1$ if worker $p \in P$ is being trained on machine $m \in M$ on day $d \in D$, 0 otherwise.
$y_{pd} = 1$ if worker $p \in P$ switches to a different machine on day $d \in D$, 0 otherwise.
z_{md}: the number of interim workers assigned to machine $m \in M$ on day $d \in D$.

The following is an IP formulation for the MPATI.

$$(IP) \quad \min \sum_{m \in M^I} \sum_{d \in D} f z_{md} + \sum_{p \in P} \sum_{d \in D_p} \pi y_{pd} \tag{1}$$

$$\text{s.t.} \sum_{m \in M} x_{pmd} = 1, \qquad \forall p \in P, d \in D_p, \tag{2}$$

$$w_{pmd} + \tau_{pmd} = x_{pmd}, \qquad \forall p \in P, m \in M, d \in D_p, \tag{3}$$

$$\sum_{p \in P: d \in D_p} x_{pmd} \geq n_{md}, \qquad \forall m \in M^N,\ d \in D, \tag{4}$$

$$z_{md} + \sum_{p \in P: d \in D_p} x_{pmd} \geq n_{md}, \qquad \forall m \in M^I,\ d \in D, \tag{5}$$

$$\tau_{pmd} \leq \sum_{q \in P: d \in D_q} w_{qmd}, \qquad \forall p \in P, m \in M, d \in D_p, \tag{6}$$

$$x_{pmd} - x_{pme_{dp1}} \leq y_{pd}, \qquad \forall p \in P, m \in M, d \in D_p, \tag{7}$$

$$x_{pme_{dp1}} - x_{pmd} \leq y_{pd}, \qquad \forall p \in P, m \in M, d \in D_p, \tag{8}$$

$$w_{pmd} \leq \gamma_{pmd}, \qquad \forall p \in P, m \in M, d \in D_p, \tag{9}$$

$$\gamma_{pme_{0p}} \leq s_{pm}, \qquad \forall p \in P, m \in M, \tag{10}$$

$$\gamma_{pmd} \leq \sum_{k=1}^{C_p - h_{pm}} x_{pme_{dpk}}, \qquad \forall p \in P, m \in M, d \in D_p : e_{dp(C_p - h_{pm})} = 0, \tag{11}$$

$$\gamma_{pmd} \leq \sum_{k=1}^{C_p} x_{pme_{dpk}}, \qquad \forall p \in P, m \in M, d \in D_p : e_{dpC_p} \geq 0, \tag{12}$$

$$\sum_{j=0: j \in D_p}^{d} \tau_{pmj} \geq (l_{pm} - t_{pm}) \gamma_{pmd}, \qquad \forall p \in P, m \in M, d \in D_p : s_{pm} = 0, \tag{13}$$

$$\gamma_{pme_{dp1}} - \gamma_{pmd} \leq \alpha_{pmd}, \qquad \forall p \in P, m \in M, d \in D_p, \tag{14}$$

$$\sum_{j=i: j \in D_p}^{d} \tau_{pmj} \geq R_p \gamma_{pmd} + R_p(\alpha_{pmi} - 1), \qquad \forall p \in P, m \in M, d, i \in D_p : i \leq e_{dpR_p}, \tag{15}$$

$$x_{pmd}, \gamma_{pmd}, w_{pmd}, \alpha_{pmd}, \tau_{pmd} \in \{0, 1\}, \qquad \forall p \in P, m \in M, d \in D_p, \tag{16}$$

$$y_{pd} \in \{0, 1\} \qquad \forall p \in P, d \in D_p, \tag{17}$$

$$z_{md} \in \{0\} \cup \mathbb{Z}^+ \qquad \forall m \in M, d \in D. \tag{18}$$

Objective function (1) minimizes the total cost of hiring interim workers and switching machines. Constraints (2) assign a machine to each worker on each of their working days, while Constraints (3) make sure that this worker is either training or qualified (but not both) for that machine on that day. Constraints (4) and (5) assign a sufficient number of operators to each machine for each day. Constraints (6) ensure that trainees are assigned to a machine only if a qualified worker is also assigned to the same machine. For each working day of each worker, Constraints (7) and (8) introduce a switch to be penalized in the objective function if on this day this worker works on a different machine than the one they worked on their previous working day.

Constraints (9) forbid a worker to work on a machine without supervision unless they are qualified for that machine on that working day. Constraints (10) indicate whether or not a worker is qualified to work on a machine independently on their first working day. Constraints (11) and (12) restrict the number of consecutive working days that a qualified worker can avoid working on a given machine and yet remain qualified for it. Constraints (13) ensure that a worker obtains the qualification to work on a machine without supervision only after sufficient training. The minimum number of required training days for this first qualification varies among workers and depends on their training level at the beginning of the planning horizon. By Constraints (14), each day a worker loses their qualification on a machine is captured and Constraints (15) ensure that upon losing the qualification this worker can become qualified on this machine again only after receiving the short retraining. Finally, Constraints (16)-(18) are binary and integer restrictions.

Note that the integer restrictions on $\mathbf{y}$ variables can be relaxed as the right hand sides of (7) and (8) are always integral and the coefficients of these variables are nonnegative in the minimization-type objective function. Given that n_{md} values are integer, a similar reasoning allows us to also relax the integrality restrictions on $\mathbf{z}$ variables as well, leading to an MIP variant. Additionally, the requirements satisfied by Constraints (4) and (5) can also be expressed as in Constraints (19) and (20), respectively, since an assigned personnel is either a trainee or a qualified worker. Utilizing (19) and (20) further enables replacing equality (3) with an inequality as in (21).

$$\sum_{p\in P:d\in D_p} w_{pmd} + \sum_{p\in P:d\in D_p} \tau_{pmd} \geq n_{md}, \qquad \forall m \in M^N,\ d \in D, \tag{19}$$

$$z_{md} + \sum_{p\in P:d\in D_p} w_{pmd} + \sum_{p\in P:d\in D_p} \tau_{pmd} \geq n_{md}, \qquad \forall m \in M^I,\ d \in D, \tag{20}$$

$$w_{pmd} + \tau_{pmd} \leq x_{pmd}, \qquad \forall p \in P, m \in M, d \in D_p, \tag{21}$$

These modifications lead us to the following MIP for the EMPATI:

$$\begin{aligned} (MIP) \qquad & \min\ (1) \\ & \text{s.t. } (2), (6)-(16), (19)-(21) \\ & 0 \leq y_{pd} \leq 1 \qquad \forall p \in P, d \in D_p, \qquad (22) \\ & z_{md} \geq 0 \qquad \forall m \in M, d \in D. \qquad (23) \end{aligned}$$

We obtain several valid inequalities by further analyzing the problem characteristics. These inequalities can be classified into two groups. The first group ensures that the binary variables corresponding to the following decision pairs cannot cannot take value one at the same time for a worker-machine combination:

Pair 1: lose qualification and be qualified on the same day (24)
Pair 2: lose qualification and work as a qualified worker on the same day (25)
Pair 3: lose qualification on a day and work on the previous day (26), (27)

$$\alpha_{pmd} + \gamma_{pmd} \leq 1, \qquad \forall p \in P, m \in M, d \in D_p, \tag{24}$$

$$\alpha_{pmd} + w_{pmd} \leq 1, \qquad \forall p \in P, m \in M, d \in D_p, \tag{25}$$

$$\alpha_{pmd} + \tau_{pme_{dp1}} + w_{pme_{dp1}} \leq 1, \qquad \forall p \in P, m \in M, d \in D_p, \tag{26}$$

$$\alpha_{pmd} + x_{pme_{dp1}} \leq 1, \qquad \forall p \in P, m \in M, d \in D_p. \tag{27}$$

Similarly, the second group ensures that the binary variables corresponding to the following decision pairs cannot take value one at the same time for a worker-machine combination:

Pair 1: train and be qualified on the same day (28)
Pair 2: train on a day and be qualified on the previous day (29)
Pair 3: train on a day and work as a qualified worker on the previous day (30)

$$\tau_{pmd} + \gamma_{pmd} \leq 1, \qquad \forall p \in P, m \in M, d \in D_p, \tag{28}$$

$$\tau_{pmd} + \gamma_{pme_{dp1}} \leq 1, \qquad \forall p \in P, m \in M, d \in D_p, \tag{29}$$

$$\tau_{pmd} + w_{pme_{dp1}} \leq 1, \qquad \forall p \in P, m \in M, d \in D_p. \tag{30}$$

3 Iterated local search

The MPATI is typically solved for a planning horizon of multiple months as the required training for a worker to qualify for a new machine typically takes several weeks. Preliminary experiments have shown that while integer programming may be used to find solutions when considering planning horizons of a few weeks, large-scale problems cannot be solved with available solvers. To address these problem instances, an Iterated Local Search (ILS) algorithm is introduced. Algorithm 1 outlines the main components of the proposed ILS, where S_0 is the initial solution and $f(S)$ is an evaluation function which returns the cost of solution S.

Algorithm 1: Iterated local search

```
Data: S_0, f(S)
Result: S
S ← localSearch(S_0);
while time limit not exceeded do
    S' ← perturb(S);
    S' ← localSearch(S');
    if f(S') ≤ f(S) then
        S ← S';
return S;
```

To obtain an initial solution S_0, a constructive heuristic first assigns one qualified worker to each machine in M^N. Then, randomly selected workers who are available are assigned until the minimum number of operators n_{md} is reached. Finally, any workers still available are assigned to the machines in M^I. These steps are repeated for each day in the planning horizon.

3.1 Solution representation

The proposed ILS operates on a direct solution representation consisting of a two dimensional matrix $S = (P \times D)$ whose values correspond to the machine assigned to worker $p \in P$ on day $d \in D$. To avoid feasibility issues, two hard constraints are relaxed during the search: machine staffing requirements (Constraints (4) and (5)) and trainee supervision (Constraints (6)). Violations of these constraints are penalized in the evaluation function $f(S)$. Possible compensation between the constraint penalties and the problem's real objective function (Equation (1)) is avoided by using a two-level lexicographic evaluation function. The first level sums all violations of the relaxed hard constraints while the second corresponds to the problem's original weighted sum function objective.

To identify violations of the trainee supervision constraint, an auxiliary datastructure is employed which maintains a training label $T_{pmd} = (l, c, \delta, s)$ for each worker $p \in P$, machine $m \in M^N$ and day $d \in D$. The label indicates the current skill level $l \in \{\text{none}, \text{trainee}, \text{qualified}\}$, the number of days c the worker has been level l, the previous day δ the worker was assigned to machine m and a boolean s which indicates whether or not the worker is eligible for a short retraining on this machine. Algorithm 2 sets the training labels for worker p on machine m from day d' onwards. The procedure updateWorking() adjusts the training label on day d after working an additional day on the machine. It is possible for a worker to transition from trainee to qualified if the required number of training days is reached. However, if the last time the worker was assigned to the machine was too long ago, that is, $d - \delta > C_p$, it is also possible to transition back to the trainee status. Similarly, updateIdle() sets the training label on day d when the worker is not assigned to the machine. All transitions are possible in this case, even to the *none* status when considering the situation where a worker was previously qualified but has lost it on this day.

Algorithm 2: Setting training labels

```
Data: p, m, d'
foreach d ∈ D : d ≥ d' do
    if S(p, d) = m then
        T_pmd ← updateWorking();
    else
        T_pmd ← updateIdle();
```

As Algorithm 2 only updates the training labels from day d' onwards, delta evaluation of the trainee supervision constraint is straightforward to implement. Similarly, identifying violations of Constraints (4)

and (5) and calculating the problem's real objective may be accelerated by only re-calculating the parts of the solution which have been changed.

3.2 Local search and perturbation

The main drivers of ILS' performance are the local search and perturbation procedures. In the local search performed at lines 1 and 4 in Algorithm 1, two parametrized neighborhood structures are used:

- *change*(k): assigns a worker to a machine for k consecutive days on which the worker is available,
- *swap*(k): swaps the assigned machines of two workers for k consecutive days on which these workers are available.

The local search procedure is instantiated with two variants of each neighborhood structure: one in which k is randomly chosen in each iteration from the interval $[1, 30]$ and another with $k = 1$. Rather than performing a full neighborhood search in each iteration, one new solution is randomly sampled and evaluated.

The perturbation procedure at line 3 in Algorithm 1 is used to escape from local optima reached by the local search procedure. This is achieved through a random walk of γ steps in the *change*(k) and *swap*(k) neighborhoods. Based on preliminary experiments, this parameter was set to $\gamma = 100$.

4 Computational study

In order to evaluate the performance of our methods under different planning settings, we randomly generated five sets of instances based on the real data provided by the company. In all instances, the interim cost $f = 264$, the switching cost $\pi = 60$, the number of workers $|P| = 40$ and the number of machines $|M| = 15$. Only three of these machines can be operated by interim workers, that is, $|M^I| = 3$. Each instance set corresponds to a fixed-length planning horizon, as shown in Table 1. For all instances in the same set, the number of days required for training is identical for all machine-personnel pairs, that is, $l_{pm} = \lambda$ for all $m \in M$ and $p \in P$. However, not every worker can be trained on a machine; this 0-1 (binary) parameter value is chosen randomly and may vary among instances of the same set. Similar to training, $C_p = C$ and $R_p = R$, $\forall p \in P$ in all instances of the same set. In addition to $|D|$, Table 1 reports the l, C and R values for each instance set.

Table 1 Planning horizon, training, skill memory and short retraining lengths (in days) for each instance set.

Instance set	Horizon $\lvert D\rvert$	Training λ	Skill memory C	Short training R
1	5	1	∞	0
2	10	2	5	1
3	20	4	10	2
4	60	12	20	3
5	260	60	20	3

Each set contains four instances which can be subdivided into two groups. For each group, a unique combination of random data generation probabilities is employed. These probability combinations imply that workers in the first group have fewer skills and less past training (yet to get the qualification) than those in the second group at the beginning of the planning horizon. Moreover, in the long run, for the first group fewer machines are available and a higher number of workers are needed.

We implement all mathematical models and the ILS algorithm in Java and use CPLEX 12.8 to solve the mathematical formulations. The experiments are run on a cluster of Intel Xeon CPU E5-2860 @2.50GHz with 24 cores and 64GB of RAM. The time limit for each run is set to five hours for each method. At most four threads are allowed for CPLEX runs. The ILS is run ten times per instance with different seed values for the algorithm's random number generator.

The first column of all the remaining tables in this section provides the instance code in format '$\#^1\_\#^2$' where $\#^1 = |D|$ indicates the length of the planning horizon and $\#^2$ is a unique identifier to associate the instance with the aforementioned groups and seeds. More specifically, instances with $\#^2 = 1, 2$ belong to the first group while those with $\#^2 = 3, 4$ belong to the second group. Columns 'Obj' provide the value of the solution obtained from the corresponding method whereas 'g%' indicates the gaps reported by CPLEX at the end of the time limit.

Table 2 Comparison of mathematical models with and without valid inequalities

		IP		MIP		IP + (24)-(30)		MIP + (24)-(30)	
Inst.	Best Obj	Obj	g%	Obj	g%	Obj	g%	Obj	g%
5_1	360	**360**	73.3	**360**	73.3	**360**	**72.8**	**360**	73.3
5_2	120	**120**	65.6	**120**	**50.0**	**120**	**50.0**	**120**	**50.0**
5_3	10044	**10044**	1.8	**10044**	1.8	**10044**	2.7	**10044**	**1.2**
5_4	8952	**8952**	6.0	**8952**	5.4	**8952**	**4.7**	**8952**	6.0
Avg.	4869.0	**4869.0**	36.7	**4869.0**	32.6	**4869.0**	**32.5**	**4869.0**	32.6
10_1	540	**540**	**65.4**	**540**	67.3	**540**	67.3	**540**	67.3
10_2	360	**360**	**68.6**	**360**	70.1	420	74.4	420	74.4
10_3	16344	**16344**	**17.3**	16644	18.8	**16344**	**17.3**	18084	25.2
10_4	12972	**12972**	22.2	**12972**	**21.7**	**12972**	22.7	**12972**	22.2
Avg.	7554.0	**7554.0**	**43.4**	7629.0	44.5	7569.0	45.4	8004.0	47.3
20_1	13044	**13044**	**94.8**	27912	97.6	41556	98.4	40152	98.3
20_2	12384	14700	96.7	34608	98.6	**12384**	**96.1**	44640	98.9
20_3	54084	**54084**	**45.2**	-	-	-	-	-	-
20_4	47292	**47292**	**39.8**	60108	52.7	54120	47.4	54120	47.4
Avg.	31701.0	**32280.0**	**69.1**	40876.0	83.0	36020.0	80.6	46304.0	81.6
Avg.	14708.0	14901.0	**49.7**	15692.7	50.7	14346.5	50.3	17309.5	51.3

Table 2 provides the results obtained from the two mathematical formulations with and without the valid inequalities presented in Section 2. These models are not tested on instances with a planning horizon longer than 20 days as they are too large for CPLEX to handle. The best solution value obtained among the four models is reported under column 'Best Obj'. Although the MIP combinations terminate with a smaller dual gap for some instances, the IP provides the best gaps on average and the best solution values in all but one instance where the IP with the valid inequalities produces the best solution. We also observe that the gaps are larger for instances from the first group in each set.

Table 3 Comparing the ILS solution values with the best IP/MIP results

		Best IP/MIP		ILS		
Inst.	Best Obj	Obj	LB	Min Obj	Max Obj	Avg. Obj
5_1	360	**360**	97.9	**360**	**360**	**360.0**
5_2	120	**120**	60.0	**120**	**120**	**120.0**
5_3	10044	**10044**	9923.0	**10044**	**10044**	**10044.0**
5_4	8952	**8952**	8532.0	**8952**	**8952**	**8952.0**
Avg.	4869.0	**4869.0**	4653.2	**4869.0**	**4869.0**	**4869.0**
10_1	540	**540**	186.0	**540**	**540**	**540.0**
10_2	360	**360**	112.9	**360**	**360**	**360.0**
10_3	16344	**16344**	13524.0	**16344**	**16344**	**16344.0**
10_4	12972	**12972**	10152.0	**12972**	**12972**	**12972.0**
Avg.	7554.0	**7554.0**	5993.7	**7554.0**	**7554.0**	**7554.0**
20_1	1980	13044	681.9	**1980**	2040	2028.0
20_2	1800	12384	488.0	**1800**	1920	1890.0
20_3	35844	54084	29664.0	**35844**	**35844**	**35844.0**
20_4	37212	47292	28452.0	**37212**	**37212**	**37212.0**
Avg.	19209.0	31701.0	14821.5	**19209.0**	19254.0	19243.5
Avg.	10544.0	14708.0	8489.5	**10544.0**	10559.0	10555.5

In Table 3, we compare the solution values of the ILS with the best upper and lower bound values obtained from the mathematical formulations. The column 'Best Obj' reports the best solution value obtained from

the four mathematical models and the ten runs of the ILS. The last three columns of this table provide the minimum, maximum and the average solution values from the ten runs of the ILS for each instance. The ILS is able to find a solution with the same objective value as the best model solution in all instances with $|D| = 5$, 10, and for the instances with $|D| = 20$ it provides a better solution than the best obtained by the formulations. Although the solution values may seem far from the lower bound values, by introducing the ILS solutions to CPLEX via a warm start function (addMIPStart) we are able to confirm that the heuristic solutions are in fact optimal for the instances with $|D| = 5$. Therefore, it is likely that the solutions provided by the ILS for larger instances are also optimal or near-optimal.

Table 4 Detailed results for all instances solved by the ILS approach

	ILS					
	Obj				Average number of	
Inst.	Min	Max	Avg,	Std.	Switches	Interims
5_1	360	360	360	0.00	6	0
5_2	120	120	120	0.00	2	0
5_3	10044	10044	10044	0.00	53	26
5_4	8952	8952	8952	0.00	48	23
10_1	540	540	540	0.00	9	0
10_2	360	360	360	0.00	6	0
10_3	16344	16344	16344	0.00	92	41
10_4	12972	12972	12972	0.00	93	28
20_1	1980	2040	2028	25.30	33.8	0
20_2	1800	1920	1890	42.43	31.5	0
20_3	35844	35844	35844	0.00	197	91
20_4	37212	37212	37212	0.00	233	88
60_1	7260	7560	7416	117.30	123.6	0
60_2	4500	4800	4644	110.27	77.4	0
60_3	109800	109800	109800	0.00	664	265
60_4	108192	108192	108192	0.00	646	263
260_1	37008	39828	38136	945.64	626.8	2
260_2	28680	30480	29730	476.03	495.5	0
260_3	471396	471516	471450	52.54	2823.9	1144
260_4	482040	482220	482112	61.97	2909.2	1165

Table 4 presents the detailed results obtained from the ILS when solving all instances. In addition to the minimum, maximum and average, this table also provides the standard deviation of the solution values of the ten runs in the fifth column. The solution values of the ten runs are identical for 60% of the instances. The average standard deviation across all instances is 112.3. The last two columns of this table provide the average number of switches and interim workers hired. We observe that the solutions for the first group of instances have fewer switches and interim workers compared to the second group. Although it is costly, it is worth noting that switching machines is unavoidable in order to obtain a feasible solution in these instances.

5 Conclusion

This paper addressed a new staff assignment problem where daily personnel requirements of different skills may vary over the planning horizon. This suggests varying duty assignments for the personnel which may eventually lead to losing skills if not used for too long. It is possible to train or retrain for different skills, however, these training decisions result in a very challenging problem for which even finding a feasible solution necessitates a significant amount of computational effort. Another challenging component penalizes switching duties as it requires an additional adjustment period and is undesirable by personnel. Together with the decisions concerning the number of interim workers to hire on a daily basis, reaching high quality solutions in a reasonable amount of time is only possible via tailored methods.

We introduced integer and mixed integer programming formulations with several valid inequalities. In our experiments, these formulations reached high quality solutions within a reasonable amount of time for

problems with short planning horizons. However, they were unable to handle problems with long planning horizons with the commercial solver that was utilized. The solver also had difficulties in closing the dual gap, especially, for the first group of instances where the employees have fewer skills and less training at the beginning of the planning horizon. This motivated the development of an iterated local search algorithm to solve larger problem instances, which produced high quality solutions when solving the instances generated.

An interesting extension to the problem could be to include shift rostering as an additional decision as this would enable grouping operators with complementary skills in addition to cross-training them. Alternatively, the problem could be extended to enable training of the interim workers so that they can be allocated to the machines which require more complicated skills, more specifically, the non-interim machines. From a methodological perspective, it is worthwhile investigating the structural properties of the proposed mathematical models for developing possibly competitive matheuristic approaches.

Acknowledgments

This research was supported by Dynamical Combinatorial Optimization (KU Leuven C2 project C24/17/012) and Data-driven logistics (FWO-S007318N). Editorial consultation provided by Luke Connolly (KU Leuven).

References

1. Azzouz, A., Ennigrou, M., Ben Said, L.: Scheduling problems under learning effects: classification and cartography. International Journal of Production Research **56**(4), 1642–1661 (2018)
2. Biskup, D.: A state-of-the-art review on scheduling with learning effects. European Journal of Operational Research **188**(2), 315–329 (2008)
3. Brusco, M.J., Johns, T.R.: Staffing a multiskilled workforce with varying levels of productivity: An analysis of cross-training policies. Decision Sciences **29**(2), 499–515 (1998)
4. De Bruecker, P., Beliën, J., Van den Bergh, J., Demeulemeester, E.: A three-stage mixed integer programming approach for optimizing the skill mix and training schedules for aircraft maintenance. European Journal of Operational Research **267**(2), 439–452 (2018)
5. De Bruecker, P., Van den Bergh, J., Beliën, J., Demeulemeester, E.: Workforce planning incorporating skills: State of the art. European Journal of Operational Research **243**(1), 1–16 (2015)
6. Ernst, A.T., Jiang, H., Krishnamoorthy, M., Sier, D.: Staff scheduling and rostering: A review of applications, methods and models. European Journal of Operational Research **153**(1), 3–27 (2004)
7. Guerry, M.A., De Feyter, T., Berghe, G.V., et al.: The roster quality staffing problem–a methodology for improving the roster quality by modifying the personnel structure. European Journal of Operational Research **230**(3), 551–562 (2013)
8. López, C.E., Nembhard, D.: Cooperative workforce planning heuristic with worker learning and forgetting and demand constraints. In: Proceedings of the 2017 Industrial and Systems Engineering Conference, pp. 380–385 (2017)
9. Nembhard, D.A., Bentefouet, F.: Parallel system scheduling with general worker learning and forgetting. International Journal of Production Economics **139**(2), 533–542 (2012)
10. Van den Bergh, J., Beliën, J., De Bruecker, P., Demeulemeester, E., De Boeck, L.: Personnel scheduling: A literature review. European Journal of Operational Research **226**(3), 367–385 (2013)
11. Wirojanagud, P., Gel, E.S., Fowler, J.W., Cardy, R.: Modelling inherent worker differences for workforce planning. International Journal of Production Research **45**(3), 525–553 (2007)

Personnel scheduling considering employee well-being: insights from case studies

Sanja Petrovic[1], Jane Parkin[2], David Wrigley[3]

[1] Nottingham University Business School, Nottingham, UK, sanja.petrovic@nottingham.ac.uk

[2] Carr House Consulting, UK Jane Parkin, janeparkinch@gmail.com

[3] Orvis Consulting Ltd, UK, david.wrigley@ntlworld.com

Abstract

This research is concerned with new metrics we suggest for measuring employee well-being to be included in personnel scheduling. The well-being of employees has attracted the interest of researchers mostly in the field of Occupational Medicine, over the past decade. The aim of this paper is to bring the issue of employee well-being to the Timetabling community. We provide an overview of well-being literature followed by a description of the well-being measures that we propose to be included in the objective function while generating a roster. These measures include work-life balance measures, Fatigue and Risk indices, and compliance with Health and Safety Executive (HSE) guidelines. As an experimental environment, we use three case studies of personnel scheduling, which are of very different nature: scheduling of fixed shift patterns, nurse rostering and cyclic scheduling of flexible shifts in a call-centre service industry. Our aim is to investigate to what extent employee well-being can be improved while maintaining high performance of rosters. We analyse and compare proposed rosters generated manually and by computerised algorithms, with and without well-being measures.

Key words: personnel scheduling, well-being measures, fatigue, case studies

1. Introduction

Personnel scheduling has been a subject of interest within both Operational Research and Artificial Intelligence research communities for a number of decades. It can be stated as an assignment problem in which personnel are assigned to shifts to cover the demand for resources that varies over time (Ernst et al. 2004b). The assignment is subject to given rules/constraints, which are used to apply restrictions to the timetable. These are usually divided into two categories: "hard" restrictions which are rigidly enforced and "soft" restrictions whose satisfaction is desirable but not essential because it is often difficult or impossible to satisfy all of them. In general, the level of satisfaction with the soft constraints in a generated timetable determines its quality. Usually, workforce planning/staffing, which deals with strategic decisions related to the optimal size or mix of a workforce, precedes timetabling, i.e. in timetabling it is assumed that staffing levels are given. Also, in each time period of the rostering horizon the demand is calculated and serves as input to timetabling. In practice, personnel scheduling problems can be very hard to solve, and their manual solution requires much effort.

Separately from these research efforts, there is increasing research into well-being of employees related to work, but mostly reported in the field of occupational medicine. In the literature review of the effect of employee well-being on their work, it was stated that employees high in well-being are generally more productive at work, have higher incomes, and have better physical and mental health (Jeffrey et al, 2014). There is evidence that improving employee well-being improves organisational outcomes and also has impact on sickness absence, presenteeism, employee engagement, retention and performance (Donald et al, 2005). This particularly applies to tasks that require vigilance and monitoring, decision making, awareness, fast reaction time, tracking ability, and/or memory. Generally, personnel scheduling methods do not measure employee well-being in any formal or objective way, apart from imposing some constraints, hard or soft, to reflect the EU Working Time Directive (2003), restrictions on night shifts, etc. The aim of this paper is to bring the research into well-being of employees related to work to the attention of the timetabling community. The paper is organised as follows. First a literature review of main findings about well-being at work is given, followed by a description of the proposed measures for employee well-being. We report our insights into these measures and their effect on the performance of the rosters based on three case studies.

2. Literature review on well-being at work

We group the papers into different categories based on aspects of employee well-being they investigate. The categories and the selection of relevant papers are presented in Table 1.

There is a large number of papers that report on the **effects of well-being on organisational outcomes**, such as productivity, absenteeism, retention, etc. An extensive piece of research was carried out by Donald et al. (2005), which involved 15 different organisations in the public and private sectors in the UK spanning a range of occupations, from professional to administrative and manual roles. Young and Bhaumik, (2011) carried out a Health and Well-being Employee Survey for Department for Work and Pensions in the UK. The conclusions were that better psychological well-being resulted in higher employee productivity; there was an association between high employee engagement, positive views about work life balance and high employee retention/low sickness absence. Factors that affect the job performance, retention and sickness absence were investigated by Zedeck et al (1983), Diener and Seligman (2004), and others. Warr and Nielsen (2018) discussed different types of context-free and job-related well-being, and reviewed research into association between well-being and work performance through specific factors such as being creative or proactive at work.

Many employees work alternating shifts, which usually last between 6 to 12 hours (Harrington, 2001). The traditional three 8-hour shifts start at 06:00, 14:00, and 22:00 hours. However, there are many variations of shifts and employees can rotate through different shifts with variable degrees of speed of rotation and direction of rotation. A summary of **effects of shift work on health and safety**

reported by Harrington (2001) initiated other research into design of work schedule (see for example, (Folkard et al, 2005, Muecke, 2005, Fransen et al, 2006, Lowdan et al, 2010, Kubo et al, 2011, Schernhammer et al, 2011; Ferri et al 2016; Gärtner et al, 2019, etc.).

Table 1. Selection of the literature on well-being at work

Topic of the research	**Publications**
Effects of well-being on organisational outcomes	(Zedeck et al, 1983), (Diener and Seligman, 2004), (Donald et al, 2005), (Young and Bhaumik, 2011), (Warr and Nielsen, 2018),
Effects of shift work on health and safety	(Folkard et al, 2005), (Muecke, 2005), (Fransen et al, 2006), (Lowdan et al, 2010), (Kubo et al, 2011), (Schernhammer et al, 2011), (Gärtner at al, 2019)
Giving staff control over their shift patterns	(Ala-Mursula et al 2006), (Ingre et al 2012)
Effect of overtime work on hazard rate and health	(Smith et al, 1998), (Dembe et al, 2005), (Bell, 2012), (Wong et al, 2019)
Different shift lengths	(Cunningham, 1982), (Northrup, 1989), (Tucker et al, 1996), (Ellis, 2008)
Rest days	(Tucker et al, 1999), (Flo et al, 2014)
Delaying shift start and end times	(Rosa et al, 1996)
Forward-rotating schedules vs backward-rotating schedules	(Barton et al, 1994), De Leede and McCarrick (2011)
Comparison of fixed shifts to rotating shifts	(Martens et al, 1999), (Fereshteh et al, 2011)
Regular work more than 48 hours a week	(Spurgeon et al, 1997)
Night shift	(Khaleque, 1999), (Gander et al, 2007), (Ferri et al, 2016)
Comparison of different shift patterns	(Baxter and Mosby, 1988)

The positive effects of **giving employees control over their shift patterns,** so-called self-rostering, were reported by Ala-Mursula et al (2006), Ingre et al (2012), etc.

A number of papers quantified the **effect of overtime work on hazard rate and health**. Dembe et al (2005) found that working in jobs with overtime schedules was associated with a 61% higher injury rate compared to jobs without overtime. In line with that, working at least 12 hours per day and at least 60 hours per week was associated with a 37% and 23% increased hazard rate. Smith et al (1994) investigated the risk of injury at an engineering firm and found that it was 20% higher for staff on night shifts than for those on day shifts. Additional research on this topic includes (Bell, 2012) and (Wong et al, 2019).

Particular attention was paid to the research into the effect on well-being of different shift patterns, timings, lengths and distribution of rest days. There is considerable research into impacts of **different shift lengths**. Long shifts (e.g. 12 hour shifts) were popular with employees, largely because of the longer periods of time off and greater freedom on weekends and evenings (Northrup ,1989). However, Tucker et al (1996) reported that towards the end of a 12 hour shift there is a risk of fatigue and decreased alertness of employees especially when the job role is highly monotonous and sedentary.

The distribution of **rest days** was investigated, especially between day and night shifts, and their effect on alertness, chronic fatigue and other health problems were investigated by Tucker et al (1999), Flo et al (2014), etc.

Delaying shift start and end times also have an effect on the performance at work (Rosa et al, 1996).

Rotating schedules can move forwards (so-called **forward-rotating schedules**), which means that employees shifts progress from morning to afternoon to night in a clockwise direction, and opposite i.e. backward (so-called **backward-rotating schedules**). The research into the effects of the two types of rotation and of alternating them was conducted by Barton et al (1994) and De Leede and McCarrick (2011).

Generally employees can work fixed shifts, which are always the same, or rotating shifts which change often on a weekly basis. Research into **comparison of fixed shifts to rotating shifts** revealed it was generally more difficult for the body to adjust to rotating shifts and working shifts might lead to health complaints **(**Martens, et al, 1999, Fereshteh et al 2011).

Negative effects of **regular work more than 48 hours a week** and **night shifts** were reported by Spurgeon et al (1997) and Khaleque (1999), respectively. Ferri et al 2016 concluded that nurses with rotating night schedules had a higher risk of both job dissatisfaction and undesirable health effects. It was also found that night work and schedule instability were independently associated with more fatigue measures than was total hours worked (Gander et al, 2007).

Baxter and Mosby (1988) compared **different shift patterns** by using a function to penalise long work periods unless followed by free periods long enough to compensate. However, this research considered only the pattern of working and free days, but not shifts of different lengths and different start times.

3. Well-being measures

Based on the literature on well-being, we propose the following well-being measures: work-life balance, Fatigue and Risk indices, and deviations from the Health and Safety Executive (HSE) guidelines (Parkin

and Petrovic, 2015). HSE is an organisation which provides a regulatory framework for work place health and safety in Great Britain.

3.1. Work-life balance measure

This measure serves as an indicator of the quality of the free time in an employee's roster. It is adapted from Cunningham's research work (1982) and counts the number of 2 day free weekends, free weekend days, free evenings, free blocks of at least 2 days, and free days in the given rostering period. These components are assigned weights in a subjective manner. In order to assign a single value of this metric for each employee in the given rostering period, a 'traditional' working pattern, which consists of 5 working days (Monday-Friday) with working hours 9am-5pm followed by 2 free days is used as basis. For each component, the deviation of its value from the corresponding value in a 'traditional' working pattern is calculated. Then the work-life balance measure for each employee is the average of the weighted sum of deviations of all work-life balance components. The higher value the better work-life balance measure for the given employee.

3.2. Fatigue and Risk indices

The HSE introduces two indices, Fatigue and Risk, and proposes how to calculate them. The value of Fatigue index relates to the probability of high levels of sleepiness, while Risk index is expressed in terms of the relative risk of an accident/incident occurring. Both indices are constructed from three separate components:

1. A cumulative component relates to the way in which individual duty periods or shifts are put together to form a complete schedule. The cumulative component associated with a particular shift depends on the pattern of work immediately preceding that shift.
2. A component associated with duty timing includes the effect of start time, shift length and the time of day throughout the shift.
3. A job type/breaks component relates to the content of the shift in terms of the activity being undertaken and the provision of breaks during the shift. Factors considered in this component include typical commuting time to or from work, workload, whether the job requires continuous attention, if rest breaks are taken, longest period of work before a break, and rest break after longest work period.

It is based on the Karolinska Sleepiness Scale, which ranges from 'extremely alert' to 'extremely sleepy – fighting sleep', and takes a value from the [0,100] interval (Akerstedt and Gillberg, 1990). The Fatigue index is calculated for each shift of an employee and then the average and maximum values are calculated over the whole rostering period.

3.3. Deviations from HSE guidelines

HSE investigated the effect of the shift-work on employees, and recommended the following good practice guidelines for shift-work schedule design (HSE, 2006):

- Avoid placing workers on permanent night shifts.
- If possible, offer workers a choice between permanent and rotating shift schedules.
- Where possible, adopt a forward-rotating schedule for rotating shifts rather than a backward-rotating schedule
- Either rotate shifts very quickly, e.g. every 2-3 days or slowly, e.g. every 3-4 weeks and avoid weekly/fortnightly rotating shift schedules.
- Limit shifts to a maximum of 12 hours (including overtime)
- Limit night shift or shifts where work is demanding, monotonous, dangerous and/or safety critical to 8 hours
- Consider if shifts of a variable length or flexible start/end times could offer a suitable compromise
- Avoid split shifts unless absolutely necessary to meet business needs.
- In general, limit consecutive working days to a maximum of 5-7 days and make sure there is adequate rest time between successive shifts.
- Where shifts are long (> 8 hours), for night shifts and for shifts with early morning starts, it may be better to set a limit of 2-3 consecutive shifts.
- When switching from day to night shifts or vice versa, allow workers a minimum of 2 nights' full sleep.
- Build regular free weekends into the shift schedule.
- Control overtime and shift swapping by monitoring and recording hours worked and rest periods.

For each employee, the compliance to the HSE guideline in the roster is measured, having value 1 if the corresponding guideline is followed, 0 otherwise. Our third proposed well-being measure calculates the overall compliance, where 100% is assigned to rosters which meet all the guidelines.

4. Insights obtained from using three case studies

We selected three case studies, which dealt with personnel scheduling problems of very different nature to obtain insights into the appropriateness of the definition of well-being measures and their use in the generation of schedules together with the roster performance metrics.

4.1. Shift-staff work across multiple sites

Our first case study was concerned with a large governmental organisation, which operated across a number of sites and whose staff worked shifts that cycled over several weeks. Our task was to review the effectiveness of personnel scheduling in the organisation focussing on staff welfare and the efficient deployment of resources. At each site, a different set of fixed shift patterns was used. Interviews with managers, Human Resources representatives, rostering staff and Trade Union representatives were conducted to explore their views on what constitutes a good/bad roster, and on elements of staff well-being. Staff were rather unhappy with their schedules at one site compared to other and complained on the effect that schedules had on their life and health. Our proposed well-being measures had different values among the sites and they vastly matched the views of the interviewed staff on their well-being.

4.2. Nurse rostering problems

Nurse rostering problems are known to be notoriously hard to solve due to a large number of constraints to consider. In our research, we used some of the nurse rostering instances from the Web page http://www.cs.nott.ac.uk/~psztc/NRP/, which are mostly based on real-world problems. As an illustration, we show the results obtained on an instance referred to as MER. In the MER instance, there was a large number of nurses (54); the rostering period was large (6 weeks), and that enabled us to adequately investigate the impact of introducing well-being measures. There was a great variety of shift types (12), so we could investigate the effects of fewer longer shifts versus more shorter shifts on the fatigue indicators. Cover constraints were defined for time periods rather than for shift types. Hard constraints included bounds on the required cover constraints and the minimum and maximum number of hours each nurse could work per fortnight. The soft constraints included preferred levels of the cover and requests for shifts and days off.

A rostering algorithm based on Large neighbourhood search was employed (Burke et al, 2013). The performance of the rosters was determined by the total penalty caused by the violation of soft cover constraints and shift and days-off requests. We compared the results obtained without considering well-being (referred to as baseline) and results obtained with the modified objective function that included the well-being measures with the aim to maximise the work-life balance measure, minimise maximum fatigue and risk indicators, and maximise adherence to HSE guidelines over all employees. Also, all the components in the original objective function that were related to well-being of employees were excluded, such as the constraint on maximum number of consecutive days on, maximum number of consecutive night shifts, etc. because if included they would reinforce some of the well-being measures.

The weights of the well-being components in the objective function were set based on the values of the corresponding measures obtained in the initial real-world problem that initiated our research into

well-being measures in rostering, and also the values obtained in the rosters created by the original nurse rostering algorithm. The weights are given in Table 2.

Table 2. Weights assigned to well-being measures

Work life balance	**Maximum fatigue**	**Maximum risk**	**HSE guidelines**
1000 if score<65% 100 if 65%≤score<75% 10 if 75%≤score<85%	1000 if score>30 500 if 30≥score>25 100 if 25≥score> 20 10 if 20≥score>15	1000 if max risk>1.30 100 if 1.30≥max risk>1.20 10 if 1.20≥max risk>1.10	1000 if score≤7 100 if 7 <score≤8 10 if 8<score≤9

Table 3 presents the average and worst values of well-being measures over all nurses, the number (percentage) of nurses who had the same or better value of the corresponding well-being measure in the new roster than in the baseline, and the performance of the roster. Best results are given in bold. The new roster had all well-being measures the same or better compared to the baseline roster. For example, in the new roster, maximum fatigue was much smaller than in the baseline solution with no nurse having maximum fatigue above 42.7 compared to 53.1 in the original roster. Also, no nurse had average fatigue above 33.8 compared to 43.3 in the original one. Similarly, 83% of the nurses had the same or smaller average risk than in the baseline roster.

The improvements of all well-being measures came at the price of the performance of the roster; namely, the trade-off between the well-being measures and performance had to be made. In the new roster the penalty of the cover constraints and shift requests constraints were 1300 and 2720, respectively, while they were smaller in the baseline, 256 and 2040, respectively. In the baseline solution, 204 shift requests out of 1008 were not satisfied, compared to the new roster in which 272 requests were not satisfied. A shift request was a preference by a nurse to work a specific shift on a certain day or conversely not work at all on a certain day, and the weight assigned to the violation was 10. On examining the solutions it could be seen that some of these requests were actually in conflict with the good work-life balance measures and fatigue indicator values, which explained why they were less satisfied in the new roster. In particular, one nurse requested to work all night shifts, which she was assigned, causing a low work-life balance measure, 17.9%, and maximum fatigue of 42.7.The objective function compensated low well-being measures with a good performance measure and that led to the roster assigned to the nurse which satisfied her/his preferred night shifts. Similarly, in the new roster, the cover constraints were slightly less satisfied compared to the baseline solution. In particular, a maximum cover preference constraint was often broken. This constraint referred to a low-weighted preference to restrict the number of nurses scheduled during day shifts. In the baseline solution this was quite well satisfied. The reason for more violations in the new roster was that work-life balance and fatigue indicator encourage assignment of night shifts to as few nurses as possible. However, the nurses

had minimum contracted hours to be satisfied so these night shifts were replaced in the solutions with day shifts but at the expense of breaking the low-weighted maximum day time cover requirements.

In our further experiments, with a careful change of weights of the well-being components in the objective function, and inclusion of additional constraints (e.g. a maximum number of night shifts) it was possible to improve the performance of the rostering algorithm and at the same time some well-being measures.

Table 3. Well-being measures and performance achieved in the baseline and the new rosters for MER instance

	Baseline	**Roster with well-being**
WLB Average (worst) As well as/better off (percentage)	75.5% (**27.0%**)	**86.8%** (17.9%) 38 (70%)
HSE Average (worst) As well as/better off (percentage)	61.9% (33.3%)	**83.5%** (**66.7%**) 48 (89%)
Average fatigue of employee Average (worst) As well as/better off (percentage)	17.6 (43.3)	**5.4** (**33.8**) 49 (91%)
Maximum fatigue of employee As well as/better off (percentage)	34.7 (53.1)	**9.5** (**42.7**) 46 (85%)
Average risk of employee Average (worst) As well as/better off (percentage)	0.90 (1.20)	**0.84** (**1.05**) 45 (83%)
Maximum risk of employee As well as/better off (percentage)	1.20 (2.33)	**1.03** (**1.45**) 44 (81%)
Performance	Cover constraints 256 Shift constraints 2040	Cover constraints 1300 Shift constraints 2720

When comparing rosters generated by using original objective functions with modified ones, we noticed that increasing the number of working shifts did not affect linearly the well-being measures. An appropriate distribution of working shifts was more important for employees' work-life balance measure than the number of shifts. Similarly, it was possible to increase the number of working shifts for a nurse in her/his roster without compromising her/his fatigue. Even more, it happened that in the roster with increase in the number of shifts worked compared to a baseline, the average fatigue in the roster was decreased, while only marginally increasing the maximum fatigue.

The well-being measures can be easily interpreted when staff have to work to standard rosters. However, nurses often use a self-rostering system and express their preferences for working and/or non-working particular shifts. This implies interesting questions. Is there a need to measure the work-life balance, which is highly subjective, given that staff have implicitly considered their own work-life balance when stating their preferences? Our view is that, organisations could use the well-being measures to advise staff of the implications of their preferences and to warn them if the values of the

measures are particularly low. For example, in general, staff prefer longer, fewer shifts (e.g. 12 hour shifts, 4 days on, 4 days off) but this considerably worsens the Fatigue index.

4.3. Shift scheduling in a call-centre

Recent years have witnessed a large growth in the call-centre industry. Call-centres deal with phone calls (and possibly e-mails) and usually have to provide 24/7 service. This case study was concerned with a large call-centre in the UK with a target to answer 90% of calls within 20 seconds (Petrovic et al, 2018). This service had to be provided with a given number of employees. The initial consultancy work focused on the analysis of different shift patterns and their effect on roster performance using simulation modelling. The next step was to generate optimal schedules given a number of employees and shift patterns agreed with the Performance manager and Human Resource manager. We developed an optimisation model for designing shift patterns and personnel scheduling. The input to our problem was a shift pattern i.e. a sequence of shifts. Every employee worked one shift per day and shifts allocated to the employee cycled throughout the shift pattern. In our model, there were two types of decision variables: to determine a start time and duration of each shift, while shifts were subject to constraints, and to allocate to each employee a start day in a shift pattern while satisfying the constraints on the employees' contracts. The constraints on each shift included the earliest and latest start and end time, and the minimum and maximum duration. The allocated start day determined the roster for that employee for the whole rostering horizon. Two metrics were defined as performance measures for the generated roster: (1) ineffectiveness, which measured the degree to which the business requirement has not been met, i.e. there was a shortage of employees required to meet the demand predicted for the time period, and (2) inefficiency, which measured the degree to which nugatory hours have been scheduled. The objective function was defined as a weighted sum of ineffectiveness, inefficiency, Fatigue index, work-life balance, and HSE Guideline compliance index. In our previous case study, it was noted that Fatigue and Risk indices were highly positively correlated, and consequently we decided to keep only the Fatigue index. The work-life balance indicator was predefined so that smaller values were better. The objective function had to be minimised. An Evolutionary Algorithm to search for best schedules was developed using Solver add-in within Excel. Our problem instance was formulated based on the call-centre and had 20 employees, while the rostering horizon was 56 days. The generated schedules were analysed and compared with the schedules generated manually in the call-centre and schedules generated by the Evolutionary Algorithm without well-being components in the objective function. The results are presented in Table 4. The weight of ineffectiveness was larger than of other objectives because the shortage of staff had to be met via the allocation of additional staff, which was a costly solution. All schedules were 100% efficient, i.e. did not have employees unnecessarily scheduled in a time period. However, our schedule improved ineffectiveness by 20% i.e. matched better the predicted demand, improved considerably the adherence to HSE guidelines, and improved slightly the work-life

balance. The Fatigue index was not improved probably due to the constraints on shifts design, which implicitly took care of fatigue.

Table 4. Results

Objective	Weight	Call centre	Evolutionary Algorith without well-being	Evolutionary Algorith witht well-being
Inefficiency	1	**0.0000**	**0.0000**	**0.0000**
Ineffectiveness	1.4	0.0395	**0.0257**	0.0315
Fatigue	1	**0.0758**	0.0921	0.0795
HSE	1	0.1167	0.1167	**0.0467**
Work-life balance	1	0.0254	0.0254	**0.0246**
Objective function		0.2733	0.2701	**0.1948**

We observed that Evolutionary Algorithm was able to generate several different schedules with similar values of the objective function, i.e. similar values of the performance and well-being components of the objective function. However, they were very different in terms of shift hours. This may serve very well in the discussion between the management and staff, who assess schedules from different aspects. It may give an opportunity to staff to have more power in shift negotiations without compromising the performance of the rosters.

5. Conclusions

The presented research is concerned with the important issue of addressing employee well-being in modelling and solving personnel scheduling problems. Based on the available literature we propose measures for well-being which should be explicitly included in personnel scheduling. Three case studies were used to evaluate the appropriateness of the proposed measures, and the effect of having well-being measures on the quality of the generated rosters. Our research shows that it is possible to improve staff well-being with appropriately constructed rosters without seriously compromising business efficiency and effectiveness.

Recently, suggestions to include well-being of employees in personnel scheduling problems and also in future rostering competitions has emerged (Gärtner et al, 2018). The researchers propose additional constraints to address employees' well-being (opposite to our approach in which explicitly defined well-being measures are included in the objective function). Especially encouraging is that these suggestions were made by practitioners with many years of experience in personnel scheduling. We believe that employee well-being measures should also be taken into account in other phases that precede and follow the generation of rosters, i.e. the planning/staffing decision on the number of employees required to cover the predicted demand and the re-scheduling activities which often happen in the real-world and involve changes to the roster to respond to unforeseen circumstances.

Acknowledgement. The authors would like to thank Dr Tim Curtois for the collaboration on the Case study on well-being in nurse rostering.

References

Akerstedt T and Gillberg M (1990). Subjective and objective sleepiness in the active individual. *International Journal of Neuroscience* **52**: 29–37.

Ala-Mursula L, Vahtera J, Kouvonen A, Väänänen A, Linna A, Pentti J, Kivimäki M (2006). Long Hours in Paid and Domestic Work and Subsequent Sickness Absence: Does Control over Daily Working Hours Matter? *Occupational and Environmental Medicine.* 63(9): 608-616.

Barton J, Folkard S, Smith L and Poole C J M (1994). Effects on Health of a Change from a Delaying to an Advancing Shift System. *Occupational and Environmental Medicine* **51**(11): 749-755

Baxter J and Mosby M (1988). Generating Acceptable Shift-Working Schedules. *Journal of the Operational Research Society* **39**: 537–542.

Bell D, Otterbach S and Sousa-Poza A (2012). Work Hours Constraints and Health. *Annals of Economics and Statistics*, **105**(106): 35-54.

Burke E.K., Curtois, T. Qu E. and Vanden Berghe G. (2013), "A Time Predefined Variable Depth Search for Nurse Rostering". *INFORMS Journal on Computing*, Vol 25, No 3, pp. 411-419.

Cunningham J B (1982). Compressed Shift Schedules: Altering the Relationship between Work and Non-Work. *Public Administration Review*, Wiley, 42(5): 438-447.

de Leede J and McGarrick J L (2011). The effect of shift systems on worker health and safety. *Ergonomist*, May issue: 12-13.

Dembe A E, Erickson J B, Delbos R G, Banks S M (2005). The Impact of Overtime and Long Work Hours on Occupational Injuries and Illnesses: New Evidence from the United States. *Occupational and Environmental Medicine.* 62(9): 588-597.

Diener E and Seligman M E P (2004). Beyond Money: Toward an Economy of Well-Being. *Psychological Science in the Public Interest* 5(1): 1-31.

Donald I, Taylor P, Johnson S, Cooper C, Cartwright S, and Robertson S (2005). Work Environments, Stress, and Productivity: An Examination using ASSET. *International Journal of Stress Management* **12**(4): 409-423.

Ellis J R (2008). *Quality of Care, Nurses' Work Schedules, and Fatigue: A White Paper*, Seattle: Washington State Nurses Association, pp.1-24.

Ernst A T, Jiang H, Krishnamoorthy M and Sier D (2004). Staff scheduling and rostering: A review of applications, methods and models, *European Journal of Operational Research* 153(1): pp 3–27.

Fereshteh G, Mahin N and Mahnaz G (2011). Comparison of general health status and sleep quality between nurses with fixed working shifts and nurses with rotating working shifts. *Zahedan Journal of Research in Medical Sciences*, **13**: 48-50.

Ferri P, Guadi M, Marcheselli L, Balduzzi S, Magnani D, Di Lorenzo R (2016). The impact of shift work on the psychological and physical health of nurses in a general hospital: a comparison between rotating night shifts and day shifts. *Risk Management and Healthcare Policy* **9**: 203–211.

Flo E, Pallesen S, Moen B and Waage, B (2014). Short rest periods between work shifts predict sleep and health problems in nurses at 1-year follow-up, *Occupational and Environmental Medicine*, **71**:555-561.

Folkard S, Lombardi D A and Tucker P T (2005). Shiftwork: safety, sleepiness and sleep. *Industrial Health*, **43**: 20-3.

Fransen M, Wilsmore B, Winstanley J, Woodward J M, Grunstein R, Ameratunga S and Norton R (2006). Shift Work and Work Injury in the New Zealand Blood Donors' Health Study. *Occupational and Environmental Medicine*, **63**(5): 352-358.

Gander P, Purnell P, Garden A and Woodward A (2007). Work Patterns and Fatigue-Related Risk among Junior Doctors. *Occupational and Environmental Medicine*, **64**(11): 733-738.

Gärtner J, Bohle P, Arlinghaus A, Schafhauser W, Krennwallner T, Widl M (2018). Scheduling Matters-Some Potential Requirements for Future Rostering Competitions from a Practitioner's View, in the Proceedings of the12th International Conference on the Practice and Theory of Automated Timetabling (PATAT-2018),Vienna, Austria, August28-31,2018, pp. 33-42.

Gärtner J, Rosa R, Roach G, Kubo T, Takahashi M (2019). Working Time Society consensus statements: Regulatory approaches to reduce risks associated with shift work-a global comparison. *Industrial Health*, **57**: 245-263.

Harrington J (2001). Health effects of shift work and extended hours of work, *Occupational and Environmental Medicine* **58**(1): 68-72.

HSE (Health and Safety Executive) (2006). *Managing shiftwork*, *Health and safety guidance*, Service code HS6256, The Office of Public Sector Information, Information Policy Team, ISBN:9780717661978, (http://www.hse.gov.uk/pubns/books/hsg256.htm) accessed on February 20, 2020.

Ingre M, Åkerstedt T, Ekstedt M, and Kecklund G (2012). Periodic self-rostering in shift work: correspondence between objective work hours, work hour preferences (personal fit), and work schedule satisfaction, *Scandinavian Journal of Work, Environment & Health*. 38(4): 327-336.

Jeffrey K, Mahony S, Michaelson J and Abdallah S (2014). Well-being at work: A review of the literature. *New Economics Foundation*, ISBN 978-1-908506-57-3, pp. 1-56.

Klein Hesselink J, de Leede J and Goudswaard A (2010). Effects of the New Fast Forward Rotating Five-Shift Roster at a Dutch Steel Company. *Ergonomics* **53**(6): 727-738.

Khaleque A (1999). Sleep Deficiency and Quality of Life of Shift Workers. *Social Indicators Research* **46**(2): 181-189.

Kubo T, Oyama I, Nakamura T, Shirane K, Otsuka H, Kunimoto M, Kadowaki K, Maruyama T, Otomo H, Fujino Y, Matsumoto T, Matsuda S (2011). Retrospective cohort study of the risk of obesity among shift workers: findings from the Industry-based Shift Workers' Health study, .Japan *Occupational and Environmental Medicine*, 68(5): 327-331.

Lowden A, Moreno C, Holmbäck U, Lennernäs M, Tucker P (2010). Eating and shift work — effects on habits, metabolism, and performance. *Scandinavian Journal of Work, Environment & Health*, **36**(2):150-162

Martens M FJ, Nijhuis F J N, van Boxtel M P J and Knottnerus J A (1999). Flexible Work Schedules and Mental and Physical Health. A Study of a Working Population with Non-Traditional Working Hours. *Journal of Organizational Behavior* **20**(1): 35-46

Muecke S (2005). Effects of rotating night shifts: literature review. *Journal of Advanced Nursing*, **50**(4): 433–439.

Northrup H R (1989). The Twelve-Hour Shift in the Petroleum and Chemical Industries Revisited: An Assessment by Human Resource Management Executives. *Industrial and Labor Relations Review* **42**(4): pp. 640-648.

Parkin J, Petrovic S (2015). Staff well-being in rostering, *Abstracts, 27th European Conference on Operational Research*, Glasgow, UK, July 12-15, 2015, pp 66.

Petrovic S, Parkin J, Wrigley D (2018). Shift work scheduling of a 24/7 service operation considering staff well-being *Abstracts, 29h European Conference on Operational Research*, Valencia, Spain, July 8-11, 2018.

Petrovic S, Parkin J, Curtois T, Wrigley D (2017). Nurse rostering with well-being measures, in the Technical Program, *Conference of the International Federation of Operational Research Societies, IFORS 2017*, Quebec City, Canada, July 17-21, 2017, pp. 62.

Rosa R R, Härmä M, Pulli K, Mulder M and Näsman O (1996). Rescheduling a Three Shift System at a Steel Rolling Mill: Effects of a One Hour Delay of Shift Starting Times on Sleep and Alertness in Younger and Older Workers, *Occupational and Environmental Medicine* **53**(10): 677-685.

Schaumberg R L, and Flynn F J (2017). Clarifying the link between job satisfaction and absenteeism: The role of guilt proneness. *Journal of Applied Psychology*, 102*(*6), 982–992.

Schernhammer E S, Thompson C A (2011). Light at night and health: the perils of rotating shift work. *Occupational and Environmental Medicine* **68**(5): 310-311.

Smith L, Folkard S, Tucker P and MacDonald I (1998). Work Shift Duration: A Review Comparing Eight Hour and 12 Hour Shift Systems. *Occupational and Environmental Medicine* **55**(4): 217-229.

Spurgeon A, Harrington J M and Cooper C L (1997). Health and Safety Problems Associated with Long Working Hours: A Review of the Current Position. *Occupational and Environmental Medicine* **54**(6): 367-375.

Tucker P, Barton J, Folkard S (1996). Comparison of Eight and 12 Hour Shifts: Impacts on Health, Wellbeing, and Alertness during the Shift. *Occupational and Environmental Medicine* **53**(11): 767-772.

Tucker P, Smith L, Macdonald I and Folkard S (1999). Distribution of Rest Days in 12 Hour Shift Systems: Impacts on Health, Wellbeing, and on Shift Alertness. *Occupational and Environmental Medicine* 56(3): 206-214.

Warr, P., & Nielsen, K. (2018). Wellbeing and work performance. In E. Diener, S. Oishi, & L. Tay (Eds.), Handbook of well-being. Salt Lake City, UT: DEF Publishers.

Wong K, Chan A H S, Ngan S C (2019) The Effect of Long Working Hours and Overtime on Occupational Health: A Meta-Analysis of Evidence from 1998 to 2018, *International Journal of Environmental Research and Public Health*, 16(12): 2102.

Young V, Bhaumik C (2011). *Health and well-being at work: a survey of employees*. Department for Work and Pensions, Research Report No **751**, ISBN 978 1 84712 985 7.

Zedeck S, Jackson S E and Summers E (1983). Shift Work Schedules and Their Relationship to Health, Adaptation, Satisfaction, and Turnover Intention. *The Academy of Management Journal* **26**(2): 297-310.

Web references

EU Working Time Directive (2003), https://ec.europa.eu/social/main.jsp?catId=706&langId= en& intPageId=205 accessed on 28 June, 2020.

Hierarchical constraints and their applications in staff scheduling problems

Chao Li · Pieter Smet · Patrick De Causmaecker

Abstract In this work, the combinatorial structures which imply polynomial-time solvability in staff scheduling problems are investigated. We introduce *hierarchical constraints* to emphasize the hierarchical relation among constraints and contribute a characterization for a large class of tractable optimization problems with totally unimodular matrices in their integer linear programs. As a result, polynomial-time solvable personnel scheduling problems in literature can be further extended and generalized, as hierarchical management requirements are often considered in practice. Furthermore, an approach to derive the minimum cost network flow problems from the proposed *hierarchical constraints* is established. The newly obtained insight into the generalized boundary between tractable and intractable staff scheduling constraints enriches the theoretical studies of staff scheduling problems' complexity and may lead to efficient models and methods for complex variations.

Keywords Integer programming · Combinatorial optimisation · Staff scheduling · Polynomial-time models · Hierarchical constraints · Total unimodularity

AI in Flanders, KU Leuven Research Fund RKU-D2932-C24/17/012
Grants from the China Scholarship Council (No. 201703170230)

C. Li
KU Leuven, Department of Computer Science, CODeS, KULAK
E. Sabbelaan 53, 8500 Kortrijk, Belgium
E-mail: chao.li@kuleuven.be

P. Smet
KU Leuven, Department of Computer Science, CODeS
Gebroeders De Smetstraat 1, 9000 Gent, Belgium
E-mail: pieter.smet@kuleuven.be

P. De Causmaecker
KU Leuven, Department of Computer Science, CODeS, KULAK
E. Sabbelaan 53, 8500 Kortrijk, Belgium
E-mail: patrick.decausmaecker@kuleuven.be

1 Introduction

Staff scheduling or personnel scheduling is a common operational management challenge with significant impact on operating costs, employee satisfaction, etc. As practical personnel scheduling problems constantly arise from real world applications, various methods including exact algorithms, heuristics and metaheuristics have been extensively investigated [4, 12, 2]. However, theoretical studies of their models are relatively limited, as their importance has been underestimated [11].

Many staff scheduling problems are NP-hard due to the presence of specific constraints [8]. Recently, polynomial time solvable models in staff scheduling have attracted growing attention, since they incorporate representative and fundamental constraints involved in many industrial variants and they are straightforward to analyze. The first systematic study on staff scheduling models was presented by Brucker, et al [3], where several polynomial-time solvable rostering problems were recognized using minimum cost network flow models. The class of tractable rostering problems was extended by applying new techniques of network flow reformulation [1, 10, 11, 7]. However, these studies lack generality, as they are based on network flow models to be customized for different problems.

To obtain general insights into characteristics of tractable staff scheduling problems, we consider the polyhedra of associated linear programs and find special hierarchical relations between constraints to develop a sufficient condition for identifying tractable scheduling problems. Models in literature [10, 11] are further extended to incorporate more realistic hierarchical requirements while preserving the polynomial-time solvability. Furthermore, a general method to derive minimum cost network flow problems from a collection of the proposed *hierarchical constraints* is presented to link this work with previous studies of staff scheduling based on network flow models.

This paper is organized as follows. Section 2 introduces basic terminology and properties of integer linear programs. Section 3 presents the hierarchical structures in constraints which make its integer program tractable and a consequent algorithm to identify the polynomial-time solvability of integer linear programs. Examples and applications of hierarchical constraints in staff scheduling problems are explored in Section 4. Finally, Section 5 includes conclusions and future work.

2 Preliminaries

The integer linear program (ILP) is one of the most frequently used models in staff scheduling. We consider an ILP P_0 in a generic form:

$$P_0 : min\{\mathbf{c}^{\mathrm{T}}\mathbf{x} : A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq 0, \mathbf{x} \in \mathbb{Z}^n\} \tag{1}$$

of which $\mathbf{c}, \mathbf{b}$ are vectors and A is a matrix with integer entries.

$H = \{x|A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq 0\}$ defines the polyhedron of the linear programming relaxation (LPR) of P_0. In general, optimal solutions of the LPR of P_0 can be fractional and infeasible, but total unimodularity of A is an important property which guarantees the integrality, as shown in Theorem 1.

Definition 1 *A matrix $\mathbf{A}_{m\times n}$ is totally unimodular (TU) if each square submatrix of A has determinant in $\{0, 1, -1\}$ [9].*

Theorem 1 *A matrix $\mathbf{A}$ is totally unimodular if and only if the polyhedron $\{x|A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq 0\}$ is integral for any integral vector $\mathbf{b}$ in P_0 [5].*

According to Theorem 1, if $\mathbf{A}$ is TU and constants $\mathbf{b}$ are all integers, P_0 is is solvable in polynomial time, since its optimal solutions are the same as its LPR's solutions.

3 Hierarchical constraints and their systems

In this section, we introduce hierarchical constraints which are key in establishing the tractability of the aforementioned ILP formulation P_0.

As shown in Section 2, constraints coefficients determine the complexity of an ILP. We consider the coefficients in each constraint as a vector and apply the Hadamard (entry-wise) product of constraint vectors to define the hierarchical relationship between constraints.

Definition 2 *The Hadamard product of two vectors $\mathbf{a} = (a_1, ..., a_n)$ and $\mathbf{b} = (b_1, ..., b_n)$ is denoted as $\mathbf{a} \odot \mathbf{b} = (a_1b_1, ..., a_nb_n)$ [6].*

Definition 3 *Two constraints C_1, C_2 are hierarchical, if their coefficient vectors $\mathbf{c}_1, \mathbf{c}_1$ has a Hadamard product such that $\mathbf{c_1} \odot \mathbf{c_2} \in \{\mathbf{0}, \pm\mathbf{c_1}, \pm\mathbf{c_2}\}$. Additionally, every entry of $\mathbf{c_2}, \mathbf{c_2}$ must be in $\{0, 1, -1\}$ and constants in their constraints must be integers.*

The following is an explanation of the hierarchy of two constraints C_1, C_2, based on the vectorized computation of their constraint coefficient vectors $\mathbf{c_1}, \mathbf{c_1}$:

1) $\mathbf{c_1} \odot \mathbf{c_2} = \mathbf{0} \iff$ the two constraints are disjoint, i.e., they contain no common variable (with nonzero coefficients). For example, $C_1 : 1x_1 + 1x_2 + 0x_3 + 0x_4 \leq b_1$ and $C_2 : 0x_1 + 0x_2 + 1x_3 + 1x_4 \leq b_2$.

2) $\mathbf{c_1} \odot \mathbf{c_2} = \mathbf{c_1}$ or $-\mathbf{c_1} \iff$ all the variables in C_1 are included in C_2. For instance, $C_1 : 1x_1 + 1x_2 + 0x_3 + 0x_4 \leq b_1$ and $C_2 : -1x_1 - 1x_2 - 1x_3 - 1x_4 \leq b_2$;

3) $\mathbf{c_1} \odot \mathbf{c_2} = \mathbf{c_2}$ or $-\mathbf{c_2} \iff C_1$ has all the variables in C_2. In other words, C_1 contains C_2;

4) $\mathbf{c_1} \odot \mathbf{c_2} \notin \{0, \pm\mathbf{c_1}, \pm\mathbf{c_2}\} \iff C_1$ and C_2 are not hierarchical.

The relationship level of two hierarchical constraints is defined as follows.

Definition 4 *The hierarchical level of a constraint C_1 is lower than that of a constraint C_2 denoted as $C_1 \leq_H C_2$, if their coefficient vectors have a Hadamard product $\mathbf{c_1} \odot \mathbf{c_2} \in \{\pm\mathbf{c_1}\}$ and $\mathbf{c_1} \neq \pm\mathbf{c_2}$. Furthermore, $C_1 <_H C_2$ donates the case that there is no other constraint C_3 with a hierarchical level between that of C_1 and C_2, i.e., $C_1 \leq_H C_3 \leq_H C_2$.*

Since there are usually more than two constraints in an ILP of P_0, a special collection of constraints is defined in Definition 4.

Definition 5 *A hierarchical constraint system is a collection of constraints in which every pair of constraints is hierarchical.*

For example, constraints $x_1 + x_2 + x_3 \leq b_1$, $-x_2 - x_3 \leq b_2$ and $-x_1 \leq b_3$ are a system of hierarchical constraints.

Theorem 2 *If the constraints of an ILP consist of no more than two hierarchical constraint systems, the problem can can be solved in polynomial time.*

Proof According to Theorem 1, Definition 3 and Definition 5, we only need to prove total unimodularity in the constraint coefficient matrix A of an ILP with at most two hierarchical constraint systems $\mathcal{H}_1$ and $\mathcal{H}_2$.

Mathematical induction is applied to show that an arbitrary square submatrix B of A, has determinant $det(B) \in \{0, \pm 1\}$. If only one element is in B, $det(B) \in \{0, \pm 1\}$ by definition. Assume $\det(B_{k\times k}) \in \{0, \pm 1\}$ for any square submatrix with k dimensions ($k > 1$), the case ($det(B_{(k+1)\times(k+1)}) \in \{0, \pm 1\}$) of any k+1 dimensions square submatrix remains to be verified.

It is noted that $det(B_{(k+1)\times(k+1)})$ will at most change its sign and preserve magnitude by applying the following elementary row operations. For a row in $B_{(k+1)\times(k+1)}$ from a constraint with level n in $\mathcal{H}_1$ or $\mathcal{H}_2$, we replace it by the entrywise sum or difference with other rows from constraints in level $n-1$ of the same hierarchical constraint system. Repeating this procedure until we get a new matrix $B'_{(k+1)\times(k+1)}$ of which at most two nonzero entries are in each column, because there are at most two hierarchical constraint systems. Finally, the proof is concluded as follows.

1) If $B'_{(k+1)\times(k+1)}$ has a column with only zero entries, $det(B'_{(k+1)\times(k+1)}) = det(B_{(k+1)\times(k+1)}) = 0$;

2) If $B'_{(k+1)\times(k+1)}$ contains a column of a single nonzero entry (1 or -1), $det(B'_{(k+1)\times(k+1)}) = \pm det(B_{(k+1)\times(k+1)}) \in \{0, \pm 1\}$, using determinant expansion by minors;

3) If all columns in $B'_{(k+1)\times(k+1)}$ have two nonzero entries, the sum of the rows from $\mathcal{H}_1$ must equal to that from $\mathcal{H}_2$ by Definition 3 and Definition 5. Consequently, $det(B'_{(k+1)\times(k+1)}) = det(B_{(k+1)\times(k+1)}) = 0$.

Hierarchical constraint systems can be used for fast identification of a wide range of polynomial-time solvable problems with no more than two hierarchical constraint systems according to Theorem 2. The number of hierarchical constraint systems is counted by using the Hadamard products of pairwise constraints' coefficient vectors in Definition 3. If two constraints are not hierarchical, they must be in different hierarchical constraint systems.

4 Applications of hierarchical constraints in staff scheduling

4.1 Tractable personnel scheduling problems

A general personnel rostering problem is to assign employees $i \in E = \{1, 2, ..., |E|\}$ to shifts $k \in S = \{1, 2, ..., |S|\}$ on days $j \in T = \{1, 2, ..., |T|\}$, considering operational constraints and objectives (e.g., costs or employee satisfaction) [11]. The assignment of employee i to shift k on day j is denoted as $x_{ijk} = 1$, otherwise $x_{ijk} = 0$.

Problem P_1 [10] is an example of staff rostering with hierarchical constraints and is presented here to demonstrate an application of Theorem 2 to other real problems. A cost c_{ijk} is incurred by the assignment $x_{ijk} = 1$. The objective is to minimize the total cost of the final schedule. The total number of shifts (or days) that employee i should work is a_i. The minimum and maximum number of employees required for shift k on day j are d^l_{jk} and d^u_{jk} respectively. The ILP of P_1 is formulated as follows.

$$P_1: \quad Min \sum_{i\in E}\sum_{j\in J}\sum_{k\in S} c_{ijk}x_{ijk} \tag{2}$$

$$s.t. \quad \sum_{k\in S} x_{ijk} \leq 1, \ \forall i \in E, j \in T \tag{3}$$

$$\sum_{j\in T}\sum_{k\in S} x_{ijk} = a_i, \ \forall i \in E \tag{4}$$

$$\sum_{i\in E} x_{ijk} \leq d^u_{jk}, \ \forall j \in T, k \in S \tag{5}$$

$$\sum_{i\in E} -x_{ijk} \leq -d^l_{jk}, \ \forall j \in T, k \in S \tag{6}$$

$$x_{ijk} \in \{0, 1\}, \ \forall i \in E, j \in T, k \in S \tag{7}$$

Inequalities (3) are *single assignment* constraints, which restrict that an employee work at most one shift per day. The *total assignment constraints* (4) ensure that the total assignments of employee i equals a_i. The *coverage constraints* (5) and (6) define the range of requirements of employees for shift k on day j. Integrality of decision variables is constrained by (7). As the decision variables and their coefficients in constraints of P_1 are integers, all the constants $(a_i, d^l_{jk}, d^u_{jk})$ in the form in constraints must be integers, otherwise we just round them into integers.

It is trivial to convert P_1 into the generic problem P_0 by replacing the *total assignment constraints* (4) with two inequality constraints. Fig. 1 shows the constraint coefficient matrix of P_1 with $|E| = |T| = |S| = 2$. There are two hierarchical constraint systems partitioned by the line in Fig. 1, of which the correctness can be easily verified by Definition 3. It is noted that the integer constraints (7) are hierarchical with all constraints in P_1, since their linear relaxations have an identity matrix of coefficients. Therefore, P_1 not only has

a polynomial-time solvable network flow model as shown in [10], the ILP of P_1 is also tractable by Theorem 2.

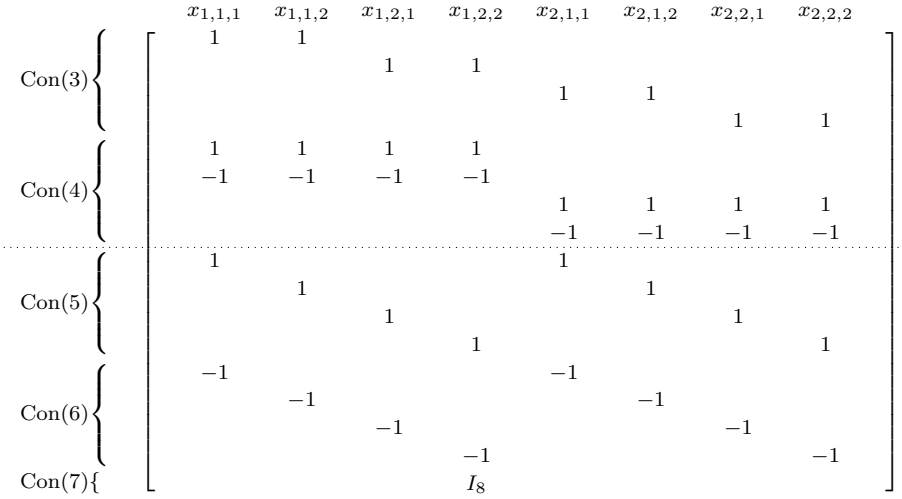


Fig. 1: The partition of constraint matrix of A_1

An important tractable extension to P_1 is adding constraints (8), where $\bar{D}_i$ is a set of pairwise disjoint subsets of the day set T for each employee i, i.e., $F_1 \cap F_2 = \emptyset, F_1 \subseteq T, F_2 \subseteq T, \forall F_1, F_2 \in \bar{D}_i$. This new type of constraint can limit the number of working shifts (or days) of an employee i in the range $[m^l_{iF}, m^u_{iF}]$ within disjoint periods such as weekends [11].

$$m^l_{iF} \leq \sum_{j \in F} \sum_{k \in S} x_{ijk} \leq m^u_{iF}, \ \forall i \in E, F \in \bar{D}_i \tag{8}$$

Constraints (8), (3) and (4) are in the same hierarchical constraint system according to Definition 5. Here we present a more general set of constraints (9) replacing constraints (8) to preserve polynomial-time solvability. In constraints (9), G_i is a set of subsets of the day set T for employee i, and $F_1 \cap F_2 \in \{F_1, F_2, \emptyset\}, F_1 \subseteq T, F_2 \subseteq T, \forall F_1, F_2 \in G_i$. This extension enables the inclusion of assignment restrictions of employees within hierarchical periods in P_1. For example, the constraints that restrict the range of assignments (workload) of an employee in hierarchical periods such as weekends, weeks and months.

$$m^l_{iF} \leq \sum_{j \in F} \sum_{k \in S} x_{ijk} \leq m^u_{iF}, \ \forall i \in E, F \in G_i \tag{9}$$

Furthermore, constraints with a hierarchy among employees can also be included in P_1. A common situation is that there are coverage constraints in terms of particular groups of employees (such as skilled workers, interns and

general workers) required for a shift k on a day j, as formulated by constraints (10). Q_k are a collection of subsets of the employee set E such that $L_1 \cap L_2 \in \{L_1, L_2, \emptyset\}, L_1 \subseteq E, L_2 \subseteq E, \forall L_1, L_2 \in Q_k$. In other words, all sets in Q_k are hierarchical instead of pairwise disjoint.

$$m^l_{kL} \leq \sum_{i \in L} \sum_{j \in T} x_{ijk} \leq m^u_{kL}, \ \forall k \in S, L \in Q_k \tag{10}$$

Similarly, there are constraints with a hierarchy in terms of shifts, represented by constraints (11). This kind of constraints models the restriction on the number of shifts in several categories ($Y \in W_i$) worked by an employee i, where $Y_1 \cap Y_2 \in \{Y_1, Y_2, \emptyset\}$, $Y_1 \subseteq T, Y_2 \subseteq S$, $\forall Y_1, Y_2 \in W_i$. For example, contractual constraints restrict the maximum workload in terms of morning shifts, daytime shifts and evening shifts defined in W_i. In this case, the daytime shifts include morning shifts (their intersection equals to the morning shifts). The set of evening shifts are disjoint with the morning shift set and the daytime shift set.

$$m^u_{iY} \leq \sum_{i \in E} \sum_{k \in Y} x_{ijk} \leq m^u_{iY}, \ \forall i \in E, Y \in W_i \tag{11}$$

However, at most two kinds of constraints from (9), (10) and (11) can be included in a polynomial-time solvable model, as there are at most two hierarchical constraints systems according to Theorem 2.

It is also noted that the inclusion of soft constraints [7] to P_1 is tractable, if no additional hierarchical constraints systems are introduced.

4.2 Derivation of network flow problems

Network flow models can be more efficient than the ILP formulations for a polynomial-time solvable rostering problem [10]. The challenge is that the network layout varies from problem to problem. Hence, this section presents a general method to derive a minimum cost flow problem from an ILP with at most two hierarchical constraint systems ($\mathcal{H}_1$, $\mathcal{H}_2$), as an application of the proposed hierarchical constraints. The main procedures are described below. Without loss of generality, we assume that $\mathcal{H}_1$ and $\mathcal{H}_2$ are not empty.

1. Node generation: add a *variable node* for each decision variable; generate *constraint nodes* corresponding to constraints in $\mathcal{H}_1$ and $\mathcal{H}_2$

2. Arc generation I: add an arc from a *variable node* to *constraint nodes* associated with the lowest-level constraints in $\mathcal{H}_1$, if the corresponding variable has nonzero coefficient (± 1) in that constraint; connect *variable nodes* with the *constraint nodes* from $\mathcal{H}_2$ with the same manner but in the opposite direction.

2. Arc generation II: add an arc (u, v) between *constraint nodes* u and v which correspond to constraints C_1 and C_2 with adjacent hierarchical levels

in $\mathcal{H}_1$ respectively such that $C_1 <_H C_2$; develop arcs in the reverse direction among *constraint nodes* from $\mathcal{H}_2$ using the same rule.

3. Parameter configuration: set the supply and demand of a source node s and a sink node t and connect them with the *constraint nodes* associated with top-level constraints in $\mathcal{H}_1$ and $\mathcal{H}_2$ respectively; set capacity of arcs according to the constants of their relevant constraints in $\mathcal{H}_1$ and $\mathcal{H}_2$.

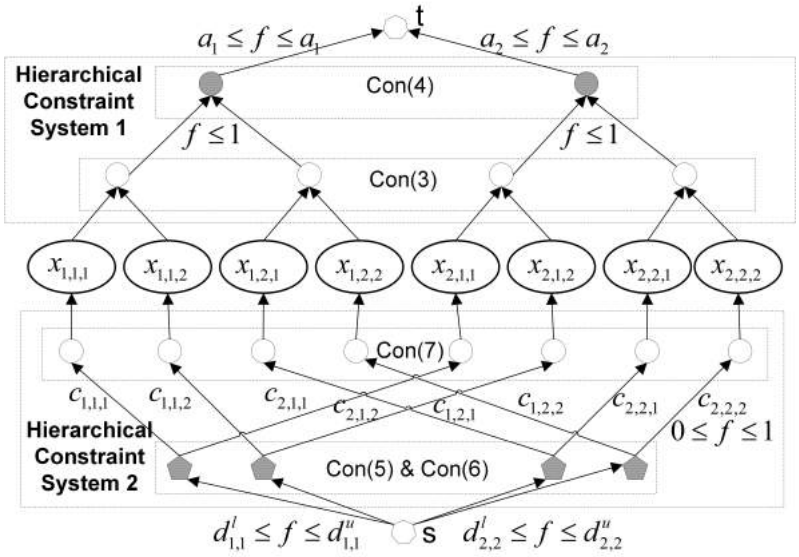


Fig. 2: The derived network flow problem from P_1

Generating the network is generic for any ILP with one or two hierarchical constraint systems, but the parameter configuration is problem dependent. Fig. 2 illustrates the network created by applying the presented approach on an instance of P_1. The supply of the source s and the demand of the sink t are equal to $\sum_{i \in E} a_i$. The lower and upper bound of flows from s to *constraint nodes* associating the *coverage constraints* (5) and (6) in $\mathcal{H}_2$ are d^l_{jk} and d^u_{jk} respectively. Then a flow from these *constraint nodes* to their upper *constraint nodes* corresponding to constraints (7) is limited in the range of [0,1]. There is a unit cost c_{ijk} of the flow to the *variable node* associating the decision variable x_{ijk} from the *constraint nodes* associated with constraints (7). As a result, costs of assignments can be calculated as the costs of flows. Furthermore, the flow from the bottom hierarchical level nodes associated with constraints (3) in $\mathcal{H}_1$ to their upper nodes has a capacity of one unit, according to the *single constraint* (3). Finally, the flow from the top-level *constraint nodes* representing *total assignment constraints* (4) in $\mathcal{H}_1$ to t is bounded by a_i, i.e., the required number of total assignments (working days) of employee i. Therefore, the derived minimum cost network flow problem is an equivalent of P_1.

5 Conclusions

This paper presents theoretical results concerning the constraints' effects on the complexity of optimization problems in a general setting of integer linear programs. Hierarchical constraints are identified and formulated to characterize the ILPs with polynomial-time solvability for a large class of personnel scheduling problems, without problem-dependent reformulations into tractable network flow problems [3, 10, 11, 7]. Consequent applications are introduced, including vectorized testing of polynomial-time solvability, several extensions of well-known tractable problems and the derivation of network flow problems from hierarchical constraints systems. These examples and applications validate that the hierarchy in constraints is one of the structural reasons why some staff scheduling problems/models are easy and can be remodeled as tractable minimum cost network flow problems in previous work [3, 10, 11, 7].

In the future, our focus will shift to the integration of tractable models into solution methods for complex personnel scheduling problems.

References

1. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows: Theory, Algorithms and Applications. Prentice Hall, Upper Saddle River, New Jersey, USA (1993)
2. Van den Bergh, J., Beliën, J., De Bruecker, P., Demeulemeester, E., De Boeck, L.: Personnel scheduling: A literature review. European Journal of Operational Research **226**(3), 367–385 (2013)
3. Brucker, P., Qu, R., Burke, E.: Personnel scheduling: Models and complexity. European Journal of Operational Research **210**(3), 467–473 (2011)
4. Ernst, A.T., Jiang, H., Krishnamoorthy, M., Sier, D.: Staff scheduling and rostering: A review of applications, methods and models. European journal of operational research **153**(1), 3–27 (2004)
5. Hoffman, A.J., Kruskal, J.B.: Integral boundary points of convex polyhedra. In: 50 Years of Integer Programming 1958-2008, pp. 49–76. Springer (2010)
6. Horn, R.A., Johnson, C.R.: Matrix analysis. Cambridge university press (2012)
7. Li, C., Smet, P., De Causmaecker, P.: Polynomial-time personnel scheduling with soft constraints. In: Proceedings of the 12th international conference on the practice and theory of automated timetabling, pp. 501–505. PATAT (2018)
8. Osogami, T., Imai, H.: Classification of various neighborhood operations for the nurse scheduling problem. In: International Symposium on Algorithms and Computation, pp. 72–83. Springer (2000)
9. Schrijver, A.: Combinatorial optimization: polyhedra and efficiency, vol. 24. Springer Science & Business Media (2003)

10. Smet, P., Brucker, P., De Causmaecker, P., Vanden Berghe, G.: Polynomially solvable formulations for a class of nurse rostering problems. In: Proceedings of the 10th international conference on the practice and theory of automated timetabling, pp. 408–419 (2014)
11. Smet, P., Brucker, P., De Causmaecker, P., Vanden Berghe, G.: Polynomially solvable personnel rostering problems. European Journal of Operational Research **249**(1), 67–75 (2016)
12. Vermuyten, H., Rosa, J.N., Marques, I., Belien, J., Barbosa-Póvoa, A.: Integrated staff scheduling at a medical emergency service: An optimisation approach. Expert Systems with Applications **112**, 62–76 (2018)

Scheduling Bus Drivers in Real-Life Multi-Objective Scenarios with Break Constraints

Extended Abstract

Lucas Kletzander · Nysret Musliu

Received: date / Accepted: date

1 Introduction

When there is varying demand for employees at different times of the day, it is important to have efficient schedules for the employees in order to cover the demand with minimal cost. On the other hand, there is a range of legal requirements, collective agreements and company policies that need to be taken into account to create feasible schedules. Further, not every schedule that is feasible will be readily accepted by the employees, purely optimizing cost might result in reduced employee satisfaction and potential conflicts with labour unions.

An area that is especially restricted by various constraints is scheduling for drivers in public transport. As these employees have a great responsibility keeping their passengers safe, legal requirements enforce strict break assignments in order to maintain concentration. In addition to that a spatial component needs to be considered. This makes the goal to create cost-efficient and employee-friendly schedules even more challenging. This paper deals with optimizing schedules for bus drivers in Austria, using the regulations from

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.

L. Kletzander
Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling
TU Wien, Vienna, Austria
E-mail: lkletzan@dbai.tuwien.ac.at

N. Musliu
Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling
TU Wien, Vienna, Austria
E-mail: musliu@dbai.tuwien.ac.at

the Austrian collective agreement for employees in private omnibus providers serving regional lines.

The contributions of this work are as follows. We extend previous work [12] on the problem with a focus on more complex objectives including various new criteria that are relevant in practice. As in [12], we also apply a Simulated Annealing approach, but we additionally propose new moves that take into account the characteristics of the extended problem. Based on this, we can provide high quality solutions for real-life scenarios.

2 Related Work

Due to its high practical relevance, the topic of employee scheduling has seen tremendous research for many years. Several surveys [8,3] provide a good overview of work in different areas. A survey for the different objectives in operating bus transport systems is provided by [11]. Driver scheduling is located between vehicle scheduling and driver rostering in a six step process. Driver scheduling belongs to the area of crew scheduling problems [8] that is also frequently applied to airline [9] and train crew scheduling.

Research on Bus Driver Scheduling (BDS) Problems has started decades ago [24]. Previous work explored different solution methods. Exact methods mostly use column generation with a set covering or set partitioning master problem and a resource constrained shortest path subproblem [19,7,17,14]. Heuristic methods like greedy [16,6,20] or exhaustive [4] search, tabu search [15,18], genetic algorithms [15,13] or assignment problems [5] are used in different variations. The scheduling of breaks within shifts is considered by several authors [1,2,22].

[12] presents a complex version of the BDS problem based on the Austrian collective agreement for employees in private omnibus providers [23], using the rules for regional lines (up to 50 km per line). New benchmark and real life instances are solved using Simulated Annealing.

3 Problem Description

The Bus Driver Scheduling Problem deals with the assignment of bus drivers to vehicles that already have a predetermined route for one day of operation. The shifts that are generated need to respect a range of constraints regarding length and complex break assignment rules. The specification presented here extends [12]. New extensions are presented in sections 3.3 and 4.

3.1 Problem Input

The bus routes are given as a set of individual bus legs $\mathbf{L}$, each leg $\ell \in \mathbf{L}$ is associated with a tour $tour_\ell$ (corresponding to a particular vehicle), a start time $start_\ell$, an end time end_ℓ, a starting position $startPos_\ell$ and an end position

Table 1 Example bus tour

ℓ	$tour_\ell$	$start_\ell$	end_ℓ	$startPos_\ell$	$endPos_\ell$
1	1	360	395	0	1
2	1	410	455	1	2
3	1	460	502	2	1
4	1	508	540	1	0

$endPos_\ell$. The amount of time within the leg that is actually spent actively driving is specified as $drive_\ell$. This problem uses $drive_\ell = length_\ell = end_\ell - start_\ell$.

Table 1 shows a short example of one particular bus tour. The vehicle starts at time 360 (6:00 as our time units are minutes) at position 0, which could be the bus depot. 35 minutes later it arrives at position 1. Before the next leg of the bus tour there is a 15 minutes waiting time which might qualify as a break for the employee depending on the constraints explained later. After four legs, the bus returns to the depot at time 540. Valid input never has overlapping bus legs for the same tour and consecutive bus legs i, j of the same tour always respect $endPos_i = startPos_j$.

Further input is a distance matrix, which, for each pair of positions i and j, denotes a time $d_{i,j}$ it takes a driver to get from i to j when not actively driving a bus. If no transfer is possible, we set $d_{i,j} = \infty$. $d_{i,j}$ with $i \neq j$ is called passive ride time. $d_{i,i}$ represents the time it takes to switch tour at the same position, but is not considered passive ride time. We define the occurrence of a tour change as when a driver has an assignment of two consecutive bus legs i and j with $tour_i \neq tour_j$.

Finally, for each position i an amount of working time for starting a shift at that position $startWork_i$ and for ending a shift $endWork_i$ are given. At any depot d preparing the bus ($startWork_d = 15$) and finishing the bus ($endWork_d = 10$) are considered, for other positions the value is 0.

3.2 Solution

A solution to the problem is an assignment of exactly one driver to each bus leg. A feasible solution must satisfy the following criteria:

- No overlapping bus legs are assigned to the same driver.
- Whenever tour or position changes for a driver between assigned bus legs i and j, then $start_j \geq end_i + d_{i,j}$.
- Each shift respects all hard constraints regarding work regulations as specified in the next section.

Within the set of feasible solutions, different criteria might be optimized as explained later.

3.3 Work and Break Regulations

Valid shifts for drivers are constrained by work regulations and require frequent breaks. There are many constraints related to different measures of the schedule.

- Driving time: The time actually spend driving the vehicle is constrained by a maximum value of 9 hours and the requirement for breaks after at most 4 hours of driving that might be split into smaller parts.
- Total time: The time between the start and end of the shift is limited to 14 hours.
- Working time: The working time does not include certain unpaid breaks or shift splits, there are complex rules which breaks are unpaid according to their length and location within the shift. The working time should be within 6.5 and 10 hours except for part time employees whose working time may last only three hours.

This work extends the problem by looking at different vehicle types as well as training of employees. First, this leads to the notion of the level of a duty, based on the different vehicles and the different lines that a duty contains. More different vehicles and lines require an employee to be trained for all of them, therefore the level of the duty is higher. Second, when optimizing duties for both bus and tram lines, some tram lines have different driving break requirements compared to the bus lines. Therefore, the driving break requirements become dependent on the current line of a duty.

4 Objectives

There are several optimization criteria, setting a different and often conflicting focus on the resulting schedules. These include both cost objectives and objectives to obtain schedules that are actually workable in practice considering the needs of the employees. The following minimization objectives are considered in our real-life application:

- Number of employees (cost objective)
- Sum of working times (cost objective)
- Sum of missing working time (shifts below 6.5 hours need to be paid 6.5 hours anyway, in combination with the previous objective this enforces shifts to be well distributed)
- Sum of long unpaid break time (time above a limit of 1.5 hours)
- Sum of passive ride times (drivers are riding as a passenger or walking to a different location)
- Number of major location changes (drivers change to a different location that is very far away, including a hard maximum of one such change per duty)
- Number of duties where the second part is longer than the first part (to achieve a favourable location of the main break)

- Number of duties with more than two stretches (a stretch is defined as a consecutive assignment of bus legs on the same tour, i.e., this objective minimizes vehicle changes, also including a hard maximum of three parts per duty)
- Sum of missing break safety time (driving breaks should be several minutes above the minimum length in order to have a buffer for minor operational delays, this objective sums missing buffer time)
- Sum of missing stretch time (a stretch should be at least 1.5 hours, this objective sums the difference in case a stretch is shorter)
- Sum of the squared duty levels (reduce especially high levels)

5 Solution Method and Results

The solution method is based on a construction heuristic and Simulated Annealing. The objectives are combined using a linear objective function. The weights are set based on the goals of the bus operator. Compared to previous schedules, the importance of the different goals are set (should be improved, should not get worse, might get worse in a certain range) and the weights are repeatedly tuned and carefully evaluated to match those goals.

The construction heuristic uses a greedy approach trying to assign consecutive bus legs of the same tour to the same duty. Simulated Annealing uses different moves that are applied to duties with high objective values with higher probability.

Different moves are used for the problem:

- Moving a bus leg to a different duty
- Swapping bus legs between different duties
- Swapping a range of bus legs between different duties
- Swapping stretches between different duties

Regarding the selection of the duties for the application of a move, with higher probability we select duties such that consecutive elements of the same tour are placed next to each other. As duties with many stretches are unwanted, this selection of moves combined with their application helps to reduce the number of tour changes in the solution.

The method has been deployed in practice just recently. We have applied it to a real-world scenario where solutions calculated with different weight distributions allow to compare different options. Compared to existing solutions the initial results can provide solutions that greatly improve important characteristics of the duties like the long break times while moderately raising less important characteristics in a controlled way. Table 2 shows a comparison of the results for the focus on improving long breaks and passive ride time. The total paid working time can be slightly improved, unpopular break over-length can be reduced by more than half, passive ride time by more than a third, and major location changes by two thirds, while increased short breaks and duty levels are still acceptable.

Table 2 Objective improvements

Objective	Goal	Previous	New
Employees	keep	133	134
Working time (inc. missing)	not worse	64904	64722
Long break time	better	2905	1261
Passive ride time	better	810	525
Major location changes	better	15	5
Second > first	not worse	49	46
3 stretches	better	14	13
Short break time	worse	29	47
Missing stretch time	not worse	95	98
Duty levels	worse	374	767

As future work we will provide more detailed experimental results. It would also be interesting to explore computing a Pareto front for the problem. However, due to the large number of objectives this will be difficult and will require methods from the area of many-objective optimization [21,10].

References

1. Beer, A., Gaertner, J., Musliu, N., Schafhauser, W., Slany, W.: Scheduling Breaks in Shift Plans for Call Centers. In: Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling, pp. 1–17 (2008)
2. Beer, A., Gärtner, J., Musliu, N., Schafhauser, W., Slany, W.: An AI-Based Break-Scheduling System for Supervisory Personnel. IEEE Intelligent Systems **25**(2), 60–73 (2010). DOI 10.1109/MIS.2010.40
3. Van den Bergh, J., Beliën, J., De Bruecker, P., Demeulemeester, E., De Boeck, L.: Personnel scheduling: A literature review. European Journal of Operational Research **226**(3), 367–385 (2013). DOI 10.1016/j.ejor.2012.11.029
4. Chen, S., Shen, Y., Su, X., Chen, H.: A Crew Scheduling with Chinese Meal Break Rules. Journal of Transportation Systems Engineering and Information Technology **13**(2), 90–95 (2013). DOI 10.1016/S1570-6672(13)60105-1
5. Constantino, A.A., de Mendonça Neto, C.F.X., de Araujo, S.A., Landa-Silva, D., Calvi, R., dos Santos, A.F.: Solving a large real-world bus driver scheduling problem with a multi-assignment based heuristic algorithm. Journal of Universal Computer Science **23**(5), 479–504 (2017)
6. De Leone, R., Festa, P., Marchitto, E.: A Bus Driver Scheduling Problem: a new mathematical model and a GRASP approximate solution. Journal of Heuristics **17**(4), 441–466 (2011). DOI 10.1007/s10732-010-9141-3
7. Desrochers, M., Soumis, F.: A Column Generation Approach to the Urban Transit Crew Scheduling Problem. Transportation Science **23**(1), 1–13 (1989). DOI 10.1287/trsc.23.1.1
8. Ernst, A., Jiang, H., Krishnamoorthy, M., Sier, D.: Staff scheduling and rostering: A review of applications, methods and models. European Journal of Operational Research **153**(1), 3–27 (2004). DOI 10.1016/S0377-2217(03)00095-X
9. Gopalakrishnan, B., Johnson, E.L.: Airline Crew Scheduling: State-of-the-Art. Annals of Operations Research **140**(1), 305–337 (2005). DOI 10.1007/s10479-005-3975-3
10. Hisao Ishibuchi, Noritaka Tsukamoto, Yusuke Nojima: Evolutionary many-objective optimization: A short review. In: 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence), pp. 2419–2426 (2008)
11. Ibarra-Rojas, O., Delgado, F., Giesen, R., Muñoz, J.: Planning, operation, and control of bus transport systems: A literature review. Transportation Research Part B: Methodological **77**, 38–75 (2015). DOI 10.1016/j.trb.2015.03.002

12. Kletzander, L., Musliu, N.: Solving large real-life bus driver scheduling problems with complex break constraints. In: Proceedings of the International Conference on Automated Planning and Scheduling, vol. 30, p. accepted for publication (2020)
13. Li, J., Kwan, R.S.: A fuzzy genetic algorithm for driver scheduling. European Journal of Operational Research **147**(2), 334–344 (2003). DOI 10.1016/S0377-2217(02)00564-7
14. Lin, D.Y., Hsu, C.L.: A column generation algorithm for the bus driver scheduling problem: Bus Driver Scheduling Problem. Journal of Advanced Transportation **50**(8), 1598–1615 (2016). DOI 10.1002/atr.1417
15. Lourenço, H.R., Paixão, J.P., Portugal, R.: Multiobjective Metaheuristics for the Bus Driver Scheduling Problem. Transportation Science **35**(3), 331–343 (2001). DOI 10.1287/trsc.35.3.331.10147
16. Martello, S., Toth, P.: A heuristic approach to the bus driver scheduling problem. European Journal of Operational Research **24**(1), 106–117 (1986). DOI 10.1016/0377-2217(86)90016-0
17. Portugal, R., Lourenço, H.R., Paixão, J.P.: Driver scheduling problem modelling. Public Transport **1**(2), 103–120 (2009). DOI 10.1007/s12469-008-0007-0
18. Shen, Y., Kwan, R.S.K.: Tabu Search for Driver Scheduling. In: G. Fandel, W. Trockel, C.D. Aliprantis, D. Kovenock, S. Voß, J.R. Daduna (eds.) Computer-Aided Scheduling of Public Transport, vol. 505, pp. 121–135. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)
19. Smith, B.M., Wren, A.: A bus crew scheduling system using a set covering formulation. Transportation Research Part A: General **22**(2), 97–108 (1988). DOI 10.1016/0191-2607(88)90022-2
20. Tóth, A., Krész, M.: An efficient solution approach for real-world driver scheduling problems in urban bus transportation. Central European Journal of Operations Research **21**(S1), 75–94 (2013). DOI 10.1007/s10100-012-0274-3
21. Wagner, T., Beume, N., Naujoks, B.: Pareto-, aggregation-, and indicator-based methods in many-objective optimization. In: International conference on evolutionary multi-criterion optimization, pp. 742–756. Springer (2007)
22. Widl, M., Musliu, N.: The break scheduling problem: complexity results and practical algorithms. Memetic Computing **6**(2), 97–112 (2014). DOI 10.1007/s12293-014-0131-0
23. WKO.at: Kollektivvertrag für Dienstnehmer in privaten Autobusbetrieben gültig ab 1.1.2019. https://www.wko.at/service/ kollektivvertrag/kv-private-autobusbetriebe-2019.html (2019). Accessed 8 Jul. 2019
24. Wren, A., Rousseau, J.M.: Bus Driver Scheduling — An Overview. In: G. Fandel, W. Trockel, J.R. Daduna, I. Branco, J.M.P. Paixão (eds.) Computer-Aided Transit Scheduling, vol. 430, pp. 173–187. Springer Berlin Heidelberg, Berlin, Heidelberg (1995)

Rehearsal Scheduling: Developing An Optimization Solution With Practitioner Input

Emily Hill · Mark Velednitsky

the date of receipt and acceptance should be inserted later

Abstract We consider the REHEARSAL SCHEDULING problem, typically stated in the context of theater, film, and performing arts. We are given a set of actors and scenes. Scenes may consist of several actors and actors may be in several scenes. We are also given a set of timeslots when scenes can be scheduled. The objective is to arrange the scenes into timeslots to produce a "good" schedule.

In theoretical work, "good" means minimizing the total *hold time*: the time between an actor's first scheduled scene and last scheduled scene during which they are not working. The theoretical setting is idealized to ignore actor conflicts, precedence between scenes, uneven scene lengths, optional actors, and other practical considerations. In practice, REHEARSAL SCHEDULING is as much an art as a science, as there is no concise, agreed-upon definition of what constitutes a good schedule.

In this work, we conduct a survey of practitioners who regularly solve the REHEARSAL SCHEDULING problem. Almost all of them currently solve the optimization problem by hand. We ask them to prioritize features of a good schedule. As a control, we also ask them to prioritize various utility features: features which do not affect the optimization, but affect how the user interacts with the system that generates the schedule. Based on the survey, we formulate the REHEARSAL SCHEDULING problem as a series of integer programs. We then build a tool for practitioners to schedule their shows.

We test the tool in 14 real-world use cases. We find that practitioners respond very positively to the results of the optimization solver. However, we learn that there are many other utility features which would be necessary for the optimization tool to gain widespread use.

E. Hill
University of California, Berkeley
E-mail: emilyhill@berkeley.edu

M. Velednitsky
University of California, Berkeley
E-mail: marvel@berkeley.edu

1 Introduction

In the REHEARSAL SCHEDULING problem, we are given a set of n actors and m scenes. Scenes require a certain set of actors. Actors may be required by several scenes. We are also given a set of timeslots into which scenes can be scheduled. The objective is to arrange the scenes into timeslots to produce a "good" schedule.

The TALENT SCHEDULING problem is the study of the REHEARSAL SCHEDULING problem in the idealized setting, with a single, clear objective function, namely to minimize *hold time*. We assume that the actor arrives for the first scene they are in and leaves after the last scene. Any time between their arrival and departure when they are not rehearsing is hold time. The TALENT SCHEDULING problem is sometimes called the HOLD COST MINIMIZATION problem.

The input to the TALENT SCHEDULING problem is typically presented as a binary matrix, where there is one row for each actor and one column for each scene. The output is an ordering of the columns of the matrix. For each row, its "hold time" is the number of zeros between the first and last one in the new column ordering. The objective is to minimize total hold time. See figure 1.

There is a dynamic programming algorithm for solving TALENT SCHEDULING optimally, though it has an exponential running time [1]. For solving the TALENT SCHEDULING problem on instances of realistic sizes, custom branch-and-bound algorithms have been effective. The key idea in most of these algorithms is to first specify the first scene, then specify the last scene, and gradually work "inwards" towards the middle scenes. The first such algorithm was proposed by Cheng, Diamond, and Lin [5] and later refined by Garcia de la Banda et al. [7], Qin et al. [10], and Cheng et al. [4]. Others have applied meta-heuristics to the problem, such as genetic algorithms [9].

Other authors have considered special cases of the *talent scheduling* problem. Determining if there is a solution with no hold time is equivalent to the CONSECUTIVE 1S problem, which can be solved in polynomial time [3]. The case where each actor appears in exactly two scenes is the LINEAR ARRANGEMENT PROBLEM, which is known to be NP-hard [5]. There exists a polylog-

Fig. 1: Example of an input to and output from the TALENT SCHEDULING problem, an idealized version of REHEARSAL SCHEDULING. Each row corresponds to an actor and each column corresponds to a scene.

$$\begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

(a) An example input to the TALENT SCHEDULING problem. In this ordering, the total hold time is 2.

(b) An example output from the TALENT SCHEDULING problem. The scenes are reordered to minimize hold time.

arithmic approximation algorithm for the LINEAR ARRANGEMENT PROBLEM [6].

The TALENT SCHEDULING problem is widely understood to be an idealized simplification of the more amorphous REHEARSAL SCHEDULING problem. TALENT SCHEDULING ignores actor conflicts, precedence between scenes, uneven scene lengths, optional actors, and other practical considerations. The custom branch-and-bound algorithms do not accommodate these complexities. As such, they are not adequate for practitioners. In practice, REHEARSAL SCHEDULING is as much an art as a science, as there is no concise, agreed-upon definition of what constitutes a "good" schedule.

Several authors have approached REHEARSAL SCHEDULING by extending the models and solution ideas used for TALENT SCHEDULING. Examples include Sakulsom et al., who consider the problem the problem of scheduling rehearsals of unequal length [11]. Another generalization comes from Wang et al., who recognized that daily and hourly scheduling should be considered separately: with separate costs for intra-day holding and inter-day holding [12].

Even after determining a suitable model that accounts for practitioners needs, there is considerable engineering work. A comprehensive system was developed by Bomsdorf and Derigs [2]. In their paper, they acknowledge "to be accepted in practice, any planning methodology has to be highly interactive, allowing fast and flexible rescheduling, and has to respect the planners problem solving style, allowing them to bring in their experience."

To develop a scheduling tool with the potential to be accepted for practical use, we took the insight of Bomsdorf et al. to heart. In our work, we take a user-first approach to the REHEARSAL SCHEDULING problem. Our primary contributions are:

- We conduct a survey of 44 practitioners of REHEARSAL SCHEDULING to determine what constitutes a "good" schedule. In addition to asking about the optimization features of a hypothetical scheduling system, we also ask about utility features: features which affect users' interactions with the tool, but not the optimization solution.
- Based on the survey results, we describe the REHEARSAL SCHEDULING problem as multi-objective optimization problem with three objectives. We treat the objective functions hierarchically and solve the problem as a series of three integer programs.
- We test our solution in 14 real-world use cases. We find that the optimization is adequate, but additional utility features would be required in order for the tool to have a possibility of wider adoption.

Our paper is organized as follows: In section 2, we describe our survey and its results. In section 3, we describe the integer programming formulation. In section 4, we describe our experimental set-up and results. In section 5, we conclude with a few final remarks.

Remark 1 (Terminology) There is some disagreement between the terminology used in the academic literature and the terminology used by practitioners.

In the academic literature, it is assumed that each scene is rehearsed once. In practice, a scene can be rehearsed several times. Switching between these contexts is not a problem, mathematically: if a scene needs to be rehearsed several times, then we can create several copies of it in the input.

For the sake of succinctly presenting our mathematical formulation, we will assume the convention that each scene is rehearsed once. However, in our survey and scheduling tool developed for practitioners, we will use the industry-standard terminology: a scene is defined once and potentially rehearsed several times.

2 Survey of Practitioners

Surveys are effective, low-cost, and non-intrusive tools to collect primary data about users' pain points, needs, and preferences surrounding an existing workflow. Human-centered design practitioners regularly implement insights derived from survey results into software development cycles [8].

We surveyed 44 active practitioners to understand their current scheduling workflow and factors that matter the most when they schedule. We collected the type of productions the practitioners do, the sizes of their respective production, and their current methods of creating schedules. The responses were collected through Google Form.

2.1 Survey Design

To formulate the REHEARSAL SCHEDULING problem in a way that appeals to practitioners, we needed to understand features of good schedules and their relative importance. The first section of our survey was dedicated to understanding the considerations practitioners take into account when they create a schedule. These features are described in section 2.1.1. As a control, we included a second section, in which we asked practitioners to rank certain usability features unrelated to the scheduling optimization. These control questions served as a useful insight into the relative importance of optimization features. They are described in section 2.1.2.

2.1.1 Optimization Features

Our survey asked participants how often they considered each of the following features when they designed a "good" schedule. Participants rated the frequency on a Likert Scale. 1 being "never considered" the feature, and 5 being "always considered" the feature.

Hard Conflicts Times when an actor cannot make it to rehearsal. It may be possible to rehearse a scene without a certain actor present, but, if possible, it is best to rehearse the scene when all the actors are present.

Soft Conflicts Times when an actor would prefer not to make it to rehearsal. It is nice to honor these if possible to keep the actors happy, but the actor will attend if called.

Space Constraints The availability of certain rooms in which the rehearsal can be held. For example, it may be desirable to only rehearse a certain scene when a music room is available or when a large room is available.

Time Efficiency Using the actors' time as efficiently as possible.[1]

Spacing In some cases, it may be important to space certain rehearsals *apart* from each other. For example, in one rehearsal the actors might learn choreography and then several days later they review it in another rehearsal. Even though these rehearsals use the same actors, it will be illogical to rehearse them consecutively. See remark 1.

Precedence Assuring that certain rehearsals happen before others. This may be necessary when there is a musical number that needs separate vocal and dance rehearsals before they can be combined in a joint rehearsal.

Parallelization Allow several rehearsals to be scheduled in the same timeslot. This is particularly useful in musicals, where there may be a separate vocal and dance rehearsals. No actor can participate in two rehearsals simultaneously.

Breaks Create a schedule in which there are intentional temporal gaps between rehearsals. These gaps serve a dual purpose. The first is to allow actors in several consecutive scenes some time to rest. The second is to serve as a buffer in case certain scene rehearsals take longer than expected.

To test the comprehensiveness of our list of features, our survey also asked practitioners if there were any features they regularly considered which we had not asked about: "What else do you think about when designing a good rehearsal schedule?"

2.1.2 Utility Features

Our survey asked participants, for them to want to use a scheduling tool, how important each of the following features is to them. Participants rated the importance on a Likert Scale. 1 being "not important," and 5 being "extremely important."

Manager Enters Availability Allow a central manager to enter the availability of each actor.

Actors Enter Availability Allow the individual actors to enter their availability themselves, so that a central manager does not have to enter it on the actors' behalf.

Schedule Chunks Rather than scheduling all the scenes into all the timeslots at once, allow the practitioner to select a subset of the scenes and a subset of the timeslots to schedule.

[1] For the purposes of the survey, this features was intentionally left vague, since "hold time" is not an industry-standard term. Indeed, since our timeslots are not all contiguous, we require a more nuanced notion of efficient time use. For more details, see section 3.

Feature	Average	**5**	**4**	**3**	**2**	**1**
Hard Conflicts	4.9	33	9	1	0	1
Manager Enters Availability	4.6	30	9	5	0	0
Prescheduling	4.5	30	10	1	2	1
Rescheduling	4.4	27	11	4	2	0
Space Availability	4.4	24	8	4	3	0
Time Efficiency	4.2	19	18	5	2	0
Schedule Chunks	4.2	25	8	4	4	2
Mobile	4.2	23	12	4	3	2
Choose Several Schedules	3.9	15	17	5	5	2
Integrations	3.9	18	14	5	3	4
Parallel Rehearsals	3.7	9	13	11	1	2
Scene Spacing	3.5	9	12	14	7	1
Actors Enter Availability	3.4	13	8	11	7	5
Soft Conflicts	3.1	1	14	18	11	0
Breaks	3.2	5	15	13	6	5
Revise Conflicts	2.8	6	7	11	10	10
Scene Ordering	2.8	1	8	13	11	7

Table 1: For each feature, we report the number of respondents who gave it a particular score on the Likert scale. The optimization features are highlighted blue. The utility features are white. The features are sorted by their average score in descending order.

Prescheduling Manually fix certain scenes to certain timeslots before calculating an optimal schedule.

Choose Several Schedules Choose from several possible schedules suggested by the algorithm.

Revise Conflicts Allow conflicts to be edited after the schedule has been created, possibly leading to rescheduling.

Rescheduling Allow the manager to reschedule all or a subset of the scheduled rehearsals.

Integrations With popular calendar products, for example.

Mobile Provide a mobile-friendly interface where the manager and actors can access the scheduling optimization input and output.

2.2 Survey Results

Practitioners had, on average, 3 productions in the last year that required them to spend more than 1 hour per week on scheduling rehearsals alone. Each production, on average, rehearsed for 6 weeks, with, on average, 35 actors and crew members in each production. In total, practitioners were spending, on average, 9.6 hours per production doing the scheduling by hand.

In table 1, for each feature we report the number of respondents who gave it a particular score on the Likert scale. The optimization features are highlighted blue. The utility features are white. The features are sorted by their average score in descending order.

To determine the highest-priority features, we used a t-test to determine which features were statistically distinguishable from each other. Among the top five features, for each pair we found $p > 0.05$. Among the top eight features, for each pair we found $p > 0.01$. On the other hand, **Hard Conflicts** and **Choose Several Schedules** were statistically significantly different from each other at the $p < 0.01$ level. Thus, we decided to distinguish the top eight features as the *high priority* features. The top optimization features were **Hard Conflicts**, **Space Availability**, and **Time Efficiency**. The top utility features were **Manager Enters Availability**, **Prescheduling**, **Rescheduling**, **Schedule Chunks**, and **Mobile**.

When asked if there where any features not covered in our survey, eight of the practitioners mentioned accounting for union contracts and labor laws, especially for shows involving child actors. These rules limit the number of consecutive hours an actor can work or the number of hours they can work in a day. While this feature was not applicable in our 14 real-world use cases, the fact that we omitted it may be a barrier to wider adoption. Aside from accounting for contracts and labor laws, no other missing features were reported by the practitioners, which suggests that our list of features was reasonably comprehensive.

From our survey, two results stand out:

- Among the optimization features, we discovered that accounting for actors' hard conflicts was even more important to the practitioners than making the schedule temporally efficient for the actors. In the theoretical study of TALENT SCHEDULING, conflicts are typically ignored. This result highlights the need to reformulate the optimization problem for practitioners.
- Many of the utility features we asked about as controls were *more* important to the practitioners than optimization features. For example, more than half of the practitioners responded that having a mobile version of the scheduling tool is an "extremely important" determinant to whether they would use the tool. More than half also considered the ability to reschedule to be "extremely important."

These survey results indicate that, for practitioners to adopt a scheduling tool in practice, they expect a highly flexible scheduling tool that addresses a few core optimization concerns. Out of 44 practitioners, only 14 reported that they were currently using one or more online tools to help them schedule rehearsals. Among the 14 who are using one or more online tools, only 1 uses professional artistic management and planning software specialized for theatre scheduling. The remaining 13 use Google Forms, when2meet, and Doodle to collect availability, Excel to organize data, and Google Calendar to provide visualizations. Given the proliferation of scheduling algorithms and specialized scheduling software, we found this number to be surprisingly low. Our survey results suggest that usability plays a large role in this phenomenon and may be the largest obstacle preventing practitioners from taking advantage of scheduling algorithms. We will revisit this idea in later sections.

3 Integer Programming Formulation

In addition to the aforementioned high-priority optimization features, there were two departures from the idealized setting we considered "obvious" and therefore did not ask about in the survey. The first was a distinction between days and hours. In the idealized setting, it is typical to assume that the timeslots are contiguous: either consecutive hours on the same day or consecutive days. In the practical setting, hours and days must be distinguished. The second was handling scenes of unequal length. For the sake of formulating the problem, we assume that the timeslots have equal length and that scenes require a positive integer number of timeslots. Both of these features have been recognized as important in previous works. For example, Wang, Chuang, and Lin [12] make a distinction between days and hours. Sakulsom and Tharmmaphornphilas [11] make a distinction between days and hours and also consider rehearsals of unequal length. In our case, we also consider actor conflicts, which is not addressed in either of the aforementioned works.

3.1 Variables and Objectives

We will use A to denote the set of actors, S to denote the set of scenes, and T to denote the set of timeslots. We assume that the timeslots have equal length and are numbered in temporal order. Let D be the set of days. The timeslots can be partitioned into days: $T_0, \ldots, T_{|D|}$. We use the notation ℓ_i to denote the *length* of scene i: the number of consecutive timeslots which scene i requires. For most practical instance of the REHEARSAL SCHEDULING problem, $|A| \lessapprox 100$, $|S| \lessapprox 100$, and $|T| \lessapprox 1000$.

We define the following binary variables. The variable y_{ij} will be 1 if scene i starts in timeslot j, 0 otherwise. The variable x_{ij} will be 1 if scene i is happening in timeslot j, 0 otherwise. The variable z_{jk} will be 1 if actor k is present in timeslot j, 0 otherwise. We intentionally avoided introducing any variable indexed over $|S| \times |T| \times |A|$, which would create a prohibitively large number of variables.

In our formulation, we group **hard conflicts** and **space constraints** into a single utility score for scene i in timeslot j: $R_{ij} \in \mathbb{R}$. The distinguishing property of these features is that the utility of scheduling a certain scene in a certain timeslot can be calculated independently of the rest of the scenes. On the other hand, **time efficiency** depends on the relative positions of *all* the scenes.

If a scene must be scheduled at a time when one of the actors in it has hard conflict, the effect on utility depends on the actor and the scene. For example, it is better to schedule a 16-person scene with 1 actor missing than it is to schedule a 2-person scene with 1 actor missing. For the sake of our experiments, the reward function we use is the fraction of actors who could attend the whole rehearsal multiplied by the length of the rehearsal. If the only space available at time j was undesirable for rehearsing scene i, then R_{ij} was

further multiplied by $\frac{1}{2}$. Formally, let s_i be the set of actors needed for scene i and let t_j be the set of actors available at time j. Then, for our experiments,

$$R_{ij} = \begin{cases} \ell_i \frac{|s_i \cap t_j|}{|s_i|} & \text{if desirable space available} \\ \frac{1}{2}\ell_i \frac{|s_i \cap t_j|}{|s_i|} & \text{if desirable space unavailable} \\ -1 & \text{if scene } i \text{ impossible at time } j. \end{cases}$$

Though our experiments use a relatively simple formula for the reward, one could imagine expressing a much richer set of features. For example, each scene could be assigned a scalar representing its importance. Then, the reward of a certain scene in a certain timeslot could be multiplied by the importance of the scene. Similarly, each actor could be assigned an importance in each scene and the reward could reflected a weighted average of their availability, weighted by their importance. We intentionally write our formulation in terms of "reward" to leave open the possibility of adding these features in the future.

The final feature we would like to capture is **time efficiency**. Recall that the *hold time* is typically defined in settings where all the timeslots can be thought of as contiguous. In our setting, we consider separate days. In order to account for this distinction, we need to update our understanding of hold time.

It is tempting, as a first pass, to simply optimize the total hold time for all the actors across all the days. However, this results in peculiar behavior. For example, consider an actor which is in exactly three scenes. Assume that there are three days worth of timeslots. To minimize their total hold time across all the days, one solution would be to schedule all three of their scenes consecutively on the same day. Since the three scenes are consecutive, the actor experiences no hold time between them. However, an equally optimal solution would schedule each of the actor's scenes on a different day. In this solution, the actor also experiences no hold time on any of the days because their first and last scene each day is the one and only scene they are called for.

For the above example, it is clear that we need to consider more than just hold time when deciding if we have used actors' time efficiently. We resort to considering two quantities: the total number of days for which each actor is called as well as the total hold time across all days.

In total, we consider three objectives:

1. maximize reward (accounting for hard conflicts and for space constraints)
2. minimize the number of days called, summed across actors (an aspect of time efficiency)
3. minimize the hold time, summed across actors (another aspect of time efficiency)

3.2 Optimizing Multiple Objectives

We considered several options for handling our multi-objective optimization problem. Ultimately, we opted to treat the objectives hierarchically, leading

to a three-stage solution process. The first stage finds a solution with the maximum possible reward. The second stage finds, among all solutions with maximum reward, the solution which minimizes the sum across actors of the number of days called. The third stage, calculated separately for each day, minimizes the total hold time across actors that day.

Our decision to treat the objectives hierarchically was based on our survey results and our preliminary experiments. In our survey, we found that the **hard conflicts** feature was significantly more important to the practitioners than the **time efficiency** feature ($p < 0.05$). In a hierarchical ordering of objectives, the former feature is prioritized over the latter, consistent with the survey. When we did try combining the objectives by introducing weights, our preliminary results suggested that the integer program took a prohibitively long time to solve. Treating the objectives hierarchically limits the search space, but makes the integer programs tractable. A reasonable solve time is crucial for practical usage.

Other researchers have also chosen to solve REHEARSAL SCHEDULING in phases. Sakulsom and Tharmmaphornphilas [11] and Wang, Chuang, and Lin [12] treated REHEARSAL SCHEDULING as a multi-objective optimization problem with two objectives, corresponding to our second and third objectives (days and hold time, respectively). In both cases, they ultimately solved the problem in two phases: one for each objective. Wang, Chuang, and Lin used heuristics to produce good feasible solutions in both phases. Sakulsom and Tharmmaphornphilas used a heuristic in the first phase and then solved an integer program in the second phase. In this work, we have three phases and solve them all with integer programming.

3.3 Reward Optimization

The aforementioned reward of scheduling scene i to start in timeslot j is $R_{ij} \in \mathbb{R}$. In this formulation, we find an assignment of scenes to timeslots that maximizes total reward.

$$\max \sum_{i \in S, j \in T} R_{ij} y_{ij} \tag{IP1}$$

$$\sum_{j \in T} y_{ij} \leq 1 \quad \forall i \in S \qquad \% \text{ Each scene rehearsed at most once.}$$

$$\sum_{j'=j-\ell_i+1}^{j} y_{ij'} = x_{ij} \quad \forall i \in S, j \in T \qquad \% \text{ Define the variable x.}$$

$$\sum_{i \in S} x_{ij} \leq 1 \quad \forall j \in T \qquad \% \text{ At most one rehearsal at a time.}$$

$$x_{ij} \in \{0,1\} \quad \forall i \in S, j \in T$$

$$y_{ij} \in \{0,1\} \quad \forall i \in S, j \in T$$

Notice that if all the scenes have length $\ell_i = 1$, then $x_{ij} = y_{ij}$, and this problem becomes a maximum bipartite matching problem between scenes and timeslots.

On the other hand, if R_{ij} depends only on the scene i, not on the timeslot j, then $\sum_{j \in T} y_{ij}$ becomes a binary indicator of whether or not scene i gets assigned to any timeslot. Finding the set of rehearsals which maximize reward while not exceeding the number of timeslots is equivalent to the Knapsack problem.

3.4 Time Efficiency Optimization

Daily For the next phase in our optimization formulation, we introduce the variable δ_{dk}. This variable will be 1 if actor k is called on day $d \in D$, 0 otherwise. Let R be optimal objective value found in formulation IP1.

The idea of this formulation is to find, among the solutions with maximum reward, the one that minimizes the sum over all the δ_{dk}.

$$\min \sum_{d \in D, k \in A} \delta_{dk} \qquad \text{(IP2)}$$

$$\sum_{j \in T} y_{ij} \leq 1 \quad \forall i \in S \qquad \% \text{ Each scene rehearsed at most once.}$$

$$\sum_{j'=j-\ell_i+1}^{j} y_{ij'} = x_{ij} \quad \forall i \in S, j \in T \qquad \% \text{ Define the variable x.}$$

$$\sum_{i \in S} x_{ij} \leq 1 \quad \forall j \in T \qquad \% \text{ At most one rehearsal at a time.}$$

$$z_{jk} = \sum_{\{i | k \in s_i\}} x_{ij} \quad \forall k \in A, j \in T \qquad \% \text{ Define the variable z.}$$

$$\delta_{dk} \geq z_{jk} \quad \forall d \in D, j \in T_d, k \in A \qquad \% \text{ Define the variable } \delta.$$

$$\sum_{i \in S, j \in T} R_{ij} y_{ij} = R \qquad \% \text{ Maintain reward.}$$

$$x_{ij} \in \{0,1\} \quad \forall i \in S, j \in T$$
$$y_{ij} \in \{0,1\} \quad \forall i \in S, j \in T$$
$$z_{jk} \in \{0,1\} \quad \forall j \in T, k \in A$$
$$\delta_{dk} \in \{0,1\} \quad \forall d \in D, k \in A$$

Hourly In the previous formulation, we assigned scenes to days. Let S_d be the set of scenes which were assigned to day $d \in D$ and let R_d be the reward achieved that day. Our final formulation assumes the scenes are fixed to certain days and refines the hold time within the days. Because the formulations for each day do not affect each other, they can be solved in parallel.

For the sake of this program, we will only consider the timeslots which fall on a particular day d. Recall that T_d denotes these timeslots. We introduce two new variables, $h^+_{jk} \in \{0,1\}$ and $h^-_{jk} \in \{0,1\}$. The variable h^+_{jk} will be 1 if and only if actor k is called for a scene at *or after* timeslot j. The variable h^-_{jk} will be 1 if and only if actor k is called for a scene at *or before* timeslot j. Thus, the sum $h^+_{jk} + h^-_{jk}$ is 2 for actor k if they are rehearsing or held at timeslot j. Otherwise, the sum is 1.

$$\min \sum_{j \in T_d, k \in A} h^+_{jk} + h^-_{jk} \tag{IP3}$$

$$\sum_{j \in T_d} y_{ij} \leq 1 \quad \forall i \in S_d \qquad \% \text{ Each scene rehearsed at most once.}$$

$$\sum_{j'=j-\ell_i+1} y_{ij'} = x_{ij} \quad \forall i \in S_d, j \in T_d \qquad \% \text{ Define the variable x.}$$

$$\sum_{i \in S_d} x_{ij} \leq 1 \quad \forall j \in T_d \qquad \% \text{ At most one rehearsal at a time.}$$

$$z_{jk} = \sum_{\{i | k \in s_i\}} x_{ij} \quad \forall k \in A, j \in T_d \qquad \% \text{ Define the variable z.}$$

$$\sum_{i \in S_d, j \in T_d} R_{ij} y_{ij} = R_d \qquad \% \text{ Maintain reward.}$$

$$h^+_{jk} \geq z_{jk} \quad \forall j \in T_d, k \in A \qquad \% \ h^+\text{: if an actor } \textit{is} \text{ called.}$$

$$h^-_{jk} \geq z_{jk} \quad \forall j \in T_d, k \in A \qquad \% \ h^-\text{: if an actor } \textit{is} \text{ called.}$$

$$h^+_{jk} \geq h^+_{j+1,k} \quad \forall j \in T_d, k \in A \qquad \% \ h^+\text{: if an actor } \textit{will be} \text{ called.}$$

$$h^-_{jk} \geq h^-_{j-1,k} \quad \forall j \in T_d, k \in A \qquad \% \ h^-\text{: if an actor } \textit{was} \text{ called.}$$

$$x_{ij} \in \{0,1\} \quad \forall i \in S_d, j \in T_d$$

$$y_{ij} \in \{0,1\} \quad \forall i \in S_d, j \in T_d$$

$$z_{jk} \in \{0,1\} \quad \forall j \in T_d, k \in A$$

$$h^+_{jk} \in \{0,1\} \quad \forall j \in T_d, k \in A$$

$$h^-_{jk} \in \{0,1\} \quad \forall j \in T_d, k \in A$$

3.5 Solutions

The relation among the integer programs is illustrated in figure 2. We solved the integer programs using the open-source solver COIN-OR Branch and Cut (CBC) with a one-minute time limit. In practice, on instances of practical size and structure, IP1 was typically solved to optimality within the time limit, IP2 typically terminated at the time limit with an optimality gap less than 20%, and IP3 was typically solved to optimality within the time limit.

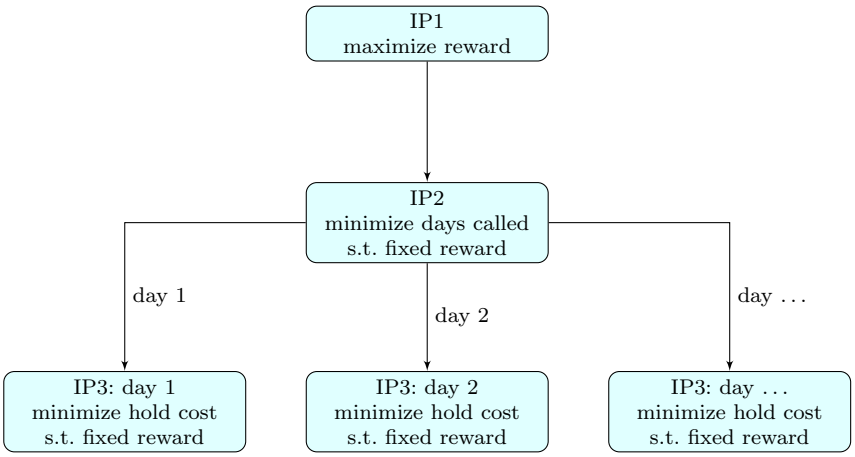


Fig. 2: A flowchart illustrating the series of integer programs we use to find a "good" feasible solution to the REHEARSAL SCHEDULING problem. Arrows indicate sending a feasible solution from one integer program to the next.

4 Experiments

4.1 Setup

We conducted user studies with a total of 14 practitioners, in two phases. We helped practitioners generate optimized rehearsal schedules for their shows using our scheduling system.

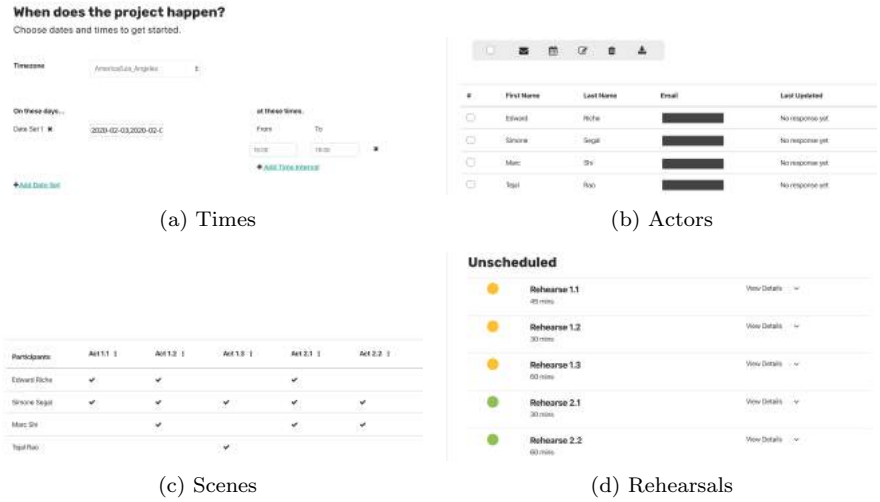
A diverse set of shows were tested, including a mix of plays and musicals, classical and new. Some of the shows used in this phase of testing included *Hamlet*, *The 25th Annual Putnam County Spelling Bee*, *Twelfth Night*, *Salomé*, *As You Like It*, *The Wizard of Oz*, *A Midsummer Night's Dream*, *Next to Normal*, and several independent works. The theater companies were predominantly located in the San Francisco Bay Area and the Greater Boston Area.

Phase I: Experimenter-in-the-Loop First, we tested our scheduling optimization algorithm by manually entering the scheduling data and constraints for two practitioners scheduling their respective shows. The practitioners were asked to evaluate the resulting schedules. Practitioners were not presented with an interface.

Next, we built a software tool for the practitioners to enter inputs to the algorithm, themselves. We designed and developed the software with the high-priority usability features in mind (see section 2.2). In particular, we implemented **Manager Enters Availability**, **Prescheduling**, **Rescheduling**, and **Schedule Chunks**. Images of the interface can be seen in figure 3.

In this phase, we used "experimenter-in-the-loop" testing to assist practitioners navigating the interface. The practitioners were prompted to use the

Fig. 3: Images of the interface in which users entered data for the scheduling algorithm.



(a) Times (b) Actors

(c) Scenes (d) Rehearsals

interface to generate a rehearsal schedule for a show that they had managed in the past. Six practitioners participated. In the software interface, the practitioners were prompted to input the following data:

Times General dates and times when rehearsals could happen.
Actors Names and emails of actors.
Scenes The set of actors required for rehearsing a section of the play. See remark 1.
Rehearsals The duration and actors needed for each rehearsal. See remark 1.

The practitioners were asked to evaluate the quality of the schedule generated by the algorithm. They were also asked to evaluate their experience of using the scheduling tool.

Phase II: Experimenter-out-of-the-Loop After iterating our interactive software prototype, incorporating the feedback we collected from *Phase I*, we sent the updated software prototype to six participants to complete the task of scheduling rehearsals for their respective shows. Participants were asked to perform the scheduling independently, and on their own time. They were asked to evaluate the resulting schedule generated by the algorithm, as well as the experience of using the scheduling software.

4.2 Results

Participants responded very positively to the schedules produced by the algorithm, and were relieved by the ease of automating scheduling. Notably, in

Phase I of the experiment, all participants were able to complete scheduling rehearsals for the entire production in 2 hours, in contrast to the average 9.6 hours spent scheduling each production using their current methods.

Phase I: Experimenter-in-the-Loop All the participants reported that they were satisfied with the optimized schedules that the algorithm generated. Participants said that the automatically optimized rehearsal schedules were similar to schedules they manually created. Participants further stated that deviations from their manually produced schedules were either insignificant (for example, swapping similar rehearsals) or reasonable (for example, scheduling some rehearsals at better times at the expense of other rehearsals). All participants finished creating rehearsal schedules for the entirety of their shows under 2 hours, a significant reduction from 9.6 hours on average spent scheduling per production as reported in the survey.

The participants were allowed to give free-form feedback. Several commented on the time and energy saved. Others commented on the quality of schedules generated. Responses included:

- "I cannot tell you how exciting it is to have all the scheduling done for me."
- "It personally would have saved me a lot of work."
- "The schedules are clearly better than last year."
- "Your algorithm really saved us time and stress."
- "This is really cool. If people can put their conflicts in, it's, like, no work."
- "The workflow can be a little simpler, but how the tool automatically created this perfect schedule was cool. I see a lot of potential in this tool."

One limitation of the user study was not including the time of rescheduling rehearsals due to changes in actor availability.

Phase II: Experimenter-out-of-the-Loop This turned out to be the more challenging testing session for users. One user confessed they in fact did not finish scheduling, saying that "[The software] is too complicated." While the practitioners who did finish scheduling were satisfied with the resulting schedule (one practitioner called the results "delightful"), they were significantly less satisfied with the process than the users who received experimenter assistance in Phase I. The practitioners expressed concern that the data entry was cumbersome and lacked flexibility. One practitioner said they would "probably not use it again." Another said they would "rather manually schedule by themselves," because of familiarity and flexibility.

The only difference between *Phase I* and *Phase II* was the guidance of researchers in navigating the interface, in addition to minor improvements to the interface. Experimenters' guidance turned out to be fundamental to the practitioners' success and positive experience using the scheduling tool.

Additionally, our user studies revealed that users have varying levels of trust in computer-generated results. Participants were curious about yet skeptical of the schedule generated by the computer throughout the data entry

process. All eight users from *Phase I* expressed the desire to tweak the outcome of the initial computer-generated schedule. Giving users the flexibility to make free-form changes is important. The ability to freely manipulate and potentially re-purpose the outcome returns agency back to the users, which positively contributes to users' experience of interacting with the scheduling software.

5 Final Remarks

In designing a scheduling system, our goal was to capture human intuition for what makes a good schedule in an integer programming formulation. Our user studies suggest that we achieved that, generating schedules which felt "good" to practitioners.

Although our tool was adequate from an optimization standpoint, a lot more work would need to be done outside of optimization in order to make the tool appealing to non-technical practitioners. In our survey, we discovered that certain utility features were as popular as, and sometimes more popular than, the optimization features.

A practical scheduling tool needs to support scheduling demands of varying intensity, to improve significantly or at least be compatible with practitioners' existing workflows, and be considerate of users' comfort level with foreign software interfaces.

Through our experiment, we found that the best practice for building a usable scheduling tool is to involve users in every cycle of the development, collecting their feedback and iterating. Rather than defer this to be handled by engineers and designers, we believe this process should start as early as developing an optimization formulation that incorporates practitioner input and is aware of their non-optimization needs.

References

[1] RM Adelson, JM Norman, and G Laporte. "A dynamic programming formulation with diverse applications". In: *Journal of the Operational Research Society* 27.1 (1976), pp. 119–121.

[2] Felix Bomsdorf and Ulrich Derigs. "A model, heuristic procedure and decision support system for solving the movie shoot scheduling problem". In: *Or Spectrum* 30.4 (2008), pp. 751–772.

[3] Kellogg S Booth and George S Lueker. "Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms". In: *Journal of computer and system sciences* 13.3 (1976), pp. 335–379.

[4] Tai Chiu Edwin Cheng, Bertrand Miao-Tsong Lin, Hsiao-Lan Huang, et al. "Talent hold cost minimization in film production". In: (2017).

[5] TCE Cheng, JE Diamond, and BMT Lin. "Optimal scheduling in film production to minimize talent hold cost". In: *Journal of Optimization Theory and Applications* 79.3 (1993), pp. 479–492.

[6] Uriel Feige and James R Lee. "An improved approximation ratio for the minimum linear arrangement problem". In: *Information Processing Letters* 101.1 (2007), pp. 26–29.

[7] Maria Garcia de la Banda, Peter J Stuckey, and Geoffrey Chu. "Solving talent scheduling with dynamic programming". In: *INFORMS Journal on Computing* 23.1 (2011), pp. 120–137.

[8] Andreas Holzinger. "Usability engineering methods for software developers". In: *Communications of the ACM* 48.1 (2005), pp. 71–74.

[9] Anna-Lena Nordstrom and Suleyman Tufekci. "A genetic algorithm for the talent scheduling problem". In: *Computers & Operations Research* 21.8 (1994), pp. 927–940.

[10] Hu Qin et al. "An enhanced branch-and-bound algorithm for the talent scheduling problem". In: *European Journal of Operational Research* 250.2 (2016), pp. 412–426.

[11] Noppadon Sakulsom and Wipawee Tharmmaphornphilas. "Scheduling a music rehearsal problem with unequal music piece length". In: *Computers & Industrial Engineering* 70 (2014), pp. 20–30.

[12] Sin-Yi Wang, Ying-Ting Chuang, and Bertrand MT Lin. "Minimizing talent cost and operating cost in film production". In: *Journal of Industrial and Production Engineering* 33.1 (2016), pp. 17–31.

A Double-Horizon Approach to a Purely Dynamic and Stochastic Vehicle Routing Problem with Delivery Deadlines and Shift Flexibility*

Nikolaus Frohner · Günther R. Raidl

Abstract We are facing a purely dynamic and stochastic vehicle routing problem with delivery deadlines motivated by a real-world application where orders arrive at an online store dynamically over a day to be delivered within short time. Pure dynamism is given since we do not know any orders in advance, whereas the stochastic aspect comes into play by having estimates for the hourly numbers of orders. The goal is to satisfy the daily demand by constructing closed routes from a single depot to the customers given a set of drivers with a predefined shift plan and the hourly demand estimates as input while first minimizing due time violations and then labor and travel costs. Labor costs are subject to optimization since the end times of shifts have a certain amount of flexibility and a decision has to made whether to send home a driver earlier than planned or to extend the shift.

In this work, we present a novel double-horizon approach based on the shifts and the hourly demand estimation. Within the shorter horizon we optimize the routes for the orders currently available whereas within the longer horizon we extrapolate until the end of the day to determine target shift end times for the drivers. Furthermore, we devise a route departure time strategy that balances between route quality and risking due time violations. The routing is performed by a classical adaptive large neighborhood search. We consider artifical instances and compare the results for the online problem with those for the offline scenario where all orders are known from the beginning. We observe superior performance of our approach as compared to fixed route departure time and driver send home strategies.

Keywords Dynamic and stochastic vehicle routing problem · double-horizon approach · adaptive large neighborhood search

* This work is supported by the Vienna Graduate School on Computational Optimization, funded by the Austrian Science Fund under grant W1260.

N. Frohner and G. R. Raidl
Institute of Logic and Computation, TU Wien, Vienna, Austria
{nfrohner|raidl}@ac.tuwien.ac.at

1 Introduction

Motivated by a real-world application where customers place orders at an online store to be delivered within a few hours, we introduce a specific vehicle routing problem (VRP) variant called *Purely Dynamic and Stochastic Vehicle Routing Problem with Delivery Deadlines and Shift Flexibility*. Orders arrive dynamically over the day, and each order is due only a couple of hours after arrival, where the specific due times vary and depend on the orders' types. These orders are picked at a single depot and are subsequently available for delivery to the customers by drivers with predefined shifts.

The goal is to assign the orders to the drivers and perform the routing in a way to avoid or minimize due time violations. Drivers perform multiple routes over the day and for each route a decision has to be made when to start it. This is crucial since after the departure of a driver, the corresponding route cannot be changed anymore. As secondary objective, the labor costs, which are determined by the actual shift end times, and the travel times, determined by the performed delivery routes, are to be minimized. The shift end times are subject to some flexibility and may be ended earlier or extended to account for the uncertainty of the actual load.

In particular, we need to account for the strong dynamism of the problem by making use of the stochastic information known in advance. As such, an estimation of the demand for each hour over the day is available upfront. To link this information to the shifts, the time-dependent average number of orders drivers can handle per hour—the driver performance—needs to be estimated. In this work, we combine well-known adaptive large neighborhood search for vehicle routing [10,13] with a double horizon approach [8] to handle dynamism and stochasticity. In the short horizon planning, we present a driver performance dependent route departure time strategy—more efficient routes are started earlier than inefficient routes, where improvement is still expected. To avoid sending drivers home too early, we look ahead until the end of the day—the large horizon—by solving a simplified assignment problem on the expected orders without concrete routing to predict target shift end times for the drivers.

In Section 2 we discuss related work. The formalization of the different problem variants (offline, point in time, online), the solution representation, and the objective function is presented in Section 3. Short horizon routes construction is done by adaptive large neighborhood search using classical insertion and regret heuristics and a diverse set of destroy operators as briefly discussed in Section 4. We present the details of our driver performance estimation in Section 5 which is crucial for our departure time strategy (Sect. 6) and our double horizon approach (Sect. 7). The latter is also used to enable informed shift ending strategies as described in Section 8. In the computational study (Sect. 9), we compare the double-horizon approach with fixed, less sophisticated strategies, on artificial instances with different load patterns (business day vs. weekend) and shift plans (generous vs. tight vs. shortage). We observe strong advantages of the former. We conclude in Section 10.

2 Related Work

For general overviews on methods to solve dynamic and stochastic VRPs, see the surveys by Ritzinger and Puchinger [12], Pillac et al. [9], and Psarafitis et al. [11]. Many existing approaches apply periodic or continuous re-optimization of the problem for the current time and essentially ignore information on expected orders. In our context such an approach would not work well as guaranteed delivery times are rather short and started routes cannot be adapted anymore (with respect to the orders they fulfill). Thus, adequately exploiting the estimations of expected orders is of crucial importance.

To handle these uncertainties existing approaches typically fall into one of two categories: those based on *sampling* and those based on *stochastic modeling* [9]. As their name suggests, sampling strategies incorporate stochastic knowledge by generating scenarios based on realizations drawn from suitable random variable distributions. Each scenario is optimized by solving the implied static and deterministic (i.e., offline) problem variant. Then a consensus solution is typically derived from all scenario solutions, which is actually applied in the next time step, until a re-optimization takes place. The advantage of sampling is its relative simplicity and flexibility on distributional assumptions, while its drawback is the massive generation and required solving of scenarios to accurately reflect reality. On the other hand, approaches based on stochastic modeling integrate stochastic knowledge analytically. They try to formally capture the stochastic nature of the problem and are usually highly technical in their formulations and require to efficiently compute possibly complex expected values. Typically, only strong abstractions from the real world allow for stochastic modeling. Applied methods to solve such stochastic models include Markov models and stochastic dynamic programming. In the case of our problem, precise and flexible enough analytical models unfortunately appear to be out of reach.

In the following we review the most relevant existing works we have found in conjunction with our specific VRP.

Bent et al. [3] were one of the first describing an event based model to solve a dynamic VRP. In their *multi plan approach* (MPA) a set of possible routing plans is maintained at any time and updated at certain events. There is one distinguished "best" plan which is determined by an appropriate selection function. The events are new customer requests, vehicle departures according to a current distinguished plan, the availability of newly generated plans, and the timeout of plans. The authors further extend the MPA by sampling to a *multiple scenario approach* (MSA) in order to obtain more robust solutions concerning the stochastic aspects. A number of scenarios is created by adding randomly sampled artificial orders, these scenarios are solved, and then a consensus solution is derived for the original online problem at a certain time. A tabu search is used for actually solving the occurring subproblems. We essentially also follow the fundamental concept of the event based model of the MPA, although with just one current solution.

Hvattum et al. [7] propose another sampling scenario-based approach in conjunction with a rather simple hedging heuristic.

Gendreau et al. [6] describe a tabu search with adaptive memory for a dynamic vehicle routing problem. Essentially, an MPA-like event model is used in conjunction with tabu search and the problem is re-solved whenever new information is available. Stochastic aspects are not considered here, but a focus lies on an effective parallelization.

Ropke and Pisinger [13] and Pisinger and Ropke [10] proposed *Adaptive Large Neighborhood Search* (ALNS) for more general vehicle routing problems, which is nowadays widely used as framework for a large variety of optimization problems. ALNS is appreciated for its practical efficiency as well as robustness on many occasions. The main idea of ALNS is to repetitively destroy a current candidate solution partially and repair it in a sensible way. Both are done by using sets of different basic operators, which are typically randomized. Improved solutions are always accepted as new current ones, while worse solutions are only accepted according to a Metropolis criterion. The application probability of the individual destroy and repair operators are adapted over the iterations based on their successes in previous iterations. ALNS is today among the most often applied metaheuristics for VRPs in general, and we find it also most useful as core optimization technique for solving our routing problem, see Section 4.

Azi et al. [1] consider a VRP with a particular focus on multiple routes per vehicle, as we also have to do. A major difference to our problem is that here the focus is on deciding upon the acceptance of requests. The solution approach is an ALNS that is in several aspects similar to those from Ropke and Pisinger. Azi et al. [1] extend this work towards the dynamic problem variant. Stochastic sampling is applied to account for unknown expected orders.

Schilde et al. [14] describe a variable neighborhood search metaheuristic for a dynamic dial-a-ride problem. The authors also apply sampling for dealing with the stochastic aspects. In their variable neighborhood search the shaking moves bear some similarities with the destroy and repair operations of ALNS.

Mitrović-Minić et al. [8] describe a double horizon approach for solving a dynamic pickup and delivery problem. A large horizon is considered for maintaining routes in a state to be able to easily respond to future dynamically appearing requests, while a short horizon is considered for the actual goal to minimize the route lengths based on the so far known requests. While the considered problem is quite different to ours, we adopt the basic idea of considering two planning horizons in our *double horizon approach* in Section 7.

3 Problem Formalization

We distinguish between three problem variants: the offline problem with full knowledge of the day in advance (OFF), the dynamic problem at a specific time $\tilde{t}$ (DYN-$\tilde{t}$), and the full dynamic problem for a whole day (DYN-DAY).

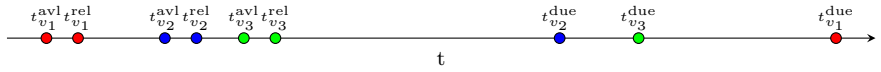


Fig. 1 Visualization of order-related times of an example route $r = \{0, v_2, v_3, v_1, 0\}$. $t_v^{\text{avl}} \leq t_v^{\text{rel}} \leq t_v^{\text{due}}$ holds for all orders: first it is placed by the customer (t_v^{avl}), then it is picked from the warehouse (t_v^{rel}) and ready for delivery by a driver, and then due (t_v^{due}). Note that orders that are placed later may be due earlier. The earliest route departure time is bound from below by the latest release time of the corresponding orders. For this particular example $\tau_r \geq t_{v_3}^{\text{rel}}$ must hold.

3.1 Full-Knowledge Offline Problem (OFF)

Here all orders of the day are known in advance together with their release times, i.e., the times the orders have been picked in the warehouse and are ready for delivery by the drivers. Although this problem variant is not what we are confronted with in reality, it is nevertheless interesting as its (optimal) solution provides a baseline of what might ideally be achieved in the online problem. We denote the set of all orders by V, with $n = |V|$, and the corresponding release times by t_v^{rel}, $\forall v \in V$. Moreover, we are given due times t_v^{due}, $\forall v \in V$, which are related to a promised maximum delivery duration starting from the time t_v^{avl} the order v was placed by the customer. Fig. 1 visualizes the order-related times of an example route consisting of three orders.

Furthermore, for all relevant vehicles $u \in U$, with $m = |U|$, planned shift time intervals $[q_u^{\text{start}}, q_u^{\text{end}}]$ and earliest shift ends $q_u^0 \in [q_u^{\text{start}}, q_u^{\text{end}}]$ are provided. Lastly, expected travel times $\delta(v, v')$ from location v to location v', where $v, v' \in V \cup \{0\}$ with 0 representing the warehouse, are given. These travel times include average stop times at the customers, average times for loading a vehicle at the warehouse, and postprocessing times when returning to the warehouse. We further assume that the triangle inequality holds w.r.t. the travel times and that they are constant throughout the day.

Solution Representation. We have to plan the drivers' routes, the route departure times, and the drivers' flexible shift end times. Hence, a candidate solution is a tuple $\langle R, \tau, q \rangle$ where

- $R = (R_u)_{u \in U}$ denotes the *ordered sequence of routes* $R_u = \{r_{u,1}, \ldots, r_{u,\ell_u}\}$ to be performed by each vehicle $u \in U$, and each *route* $r \in R_u$ is an ordered sequence $r = \{v_0^r = 0, v_1^r, \ldots, v_{l_r}^r, v_{l_r+1}^r = 0\}$ with $v_i^r \in V$, $i = 1, \ldots, l_r$, being the i-th order to be delivered and 0 representing the warehouse at which each tour starts and ends,
- $\tau = (\tau_r)_{r \in R_u, u \in U}$ are the (planned) *departure times* of the routes, and
- $q = (q_u)_{u \in U}$ are the *shift end times* of the vehicles.

The time at which the i-th order v_i^r of route r, $i = 1, \ldots, l_r$, is delivered is

$$a(r, i) = \tau_r + \sum_{j=0}^{i-1} \delta(v_j^r, v_{j+1}^r). \tag{1}$$

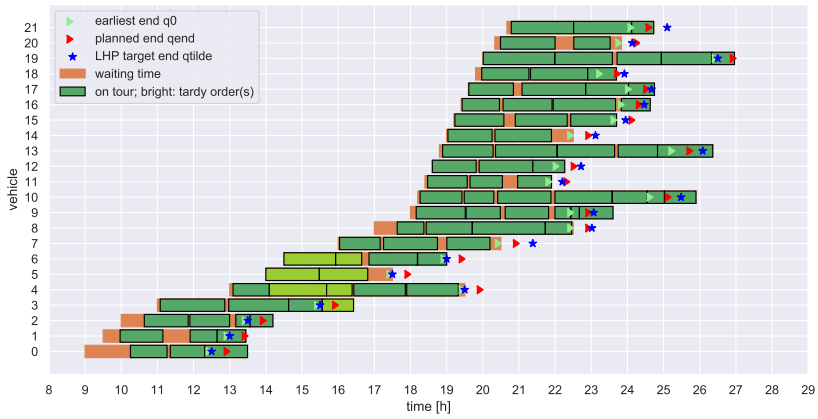


Fig. 2 Visualization of a solution for an artificial instance with 22 vehicles. The x-axis denotes the time and the discrete y-axis the drivers' shifts. The whole bar indicates the actual shift duration. The green triangle indicates the earliest shift end time for each driver, where excess labor time contributes to our considered costs. The red triangle depicts the planned shift ending, after which no route can be started, but the last route may end arbitrarily late. The distinct green bars stand for routes and contribute to the travel time part of our objective function. If the latter is shaded light green, the route contains at least one tardy order, which can be observed around hour 15. The remaining orange of the shift bars denote waiting time of the driver at the depot. The blue stars denote the target end shift times as determined by the large horizon planning in the beginning of the day.

The total duration of a route $r \in R_u$ of a vehicle $u \in U$ is

$$d(r) = \sum_{i=0}^{l_r} \delta(v_i^r, v_{i+1}^r), \tag{2}$$

and the route therefore is supposed to end at time $\tau_r + d(r)$.

Let $\tau^{\min}(r) = \max_{i=1,\dots,l_r} t_{v_i^r}^{\text{rel}}$ be the earliest feasible starting time of a route r, which corresponds to the maximum release time of the orders served in the route. Furthermore, let $\tau^{\max}(r)$ be the latest starting time without violating any due time, i.e.,

$$\tau^{\max}(r) = \min_{i=1,\dots,l_r} \left(t_{v_i^r}^{\text{due}} - \sum_{j=0}^{i-1} \delta(v_j^r, v_{j+1}^r) \right). \tag{3}$$

Feasibility. A solution is feasible when

- each order $v \in V$ appears exactly once in all the routes in $\bigcup_{u \in U} R_u$,
- each route $r \in R_u$, $u \in U$, is started in the planned shift time of the assigned vehicle, i.e., $\tau_r \in [q_u^{\text{start}}, q_u^{\text{end}}]$,
- and not started before all corresponding orders are released, i.e., $\tau_r \geq \tau^{\min}(r)$,
- the routes in each R_u, $u \in U$ start at increasing times and do not overlap, i.e., $\tau_{r_{u,i}} + d(r_{u,i}) \leq \tau_{r_{u,i+1}}$, $i = 1, \dots, |R_u| - 1$,

– and the actual shift end time is not smaller than the finishing time of the last route (if there is one) and the minimum shift time, i.e., $q_u \geq \max(q_u^0, \sup_{r \in R_u}(\tau_r + d(r))),\ u \in U$.

Objective. The primary goal is to avoid tardiness or distribute it evenly among the customers. The secondary goal is reduce labor and travel costs. This leads to the following objective function to be minimized

$$f(\langle R, \tau, q \rangle) = \mathrm{L}\left(\sum_{r \in R_u, u \in U} \sum_{i=1}^{l_r} \max(0, a(r,i) - t_{v_i^r}^{\mathrm{due}})^2,\ \gamma \cdot \sum_{u \in U}(q_u - q_u^0) + \sum_{r \in R_u, u \in U} d(r) \right). \tag{4}$$

L denotes the lexicographic combination of two terms, which are a quadratic penalty for the tardiness of deliveries and a linear combination of the sum of labor and travel costs. More precisely, the latter is calculated as the sum of the actual shift durations above the minimum shift times q_u^0 weighted by a factor γ and the sum of travel times.

In a real-world comparison of results, it is also worthwhile to view it as a multi-objective optimization problem. A small increase in tardiness may be acceptable, if it comes with a substantial reduction of costs.

3.2 Dynamic Problem at a Specific Time $\tilde{t}$ (DYN-$\tilde{t}$)

This problem variant is actually the one that needs to be iteratively solved during the whole day, for increasing current time $\tilde{t}$. It extends OFF by having as additional input the current time $\tilde{t}$ and the expected number of orders $\hat{\omega}(t)$ that become available in the time intervals $[t, t + 1\mathrm{h})$ for all relevant business hours. Moreover we assume to have knowledge about the distribution of order types w.r.t. the promised delivery durations. The set of all orders V is reduced to those which are already available at time $\tilde{t}$ and whose delivery has not yet started. The set of vehicles U is reduced to those whose shift has not been finished, and shift start times are updated to expected return times of vehicles that are currently on a tour. The route construction must now additionally consider these unknown future orders in an appropriate way. The ultimate goal is to lead to an optimal solution w.r.t. the full dynamic problem below.

3.3 Full Dynamic Problem (DYN-DAY)

This is the actual problem to be solved from the point-of-view of the whole day. Time is considered to continuously increase over the whole relevant time horizon, expected numbers of future orders are known as above, but each concrete order becomes available only at the availability time $t_v^{\mathrm{avl}},\ \forall v \in V$. The decision on each route $r \in R$ must be fixed with only the knowledge available up to the routes respective departure time τ_r. An example solution of a DYN-DAY instance with 22 vehicles is depicted in Fig. 2 as a bar chart

displaying the waiting times (orange), routes (green), and routes with tardy orders (light green) of the drivers. Stars show the target shift end times derived from our initial large horizon planning (Sect. 7).

More specifically, we solve the successive DYN-$\tilde{t}$ instances every time an order is released:

$$\tilde{t} \in \left\{ t \mid \exists v \in V \colon t = t_v^{\text{rel}} \right\} \tag{5}$$

Having obtained a solution for a time $\tilde{t}$, we extract any routes that start before the next value for $\tilde{t}$ in the above sequence, adopt these routes for the final solution of DYN-DAY, and remove all the orders served in these routes from any further consideration.

4 Routes Construction and Optimization

To be suitable for a real-time application, an important property that an optimization method must exhibit is a good *anytime behavior*: a somehow reasonable heuristic solution must be found very soon (within seconds), and over time the solution should continuously be improved up to (or close to) optimality. In other words, the optimization can be interrupted almost at any time and a reasonable solution with respect to the invested time is available. We achieve this by using a carefully designed *Adaptive Large Neighborhood Search* (ALNS) [9,13].

ALNS heuristics. As construction heuristics to insert orders into either an empty solution or to repair a partial solution in the ALNS, we use the well-known insertion and regret-k heuristics as described in [10]. We distinguish between the *zero-tardiness* and *tardiness* regimes. In a two-stage approach, we first seek to insert an order without introducing additional tardiness, which can be checked in constant time with caching of suitable slack values for existing orders and routes. If this is not possible, we search for an order position with the smallest sum of squares increase of tardiness, which is computationally more demanding by a factor of $O(n)$.

Our destruction heuristics are mostly adopted from Pisinger and Ropke [10], Ropke and Pisinger [13], Shaw [15], and Azi et al. [1] and suitably adapted to our problem. There are two kinds of destruction heuristics, those that remove a certain number of orders from routes and those that remove a certain number of whole routes. More specifically, we use *random order*, *random route*, *related order*, *related route*, *worst order*, and *worst and related order* removal.

Shift End Times. The actual shift end times q_u for the vehicles are set to $\max(q_u^0, \sup_{r \in R_u}(\tau_r + d(r)))$, i.e., for each vehicle u to the end of the last route or the earliest possible shift end time, whichever comes later. In Section 7, we introduce the large horizon planning, where we estimate desired shift ends for the vehicles in advance so that we can satisfy the expected workload. Since in

the objective function we penalize labor time after the earliest possible shift end times q^0, we grant vehicles that are below their desired shift end time $\tilde{q}_u > q_u^0$ a labor time bonus that equalizes the incurred labor time costs up until $\tilde{q}_u$—otherwise, the insertion heuristic would avoid assigning orders to vehicles after their earliest possible shift end q_u^0, in case there is no tardiness yet and other vehicles not close to their shift end are available. The labor time bonus is implemented by using the augmented objective function

$$\tilde{f}(\langle R, \tau, q\rangle)) = f(\langle R, \tau, q\rangle) - \gamma \cdot \sum_{u \in U} \min(q_u - q_u^0, \tilde{q}_u - q_u^0). \tag{6}$$

During the optimization, the route departure time is always set to the earliest possible time. Afterwards, we are free to postpone the routes up to the latest time within the departure time slacks of the routes so that the objective value is neither increased by tardiness nor by labor costs.

5 Driver Performance Estimation

For both an informed route departure time strategy and our large horizon approach, we need to estimate the driver performance of a given hour. It is the average time needed to serve an order. It is strongly related to the expected duration of all routes involved to serve the customers at the considered time interval divided by the number of customers. We introduce this as a function $\phi\colon \mathbb{R} \to \mathbb{R}$, depending on the load λ. We define the load λ to be the expected number of orders due in a given hour.

A classical result by Beardwood et al. [2] shows that the expected length of an optimal traveling salesperson tour with n randomly sampled cities given some geometry with area $\mathcal{A}$ grows with $k\sqrt{\mathcal{A}n}$. k is an empirical constant depending on the spatial distribution and the metric. This result is extended to capacitated vehicle routing problems by Daganzo [4] and refined by Figliozzi [5], from which we adapt the following model to explain $\phi(\lambda)$

$$\phi(\lambda; k_m, k_l) \approx k_m + \frac{k_l}{\sqrt{\lambda + 1}}. \tag{7}$$

k_m corresponds to constant costs occurring for each customer like the stop time at the customer. k_l relates to the empirical k from [2] and accounts together with $(\lambda+1)^{-1/2}$ for the expected travel time to a customer. We shift the load by one to avoid divergence at zero load. As we can see, it is a function that decreases with the square root of the load. As a more flexible model, we further suggest the following inverse power law

$$\phi(\lambda; k_m, k_l, \alpha) \approx k_m + \frac{k_l}{(\lambda + 1)^{\alpha}}. \tag{8}$$

To check the validity of these models in our setting and tune the parameters, we create ten artificial instances each for loads starting from 0.5 up to 20 in steps

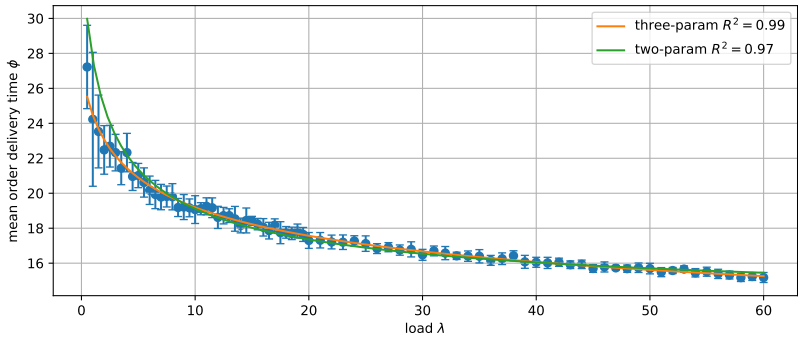


Fig. 3 Mean order delivery times ϕ in minutes with standard errors over ten instances for each load value $\lambda \in \{0.5, 1.0, \ldots, 20, 21, \ldots, 60\}$ with fitted curves for the two and three-parameter models. The three-parameter model seems to explain the region of little load $\lambda \leq 10$ better than the two-parameter model.

of 0.5 and further in steps of 1 up to 60, i.e., to one order due per minute. The geometry is the unit disk with a central depot, Euclidean metric, and vehicles driving a constant pace of 20 minutes per unit distance. Furthermore, constant stop times at the customers, loading times when leaving the warehouse, and postprocessing times when returning to the warehouse are added. Orders arrive randomly throughout a whole day at a given constant rate λ sampled from a Poisson process following a uniform spatial distribution. Optimization at each DYN-$\tilde{t}$ is done for 60 seconds using ALNS. Sufficient drivers are available so that no tardiness occurs, and the drivers wait to start their routes as long as possible. For each instance, we average over all routes the time needed to serve a customer.

In Fig. 3 we see a scatter plot of the mean order delivery times including standard errors ($N = 10$) over the different loads. Weighted least squares fits of the models are displayed. Both models explain the data starting from load $\lambda \geq 10$ similarly well with weighted R^2 values of 0.97 and 0.99. For low-load regions $\lambda \leq 10$, the model with the inverse power as an arbitrary parameter lies closer to the means.

6 Departure Time Strategies

In the dynamic problem, at every time $\tilde{t}$ we construct and optimize the routes for the drivers. After that, we have to decide when these should be started. A departure time window $[\tau_r^{\text{earliest}}, \tau_r^{\text{latest}}]$ is attributed to each route, within which the departure time τ_r of the route may be set while maintaining a feasible solution and not increasing the objective value. Setting $\tau_r < \tau_r^{\text{earliest}}$ or $\tau_r > \tau_r^{\text{latest}}$ makes the solution either infeasible or increases tardiness, labor, or travel costs. This decision is crucial since routes cannot be adapted anymore after they have been started.

Two naive strategies can immediately be devised by either always starting the route at τ^{earliest} or at τ^{latest}. τ^{latest} seems favorable in most situations since not yet started routes may later still be adapted in order to more efficiently include newly emerged orders as opposed to the earliest strategy where in the extreme case a route is immediately started with just one order. However, experiments have shown that the start-latest strategy is not always the better strategy, since we may run into tardiness at a later time when working at or shortly before critical utilization and letting vehicles wait instead of delivering orders.

A more sophisticated approach takes into account the current performance of a route, measured by its number of minutes per order d_r^{O}, i.e., the route duration divided by the number of orders served. The main idea is: the better the performance of a route, the closer we can set its departure time τ_r towards τ_r^{earliest}, the worse, the closer towards τ_r^{latest}, so that there is a performance-dependent time for improvement by further incoming orders. As we have seen in detail in Section 5, the performance depends on the load by an inverse power law.

We assume a Gaussian distribution of $d_r^{\text{O}} \sim \mathcal{N}(\phi(\lambda), \sigma_\phi^2(\lambda))$ and set the departure time of a route to

$$\tau_r(d_r^{\text{O}}, \lambda) = \tau_r^{\text{earliest}} + (\tau_r^{\text{latest}} - \tau_r^{\text{earliest}}) \cdot \Phi\left(\frac{d_r^{\text{O}} - \phi(\lambda)}{\sigma_\phi(\lambda)}\right), \tag{9}$$

where Φ is the cumulative normal distribution function. For example, when d_r^{O} corresponds exactly to the expected mean order delivery time $\phi(\lambda)$ in the given load situation, the departure time of r will be set to $(\tau_r^{\text{earliest}} + \tau_r^{\text{latest}})/2$, the middle of the route departure time slack. We estimate $\sigma_\phi(\lambda)$ by calculating sample standard deviations from our experiments described in the previous section.

We will refer to the three different strategies as τ-earliest, τ-latest, and τ-route.

7 Double Horizon Approach

This approach adopts from Mitrović-Minić et al. [8] the idea of considering in the optimization two planning horizons simultaneously, a short horizon and a large horizon. In the *Large Horizon Planning* (LHP), which we always perform as first step, we consider a strongly simplified approximate problem variant of DYN-$\tilde{t}$ where, in addition to all available requests, also all the expected future requests for either the whole day or at least several hours into the future. The primary goal is to make a rough plan on the utilization of the vehicles and recognize times where we might exceed the available capacity or have enough time to finish vehicle shifts earlier. A detailed routing is *not* done in the LHP. The short horizon problem corresponds to our definition of DYN-$\tilde{t}$ so far but utilizes an adapted objective function that includes additional terms defined by the LHP's results in order to meet the long-term goals as closely as possible.

In our case decisions on the labor time to be used beyond the minimum q_u^0 for each vehicle are most critical in the long-term in order to avoid later deliveries becoming tardy due to insufficient driving resources for the given workload.

We therefore define and solve the following LHP subproblem at time $\tilde{t}$ in order to derive *target shift end times* $\tilde{q}_u$ for each vehicle $u \in U$. We consider as V all currently relevant orders of the current DYN-$\tilde{t}$ plus expected orders V^{exp} for the remaining day. These expected orders are artificially created according to the estimated numbers of orders becoming available per hour $\hat{\omega}(t)$, equidistantly spaced over each hour. For each of these orders we further derive a due time randomly based on the distribution of expected order types and their promised maximum delivery times.

Let $z\colon \mathbb{R}^+ \to \mathbb{R}^+$ be a function that estimates the average shift duration needed to serve one order $v \in V \cup V^{\text{exp}}$ within the current hour of $\tilde{t}$ and a few subsequent hours, assuming a reasonable routing and an average number of available orders. The basis for z is the mean order delivery time $\phi(\lambda)$ derived from the routes with latest departure time strategy as presented in Section 5. To account for a slight increase due to waiting times in the depot and an intermediate departure time strategy, we introduce an additional factor $\zeta \gtrapprox 1$. With $\mathit{\Lambda}(t)$ being the load at hour t, we then calculate a weighted average to estimate the average shift duration

$$z(\tilde{t}) = \zeta \cdot \frac{\sum_{t'=\tilde{t}}^{\tilde{t}+\rho} \mathit{\Lambda}(t') \cdot \phi(\mathit{\Lambda}(t'))}{\sum_{t'=\tilde{t}}^{\tilde{t}+\rho} \mathit{\Lambda}(t')}, \tag{10}$$

with ρ corresponding to three hours in our implementation. We make the strongly simplifying assumption that any order v can be independently served by any available vehicle within time $z(\tilde{t})$ from t_v^{rel} onward. Each vehicle's shift is split into successive time slots of duration $z(\tilde{t})$, and in each of these time slots one order can be served. This implies that we do not allow arbitrary start times to serve orders but only times that are multiples of $z(\tilde{t})$ away from a vehicles' shift start time (or $\tilde{t}$). We do not have a strict last slot, i.e., in principle further orders to be served might always be appended to a vehicles shift. An instructive visualization of the LHP's view on an example DYN-$\tilde{t}$ instance is provided in Figure 4.

A solution to our LHP is a complete assignment of all the orders $V \cup V^{\text{exp}}$ to vehicle slots. As actual delivery time of an order we consider the respective time slot's middle point, i.e., the time slot's start time plus $z(\tilde{t})/2$. The objective function corresponds to our main objective function (4), but as we do not consider routing the last travel time term is omitted.

This LHP is heuristically solved by a greedy assignment procedure, in which orders are assigned in increasing due-time always to the earliest feasible time slot of a vehicle that increases the objective the least. In case of ties, a vehicle $u \in U$ whose end of the shift exceeds q_u^0 the least, i.e., where the vehicle's excess labor time is smallest, is chosen. This aspect automatically balances the deviations from the planned shift times among the vehicles if

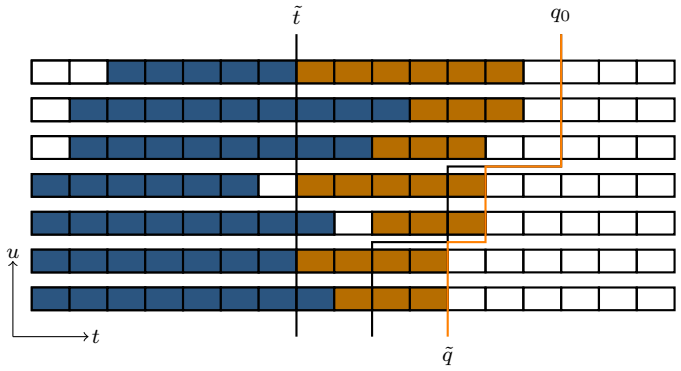


Fig. 4 View on an example DYN-$\tilde{t}$ problem instance as seen by the large horizon planning. The x-axis represents the time, discretized by time slots of the average expected shift time needed to deliver an order $z(\tilde{t})$. The drivers that are still available at or after $\tilde{t}$ are stacked on the y-axis. Blue rectangles indicate orders that have already been delivered or are en route. Brown rectangles represent greedily assigned orders, either real (available at the moment) or expected up until the end of the planning horizon. The maximum of the earliest shift end time q_u^0 and the latest assigned order define for each driver the target shift end time $\tilde{q}_u$. For the last four drivers q_0 is exceeded, since otherwise tardiness would have arisen. Note that unassigned slots may occur if no more orders are ready for delivery at that time.

there are no particular other reasons such as avoiding tardy orders. Further ties are resolved randomly by a random processing order of the vehicles.

The obtained shift end times of this solution, i.e., the end times of the last used time slots of each vehicle, are finally used as target shift end times $\tilde{q}_u$, for all $u \in U$ in the short horizon optimization, i.e., the ALNS from Section 4.

This is achieved by augmenting objective function (4) to

$$\tilde{f}(\langle R, \tau, q \rangle)) = f(\langle R, \tau, q \rangle) + \gamma \cdot \sum_{u \in U} Q_u \tag{11}$$

with

$$Q_u = -\min(q_u - q_u^0, \tilde{q}_u - q_u^0). \tag{12}$$

This non-positive term can be seen as bonus that exactly compensates any arising labor time costs above q_u^0 up to the target time $\tilde{q}_u$ for each vehicle $u \in U$. Thus, the time up to $\tilde{q}_u$ can be used "for free". Note that the factor γ by which the bonus is multiplied is the same as by which the labor time is weighted in (4).

8 Shift Ending Strategies

In the online problem, we also have to decide if a shift should be ended by sending a driver (vehicle) $u \in U$ home, providing this is allowed, i.e., $\tilde{t} \geq q_u^0$, u is in the depot, and no more routes are planned for u, or if the driver has to wait at the depot to possibly receive further orders. Again, two naive strategies

are immediately available: The first option is to send a vehicle u home as early as possible, i.e., after its last so far planned route or at q_u^0, whichever comes later. This is also the default of the insertion heuristic. The other extreme is the latest strategy that waits until q_u^{end} in any case, even if the last route ends before q_u^{end}. The earliest strategy seems to be an attractive choice, since we can save labor costs and during peak hours, it is likely that a vehicle has already a next route planned during its current route, therefore it is not sent home prematurely when arriving at the depot, if there is still enough work to do.

A more sophisticated approach makes use of the estimated shift end times $\tilde{q}$ provided by the LHP. The earliest shift end time is then modified to be $\tilde{q}_u - \tilde{d}$, where $\tilde{d}$ is a threshold duration of an efficient route. The rationale is that if a vehicle cannot start a somewhat efficient route that ends before its target shift end $\tilde{q}_u$, it is better to send it home.

We will refer to the three different strategies as q-earliest, q-latest, and q-LHP.

9 Computational Study

We conducted all our experiments on Intel Xeon E5-2640 processors with 2.40 GHz in single-threaded mode and a memory limit of 8 GB. We implemented our approach as a prototype in Python 3.7, being aware that an implementation in a compiled language would be substantially faster and have a smaller memory footprint. We consider six different instance classes, each with 20 instances: Artificial instances[1] on the Euclidean unit disk as described in Section 5 using either a business day (BD) or a weekend (WE) load profile with generous (GE), tight shift planning (TI), and with a shortage (SH) of drivers. The idea is to observe the transition from a more generous shift planning to a tighter one and simulating a driver being absent on short notice where in the latter cases more tardiness is expected to occur. Furthermore, in the generous case, dynamically ending shifts earlier is expected to have more impact where in the tight case shifts are more likely to be extended by starting long routes shortly before the ending.

We aim at comparing the performance of the naive earliest and latest strategies with the more sophisticated LHP and driver performance based route departure strategy on those DYN problem instances. In each case, we apply the ALNS with a limit of 1000 non-improving iterations and additionally a 60 seconds time limit for route optimization at each arriving order. This should be consistent with a real-time setting, where orders may arrive every minute during peak-time or on weekends and routes already including them should occasionally be started within a minute. Without LHP and driver performance estimation, we are restricted to naive earliest and latest strategies regarding the departure time of a route and the early shift termination. LHP extrapolates until the end of the horizon to set desired shift ending times for

[1] https://github.com/nfrohner/pdsvrpddsf

Algorithm 1: Simulated DYN Problem Solver with ALNS and LHP.

Input: Orders V, drivers U, shift starts q^{start}, earliest shift ends q^0, planned shift ends q^{end}, hourly expected number of orders $\hat{\omega}$, travel time matrix δ, mean order delivery time ϕ, efficient route threshold duration $\tilde{d}$.

Output: Solution $\langle R, \tau, q \rangle$ with routes R, route departure times τ, and actual shift end times q for the whole day.

$V^{\text{deliv}} \leftarrow \{\}$;
$U^{\text{home}} \leftarrow \{\}$;
$R' \leftarrow ()_{u \in U}, \tau' \leftarrow ()_{r \in R'}, \tilde{t}' \leftarrow 0, \tilde{q} \leftarrow q^{\text{end}}$;
$\langle R, \tau, q \rangle \leftarrow \langle R', \tau', q^0 \rangle$;
foreach $\tilde{t} \in \{t \mid \exists v \in V : t = t_v^{\text{rel}}\} \cup \{\infty\}$ **do**
  foreach $(u, r) \in R' : \tilde{t}' \leq \tau'_r < \tilde{t}$ **do**
    $V^{\text{deliv}} \leftarrow V^{\text{deliv}} \cup \{v_1^r, \ldots, v_{l_r}^r\}$;
    $\langle R, \tau, q \rangle \leftarrow \langle R, \tau, q \rangle \oplus (r, \tau'_r)$;
    $q_u^{\text{start}} \leftarrow u$'s return time at depot after $\tilde{t}$;
  end
  $U^{\text{home}}, q \leftarrow \text{SENDHOME}(\tilde{t}, \tilde{t}', U^{\text{home}}, U \setminus U^{\text{home}}, q^{\text{start}}, q^0, q^{\text{end}}, \tilde{q}, \tilde{d})$;
  $V^{\text{avl}} \leftarrow \{v \in V \setminus V^{\text{deliv}} : t_v^{\text{avl}} \leq \tilde{t}\}$;
  $\tilde{q} \leftarrow \text{LHP}(\tilde{t}, V^{\text{avl}}, U \setminus U^{\text{home}}, \hat{\omega}, \phi, q^{\text{start}}, q^0, q^{\text{end}})$;
  $\langle R', \tau', q \rangle \leftarrow \text{ALNS}(\tilde{t}, V^{\text{avl}}, U \setminus U^{\text{home}}, q^{\text{start}}, q^0, q^{\text{end}}, \tilde{q}, \delta)$;
  $\tau' \leftarrow \text{DEPART}(R', \phi)$;
  $\tilde{t}' \leftarrow \tilde{t}$;
end
return $\langle R, \tau, q \rangle$;

each driver, using an estimation of the average driver performance in the window of the current and the upcoming three hours. The target shift ending times may be before the planned shift ends to send drivers home early or after them so that extending shifts is favored via the augmented objective function.

In Algorithm 1 we list a high-level pseudo-code of the simulated DYN problem solver, combining the previously explained approaches based on the LHP and route performance. The main loop goes over all times $\tilde{t}$ where an order is released, where the first inner loop checks whether routes have been started between the last and the current $\tilde{t}$. If so, they are added to the current solution, the corresponding orders are removed, and the drivers' shift starts are set to their return times at the depot. Afterwards, drivers are sent home, if their target shift end time reduced by the efficient route threshold $\tilde{q} - \tilde{d}$ passed and they have no further routes planned. Then the route construction and optimization begins with the large horizon planning to update the $\tilde{q}$. It further continues with the ALNS—the optimization workhorse—that creates routes for the currently available and not yet delivered orders. Finally, the departure time of the planned routes is set according to the route performance strategy (more efficient routes are planned to start earlier).

In Table 1, we present the main results of our computational study. We compare for the different combinations of our approaches means and standard deviations of the number of tardy orders n^{tardy}, the root mean squared error

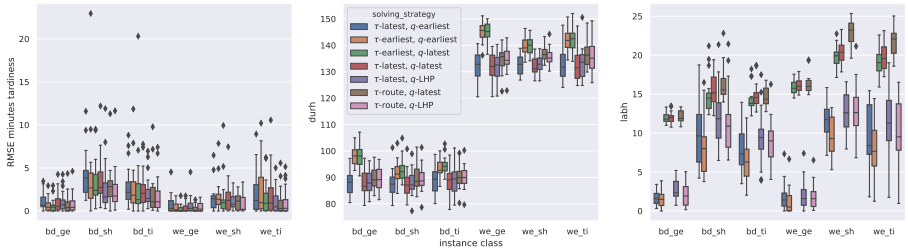


Fig. 5 Comparison of the root mean square error of the tardiness in minutes, the travel time duration, and the excess labor time of different solution strategies (without offline solution) on six different instances classes with 20 instances each. We observe that the more sophisticated strategies based on LHP and the route performance decreases the tardiness at the cost of carefully introducing additional travel time (regarding which τ-latest is best) and labor time (where q-earliest is best).

(RMSE) of the tardiness in minutes, the total travel time in hours, the labor time exceeding q_u^0 in hours, the average route duration $\bar{d}$, and the average route performance (labor time to serve an order without waiting time) in minutes d_r^{O}. The offline (full knowledge of the day) results (OFF) where we applied ALNS to convergence with a limit of 1000 non-improving iterations without additional time limit provide a performance baseline. All the other results are for the DYN-DAY problem variant and we see that the offline baseline is somewhat out of reach, which is not too surprising due to the substantially restricted knowledge that can be exploited in the online variant. Despite having used a lexicographic optimization approach, where distributing tardiness evenly and reducing it was the single most important objective, we analyze the results in the sense of a multi-objective optimization problem. Small amounts of tardiness for a few customers may in practice be acceptable when real costs may be substantially reduced. In Figure 5, we visualize the results by boxplots of the tardiness, travel time, and labor time, for the different solution strategies (excluding the offline problem) on the six different instance classes.

We observe that the τ-latest strategy provides the best route performances and therefore smallest travel costs but sometimes runs into troubles regarding tardiness, where a τ-earliest strategy would have been beneficial. Similarly, the q-latest strategy provides the most shift time resources allowing to reduce tardiness, as opposed to the q-earliest strategy. The goal of τ-route is to balance between the extremes of the τ determination strategies considering the load development of the day. Likewise, the q-LHP strategy should provide additional shift resources regarding the flexibility of shift endings times only when necessary. We observe that the τ-route strategy sacrifices a slight amount of route quality in exchange for substantially less tardiness. Likewise, the LHP carefully provides additional labor time to be used to reduce tardiness. Combining both strategies results in a reasonable trade-off over all the instances classes, where a decision maker may also select a suitable combination of strategies given the load and shift structure of the day.

Table 1 Offline problem performance (OFF) and different solution strategies applied to 20 artificial instances for each configuration using either a business day (BD) or a weekend (WE) load profile with either generous, tight, or shift planning with a driver shortage.

			n^{tardy}		RMSE [min]		dur [h]		lab [h]		$\bar{d}$ [min]		d_r^O [min]	
			mean	std	mean	std	mean	std	mean	std	mean	std	mean	std
load	shift	solving strategy												
BD	generous	τ-earliest, q-earliest	4.500	4.536	0.680	0.798	98.134	3.580	1.491	1.069	56.425	4.315	18.386	0.692
		τ-earliest, q-latest	3.850	4.771	0.592	0.795	98.087	3.984	11.940	0.698	55.930	3.707	18.372	0.631
		τ-latest, q-LHP	6.250	6.257	1.107	1.208	88.221	4.447	2.685	1.199	74.935	2.832	16.512	0.426
		τ-latest, q-earliest	7.600	6.065	1.157	1.091	87.716	4.443	1.673	0.889	74.745	2.818	16.420	0.479
		τ-latest, q-latest	5.900	5.937	1.028	1.237	87.980	5.210	11.974	0.722	75.460	2.792	16.463	0.574
		τ-route, q-LHP	4.400	4.604	0.841	1.134	89.369	4.568	2.197	1.560	71.340	2.113	16.728	0.483
		τ-route, q-latest	4.350	4.782	0.757	1.082	89.956	4.241	12.036	0.797	70.060	2.398	16.839	0.379
		OFF	0.550	1.572	0.056	0.174	78.897	4.763	0.377	0.479	64.170	2.039	14.760	0.384
	shortage	τ-earliest, q-earliest	19.950	16.804	3.991	4.966	91.863	4.294	8.334	3.843	63.415	4.985	17.259	0.468
		τ-earliest, q-latest	19.100	13.726	3.381	2.568	92.763	4.305	14.775	2.426	62.930	4.941	17.430	0.533
		τ-latest, q-LHP	14.800	10.928	2.705	2.915	88.099	4.742	12.262	3.854	74.640	3.749	16.544	0.350
		τ-latest, q-earliest	26.200	14.207	4.179	2.747	87.507	5.044	9.946	4.296	74.355	4.007	16.430	0.370
		τ-latest, q-latest	22.737	16.562	3.551	2.820	87.524	5.038	15.406	2.073	72.126	3.019	16.505	0.426
		τ-route, q-LHP	15.450	11.019	2.546	2.525	89.483	4.717	11.454	4.103	71.270	3.763	16.806	0.388
		τ-route, q-latest	18.150	11.864	2.988	2.338	89.992	4.691	16.304	2.145	70.045	3.622	16.903	0.375
		OFF	2.050	5.826	0.255	0.721	81.655	4.358	4.671	3.230	57.595	1.676	15.336	0.400
	tight	τ-earliest, q-earliest	14.158	10.673	2.720	2.928	93.582	3.530	6.319	2.803	62.416	4.261	17.507	0.746
		τ-earliest, q-latest	10.950	8.003	2.847	4.480	94.496	3.263	14.236	1.601	61.630	3.886	17.670	0.725
		τ-latest, q-LHP	14.300	11.855	2.139	1.923	89.265	5.296	9.496	3.458	73.180	2.846	16.667	0.477
		τ-latest, q-earliest	17.150	9.544	2.959	2.341	88.868	5.074	7.920	2.934	72.865	2.211	16.593	0.457
		τ-latest, q-latest	13.900	9.754	2.466	2.179	88.732	4.822	14.810	1.636	70.945	2.330	16.571	0.514
		τ-route, q-LHP	8.950	6.117	1.843	2.098	90.030	4.238	8.883	2.960	70.370	2.580	16.818	0.452
		τ-route, q-latest	10.105	7.880	2.028	2.542	90.265	4.403	14.728	1.231	68.816	1.937	16.870	0.509
		OFF	0.700	2.904	0.115	0.512	82.371	5.380	3.273	2.645	58.400	2.307	15.374	0.420
WE	generous	τ-earliest, q-earliest	2.400	3.267	0.420	0.539	145.071	3.794	1.213	1.660	62.140	3.102	17.008	0.504
		τ-earliest, q-latest	1.750	2.900	0.273	0.435	145.130	3.561	15.889	0.853	61.985	3.286	17.021	0.634
		τ-latest, q-LHP	2.850	2.661	0.701	1.056	132.071	4.851	2.024	1.885	76.835	2.059	15.477	0.343
		τ-latest, q-earliest	2.850	2.978	0.745	1.112	132.159	5.094	1.715	1.753	76.580	2.008	15.486	0.334
		τ-latest, q-latest	1.600	2.137	0.358	0.531	132.161	4.962	16.099	0.859	76.015	1.727	15.488	0.335
		τ-route, q-LHP	2.450	2.837	0.455	0.551	134.841	4.713	1.820	1.657	73.420	2.295	15.800	0.329
		τ-route, q-latest	1.650	2.641	0.271	0.411	134.682	4.460	16.286	1.248	73.945	1.934	15.787	0.435
		OFF	0.000	0.000	0.000	0.000	118.698	5.720	0.197	0.407	67.455	2.253	13.902	0.291
	shortage	τ-earliest, q-earliest	12.950	9.801	1.743	1.817	139.469	3.321	9.636	2.564	67.160	3.392	16.167	0.597
		τ-earliest, q-latest	7.400	5.623	1.475	2.305	139.888	3.335	19.844	1.440	66.135	2.164	16.215	0.571
		τ-latest, q-LHP	9.900	7.440	1.077	0.923	132.993	3.421	12.833	2.534	77.185	1.991	15.411	0.350
		τ-latest, q-earliest	14.250	9.107	1.527	1.578	132.510	3.574	11.427	2.400	75.295	1.975	15.354	0.380
		τ-latest, q-latest	11.700	9.985	1.632	1.878	132.087	3.426	20.397	1.485	75.850	2.883	15.305	0.320
		τ-route, q-LHP	8.100	8.534	0.924	1.048	135.573	3.432	12.724	3.585	74.550	1.673	15.711	0.413
		τ-route, q-latest	7.400	8.068	1.042	0.898	136.989	3.011	22.970	1.615	73.765	1.808	15.875	0.371
		OFF	0.350	0.988	0.032	0.130	123.384	3.617	4.860	2.077	61.880	1.426	14.296	0.330
	tight	τ-earliest, q-earliest	12.200	16.938	2.372	3.017	141.820	3.805	8.029	3.336	64.295	3.578	16.608	0.526
		τ-earliest, q-latest	8.350	7.707	1.558	1.924	142.144	4.282	19.164	1.851	64.055	3.246	16.643	0.528
		τ-latest, q-LHP	7.050	8.587	1.121	1.442	133.109	5.952	11.178	4.118	76.335	2.057	15.575	0.261
		τ-latest, q-earliest	10.850	10.246	2.109	2.531	132.394	5.840	9.369	3.762	75.115	1.923	15.490	0.285
		τ-latest, q-latest	8.500	8.237	1.955	2.667	132.156	5.971	19.720	1.794	73.955	2.546	15.462	0.311
		τ-route, q-LHP	6.050	6.739	1.113	1.610	135.414	5.522	10.216	4.080	73.805	1.910	15.848	0.365
		τ-route, q-latest	5.050	6.700	1.021	1.536	136.518	4.566	21.582	1.895	73.285	2.089	15.980	0.347
		OFF	0.800	2.118	0.079	0.189	122.279	6.193	3.716	2.530	62.540	1.653	14.303	0.285

For τ-latest, q-LHP we use $\zeta = 1.2$ to convert the route performance values to the average time to serve an order as described in Section 7, and for τ-route, q-LHP $\zeta = 1.15$. This is only a naive transformation rule. Further research is needed regarding the driver performance estimation, especially since the waiting time, the route departure strategy, and the driver performance have

an immanent cyclic dependency. The parameters used for the three-parameter inverse power law model to estimate the route performance $\phi(\lambda)$ given the load λ are tuned by means of least squares optimization to $k_l = 23.144, k_m = 3.951, \alpha = 0.174$, with a 25% estimated constant relative standard deviation. For the driver send home strategy, $\tilde{d}$ is set to 55 minutes.

10 Conclusions

We considered a purely dynamic and stochastic vehicle routing problem with delivery deadlines and shift flexibility arising from a real-world application. Orders arrive at an online store throughout the day, regarding which we have stochastic knowledge by means of hourly estimates. They have to be delivered within only a few hours by drivers deployed at a single depot. The goal is to reduce or evenly spread tardiness if not avoidable among the customers and to minimize travel and labor costs. Drivers may be sent home early or have their shifts extended to some degree to account for the uncertainty of the load.

Our proposed double-horizon approach is able to effectively address the dynamism and stochasticity of the problem. The large horizon planning with its simplified order-to-drivers assignment is able to derive meaningful target shift end times. These are exploited in the short horizon planning—the actual routing performed by ALNS—by augmenting its objective function, releasing additional shift resources in an informed way.

Another important aspect is to determine the route departure times, where neither the naive earliest nor the latest strategy suffices. We devised a more balanced strategy that estimates the expected route performance (average time per order in a route) depending on the current load and start routes earlier that are already close to the desired performance and later when the performance is worse.

The combination of both approaches leads to superior performance over the naive strategies or allows for trade-offs regarding substantially reduced travel and labor time versus a slight amount of more tardiness on artificially created instances for different load profiles (business day vs weekend) and shift plans (generous vs tight vs shortage).

Further research is needed to improve the estimation of the driver performance over the day, especially for real delivery areas in a city, also studying the impact of traffic. This is also a basis for the accuracy of the LHP. A difficulty lies in the cyclic dependency between the driver performance estimation and the route departure strategy, where we used a bootstrapping mechanism by fitting a function on idealized randomized instances, incorporating the driver waiting times by a constant factor over the whole day to pragmatically resolve this in a first step.

Acknowledgements We thank Daniel Obszelka for numerous fruitful discussions and the implementation and tuning of the destroy operators. We thank the anonymous reviewers for their valuable feedback resulting in several improvements of this work.

References

1. Azi, N., Gendreau, M., Potvin, J.Y.: An adaptive large neighborhood search for a vehicle routing problem with multiple routes. Computers and Operations Research **41**(1), 167–173 (2014)
2. Beardwood, J., Halton, J.H., Hammersley, J.M.: The shortest path through many points. In: Mathematical Proceedings of the Cambridge Philosophical Society, vol. 55, pp. 299–327. Cambridge University Press (1959)
3. Bent, R.W., Van Hentenryck, P.: Scenario-based planning for partially dynamic vehicle routing with stochastic customers. Operations Research **52**(6), 977–987 (2004)
4. Daganzo, C.F.: The distance traveled to visit n points with a maximum of c stops per vehicle: An analytic model and an application. Transportation science **18**(4), 331–350 (1984)
5. Figliozzi, M.A.: Planning approximations to the average length of vehicle routing problems with varying customer demands and routing constraints. Transportation Research Record **2089**(1), 1–8 (2008)
6. Gendreau, M., Guertin, F., Potvin, J.Y., Taillard, É.: Parallel tabu search for real-time vehicle routing and dispatching. Transportation Science **33**(4), 381–390 (2008)
7. Hvattum, L.M., Løkketangen, A., Laporte, G.: Solving a dynamic and stochastic vehicle routing problem with a sample scenario hedging heuristic. Transportation Science **40**(4), 421–438 (2006)
8. Mitrović-Minić, S., Krishnamurti, R., Laporte, G.: Double-horizon based heuristics for the dynamic pickup and delivery problem with time windows. Transportation Research Part B: Methodological **38**(8), 669–685 (2004)
9. Pillac, V., Gendreau, M., Guéret, C., Medaglia, A.L.: A review of dynamic vehicle routing problems. European Journal of Operational Research **225**(1), 1–11 (2013)
10. Pisinger, D., Ropke, S.: A general heuristic for node routing problems. Computers & Operations Research **34**, 2403–2435 (2007)
11. Psaraftis, H.N., Wen, M., Kontovas, C.A.: Dynamic vehicle routing problems: Three decades and counting. Networks **67**(1), 3–31 (2016)
12. Ritzinger, U., Puchinger, J.: Hybrid metaheuristics for dynamic and stochastic vehicle routing. In: E.G. Talbi (ed.) Hybrid Metaheuristics, *SCI*, vol. 434, pp. 77–95. Springer (2013)
13. Ropke, S., Pisinger, D.: An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation Science **40**(4), 455–472 (2006)
14. Schilde, M., Doerner, K.F., Hartl, R.F.: Metaheuristics for the dynamic stochastic dial-a-ride problem with expected return transports. Computers and Operations Research **38**(12), 1719–1730 (2011)
15. Shaw, P.: A new local search algorithm providing high quality solutions to vehicle routing problems. Tech. rep., APES Group, Dept. of Computer Science, University of Strathclyde, Glasgow, UK (1997)
16. Yang, Z., van Osta, J.P., van Veen, B., van Krevelen, R., van Klaveren, R., Stam, A., Kok, J., Bäck, T., Emmerich, M.: Dynamic vehicle routing with time windows in theory and practice. Natural Computing **16**(1), 119–134 (2017)

An LP-based heuristic for Inspector Scheduling

Gerwin Gamrath · Markus Reuther · Thomas Schlechte · Elmar Swarat

Received: date / Accepted: date

Abstract We present a heuristic based on linear programming (LP) for the integrated tour and crew roster planning of toll enforcement inspectors. Their task is to enforce the proper paying of a distance-based toll on German motorways. This leads to an integrated tour planning and duty rostering problem; it is called Toll Enforcement Problem (TEP). We tackle the TEP by a standard multi-commodity flow model with some extensions in order to incorporate the control tours.

The heuristic consists of two variants. The first, called Price & Branch, is a column generation approach to solve the model's LP relaxation by pricing tour and roster arc variables. Then, we compute an integer feasible solution by restricting to all variables that were priced. The second is a coarse-to-fine approach. Its basic idea is projecting variables to an aggregated variable space, which is much smaller. The aim is to spend as much algorithmic effort in this coarse model as possible. For both heuristic procedures we will show that feasible solutions of high quality can be computed even for large scale industrial instances.

Keywords Crew Rostering · Column Generation · Heuristics

Mathematics Subject Classification (2010) 90B20 · 90C06

Gerwin Gamrath · Elmar Swarat
Zuse Institute Berlin, Takustr. 7, D-14195 Berlin
Tel.: +49-30-84185244
Fax: +49-30-84185269
E-mail: swarat@zib.de

Markus Reuther · Thomas Schlechte
LBW Optimization GmbH, Englerallee 19, 14195 Berlin

1 Introduction

Toll systems are an active research area, especially the aspect of toll road pricing, partly in consideration of congestion on roads. When designing toll systems a common approach is to utilize bi-level programming models as presented in [1] or [2]. Toll pricing to reduce traffic congestion is studied by [3] or more recently by [4].

We consider a rather neglected operational aspect of toll systems, namely the enforcement of the toll. But planning of control resources and in particular the rostering of employees, that conduct the enforcement, is not limited to the case of a toll. It is an important challenge in many real-world applications, e.g., police inspections, ticket inspections or other security related tasks. Here, we focus on the enforcement of the truck toll on German motorways and main roads, a network of around 50k kilometers. All trucks weighting at least 7.5 tonnes must pay a distance based toll. Fares differ according to the vehicle weight, the number of axis and the emission class. Introduced in 2005 on motorways and extended in 2018 to all main roads, it makes one of the most important contribution to the budget of public roads maintenance. Hence, it is needed to organize the limited control resource as effectively as possible.

Since the system is barrier-free, toll evasion is basically possible. The German Federal Office for Goods Transport (BAG) is responsible for the enforcement of the toll. It is conducted by a combination of traffic control gantries or devices for automatic stationary camera control and spot-checks by more than 400 mobile control inspectors. The spot checks are carried out as part of control tours by approximately 250 *control teams* of one or two inspectors. Due to practical aspects the teams are divided into more than 20 regions.

In an on-going research and development project with the BAG we develop methods and a software tool to compute optimal control tours and crew rosters of the inspectors. We have called this integrated tour planning and duty rostering problem the *Toll Enforcement Problem (TEP)*. At BAG the planning is organized as monthly planning problem. Two or four sections of the network are controlled during a mobile tour each for a fixed time interval of approximately 2 or 4 hours.

In the TEP a duty corresponds to a control tour starting at a certain time in some depot. After some hours the tour ends in the same depot. The tours are not given in advance, they have to be created by *tasks* consisting of controlling a certain *section* of a motorway (or more precisely a subarea of the toll network) in a corresponding time interval. A tour is a combination of such tasks. The main difference to standard rostering lies in the integrated optimization of tours and rosters. This integration is necessary because it is unclear what work has to be done beforehand and crews can only conduct controls in the area of their home depots. Therefore, it is important to prevent the planning of tours for which no crew is available.

The TEP was presented inter alia in [5] where an extensive description of the modeling power of the approach is given, including an analysis of the bi-criterion character of the TEP and computational results that analyze the

complexity of real-world instances. In the literature there are different approaches to solve inspection problems that are similar to our application. The authors of [8] consider the problem of fare evasion. Optimal control strategies are derived by game theory. While our focus is rostering, another recent report [9] considers the generation of duties for railway security teams by concatenating inspection tasks on trains or platforms.

In this paper, we propose a column generation approach to solve the TEP. In a standard approach we solve the LP relaxation of the TEP IP by an arc generation method. After the root node LP is solved to optimality the restricted IP is solved with the set of variables generated during column generation in the root node. In fact, we develop a heuristic since feasibility is not guaranteed.

Our main contribution - that can be seen as a modification of the standard approach - is the adoption of a method called *Coarse-to-Fine*. This method was first used by the authors of [6]. They introduced a coarsening approach to the Railway Rolling Stock Rotation Problem. It is also presented in [7]. The basic idea is to introduce a coarsening projection on the variables of the original model. It leads to a coarser model with significantly less variables. Then the problem is solved via column generation on the coarse level. We will show that the coarse reduced costs overestimate the fine reduced costs. Afterwards, again the IP is solved with the columns generated in the root node. The major benefit in comparison to other heuristic approaches is that we are able to provide a quality measure.

The paper is structured as follows: In Section 2 we shortly define the TEP and recapitulate a corresponding formulation. Thereafter, in Sections 3, 4 and 5 the Coarse-to-Fine approach is presented. Finally, Section 6 discusses computational results for both approaches and Section 7 concludes the paper.

2 A Multicommodity Flow Formulation

The TEP is formulated by a standard multi-commodity flow model with some extensions in order to incorporate the control tours. It is based on a scheduling graph $\mathcal{G} = (D, \mathcal{A})$). There, the set D corresponds to duties for the inspectors plus artificial source and sink nodes. If two duties $u \in D$ and $v \in D$ can be performed successively by the same inspector we construct an arc $(u, v) \in A$ in $\mathcal{G}$. Additionally, there are arcs $(\hat{s}, v) \in A$ leaving the source node $\hat{s} \in D$ and arcs $(u, \hat{t})$ entering the sink node $\hat{t} \in D$, respectively. Hence, a feasible duty roster corresponds to a $\hat{s}$-$\hat{t}$-path in $\mathcal{G}$. The model uses binary variables z_d to decide if a control tour $d \in D$ is chosen. In addition, there are binary flow variables x_a^m that are one if arc $a \in A$ is used by inspector $m \in M$.

Typical constraints for the tour variables are section covering and section capacity constraints. The first guarantee each section to be visited and the latter that not more than one control team conducts a control on a section during the same time. For the rostering part classical flow value and flow conservation constraints are combined with different types of resource constraints.

Coupling constraints link the tour variables with the roster graph. For more details we refer to our earlier work [5].

The overall goal is to compute a reward-maximal set of control tours. The reward of a section depends both on its amount of traffic depending on the time and the day of the week and on the quota of fare evaders in the past. In addition, costs on the sequence arcs represent penalties for soft rule violations. With this model a lot of rules can be modeled implicitly in the scheduling graph. A typical example are minimum rest times. There exist also other rules, e.g., monthly working hours, that are modeled as resource constraints.

A major task is to find a compromise between quality and quantity of controls as well as providing fair roster schedules for the inspectors such that the acceptance of the optimized schedules can be increased. The modeling power of Mixed Integer Programming and the ability of rapid model modifications in order to cope with moving targets have been an important instrument to solve real-world instances. For practical reasons the tours consist of two parts only (for an efficient control it is required to stay for some time in an area). Therefore, the main complexity of the model stems from the rostering part. The incorporation of the tours can be seen as an extension of the classical rostering flow model. According to the requirement that inspectors can only conduct controls in their regional area, the number of tours and roster sequence variables is still in a range that allows a complete tour generation.

Since the beginning of 2014, the algorithm and software based on this model and the commercial MIP solver CPLEX is in production to schedule all toll enforcement inspectors of BAG in Germany. In contrast to many other rostering problems, model TEP is directly solvable by a general-purpose solver such as CPLEX if the number of arcs and tours does not exceed, say, 2 million. But in some cases the resulting MIP formulation becomes too large and intractable or slow even for commercial MIP solvers. In addition the vast majority of arcs and tours will not be part of a feasible solution. This motivated us to develop a heuristic approach and transfer the idea of dynamic variable generation methods and Coarse-to-Fine to this application.

3 Applying an LP-based heuristic

The main contribution is a heuristic based on Linear Programming (LP) to solve the integrated problem. An important precondition is the fact that the reduced costs can easily be computed in the TEP, since all tours and duty sequence arcs are generated in advance. The heuristic uses a *Coarse-to-Fine* approach to solve the LP-relaxation of the TEP by column generation. The basic idea of Coarse-to-Fine is to identify unprofitable clusters of variables and to focus on the parts that promise an improvement of the objective value. In this concept our original model is called the *fine model*. We introduce a coarsening projection for the variables of the fine model. Several variables from the fine model are mapped to a single variable in the *coarse model*.

The general definition is as follows: We are given a linear program

$$\max p^T x, s.t. Ax \leq b, x \geq 0$$

with J as the index set of the variables. We introduce a coarsening projection

$$[\cdot] : J \to [J]$$

that maps J onto a smaller index set $[J]$, where some variables are aggregated. Therefore, the coarse model has significantly fewer variables than the fine model. Furthermore, for a matrix A let A_{ij} be the entry in the i-th row and in the j-th column. We introduce the coarse matrix $[A] \in \mathbb{R}^{I \times [J]}$ by defining for each $j \in [J]$ the coarse column vector $[A]_{\cdot j}$ as follows:

$$\begin{aligned}[A]_{ij} :=& ([A]_{ij1}, [A]_{ij2}) \\ :=& (\min \{A_{ik} \,|\, k \in J \,:\, [k] = j\}, \max \{A_{il} \,|\, l \in J \,:\, [l] = j\}).\end{aligned}$$

The coarse objective coefficients are defined by $[p]_j := \max_{k \in J}\{p_k \mid [k] = j\}$ for all $j \in [J]$. A similar approach where rows were aggregated was introduced to the Railway Rolling Stock Rotation Problem in [6].

The trick is now to solve the fine problem by a column generation algorithm that operates mainly on the coarse level. To price the variables in the coarse model, a proper definition of the *coarse reduced cost* is necessary. Let $\alpha \in \mathbb{R}^I$ be an optimal dual solution for the restricted master problem in the fine layer. Then let us define how to multiply such a vector with a coarse matrix:

$$\hat{x} \in \mathbb{R}^a, \hat{y} \in (\mathbb{R}, \mathbb{R})^a, a \in \mathbb{N}, a \geq 1$$

that

$$\hat{x}^T * \hat{y} := \sum_{i=1}^{a} \min \{\hat{x}_i \hat{y}_{i1}, \hat{x}_i \hat{y}_{i2}\}$$

where $\hat{y}_i = (\hat{y}_{i1}, \hat{y}_{i2})$. Then we can define the coarse reduced cost as:

$$[\tau]_j := [p]_j - \alpha^T * [A]_{\cdot j}, j \in [J].$$

We are now ready to state the crucial property that the coarse reduced costs overestimate the (fine) reduced costs.

Lemma 1 (Coarse Reduced Cost Lemma) *The coarse reduced cost overestimate the (fine) reduced cost:*

$$[\tau]_j = [p]_j - \alpha^T * [A]_{\cdot j} \geq p_k - \alpha^T \cdot A_{\cdot k} = \tau_k \qquad \forall k \in J : [k] = j, j \in [J].$$

Proof Since $[p]_j = \max_{k \in J}\{p_k | [k] = j\}$ it holds that $[p]_j \geq p_k$. It also holds that $[A]_{ij1} \leq A_{ik} \leq [A]_{ij2}$ and with the $*$-operation we get $\alpha^T * [A]_{\cdot j} = \sum_{i \in I} \min \{\alpha_i \cdot [A]_{ij1}, \alpha_i \cdot [A]_{ij2}\} \leq \sum_{i \in I} \alpha_i \cdot A_{ik} = \alpha^T \cdot A_{\cdot k}$. Hence, $[p]_j - \alpha^T * [A]_{\cdot j} \geq p_k - \alpha^T * [A]_{\cdot j} \geq p_k - \alpha^T \cdot A_{\cdot k}$.

4 Coarse-to-Fine column generation

The bounding property of the coarse reduced cost is used to design a Coarse-to-Fine column generation algorithm. Briefly, it works as follows: In the first place, we only have to compute the reduced costs of the coarse variables. If these have a positive value then the reduced costs of the corresponding fine variables will be computed afterwards. Note that we maximize and search for positive reduced cost in this setting. It is described in Algorithm 4.1. There we denote the master problem by MP and the restricted master problem by RMP.

```
1  Input feasible RMP with start columns J_0, coarsening projection [·] and finite set
     of total columns J (not part of the RMP yet)
2  Output RMP with columns J' and an optimal solution for the MP
3  init J' = J_0
4  solve RMP with columns J'
5  compute coarse reduced costs [τ]
6  let Ĵ := {j ∈ [J] : [τ]_j > 0}
7  if Ĵ ≠ ∅ then
8  |   compute fine reduced costs τ_k   ∀k ∈ J : [k] ∈ Ĵ
9  |   let J* := {k ∈ J \ J' : [k] ∈ Ĵ : τ_k > 0}
10 |   if J* ≠ ∅ then
11 |   |   add J̃ ⊆ J* to J', J̃ ≠ ∅
12 |   |   goto 4
13 |   end
14 end
```

Algorithm 4.1: Coarse-to-Fine column generation algorithm.

The input of Algorithm 4.1 is a feasible RMP and a given coarsening projection $[\cdot]$. In line 4 we solve the current RMP with columns J'. In the first iteration these are only the initial columns J_0 that ensure feasibility. As a next step we compute the coarse reduced costs in line 5. If there are no coarse reduced costs with positive value, the algorithm terminates. Otherwise we compute the fine reduced costs for all fine columns which are projections of the coarse columns with positive coarse reduced costs. If none of the fine columns have positive reduced costs, the algorithm finishes as well. But if there are fine columns with positive reduced cost, then we add at least one of them to the model, jump back to line 4 and repeat the same procedure again.

5 Coarse-to-Fine for the TEP

We consider a problem specific approach for the coarsening projection. Namely, for the TEP, the coarsening is processed on the arc variables. Let us consider the scheduling graph $\mathcal{G} = (D, \mathcal{A})$ introduced in Section 2. To prepare the coarsening projection for each day and possible tour, all duty nodes that have similar starting times are aggregated.

We divide a day into blocks of two hours and create coarse nodes (one for each tour, i.e., for each feasible sequence of sections) representing these blocks. The motivation is basically that the expected traffic does not change dramatically during neighboring hours as opposed to considering the whole day. Thus, a coarse decision when a mobile tour starts (or ends) has already a considerable significance.

This coarsening projection on the nodes induces a coarsening projection on the arcs: We aggregate all arcs into a coarse arc that share the same coarse tail node and the same coarse head node. Therefore, for each coarse arc we introduce a coarse arc variable and map the variables belonging to the fine arcs that are aggregated in the current coarse arc to the corresponding coarse variable. In addition, we use a slightly modified definition of coarse reduced costs that takes the particular structure of the TEP scheduling graph into account. All constraints, that correspond to the sequence arc variables, only depend on the corresponding tail and head nodes.

The Coarse-to-Fine approach handles only the inspector arc variables. An artificial tour variable is maintaining feasibility in the beginning and we do one step of column generation on the tour variables in each iteration of the Coarse-to-Fine algorithm. Algorithm 4.1 is applied to the root node LP and afterwards the IP of the fine model is solved with the columns generated during the Coarse-to-Fine algorithm.

We compare this approach with a standard column generation approach, called Price & Branch, to solve the model's LP relaxation by pricing tour and roster arc variables. We can deviate this approach from Algorithm 4.1 by omitting lines $(5-7)$. Then, we again compute an integer feasible solution by restricting to all variables that were priced.

6 Computational Results

In this section we present several computational results for the two presented heuristic approaches. We show how the column generation algorithms decrease the model size of several industrial instances and that both heuristics (Price-and-Branch and Coarse-to-Fine) lead in almost all cases to feasible solutions with a high quality. We aggregated duty nodes sharing the same day, the same control tour, and starting in the same time block, as described in Section 5. The time blocks have a length of two hours.

Table 1 gives the basic data for each instance. Every instance represents a planning scenario for a control region and comprises a time horizon of several weeks. The length of a section control equals four hours. We set a time limit of one day for solving the IP. We performed all computations on an Intel(R) Xeon(R) CPU E5-2670 v2 machine with 2.50GHz and 10 CPU cores. For all computations we used CPLEX (version 12.6.0.0) as LP and IP solver.

There is a broad range of instances from different regions, with a varying number of duty types and fixed duties. Instance *I3* deviates from the other instances in that it is only generated for testing purposes. It covers only a plan-

Inst.	Region	Inspectors	Sections	Fixed Duties	Duty Types	Rows	Columns
I1	r_1	21	28	308	8	16469	164775
I2	r_2	21	13	206	8	14583	347128
I3	r_4	6	28	0	8	2242	27160
I4	r_6	22	22	64	9	20727	705410
I5	r_3	24	28	137	14	31758	2008131
I6	r_5	20	20	37	16	32080	2178483

Table 1 Key characteristics of the TEP instances

ning horizon of one week. Instances I1, I2 and I3 can be solved to optimality without column generation within one day.

For each instance we ran three different variants. The first, the default IP run, is called `noCG`. The second is the Price-and-Branch approach and the third Coarse-to-Fine. Table 2 presents in columns two, three and five the number of variables of the original (master) problem, the number of variables priced by Algorithm `CG`, and by algorithm `CtF`, respectively. In column four and six the relation of the restricted LP for `CG` and `CtF` to the `noCG` is shown in percent. As expected the number of priced variables is much smaller compared to the number of variables in the original model. As an example, instance *I4* has in total around 705,000 variables. The restricted problem after running the Price-and-Branch algorithm has only 97,000 columns. In case of Algorithm 4.1 the column size even reduces a litte more to 84,000. Instance *I5* has even a higher reduction with more than 90%.

Table 3 gives the results of solving the TEP IP both with the full model and with the columns generated by the column generation algorithms. Columns two to four give the solution of the IP run and columns five to seven the objective values either when the time limit is reached or if optimality is proven. We remind that an optimal solution of the IP restricted to the generated variables is not necessary optimal for the original problem but feasible.

On average, we achieved a considerable speed-up with all versions. For instance *I1* the decrease of running time is extreme. For *I3* there is in fact no need for column generation. Fortunately, for all instances feasible solutions could be found when solving the IP with the generated columns. Furthermore, all instances finished before reaching the time limit for `CG` and `CtF`. Raising

Instance	`noCG`	`CG`	%	`CtF`	%
I1	164775	31898	19.4	26366	16.0
I2	347128	60371	17.4	60082	17.4
I3	27160	2749	10.1	2788	10.3
I4	705410	97170	13.8	83921	11.9
I5	2008131	158497	7.9	156550	7.8
I6	2178483	143860	6.6	127646	5.8

Table 2 Model reduction measured by number of variables in the different LP relaxations

Instance	Time IP noCG	Time IP CG	Time IP CtF	Objective noCG	Objective CG	Objective CtF
I1	29,168.24	58.78	9.73	385,744	379,987	381,568
I2	15,573.29	171.86	131.76	776,919	768,434	766,635
I3	0.48	0.08	0.15	65,392	65,392	65,392
I4	86,408.14	1807.30	482.98	492,754	478,913	475,269
I5	86,400.93	49,973.54	14,930.88	530,344	518,593	517,169
I6	86,414.98	32,005.40	79,280.61	538,749	521,458	520,542

Table 3 Comparison of solution time and quality for the different methods

the time limit for I4 to I6 would lead to a longer running time for `noCG` and therefore to a smaller proportion in terms of running time for `CG` and `CtF`. On the other hand, despite not finishing, version `noCG` yields good IP solutions. In all cases, the IP best incumbent values of `noCG` are better (to a small fraction) than the ones by the heuristics. But in many cases (e.g. *I1* or *I5*) the differences are small enough to claim that our approaches yield good integral solutions. The results give a strong indication that `CtF` yields the smallest models and in many cases the best running times.

7 Conclusion

We presented two heuristics based on linear programming for a rostering problem in the area of toll enforcement on German motorways. We tackle the TEP by a standard multi-commodity flow model with some extensions in order to incorporate the control tours. One heuristic, called Price-and-Branch, is a column generation approach to solve the model's LP relaxation by pricing tour and roster arc variables.

The main contribution is a coarse-to-fine approach. There, several variables from the original model, called fine model, are mapped to a single variable in an aggregated model, the coarse model. First, we presented a generic approach to general linear programs and applied it to the TEP. We discussed the important property that the coarse reduced costs overestimate the fine reduced costs. Then column generation is performed on the coarse level. In both cases (Price-and-Branch and Coarse-to-Fine), we compute an integer feasible solution by restricting to all variables that were priced.

For both heuristic procedures we showed that feasible solutions with high quality can be computed even for large industrial instances. An important issue for future research is to solve instances with a large number of duty types (> 16) or a minor control duration that could not be solved so far. Another idea would be to add an additional algorithmic step to the Coarse-to-Fine, like the coarse reduction [6] in the railway setting, to compute additional suitable columns by fast combinatorial algorithms for a faster convergence.

References

1. Martine Labbé and Patrice Marcotte, and Gilles Savard A Bilevel Model of Taxation and Its Application to Optimal Highway Pricing *in Management Science*, 44(12-part-1):1608-1622, 1998, 10.1287/mnsc.44.12.1608.
2. Luce Brotcorne and Martine Labbé and Patrice Marcotte and Gilles Savard A Bilevel Model for Toll Optimization on a Multicommodity Transportation Network *in Transportation Science*, 35(4):345-358, 2001, 10.1287/trsc.35.4.345.10433.
3. Pia Bergendorff and Donald W. Hearn and Motakuri V. Ramana editor="Pardalos, Panos M. and Hearn, Donald W. and Hager, William W.", Congestion Toll Pricing of Traffic Networks *in Network Optimization*, 51–71, 1997, Springer Berlin Heidelberg, 10.1007/978-3-642-59179-2_4.
4. Tobias Harks and Ingo Kleinert and Max Klimm and Rolf H. Mhring Computing network tolls with support constraints *in Networks*, 65(3):262–285, 2015, 10.1002/net.21604.
5. Ralf Borndörfer and Guillaume Sagnol and Thomas Schlechte and Elmar Swarat Optimal Duty Rostering for Toll Enforcement Inspectors *in Annals of Operations Research*, 252(2):383–406, 2017, 10.1007/s10479-016-2152-1.
6. Ralf Borndörfer and Markus Reuther and Thomas Schlechte A Coarse-To-Fine Approach to the Railway Rolling Stock Rotation Problem *in 14th Workshop on Algorithmic Approaches for Transportation, Modelling, Optimization, and Systems*, 42, 79–91, 2014, 10.4230/OASIcs.ATMOS.2014.79.
7. Markus Reuther Mathematical Optimization of Rolling Stock Rotations PhD Thesis, 2016, Technische Universitt Berlin.
8. Jose Correa and Tobias Harks and Vincent J. C. Kreuzen and Jannik Matuschke Fare Evasion in Transit Networks *in Operations Research*, 65(1):165–183, 2017, 10.1287/opre.2016.1560.
9. Hilbert Snijders and Ricardo L. Saldanha Decision support for scheduling security crews at Netherlands Railways *in Public Transport*, 9(1):193–215, 2017, 10.1007/s12469-016-0142-y.

Multithreaded incremental solving for local search based metaheuristics with step chasing

Geoffrey De Smet · Tony Wauters

Received: date / Accepted: date

Abstract This work introduces a multithreaded solving methodology for local search based metaheuristics. It runs a single local search that spreads move evaluations across multiple threads. To preserve incremental score calculation (delta evaluation) capabilities, which are essential for the performance of local search methods, the child threads reproduce the step of the main thread in a method we named step chasing. The proposed method is implemented within OptaPlanner, a Java-based open source solver, and can thus be used by anyone. The effectiveness of the method is demonstrated using three metaheuristics (Tabu Search, Simulated Annealing, Late Acceptance) on four difficult combinatorial optimization problems: the nurse rostering problem, the vehicle routing problem, the curriculum course timetabling problem and the cloud balancing problem. Extensive experiments are performed using up to 16 threads with a total of 5550 runs, with significant speedups realized when more threads are available to the solver. All results are compared with a single threaded implementation, as well as a multi-walk approach. The greatest speedups take place with respect to the nurse rostering problem.

Keywords Parallel local search · Multithreading · Incremental solving · Metaheuristics · Open source

1 Introduction

Metaheuristics - and more specifically local search - are frequently used to solve difficult combinatorial optimization problems. In order to solve real-world

Geoffrey De Smet
Red Hat
E-mail: gds.geoffrey.de.smet@gmail.com

Tony Wauters
KU Leuven, Department of Computer Science, CODeS research group
E-mail: tony.wauters@kuleuven.be

problems and find high quality solutions, the speed at which these algorithms can calculate constraint and fitness scores is generally an important factor. The ability to accommodate large-scale problems is also crucial. A frequently applied technique for improving the scaling behaviour of local search is incremental score calculation (also known as delta evaluation). While this technique brings an important scaling factor, parallelization on multi-core machines and multi-node clouds enables further scaling. An opportunity which typically has been overlooked in the academic literature. Perhaps because combining parallelization and incremental score calculation poses some significant challenges.

Nevertheless, several works proposed a parallel local search method. [12] proposed multiple-walk and single walk parallel local search methods and applied it to the traveling salesman problem, the steiner tree problem and two scheduling problems. An overview of parallel metaheuristic strategies can be found in [5,1,11,6].

As highlighted by [2], most metaheuristic frameworks and libraries with parallelization focus on evolutionary algorithms. Therefore, very few frameworks support parallel local search or trajectory-based metaheuristics. Moreover, none of them support incremental score calculation.

This study introduces a parallel local search strategy with incremental score calculation, which is implemented in the open source constraint solver OptaPlanner [7]. We show that our parallelization strategy is effective for many local search algorithms (Hill Climbing, Tabu Search, Late Acceptance Hill Climbing, Simulated Annealing). Furthermore, we present benchmarks on four use cases (vehicle routing problem, nurse rostering problem, course scheduling and cloud balancing) resulting in a total of 37 datasets and 5550 benchmark runs.

The remainder of the paper is structured as follows. In Section 2 we discuss the primary implementation requirements for this research. The actual implemented method is discussed in Section 3. A thorough experimental investigation of the implemented method is provided in Section 4. Following this, Section 5 ends this paper with conclusions and directions for future work.

2 Requirements

Before we move on to the design of the implemented method, we must explore three important requirements which must be fulfilled in our implementation. These requirements are:

1. Incremental score calculation
2. Reproducible runs
3. Parallel computation

We will illustrate these requirements using the Nurse Rostering Problem (NRP) [4] as an example, an NP-hard problem which assigns shifts to nurses. Each shift must be assigned to exactly one nurse. Hard constraints typically include nurse conflicts, skill requirements and minimal rest periods. Soft constraints may include day off requests and illness affinity.

2.1 Incremental score calculation

A local search algorithm evaluates the neighborhood of proposed solutions at every iteration. It determines the quality of each proposed solution by calculating its score. In simple cases, a score is formed from two weighted numbers: one for the hard constraints (the feasibility check) and another for the soft constraints (the fitness function). Calculating these numbers is computationally expensive. For example, in the Nurse Rostering Problem, it requires detecting the number of nurse conflicts, and therefore every shift assignment must be compared with every other overlapping shift assignment to check if they are assigned to the same nurse. Given s shift assignments, this part of the score calculation requires $O(s^2)$ checks, and therefore scales quadratically. Some of the other hard or soft constraints are even more computationally expensive.

The local search algorithm triggers a score calculation for each move in its neighborhood until an acceptance criterion is met - at which point it applies the winning move and begins a new step to evaluate a new neighborhood. In the NRP, a simple change move assigns one shift to a different nurse. To calculate the score of the solution after applying such a move, we could calculate the score from scratch by iterating over all assignments. However, this is highly inefficient. Instead, we calculate it incrementally by determining the delta between the old and new score. For example in the NRP, the incremental score calculation of a move, that assigns one shift to a different nurse, need only compare that one shift assignment with every other overlapping shift assignment to determine the delta. This part of the score calculation now requires only $O(s)$ checks, instead of $O(s^2)$. This is an order of magnitude faster than non-incremental score calculation methods.

To achieve multithreaded solving, we cannot afford to forfeit incremental score calculation given that, in practice, the negative effects of being unable to calculate scores incrementally far outweigh any gain of parallel computation. For example, given a NRP with 5000 shift assignments and 250 nurses, incremental calculation of the nurse conflict constraint is up to 5000 times faster than non-incrementally. To make up for such a loss, a perfectly parallelized algorithm would require at least 5000 CPU's. From both a practical and economic perspective, this is clearly not an option.

Our implementation must therefore combine both incremental score calculation and multithreaded solving.

2.2 Reproducible runs

A constraint solver is reproducible if and only if running it twice yields the exact same solution (with the exact same score), given the exact same allocated CPU time (in the same manner). For local search and construction heuristic algorithms, this boils down to yielding the exact same solution at every step. A step is the outer iteration in these algorithms: each time they pick a winning move, it is a new step. Note that due to a difference of allocated CPU time,

the timing of each step between runs can vary and - given the same amount of time - a reproducible run still might not reach the same number of steps.

Most local search algorithms use a Pseudo Random Number Generator (PRNG), which influences reproducibility. For example, a PRNG breaks score ties in Tabu Search and influences move acceptance in Simulated Annealing. A single-threaded run can easily be made reproducible by using a single, seeded PRNG for all decisions. However, if a multi-threaded run uses a single PRNG across multiple threads, the concurrent calls on that PRNG suffer from congestion. Furthermore, we must somehow guarantee that all calls on that PRNG are always executed in the same order, given that re-ordering affects the step decision, causing the entire local search algorithm to go down a different path. As such, an efficient and reproducible multi-threaded run cannot use the same PRNG across multiple threads.

In practice, reproducibility is critical for any production solver: it allows programmers to debug their code during development as well as reproduce production issues on their own machines. Furthermore, in highly regulated enterprise environments, such as financial institutions, it enables the auditing of historic solver runs.

Our implementation of multithreaded solving must therefore not sacrifice reproducibility.

2.3 Parallel computation

Let us examine which parts of local search are suitable for parallelization. In order to improve performance over a single threaded solver, a multithreaded solver must parallelize at least parts of the algorithm. Given t threads and the same number of CPU cores, the performance of those algorithm parts increases by a factor of t, minus any overhead the multithreaded solving incurs.

Looking at the anatomy of a single threaded local search (Figure 1), there are several candidates to compute in parallel:

1. Select move: From the neighborhood, generate one move at a time, just in time. We could give each child thread a copy of the move selector (which generates the neighborhood on the fly). If we give each move selector its own PRNG (which is seeded by a common PRNG), they would consequently generate the same moves, thereby preserving reproducibility. However, we would need to force all child threads to generate and evaluate the exact same number of moves, meaning that the faster move selectors would have to wait for the slower ones. With a mix of fine and coarse grained moves, or varying CPU power per core (often the case in public clouds), the performance cost for enforcing an equal distribution of moves per child thread is unacceptable. Furthermore, move selection itself is usually inexpensive and thus we chose not to parallelize it in this work.
2. Calculate move score: Evaluating hard and soft constraints is computationally expensive given that all constraints must be evaluted. Even with incremental score calculation, evaluating the NRP's nurse conflict constraint

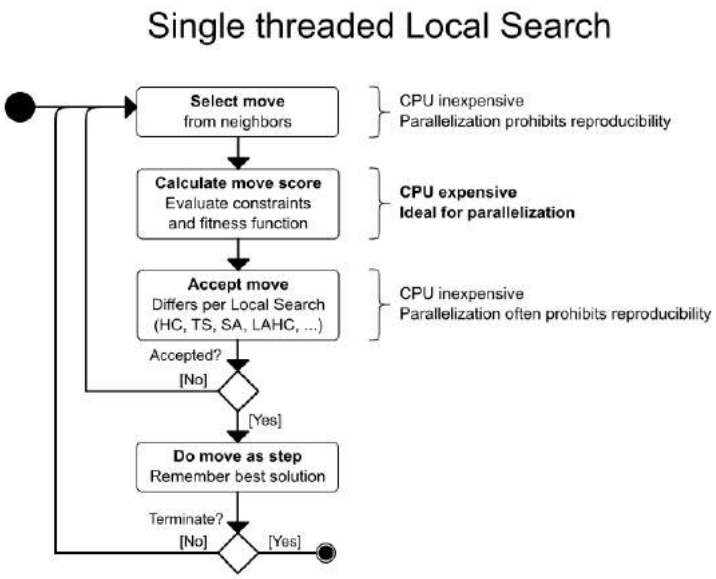


Fig. 1 Anatomy of a single threaded Local search

is still in $O(s)$ - which is still expensive because its nested in the move evaluation iteration, which is in turn nested in the step iteration. Yet each move can be scored in isolation from the others in the same step, so this is an ideal candidate for parallelization.

3. Accept move: In Hill Climbing, the acceptor simply checks if the score of a move improves upon the best score. Such an implementation can be easily parallelized. But in Simulated Annealing, the acceptance criterion uses the PRNG. Because we do not wish to enforce in advance which child thread evaluates which move (because, as explained earlier, this impairs efficiency), the acceptor must always use the same PRNG in the same order to guarantee reproducibility. Also, given that move acceptance itself is usually inexpensive, we choose not to parallelize it.

Our approach must therefore at least parallelize the score calculation of each move.

3 Method

We explain our method on local search in a general manner given that it works with respect to any local search algorithm (including Tabu Search [8], Simulated Annealing [10] and Late Acceptance Hill Climbing [3]), as shown by the implementation and benchmarks later in this paper.

The basic principle is parallelizing the score calculation by offloading it from the main solver thread (the parent thread) to n child threads, as shown in Figure 2.

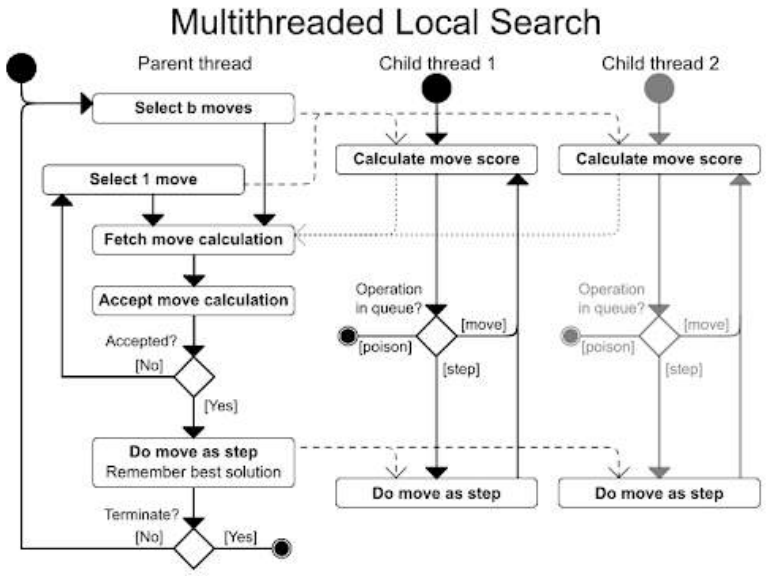


Fig. 2 Multithreaded local search

This redistribution has two noteworthy consequences:

- To evaluate multiple moves in parallel, multiple moves are active simultaneously.
- To preserve incremental score calculation, each child thread must have an isolated score calculation state that must be kept in sync with the steps on the parent thread.

3.1 Select multiple moves

One implication of calculating the score of two (or more) moves in parallel is that the second move must be generated before the calculation of the first move's score has finished. As explained earlier, we have chosen to generate all moves in the parent thread. Furthermore, the winning move of a step is also picked in the parent thread. So, given n child threads, we must at least generate n moves in the parent thread, before processing the resulting score of the first move in the same thread, to prevent the child threads from starving. All these generated moves for which the score has not been processed yet are considered active. For reproducibility, the number of active moves must be

deterministic. Generating a different number of moves, offsets the PRNG used in move selectors, causing non-reproducible runs. Furthermore, we must have more than n active moves given that a child thread can calculate multiple fine-grained moves while the parent thread is awaiting the result of a coarse-grained move from another child thread. Therefore we have b active moves, for which b is a multiple of n. There is a trade-off affecting the value of b: it must be large enough to satisfy the hunger of the child threads for maximum performance (especially with a mix of fine-grained and coarse-grained moves), but if b is too large then the generating of b moves at the beginning of each step iteration could negatively affect performance and memory usage. From our experiments, we found that setting b to 10 times n is a good default setting, where n is the number of child threads.

In the parent thread, we begin every step by selecting b moves from the neighborhoods, before receiving the first evaluated move from a child thread. Then, every time a move is received, we select a new one such there are always b active moves.

3.2 Move evaluation

The parent thread communicates with the child threads through an operation queue to send information and a result queue to receive information. The parent thread puts each selected move in the operation queue as a move calculation operation. The child threads continuously pulls these moves from that operation queue. When they do so, they calculate the score incrementally for that move and put the result in the result queue. The auxiliary data structures for such incremental score calculation are maintained by each child thread individually. Meanwhile, once the parent thread has generated b moves, it pulls the first move calculation result from the result queue. If that queue is empty, it waits until one of the child threads adds a result. After receiving a move, the parent thread accepts or rejects that move. Then it checks if an accepted move has won the step iteration. If at this point in time there is no winning step, it selects a new move.

This orchestration, shown in Figure 3, ensures that all child threads are kept busy evaluating moves, regardless of any variance concerning move evaluation duration.

Due to differences in move granularity, the score calculation results of each move come back out of order. For example, the result of the 4th generated move may arrive before the result of the 3rd generated move, especially if the 3rd move is more coarsely grained. For reproducibility, the parent thread orders the results back into their original order, on the fly. In this example, if the parent thread needs to process the 3rd move, but the 4th and 5th arrive first, it places those moves in a backlog and waits until the 3rd move arrives. Later, to return the 4th or 5th move, it first searches in the non-empty backlog before pulling a move from the result queue.

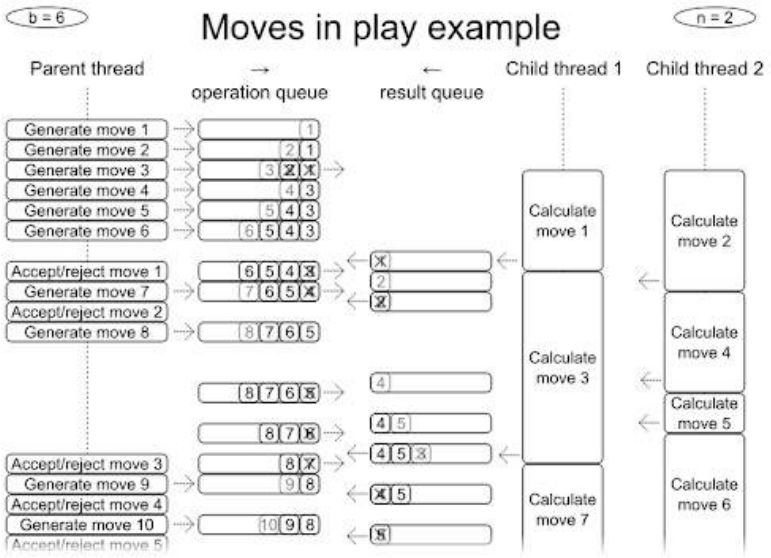


Fig. 3 Lifetime of active moves

The parent thread then accepts or rejects the retrieved move, depending on the acceptance criteria. If enough moves are accepted, it appoints the winning move of the current step. Both these decisions depend on the Local Search type and greatly affect the number of selected moves per step. For example, Tabu Search typically samples many non-tabu moves, which results in a slow stepping run. Simulated Annealing and Late Acceptance on the other hand, appoint the first accepted move, resulting in a fast stepping run - especially in the beginning.

If there is no winning move, the parent thread selects a new move and puts it in the operation queue to ensure there are always exactly b active moves, before repeating the process.

3.3 Step chasing

If there is a winning move, the local search advances: that move is applied on the working solution of the parent thread and the step ends. Before doing so, it clears the operation queue (which removes most of the b active moves except for about n active moves that already reached a child thread) and puts n step operations for that winning move in the operations queue, to sync the child threads. When a child thread pulls such a step operation it applies the winning move to keep their solution state in sync with the parent thread's state in an incremental manner (without cloning the entire solution state). After processing the step operation, the child thread also waits until all other

child threads have also pulled and processed their step operation. This is to prevent one child thread pulling two step operations of the same step from the operations queue, which would cause another child thread to not get any and go out of sync.

After applying the accepted move as a step, the parent thread also updates the best known solution if the new solution's score is better. Unless the termination criterion is reached, the process restarts and the parent thread generates b active moves. Then the parent thread continues as before: every time it receives an evaluated move, it generates a new active move. As for the old active moves that reached a child thread before the operations queue got cleared, they eventually end up in the result queue too and they are completely ignored when they finally arrive to the parent thread in the new step (based on their outdated step ID).

This orchestration, shown in Figure 4, ensures that every child thread has its own copy of an equivalent working solution, eventually in sync with the parent thread, to incrementally evaluates each move in isolation. They calculate the same score as the parent thread would, regardless of when and which child thread ends up evaluating a specific move.

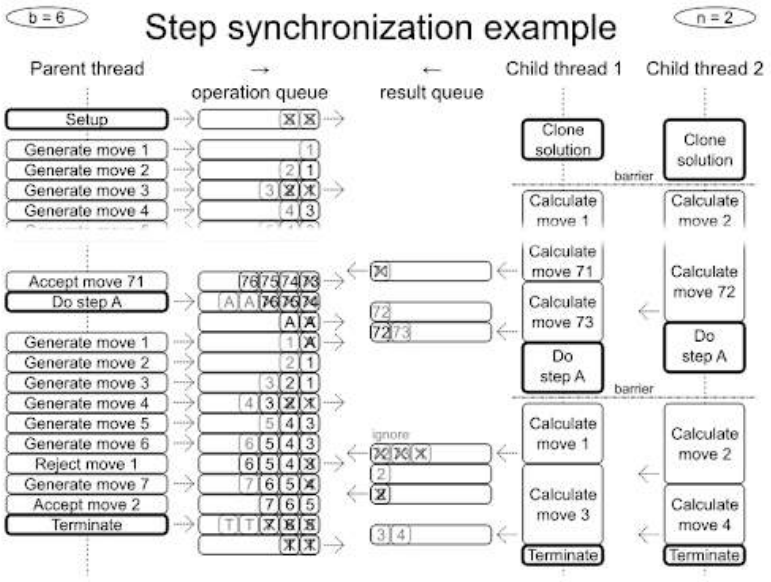


Fig. 4 Step synchronization procedure

If the termination criterion is reached, the parent thread clears the operation queue and puts n poison pill operations on the operation queue. When a child thread pulls such a poison pill, it terminates itself. Meanwhile the parent thread also terminates, returning the best solution encountered.

4 Experimental results

In order to evaluate the implemented parallelization method we ran a significant number of benchmarks on a range of optimization problems. The following problems, all of which are available as examples within OptaPlanner[1], are considered:

- Nurse rostering problem
- Vehicle routing problem
- Curriculum course timetabling problem
- Cloud balancing problem

All experiments[2] are performed on a dual Intel(R) Xeon(R) CPU E5-2660 v3 platform with 20 cores. Hyper Threading was disabled. A maximum heap size of 4GB RAM was allowed for the Java Virtual Machine (JDK 8), which was revealed to be sufficient for all experiments. OptaPlanner version 7.13.0.Final was used. Each algorithm configuration was run 10 times on each set of problem instances with different random seeds. As a termination condition, a fixed time of 300 seconds was used for all runs.

In the next sections we will first present the summarized results for each test case. Following this, Section 4.5 will provide a more thorough discussion of the results.

4.1 Nurse rostering problem results

In the nurse rostering problem, as defined by the first International Nurse Rostering Competition 2010 [9], shifts must be assigned to nurses while taking into account various constraints such as skill requirements, employee availability and unwanted shift patterns. We tested on the competition's medium, medium-late and medium-hidden[3] instances. Figure 5 shows the average speedups over all instances for Late Acceptance, Simulated Annealing and Tabu Search. For each algorithm the default version without multithreading is compared against a multithreaded version with step chasing using 2, 4, 8 and 16 threads. It is clear from the figure that the multithreaded solver offers good speedups compared to the default version, with increasing speedups up to 16 threads. The most significant speedups are reached by Tabu Search (which is up to 9.32 times faster when using 16 threads).

In addition to the realized speedups we also want to study the effect of parallelization on the quality of the solutions. Table 1 shows the comparison of solution quality (score) between a Single walk (1 thread), a multi-walk[4] (8

[1] OptaPlanner code and documentation available at: `https://www.optaplanner.org`

[2] A note on how to reproduce these results can be found at: `https://www.optaplanner.org/code/benchmarks.html`

[3] `https://www.kuleuven-kulak.be/nrpcompetition/instances-results`

[4] Since a real multi-walk is not implemented in OptaPlanner, the multi-walk results are defined by taking the best out of 8 independent walks.

independent walks/threads) and the multithreaded solver with step chasing (8 threads) on the nurse rostering problem. Scores are averaged over 10 runs. The columns **Δ single** and **Δ multi-walk** indicate the solution quality difference between the step chasing and the single, and between the step chasing and the multi-walk, respectively. The average results indicate that, when given 8 threads, multithreading with step chasing outperforms a single walk, and for Late Acceptance and Tabu Search also outperforms a multi-walk strategy.

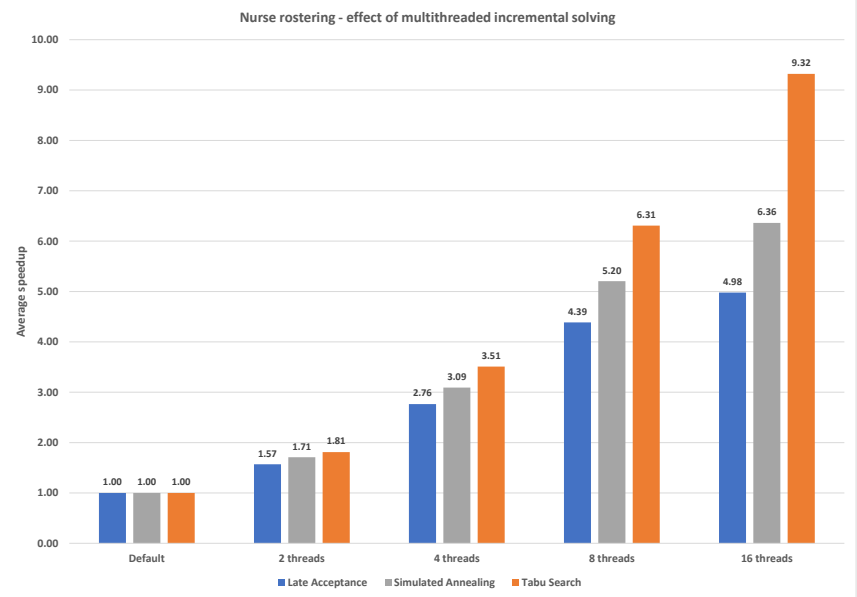


Fig. 5 Average speedups on the nurse rostering problem for Late Acceptance, Simulated Annealing and Tabu Search.

4.2 Vehicle routing problem results

In the vehicle routing problem, we tested on a set of instances from the vrp-rep benchmark website [5]: belgium-road-time-n50-k10, belgium-road-time-n100-k10, belgium-road-time-n500-k20, belgium-road-time-n1000-k20, belgium-road-time-n2750-k55.vrp. These include capacity constraints and realistic asymmetric distances calculated from OpenStreetMap. Figure 6 illustrates the average speedups over all instances for Late Acceptance, Simulated Annealing and Tabu Search on this problem. For each algorithm the default version without multithreading is again compared against a multithreaded version using 2, 4,

[5] `http://www.vrp-rep.org/datasets.html`

Alg.	Dataset	Single	Multi-walk 8T	Step chasing 8T	Δ single	Δ multi-walk
LA	medium01	346,3	334,0	334,9	3,29%	-0,27%
LA	medium02	347,1	333,0	334,4	3,66%	-0,42%
LA	medium03	334,8	328,0	326,0	2,63%	0,61%
LA	medium04	348,9	337,0	337,4	3,30%	-0,12%
LA	medium05	386,1	383,0	382,1	1,04%	0,23%
LA	medium_hint01	152,3	120,0	121,7	20,09%	-1,42%
LA	medium_hint02	268,9	190,0	160,0	40,50%	15,79%
LA	medium_hint03	314,7	187,0	190,6	39,43%	-1,93%
LA	medium_late01	330,6	242,0	247,2	25,23%	-2,15%
LA	medium_late02	102,2	99,0	96,1	5,97%	2,93%
LA	medium_late03	113,2	84,0	85,1	24,82%	-1,31%
LA	medium_late04	113,7	106,0	105,5	7,21%	0,47%
LA	medium_late05	365,8	221,0	225,6	38,33%	-2,08%
Late Acceptance average					**16,58%**	**0,80%**
SA	medium01	254,7	251,0	248,4	2,47%	1,04%
SA	medium02	253,7	248,0	248,4	2,09%	-0,16%
SA	medium03	250,6	244,0	244,3	2,51%	-0,12%
SA	medium04	251,6	246,0	245,0	2,62%	0,41%
SA	medium05	316,2	312,0	311,1	1,61%	0,29%
SA	medium_hint01	52,2	43,0	43,2	17,24%	-0,47%
SA	medium_hint02	103,6	95,0	97,1	6,27%	-2,21%
SA	medium_hint03	167,6	142,0	148,6	11,34%	-4,65%
SA	medium_late01	181,5	178,0	176,3	2,87%	0,96%
SA	medium_late02	36,1	25,0	27,5	23,82%	-10,00%
SA	medium_late03	41,7	35,0	36,7	11,99%	-4,86%
SA	medium_late04	47,4	42,0	40,8	13,92%	2,86%
SA	medium_late05	166,9	153,0	152,5	8,63%	0,33%
Simulated Annealing average					**8,26%**	**-1,28%**
TS	medium01	255,3	252,0	248,2	2,78%	1,51%
TS	medium02	252,1	248,0	245,7	2,54%	0,93%
TS	medium03	248,3	244,0	243,2	2,05%	0,33%
TS	medium04	253,1	248,0	246,8	2,49%	0,48%
TS	medium05	324,9	320,0	315,3	2,95%	1,47%
TS	medium_hint01	60,2	55,0	53,1	11,79%	3,45%
TS	medium_hint02	147,0	130,0	138,4	5,85%	-6,46%
TS	medium_hint03	203,1	189,0	185,0	8,91%	2,12%
TS	medium_late01	223,9	208,0	210,0	6,21%	-0,96%
TS	medium_late02	45,9	42,0	37,1	19,17%	11,67%
TS	medium_late03	51,2	49,0	45,1	11,91%	7,96%
TS	medium_late04	50,7	45,0	45,0	11,24%	0,00%
TS	medium_late05	218,8	198,0	200,1	8,55%	-1,06%
Tabu Search average					**7,42%**	**1,65%**

Table 1 Score comparison on the nurse rostering problem for single-walk, multi-walk (8 threads), and step chasing (8 threads). Lower scores indicate better solutions.

8 and 16 threads. The best results are obtained when using 4 threads. More significant speedups are obtained for larger instances, while the multithreaded version offers no benefit for the two smallest instances .

Table 2 shows the comparison of solution quality (score) between a Single walk (1 thread), a multi-walk (8 independent walks/threads) and the multi-threaded solver with step chasing (8 threads) on the vehicle routing problem. Scores are averaged over 10 runs. The columns **Δ single** and **Δ multi-walk** indicate the solution quality difference between the step chasing and the single, and between the step chasing and the multi-walk, respectively. Given the low speedup values on this problem when 8 threads are available, no significant benefit can be seen from the results. In fact, a multi-walk strategy performance better in most cases.

4.3 Curriculum course timetabling results

We also tested on the curriculum course timetabling problem. Lectures have to be assigned to rooms and periods, under constraints such as teacher conflicts, room capacity and curriculum compactness. Instances from the international

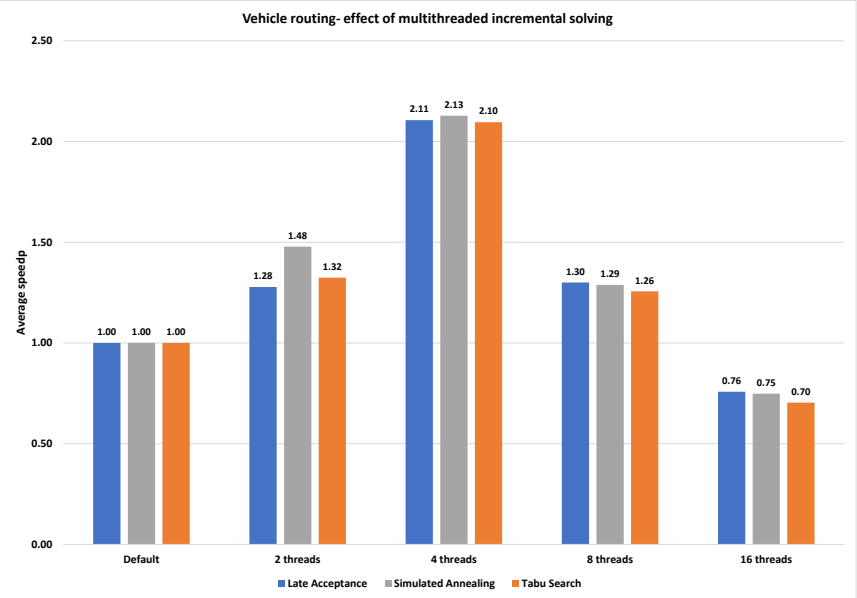


Fig. 6 Average speedups on the vehicle routing problem for Late Acceptance, Simulated Annealing and Tabu Search.

Alg.	Dataset	Single	Multi-walk 8T	Step chasing 8T	Δ single	Δ multi-walk
LA	belgium-road-time-n50-k10	114687,9	114320,2	114544,0	0,13%	-0,20%
LA	belgium-road-time-n100-k10	141631,7	140883,1	141667,2	-0,03%	-0,56%
LA	belgium-road-time-n500-k20	360078,3	356830,5	358769,3	0,36%	-0,54%
LA	belgium-road-time-n1000-k20	556317,0	555025,2	541285,1	2,70%	2,48%
LA	belgium-road-time-n2750-k55	1195007,5	1194608,9	1190600,6	0,37%	0,34%
Late Acceptance average					**0,71%**	**0,30%**
SA	belgium-road-time-n50-k10	117740,9	116871,7	117757,9	-0,01%	-0,76%
SA	belgium-road-time-n100-k10	152050,8	145775,7	152893,6	-0,55%	-4,88%
SA	belgium-road-time-n500-k20	374649,3	372377,2	375574,1	-0,25%	-0,86%
SA	belgium-road-time-n1000-k20	522698,7	512684,7	524798,4	-0,40%	-2,36%
SA	belgium-road-time-n2750-k55	1127755,5	1118702,3	1114100,7	1,21%	0,41%
Simulated Annealing average					**0,00%**	**-1,69%**
TS	belgium-road-time-n50-k10	118009,2	116340,7	117592,1	0,35%	-1,08%
TS	belgium-road-time-n100-k10	148411,5	144667,8	148509,0	-0,07%	-2,66%
TS	belgium-road-time-n500-k20	362010,5	353631,9	362017,5	0,00%	-2,37%
TS	belgium-road-time-n1000-k20	525642,8	521534,0	522372,1	0,62%	-0,16%
TS	belgium-road-time-n2750-k55	1200391,1	1200391,1	1200391,1	0,00%	0,00%
Tabu Search average					**0,18%**	**-1,25%**

Table 2 Score comparison on the vehicle routing problem for single-walk, multi-walk (8 threads), and step chasing (8 threads). Lower scores indicate better solutions.

Timetabling Competition 2007 Track 3 `http://www.cs.qub.ac.uk/itc2007/` are used.

Figure 7 shows the average speedups over all instances for Late Acceptance, Simulated Annealing and Tabu Search on this problem. For each algorithm the default version without multithreading is compared to a multithreaded version using 2, 4, 8 and 16 threads. The figure shows how good speedups are achieved when using 4 or more threads. However, no significant gains are realized when using more than 4 threads. Once again, Tabu Search benefits most from multithreaded incremental solving.

Table 3 shows the comparison of solution quality (score) between a Single walk (1 thread), a multi-walk (8 independent walks/threads) and the multithreaded solver with step chasing (8 threads) on the curriculum course timetabling problem. Scores are averaged over 10 runs. The columns **Δ single** and **Δ multi-walk** indicate the solution quality difference between the step chasing and the single, and between the step chasing and the multi-walk, respectively. It should be noted, that for this problem not all runs returned a feasible solution. Therefore, infeasible solutions are penalized with a penalty value of 100,000 for each hard constraint violation. On this problem, the multi-walk strategy is capable of finding more feasible solutions. However, when both strategies are able to find feasible solutions, the step chasing shows better results when Late Acceptance or Simulated Annealing is used.

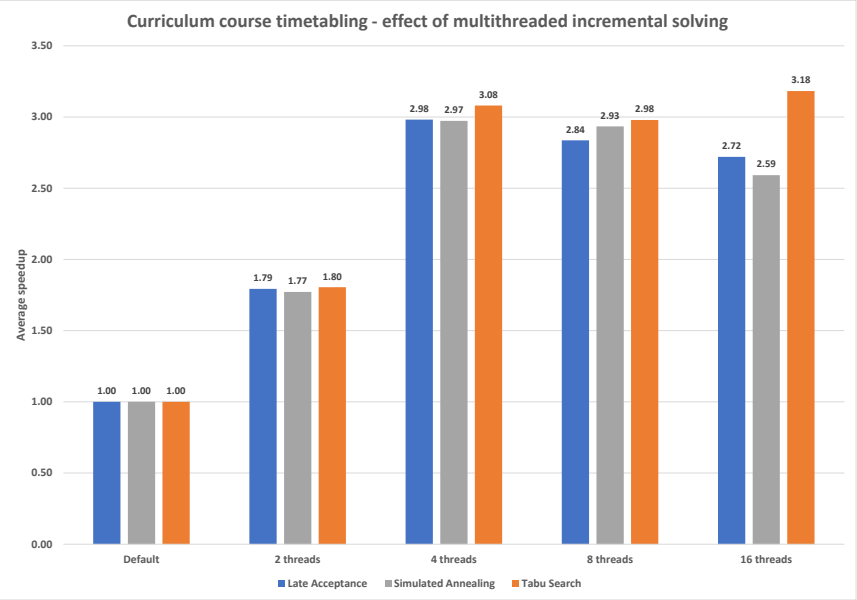


Fig. 7 Average speedups on the curriculum course timetabling problem for Late Acceptance, Simulated Annealing and Tabu Search.

4.4 Cloud balancing results

We have also tested the incremental solver on the cloud balancing problem. This problem concerns the assignment of computer processes to machines subject to CPU, RAM and network bandwidth constraints with the aim of reducing operating costs. The instances considered are 100computers-300processes, 200computers-600processes, 400computers-1200processes, 800computers-2400processes

Alg.	Dataset	Single	Multi-walk 8T	Step chasing 8T	Δ single	Δ multi-walk
LA	comp01	10,7	10,0	6,1	42,99%	39,00%
LA	comp02	214,5	191,0	199,9	6,81%	-4,66%
LA	comp03	209,5	191,0	188,7	9,93%	1,20%
LA	comp04	127,2	111,0	104,3	18,00%	6,04%
LA	comp05	250626,3	583,0	250654,5	-0,01%	-42893,91%
LA	comp06	244,3	222,0	181,2	25,83%	18,38%
LA	comp07	269,3	251,0	186,2	30,86%	25,82%
LA	comp08	147,7	130,0	120,5	18,42%	7,31%
LA	comp09	224,0	208,0	211,0	5,80%	-1,44%
LA	comp10	200,7	185,0	156,4	22,07%	15,46%
LA	comp11	1,1	0,0	0,0	100,00%	0,00%
LA	comp12	10533,3	499,0	534,8	94,92%	-7,17%
LA	comp13	163,7	154,0	147,0	10,20%	4,55%
LA	comp14	158,7	152,0	136,0	14,30%	10,53%
Late Acceptance average					**28,58%**	**-3055,64%**
SA	comp01	8,2	7,0	5,8	29,27%	17,14%
SA	comp02	40210,2	192,0	70184,3	-74,54%	-36454,32%
SA	comp03	205,5	178,0	197,3	3,99%	-10,84%
SA	comp04	99,9	92,0	79,2	20,72%	13,91%
SA	comp05	680749,7	600676,0	730764,1	-7,35%	-21,66%
SA	comp06	60183,0	167,0	30152,1	49,90%	-17955,15%
SA	comp07	184,2	167,0	125,7	31,76%	24,73%
SA	comp08	118,4	103,0	85,7	27,62%	16,80%
SA	comp09	187,2	174,0	176,0	5,98%	-1,15%
SA	comp10	147,0	135,0	113,7	22,65%	15,78%
SA	comp11	0,8	0,0	0,0	100,00%	0,00%
SA	comp12	30569,5	543,0	70568,6	-130,85%	-12896,06%
SA	comp13	126,6	120,0	106,4	15,96%	11,33%
SA	comp14	140,1	123,0	116,4	16,92%	5,37%
Simulated Annealing average					**8,00%**	**-4802,44%**
TS	comp01	7,8	6,0	5,8	25,64%	3,33%
TS	comp02	150242,3	233,0	160217,0	-6,64%	-68662,66%
TS	comp03	259,5	239,0	230,6	11,14%	3,51%
TS	comp04	129,3	105,0	114,0	11,83%	-8,57%
TS	comp05	730741,0	600693,0	790767,8	-8,21%	-31,64%
TS	comp06	120217,8	220,0	90185,4	24,98%	-40893,36%
TS	comp07	220,2	200,0	173,7	21,12%	13,15%
TS	comp08	149,9	137,0	135,8	9,41%	0,88%
TS	comp09	230,2	218,0	221,0	4,00%	-1,38%
TS	comp10	179,8	160,0	145,1	19,30%	9,31%
TS	comp11	10,2	1,0	1,9	81,37%	-90,00%
TS	comp12	120595,5	100564,0	100587,1	16,59%	-0,02%
TS	comp13	157,1	125,0	143,4	8,72%	-14,72%
TS	comp14	177,5	143,0	154,9	12,73%	-8,32%
Tabu Search average					**16,57%**	**-7834,32%**

Table 3 Score comparison on the curriculum course timetabling problem for single-walk, multi-walk (8 threads), and step chasing (8 threads). Lower scores indicate better solutions. Infeasible solutions are penalized with a penalty value of 100,000 for each hard constraint violation

and 1600computers-4800processes from optaplanner-examples. Figure 8 illustrates the average speedups over all instances when Late Acceptance, Simulated Annealing and Tabu Search are applied to this problem. For each algorithm the default version without multithreading is compared against a multithreaded version using 2, 4, 8 and 16 threads. The figure shows that the multithreaded incremental solver runs faster than the default version on this problem. The largest speedups are realized for the Tabu Search with 8 threads. However, further increasing the number of threads to 16 appears to have a negative impact on the speedup for Tabu Search.

Table 3 shows the comparison of solution quality (score) between a Single walk (1 thread), a multi-walk (8 independent walks/threads) and the multithreaded solver with step chasing (8 threads) on the cloud balancing problem. Scores are averaged over 10 runs. The columns **Δ single** and **Δ multi-walk** indicate the solution quality difference between the step chasing and the single, and between the step chasing and the multi-walk, respectively. Although a small benefit over a single walk can be observed, the step chasing speedups do

Alg.	Dataset	Single	Multi-walk 8T	Step chasing 8T	Δ single	Δ multi-walk
LA	100computers-300processes	111036,0	110650,0	110715,0	0,29%	-0,06%
LA	200computers-600processes	193061,0	191490,0	192793,0	0,14%	-0,68%
LA	400computers-1200processes	432172,0	429110,0	432230,0	-0,01%	-0,73%
LA	800computers-2400processes	890808,0	886520,0	889665,0	0,13%	-0,35%
LA	1600computers-4800processes	1788136,0	1783800,0	1784911,0	0,18%	-0,06%
Late Acceptance average					**0,14%**	**-0,38%**
SA	100computers-300processes	110335,0	109950,0	110142,0	0,17%	-0,17%
SA	200computers-600processes	192335,0	191370,0	192183,0	0,08%	-0,42%
SA	400computers-1200processes	429866,0	428300,0	430913,0	-0,24%	-0,61%
SA	800computers-2400processes	886034,0	884510,0	884609,0	0,16%	-0,01%
SA	1600computers-4800processes	1765000,0	1763950,0	1759957,0	0,29%	0,23%
Simulated Annealing average					**0,09%**	**-0,20%**
TS	100computers-300processes	110099,0	109410,0	108723,0	1,25%	0,63%
TS	200computers-600processes	191490,0	190530,0	190148,0	0,70%	0,20%
TS	400computers-1200processes	432013,0	429360,0	428019,0	0,92%	0,31%
TS	800computers-2400processes	895670,0	891230,0	881928,0	1,53%	1,04%
TS	1600computers-4800processes	1803831,0	1801300,0	1761139,0	2,37%	2,23%
Tabu Search average					**1,36%**	**0,88%**

Table 4 Score comparison on the cloud balancing problem for single-walk, multi-walk (8 threads), and step chasing (8 threads). Lower scores indicate better solutions.

not translate into significantly better solutions on this problem when compared to a multi-walk.

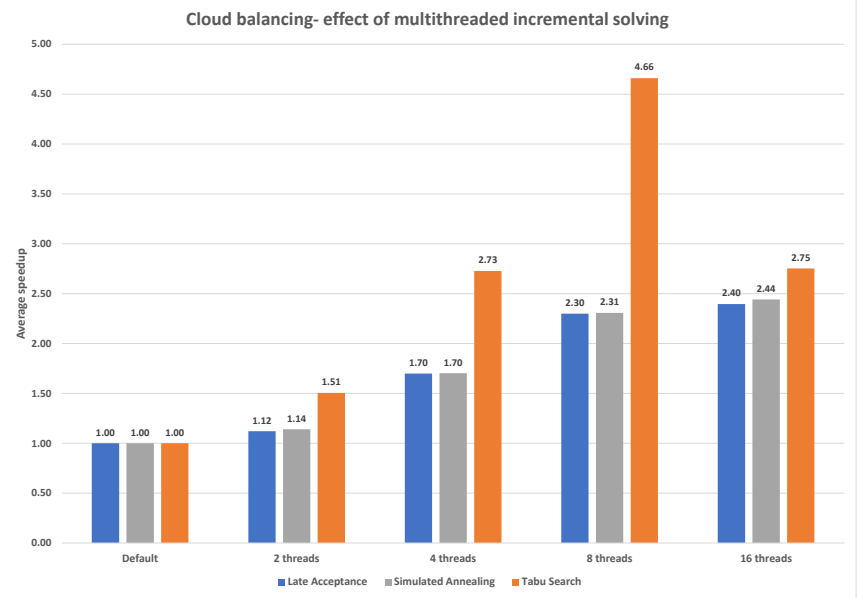


Fig. 8 Average speedups on the cloud balancing problem for Late Acceptance, Simulated Annealing and Tabu Search.

4.5 Results discussion

The results on the four tested benchmark problems show that the proposed multithreaded implementation offers significant speedups in general. However, the speedups depend upon the problem characteristics such as scale.

It can be seen that Tabu Search benefits more from the multithreaded implementation than Simulated Annealing and Late Acceptance. This can be attributed to the fact that Tabu Search typically evaluates more moves in each step than the other algorithms. This means that it performs fewer steps and therefore has less step chasing overhead.

When compared to a (non parallel) single-walk, the step chasing strategy results in significantly better solutions. However, when compared to a multi-walk strategy, which offers a more diverse search, step chasing does not always results in better solutions. In particular, when feasibility is difficult to achieve a multi-walk strategy seems to be the better choice.

5 Conclusion and future work

This paper introduced an effective multithreaded incremental solving method for metaheuristics using the concept of step chasing. This method was integrated into the OptaPlanner solver and compared against the default non-multithreaded metaheuristics on four difficult combinatorial optimization problems: the nurse rostering problem, the vehicle routing problem, the curriculum course timetabling problem and the cloud balancing problem. It shows significant speedups and better solutions.

An important requirement of the method are reproducible runs. This highly impacted architecture. Dropping this requirement could yield additional performance and scaling benefits. Future research is needed to quantify such benefits.

As it stands, experiments show diminishing returns as the child thread count increases or as move evaluation duration decreases. This is likely due to congestion in the operation and result queue. This leads to two potential improvements which ought to be addressed in future research:

- Ship multiple moves in bulk through the operation and result queues in order to decrease the frequency with which parent and child threads operate on it.
- Redesign an architecture in which the child threads do not share the same operation and result queue. For example, there could be one operation queue per child thread, such that only the parent thread and one child thread interact with the same queue (except for work stealing by other child threads which have already emptied their own operation queue). Alternatively, the result queue could be replaced by a reduce operation similar to that found in MapReduce.

Further research is also needed to determine how well this technique applies on other metaheuristics, such Genetic Algorithms, Particle Swarm Optimization and Ant Colony Optimization.

Acknowledgements Editorial consultation provided by Luke Connolly (KU Leuven).

References

1. Alba, E.: Parallel metaheuristics: a new class of algorithms, vol. 47. John Wiley & Sons (2005)
2. Alba, E., Luque, G., Nesmachnow, S.: Parallel metaheuristics: recent advances and new trends. International Transactions in Operational Research **20**(1), 1–48 (2013)
3. Burke, E.K., Bykov, Y.: A late acceptance strategy in hill-climbing for exam timetabling problems. In: PATAT 2008 Conference, Montreal, Canada, pp. 1–7 (2008)
4. Burke, E.K., De Causmaecker, P., Berghe, G.V., Van Landeghem, H.: The state of the art of nurse rostering. J. of Scheduling **7**(6), 441–499 (2004). DOI 10.1023/B:JOSH.0000046076.75950.0b. URL `https://doi.org/10.1023/B:JOSH.0000046076.75950.0b`
5. Crainic, T.G., Toulouse, M.: Parallel strategies for meta-heuristics. In: Handbook of metaheuristics, pp. 475–513. Springer (2003)
6. Crainic, T.G., Toulouse, M.: Parallel meta-heuristics. In: Handbook of metaheuristics, pp. 497–541. Springer (2010)
7. De Smet, G., open source contributors: OptaPlanner User Guide (2006). URL `https://www.optaplanner.org`. OptaPlanner is an open source constraint solver in Java
8. Glover, F., Laguna, M.: Tabu search. In: Handbook of combinatorial optimization, pp. 2093–2229. Springer (1998)
9. Haspeslagh, S., De Causmaecker, P., Schaerf, A., Stølevik, M.: The first international nurse rostering competition 2010. Annals of Operations Research **218**(1), 221–236 (2014)
10. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. science **220**(4598), 671–680 (1983)
11. Talbi, E.G.: Parallel combinatorial optimization, vol. 58. John Wiley & Sons (2006)
12. Verhoeven, M.G.A., Aarts, E.H.L.: Parallel local search. Journal of Heuristics **1**(1), 43–65 (1995). DOI 10.1007/BF02430365. URL `https://doi.org/10.1007/BF02430365`

ITC 2019: University Course Timetabling with MaxSAT

Alexandre Lemos · Pedro T. Monteiro · Inês Lynce

Abstract This paper describes the *UniCorT* tool designed to solve **uni**versity **cou**rse **t**imetabling problems specifically tailored for the 2019 International Timetabling Competition (ITC 2019). The proposed approach comprehends pre-processing, the use of a Maximum Satisfiability (MaxSAT) solver, and a local search procedure.

UniCorT is assessed with the benchmark instances from ITC 2019. The impact of a handful of techniques in the quality of the solution and the execution time is evaluated. We take into account different pre-processing techniques and Conjunctive Normal Form (CNF) encoding, as well as the combination with a local search procedure. The success of our tool is attested by having been ranked among the five finalists of the ITC 2019 competition.

Keywords ITC 2019 · MaxSAT · University Course Timetabling

1 Introduction

The University Course Timetabling Problem (UCTTP) was introduced in the context of the fourth International Timetabling Competition (ITC) 2019 [1]. UCTTP can be informally defined as two complementary problems: (i) course timetabling; and (ii) student sectioning. The goal of *course timetabling* is to find a feasible assignment for all the classes of all courses to a time slot and a room, subject to a set of time constraints. The goal of *student sectioning* is to

The authors would like to thank the reviewers for their helpful comments and suggestions that contributed to an improved manuscript. This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference SFRH/BD/143212/2019 (PhD grant), DSAIPA/AI/0033/2019 (project LAIfeBlood), DSAIPA/AI/0044/2018 (project Data2help) and UIDB/50021/2020 (INESC-ID multi-annual funding).

Instituto Superior Técnico, Universidade de Lisboa
INESC-ID, Rua Alves Redol 9, 1000-029 Lisboa, Portugal
E-mail: {alexandre.lemos,pedro.tiago.monteiro,ines.lynce}@tecnico.ulisboa.pt

section students into all the classes required by the courses they are enrolled in, subject to capacity and time constraints.

Timetabling problems have been encoded in the past into propositional logic [2, 3]. This approach has the advantage of making use of propositional satisfiability (SAT) solvers, which are well-known for being quite effective [4].

Example 1 Let us consider four Boolean variables $r^{r_1}_{c_1}$, $r^{r_2}_{c_1}$, $r^{r_1}_{c_2}$ and $r^{r_2}_{c_2}$ representing the possible assignments of the classes c_1 and c_2 to all available rooms (r_1 and r_2). Furthermore, let us consider that the classes are taught at the same time, and thus they cannot be taught in the same room. The same room constraint is encoded as follows: $\neg r^{r_1}_{c_1} \vee \neg r^{r_1}_{c_2}$ and $\neg r^{r_2}_{c_1} \vee \neg r^{r_2}_{c_2}$. A possible solution to this SAT problem is the assignment of class c_1 to room r_1 and class c_2 to room r_2, *i.e.* $r^{r_1}_{c_1} = 1$, $r^{r_2}_{c_1} = 0$, $r^{r_1}_{c_2} = 0$ and $r^{r_2}_{c_2} = 1$. Naturally, for larger domains, one may need to encode cardinally constraints.

UCTTP usually requires to optimize a set of non-mandatory (soft) constraints. Therefore, in this paper we use a maximum satisfiability solver. The maximum satisfiability problem (MaxSAT) [4] is an optimization version of the SAT problem.

This paper provides a detailed description and evaluation of the MaxSAT-based approach that was ranked amongst the five finalists of ITC 2019. The resulting tool *UniCorT* combines pre-processing methods, a MaxSAT solver and a local search procedure to solve UCTTP. We use a MaxSAT solver to find a complete solution to a problem instance, followed by a local search procedure to further optimize the solution. We evaluate two different encodings within *UniCorT* [1]. This tool is evaluated with the large data sets from the ITC 2019 benchmark [1]. Furthermore, we discuss the advantages and disadvantages of the different components of the implementation submitted to ITC 2019.

This paper is organized as follows. Section 2 provides the required background on UCTTP and MaxSAT solving. Section 3 formally describes the ITC 2019 problem. Section 4 describes *UniCorT*. Section 4.1 details the pre-processing techniques. Section 4.2 describes the two different MaxSAT encodings for the course timetabling and student sectioning problems. Section 4.3 describes the local search. Section 5 analyses the evaluation of the proposed approach considering different encodings. The impact of different pre-processing techniques is also taken into account. Finally, Section 6 concludes the paper and discusses possible future directions.

2 Background

This section provides a background on MaxSAT, followed by an overview of existing approaches to solve the UCTTP.

[1] One of these encodings has already been successfully applied to solve the *minimal perturbation problem* in a university course timetabling setting [5]. The paper describing the encoding is available at `http://web.tecnico.ulisboa.pt/~alexandre.lemos/papers/CPAIOR20.pdf`.

2.1 MaxSAT

A propositional formula in conjunctive normal form (CNF) is defined as a conjunction of clauses, where a clause is a disjunction of literals and a literal is either a Boolean variable x or its complement $\neg x$. The propositional satisfiability (SAT) problem consists of deciding whether there is a truth assignment to the variables such that a given CNF formula is satisfied. A formula is satisfied iff there is at least one assignment where all the clauses are satisfied. A clause is satisfied iff there is at least one literal satisfied. Nowadays, most SAT solvers apply conflict-driven clause learning algorithms [6, 7], which are based on the well-known Davis-Putnam algorithm [8] (see [4] for more details).

The MaxSAT problem is an optimization version of SAT, where the *objective* is to find an assignment that maximizes the number of satisfied clauses. A partial MaxSAT formula ($\varphi = \varphi_h \cup \varphi_s$) consists of a set of hard clauses (φ_h) and a set of soft clauses (φ_s). The *objective* in partial MaxSAT is to find an assignment such that all hard clauses in φ_h are satisfied, while maximizing the number of satisfied soft clauses in φ_s.

In this paper, we consider the weighted variant of partial MaxSAT where there is a function $w^{\varphi} : \varphi_s \to \mathbb{N}$ associating an integer weight to each soft clause. In this case, the *objective* is to satisfy all the clauses in φ_h and maximize the total weight of the satisfied clauses in φ_s.

Example 2 Recall example 1, where $r_{c_1}^{r_1} = 1$, $r_{c_1}^{r_2} = 0$, $r_{c_2}^{r_1} = 0$ and $r_{c_2}^{r_2} = 1$ was a feasible solution. Now, let us consider that the assignment of class c_1 to room r_1 has a penalty of 1 associated. For this reason, we add $\neg r_{c_1}^{r_1}$ as a soft clause with weight 1. The previously found solution has now a cost 1 and it is not optimal. The optimal solution is $r_{c_1}^{r_1} = 0$, $r_{c_1}^{r_2} = 1$, $r_{c_2}^{r_1} = 1$ and $r_{c_2}^{r_2} = 0$.

Most MaxSAT solvers [9, 10] call a SAT solver iteratively to improve the quality of the solution. There are different algorithms to guide the search. In this work, we use the linear search with clusters algorithm [11]. The basic idea is the following. We start with a formula where all clauses, including the soft clauses, are considered hard. If a solution is found, then the process ends with the optimal solution. Otherwise, the SAT solver is restarted with a relaxed formula. The relaxed formula consists of adding one new variable to each soft clause. Additionally, we add a constraint imposing a limit on the number of relaxed clauses. This limit is incremented each time the formula is not satisfied. This process ends when a solution is found, or when it is impossible to satisfy all the hard clauses.

In general, we assume that all formulas are encoded into CNF. Nevertheless, we will write some constraints in pseudo-Boolean (PB) form for the sake of readability. PB constraints are nothing more than linear constraints over Boolean variables, and can be written as follows: $\sum q_i x_i \ \text{OP} \ K$, where K and all q_i are integer constants, all x_i are Boolean variables, and $\text{OP} \in \{<, \leq, =, \geq, >\}$. PB constraints can be easily translated into CNF [12]. In this work, we tested different encodings for PB constraints.

Example 3 Consider the following PB constraint: $\sum_{i=0}^{2} r_{c_1}^{r_i} \leq 1$. The constraint ensures that the class c_1 can only be assigned to at most one room. One possible CNF encoding is as follows: $(\neg r_{c_1}^{r_0} \vee \neg r_{c_1}^{r_1}) \wedge (\neg r_{c_1}^{r_0} \vee \neg r_{c_1}^{r_2}) \wedge (\neg r_{c_1}^{r_1} \vee \neg r_{c_1}^{r_2})$.

2.2 University Course Timetabling

UCTTP is known to be NP-complete [13, 14]. The organization of timetabling competitions in the past has led to important advances in solving UCTTP [15, 16]. In the literature, there are several different approaches to solve UCTTP, namely: Constraint Programming (CP) [17, 18], Answer Set Programming (ASP) [19], Boolean Satisfiability (SAT) [2], Maximum Satisfiability (MaxSAT) [3, 5], Integer Linear Programming (ILP) [20–22] and local search [20, 23].

Lemos *et al.* [24] proposed two integer programming models to solve university timetabling problems. The *Boolean* model that used two decision variables to describe the assignment of a class to a time slot and the assignment of a class to a room. The authors also proposed a *mixed* model that had an integer variable representing the start time of class and a Boolean variable representing the assignment of a class to a room. The *direct* model presented in this paper can be seen as the extension of the *Boolean* model.

In the context of SAT, Asín Achá *et al.* [3] proposed a CNF encoding with four types of decision variables to solve curriculum-based course timetabling with data from ITC-2007. The authors proposed variables to described: the day of the class; the hour of the class; the room of the class, and finally the different times a curriculum is taught. Obviously, the problem is different from ours. For example, in ITC 2019 the classes can be scheduled in different weeks.

Later, Lemos *et al.* [5] proposed a CNF encoding with four types of decision variables to solve the *minimal perturbation problem* applied to UCTTP with data from ITC 2019. The authors proposed variables to describe: the week of the class; the day of the class; the hour of the class and finally the room of the class. This *linked* encoding can be seen as the extension to the work proposed in [3].

3 Problem Definition

In this section, we formally describe the ITC 2019 problem adapted from [1]. Let us consider a set of courses Co. A course ($co \in Co$) is composed by a set of classes C_{co}. These classes are characterized in configurations ($Config_{co}$) and organized in parts ($Parts_{config}$). A student must attend the classes from a single configuration. A student enrolled in the course co and attending the configuration $config \in Config_{co}$ must attend *exactly one* class from each part $Parts_{config}$. The set of classes belonging to $part \in Parts_{config}$ is represented by C_{part}.

All classes C (from different courses) must have a schedule assigned to them. Each class $c \in C$ has a set of possible periods (P_c) to be scheduled

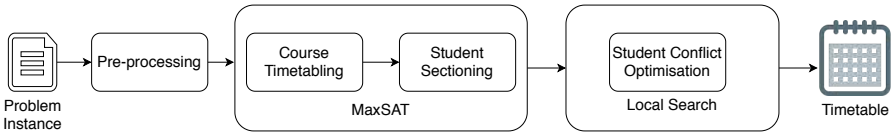


Fig. 1: Overall schema of *UniCorT*.

in. Class c has a hard limit on the number of students that can attend it (lim_c). A class c may have a set of possible rooms (R_c). Each room $r \in R_c$ has $capacity \geq lim_c$. Each class may also have a parent-child relation with another class, *i.e.*, a student enrolled in class c must also be enrolled in the parent $parent_c$.

A time period p corresponds to a 4-tuple (W_p, D_p, h_p, len_p) denoting a set of weeks (W_p), a set of days (D_p), an hour (h_p), and its duration ($len_p > 1$).

Let us consider a set of rooms R where the classes can be scheduled. The travel time, in slots, between two rooms $r_1, r_2 \in R$ is represented by $travel_{r_2}^{r_1}$. Each room $r \in R$ has a set of unavailable periods P_r.

Given a set of students S, each student $s \in S$ is enrolled in a set of courses Co_s. Furthermore, UCTTP is subject to a set of constraints ($constraint_c$ is the set of constraints relating to class c) that can be divided into hard or soft. For brevity, we defined the constraints as needed in the encoding section (for a full description see [1]).

4 Proposed Solution

In this section, we describe *UniCorT*. Figure 1 describes the overall schema of the tool, which has three separate components: pre-processing the UCTTP instance; using a MaxSAT solver to find a solution; and improving the quality of the solution with a local search procedure.

4.1 Pre-processing

The pre-processing component relies on two techniques: (i) identifying of independent sub-instances; and (ii) merging students with exactly the same course enrollment plan.

Technique (i) divides the problem instance into self-contained sub-instances. A set of sub-instances ($Inst$) are self-contained if and only if the following four constraints are upheld:

1. $\forall_{i_1,i_2 \in Inst}\ Co_{i_1} \cap Co_{i_2} = \emptyset$;
2. $\forall_{i_1,i_2 \in Inst}\ R_{i_1} \cap R_{i_2} = \emptyset$;
3. $\forall_{i_1,i_2 \in Inst}\ S_{i_1} \cap S_{i_2} = \emptyset$;
4. $\forall_{i_1,i_2 \in Inst}\ \forall_{c \in C_{i_1}} constraint_c \cap C_{i_2} = \emptyset$.

If these constraints are upheld then we can split the instances without removing any non redundant solution. Note that it is possible to consider a relaxation of this procedure.

Example 4 Let us consider an instance *Inst* with five courses $co_1, \ldots, co_5 \in Co_{Inst}$ and five rooms $r_1, \ldots, r_5 \in R_{Inst}$. The classes of the courses co_1 and co_2 can only be taught in two rooms r_1 and r_2. The classes of the courses co_3, co_4 and co_5 can only be taught in rooms r_3, r_4 and r_5, respectively. Therefore, we can create four sub-instances $i_1, \ldots, i_4 \in Inst$ such that $co_1, co_2 \in Co_{i_1}$, $co_3 \in Co_{i_2}$, $co_4 \in Co_{i_3}$ and $co_5 \in Co_{i_4}$.

Consider three students $s_1, s_2, s_3 \in S_{Inst}$ with the following enrollments: s_1 is enrolled in courses co_1, co_2; s_2 is enrolled in courses co_3, co_4; and finally s_3 is enrolled in co_5. The student enrollments reduces the number of sub-instances since sub-instances i_2 and i_3 violate constraint 3. These two sub-instances must be solved together.

Consider a *no overlap* constraint between the classes of the courses co_4 and co_5. The sub-instances i_3 and i_4 violate constraint 4. For this reason, the instance *Inst* can only be split into two self-contained sub-instances such that $co_1, co_2 \in Co_{i_1}$ and $co_3, co_4, co_5 \in Co_{i_2}$.

Technique (ii) reduces the number of variables and constraints by creating groups of students that share the same curricular plan [25, 26]. The following example illustrates the identification of groups of students with the same curricular plan.

Example 5 Let us consider three courses $co_1, \ldots, co_3 \in Co$ and six students $s_1, \ldots, s_6 \in S$ that are enrolled in courses as follows: $s_1, \ldots, s_4$ are enrolled in the courses co_1 and co_2; and s_5, s_6 are enrolled in the courses co_2 and co_3. In this example, it is possible to generate two *perfect* clusters: clu_1 for all the students enrolled in courses co_1 and co_2; and clu_2 for all the students enrolled in courses co_2 and co_3.

However, this process may remove all the feasible solutions since the classes of each course may have a limitation on the number of students enrolled. Let us denote the greatest common divisor (GCD) between the numbers n_1 and n_2 as $GCD(n_1, n_2)$. Consider now an expansion of the previous example.

Example 6 Let us revisit the Example 5 and consider that each course has two classes, and so $c_1, c_2 \in C_{co_1}, c_3, c_4 \in C_{co_2}$ and $c_5, c_6 \in C_{co_1}$. A student enrolled in the courses co_1, co_2, co_3 must attend exactly one class. The limit on the number of students that can attend is, for each class, as follows: $lim_{c_1} = lim_{c_2} = 4$; $lim_{c_3} = lim_{c_4} = 3$; and $lim_{c_5} = lim_{c_6} = 2$. Figure 2 shows the clusters defined in Example 5 and a possible student sectioning to classes. One can see that the solution is infeasible.

For this reason, we computed GCD between the total number of students enrolled in a course and the smallest capacity of the classes of that course. In this case, we obtain: $GCD(4, 4) = 4$ for course co_1; $GCD(6, 3) = 3$ for

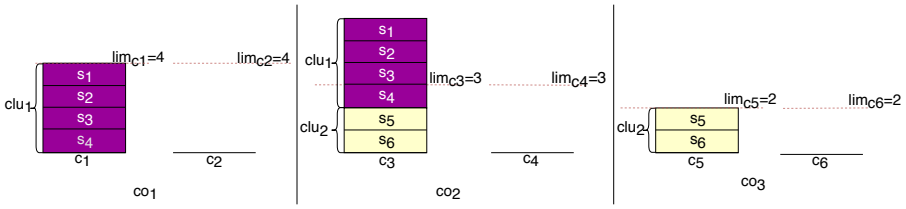


Fig. 2: An infeasible assignment of students to classes based on the clusters defined in Example 5.

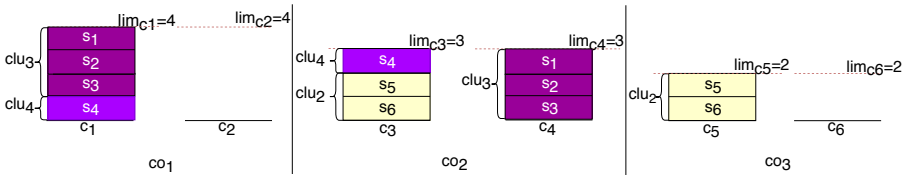


Fig. 3: A feasible assignment of students to classes based on the clusters defined in Example 6.

course co_2; and $GCD(2,2) = 2$ for course co_3. This indicates that the cluster of students enrolled in co_2 needs to be smaller or equal to 3. Therefore, we need to split clu_1. One possibility is to create two clusters. One feasible solution is shown in Figure 3.

This process ensures that it is possible to find a feasible solution to a problem instance, since it is possible to combine all groups of students into classes. However, we may remove the optimal solution by not allowing the assignment of a single student to a given class. The pros and cons of creating clusters are discussed in Section 4.2.3. The GCD can also be used to choose the number of sections of a course in order to reduce the number of conflicts *a priori* [26].

4.2 MaxSAT

In this section, we formally describe two MaxSAT encodings for course timetabling. The two MaxSAT encodings for course timetabling are denoted as: *direct* and *linked* [5]. The ITC 2019 optimization criteria are encoded as soft constraints in both encodings. Furthermore, we also describe a MaxSAT encoding for student sectioning.

4.2.1 Direct Course Timetabling

The most *direct* encoding has only one type of variable to describe the assignment of a class to an allocation slot and a room. This type of encoding

is commonly used to describe scheduling problems [20, 21, 24, 27]. For this reason, we decided to start with this type of encoding in CNF. This encoding can be seen as an expansion of the Boolean encoding we proposed in [24]. Our *direct* encoding to solve course timetabling has only two Boolean decision variables:

- t_c^{slot} represents the assignment of class c to the allocation slot $slot$, with $c \in C$ and $slot \in [0, \ldots, |P_c|]$;
- r_c^{room} represents the assignment of class c to the room $room$, with $c \in C$ and $room \in R_c$.

The separation of the decision variables into two variables allows us to reduce the number of unnecessary variables. Using only one decision variable would increase the amount of memory allocation ($|R_c| \times |P_c|$), where most of these variables are always with value 0.

Most UCTTP constraints have a similar encoding when using the *direct* encoding. For this reason, here we only show a few examples.

In the *direct* encoding we need two types of *exactly one* constraints. We need to ensure that each class is assigned *exactly one* slot and in some cases that each class is assigned *exactly one* room.

Example 7 Let us consider two classes c_1 and c_2 with the following characteristics: $D_{c_1} = D_{c_2} = \{0101, 1010\}$; $W_{c_1} = W_{c_2} = \{11110, 01111\}$; $H_{c_1} = \{10, 11\}$; $H_{c_2} = \{11\}$; $P_{c_1} = \{1, \ldots, 12\}$; $P_{c_2} = \{1, \ldots, 8\}$; $R_{c_1} = \{1, 2\}$ and $R_{c_2} = \emptyset$. In this example, we generate the following *exactly one* constraints: $\sum_{i=1}^{12} t_{c_1}^i = 1$,$\sum_{i=1}^{8} t_{c_2}^i = 1$ and $\sum_{i=1}^{2} r_{c_1}^i = 1$.

All constraints related to time allocations, can be encoded into one binary clause for each pair of classes c_i, c_j where $p_i \in P_{c_i}$ and $p_j \in P_{c_j}$:

$$\neg t_{c_i}^{p_i} \vee \neg t_{c_j}^{p_j}. \tag{1}$$

All constraints that involve both time and room allocation can be encoded as follows:

$$\neg t_{c_i}^{p_i} \vee \neg t_{c_j}^{p_j} \vee \neg r_{c_i}^{room_i} \vee \neg r_{c_j}^{room_j}. \tag{2}$$

The following constraints are encoded the same way for both encodings. In order to ensure that a given class cannot be taught in more than V different days ($MaxDays$(V)) we use an auxiliary variable $dayofweek_d^{const}$, where $const$ is the identifier of the constraint and $d \in \{1, \ldots, |Days|\}$. This variable corresponds to having at least one class, of this constraint, assigned to weekday d. Now, we only need to ensure that:

$$\sum_{c \in C} \sum_{p \in P_c} \sum_{d \in [1, \ldots, |Day_p|]} dayofweek_d^{const} \leq V. \tag{3}$$

In order to ensure that there are no more than V consecutive slots (breaks) throughout a day between a set of classes ($MaxBlock/MaxBreaks$) we need to generate all blocks. After computing all blocks, we add a clause for every class

c_1 to c_n assigned to a period ($p_1 \in P_{c_1}$ to $p_n \in P_{c_n}$) in such a way that it forms a block of classes that breaks one of these constraints:

$$\neg t_{c_1}^{p_1} \vee \ldots \vee \neg t_{c_n}^{p_n}. \tag{4}$$

Example 8 Let us consider two classes c_1 and c_2 that are taught in the same day and cannot overlap in time. Furthermore, all classes are involved in the *MaxBreaks* constraint which ensures that there are 0 breaks (of 1 time slot or more) between them. For simplicity, let us consider that there are only three time slots, t_1, t_2, t_3, per day and all the classes have the same duration of 1 time slot. To ensure that the constraint *MaxBreaks* holds, we add clauses $\neg t_{c_1}^{t_1} \vee \neg t_{c_2}^{t_3}$ and $\neg t_{c_2}^{t_1} \vee \neg t_{c_1}^{t_3}$.

4.2.2 Linked Course Timetabling

It is obvious that we do not need always to take into account the complete schedule information. For some constraints, we only need the information about week, day or hour, and not all the three. For this reason, our *linked* encoding to solve course timetabling has only *four* Boolean decision variables:

- $w_c^{Week_p}$ represents the assignment of class c to the set of weeks $Week_p$, with $c \in C$, and $p \in P_c$;
- $d_c^{Day_p}$ represents the assignment of class c to the set of days Day_p, with $c \in C$, and $p \in P_c$;
- $h_c^{hour_p}$ represents the assignment of class c to the hour $hour_p$, with $c \in C$ and $p \in P_c$;
- r_c^{room} represents the assignment of class c to the room $room$, with $c \in C$ and $room \in R_c$.

The usage of these variables can be seen as an expansion of the encoding proposed in [3]. The scheduling possibilities of a class are usually just a small part of the complete set. For this reason, we only define these variables for acceptable values of the class domain. Furthermore, the *linked* encoding reduces the number of constraints required. For example, *SameStart* constraints (*i.e.* forcing a set of classes to start at the same time) do not require information about the day or week of the class.

In contrast to the *direct* encoding we can reduce the size of each *exactly one* constraint since we have separated the variables for the time allocation. Therefore, we have four *exactly one* constraint for each class (room, hour, day and week). The reduction in the size of the *exactly one* constraints is important since it allows us to avoid a known bottleneck of timetabling encodings using CNF [2].

Example 9 Recall Example 7. The linked encoding for the same instance generates a much smaller number of *exactly one* constraints. In this example, we generate the following *exactly one* constraints: $\sum_{i=1}^{2} w_{c_1}^i = 1$, $\sum_{i=1}^{2} w_{c_2}^i = 1$, $\sum_{i=1}^{2} d_{c_1}^i = 1$, $\sum_{i=1}^{2} d_{c_2}^i = 1$, $\sum_{i=1}^{2} h_{c_1}^i = 1$, $h_{c_2}^{11} = 1$ and $\sum_{i=1}^{2} r_{c_1}^i = 1$.

Once again, most UCTTP constraints have a similar encoding. Therefore, here we only show a few examples. For constraints involving only the variable *hour* (*e.g. SameStart*) we add the clause:

$$\neg h_{c_i}^{hour_{p_i}} \vee \neg h_{c_j}^{hour_{p_j}}. \tag{5}$$

Similarly, the same type of clauses have to be added for constraints involving only the variables *day* (*e.g. SameDay*), *week* (*e.g. SameWeek*) and *room* (*e.g. SameRoom*).

However, not all constraints are so simple to write. For constraints that involve all the time assignments (week, day and hour) we add an auxiliary variable t. This variable has exactly the same meaning that the decision variable of the *direct* encoding. This allows us to generate binary clauses to encode more complex constraints (*e.g.* ensuring that two classes do not *overlap* in time).

To further reduce the size of the clauses, we define the auxiliary variables $sd_{c_j}^{c_i}$ to represent two classes taught in the same day (*i.e.* with at least one day overlap). For each two classes c_i, c_j with $i \neq j$, consider that overlap in days Day_0 to Day_n belong to the domain of class c_i, Day_{n+1} to Day_m belong to the domain of class c_j, with $0 < n < m$. Hence, we add the following equivalence:

$$sd_{c_j}^{c_i} \iff (d_{c_i}^{Day_0} \vee \ldots \vee d_{c_i}^{Day_n}) \wedge (d_{c_j}^{Day_{n+1}} \vee \ldots \vee d_{c_j}^{Day_m}). \tag{6}$$

Similarly, one can define an auxiliary variable $sw_{c_j}^{c_i}$ to represent two classes overlapping in at least one week.

To guarantee that no two classes (c_i, c_j) are taught in the same room (ro) in overlapping times, we add the clause:

$$\neg sd_{c_j}^{c_i} \vee \neg sw_{c_j}^{c_i} \vee \neg h_{c_i}^{hour_{p_i}} \vee \neg h_{c_j}^{hour_{p_j}} \vee \neg r_{c_i}^{ro} \vee \neg r_{c_j}^{ro}. \tag{7}$$

The remaining constraint types are encoded in the same way for both encodings (see previous section).

4.2.3 Student Sectioning

The usage of clusters requires us to define a set $\mathcal{C}luster$ of clusters of students. The number of students merged in the $clu \in \mathcal{C}luster$ is represented by $|clu|$.

In order to solve student sectioning our encoding is extended with *one* decision variable s_{clu}^{c}, where $c \in C$ and $clu \in [1, \ldots, |\mathcal{C}luster|]$. The advantage of the pre-processing step of creating clusters is to reduce the number of variables and constraints required to model students. Note that ITC 2019 instances do not require student sectioning to be *balanced* as in [26].

Example 10 Let us consider again Example 5. Recall that we have three courses $co_1, \ldots, co_3 \in Co$ and six students $s_1, \ldots, s_6 \in S$ that are enrolled in the following courses: students $s_1, \ldots, s_4$ are enrolled in the courses co_1 and co_2; and students s_5, s_6 are enrolled in the courses co_2 and co_3. Therefore, it is possible to generate two *perfect* clusters: clu_1 for all the students enrolled in

courses co_1 and co_2; and clu_2 for all the students enrolled in courses co_2 and co_3. Additionally, consider that $|C_{co_1}| = |C_{co_2}| = 1$ and $|C_{co_3}| = 6$.

The two clusters allow to reduce the number of variables from *22* (the number of students times the number of classes available for each student) to *9* (the number of clusters times the number of classes). Note that the impact of the clusters depends not only on the number of students merged but also on the course composition. In this case, the smaller cluster (clu_2) has a greater impact since the courses have a larger number of classes, more precisely 7 variables.

In order to ensure that a student can only be assigned to a single course configuration, we define an auxiliary variable for each pair configuration-cluster of students. The variable is denoted as $conf_{clu}^{config}$, where $clu \in [1, \ldots, |\mathcal{C}luster|]$, $config \in Config_{co}$ and $co \in Co$.

We need to add an *exactly one* constraint to ensure that each cluster of students id is enrolled in *exactly one* configuration of each course. To ensure that the class capacity is not exceeded, we add the following constraint for each class c:

$$\sum_{clu \in [1,\ldots,|\mathcal{C}luster|]} |clu| \times s_{clu}^{c} \leq lim_c. \tag{8}$$

In addition, to ensure that a cluster of students clu enrolled in a class c is also enrolled in this parent class $parent_c$, we add the following clause:

$$\neg s_{clu}^{c} \vee s_{clu}^{parent_c}. \tag{9}$$

Finally, we need to ensure that a cluster of students clu is enrolled in *exactly one* class of each part of a single configuration of the course co. The conflicting schedule of classes attended by the same cluster of students is represented by a set of weighted soft clauses. For each cluster of students id enrolled in two classes c_i, c_j overlapping in time, we add:

$$\neg s_{clu}^{c_i} \vee \neg s_{clu}^{c_j} \vee \neg sw_{c_j}^{c_i} \vee \neg sd_{c_j}^{c_i} \vee \neg h_{c_i}^{hour_{c_i}} \vee \neg h_{c_j}^{hour_{c_j}}. \tag{10}$$

4.3 Local Search: Student Conflict Optimisation

The *goal* of this procedure is to improve the quality of the solution found without changing the schedule and room assignments of the courses. Neighborhood structures are the basis of this local search (LS) procedure. In this work, the neighborhood consists of small changes in the student sectioning. To create a new neighborhood two operations can be performed: (i) allocating a cluster of students to a different class with empty seats and (ii) *swaping* two clusters of students between classes. Considering these moves, the procedure does not require the knowledge of course timetabling constraints. The LS procedure stops when the neighbors of the best solution cannot reduce the number of conflicts (*i.e.* the solution found has the best cost of its neighborhood).

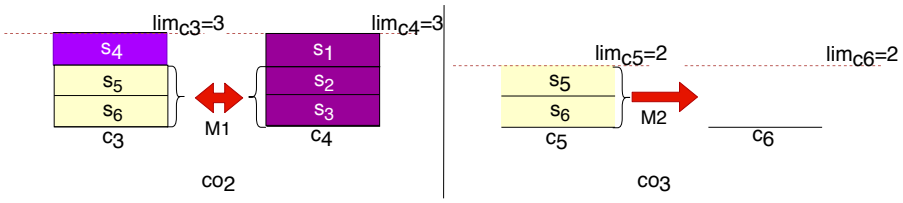


Fig. 4: The two neighbors of the solution in Example 6.

Example 11 Let us consider again Example 6. Additionally, consider that the classes c_3 and c_5 are taught at the same time, on the same day and in the same week. For this reason, the solution shown in Figure 3 has two conflicts (students s_5 and s_6 are sectioned into two classes that overlap in time). This solution would have two neighbors for which the solution would improve. These two neighbors are shown in Figure 4. The neighbor **M1** *swaps* students s_5 and s_6 with students s_2 and s_3 (from class c_3 to c_4). However, this move is not possible since the clusters do not allow to separate the students s_2 and s_3 from s_1. The neighbor **M2** results from is just sectioning students s_5 and s_6 to the class c_6 instead of c_5. This change does not require to break any clusters and reduces the number of conflicts to zero.

5 Experimental Evaluation

In this section, we discuss the main computational results obtained. First, we describe the experimental setup used to validate *UniCorT* (Section 5.1). Next, we discuss our results for UCTTP (Section 5.2). Finally, we present a summary of the results (Section 5.3).

5.1 Experimental Setup

The experimental evaluation was performed on a computer with Fedora 14, with 32 CPUs at 2.6 GHz and 126 Gb of RAM. All results were obtained when running the solver with a time out of 6,000 seconds.

We used the benchmark obtained from ITC 2019 [1] to validate our tool. The benchmark is divided into three groups of instances (early, middle, late). All results were verified by an online validation tool provided by the organizers[2].

UniCorT was implemented in C++, using the MaxSAT solver *TT-Open-WBO-Inc* [10, 28][3] as a black box. *TT-Open-WBO-Inc* is a linked MaxSAT

[2] `https://www.itc2019.org/validator`, accessed in January of 2020

[3] *TT-Open-WBO-Inc* won both the Weighted Incomplete tracks at *MaxSAT Evaluation 2019*. The results are available at `https://maxsat-evaluations.github.io/2019/`.

solver [9] that has different algorithms and encodings to solve a given problem. The results shown in this paper correspond to the best configuration of the parameters of the solver. The solver was executed with the following parameters:

- ⋆ *-algorithm=6* corresponding to the use of linear search with the clusters algorithm [11];
- ⋆ *-pb=2* corresponding to the use of the adder encoding [29] to convert the PB constraints to CNF;
- ⋆ *-amo=0* corresponding to the use of the ladder encoding [30] to convert *exactly one* constraints to CNF.

The linear search with the clusters algorithm uses a lexicographic optimization criterion [31]. Recall that ITC 2019 considers four optimization criteria: the cost of assigning a class to a room; the cost of assigning a class to a time slot; the number of students conflicts; and the weighted sum of violated soft constraints. Each instance has its own weight for each criterion. We have computed the worst possible penalization of each criterion and used the value to order the lexicographic optimization.

The XML parser used to parse the ITC 2019 input file was *RAPIDXML*[4]. Also, we make our implementation available on github (`https://github.com/ADDALemos/MPPTimetables/tree/ITC-2019`).

5.2 Computational Evaluation

In this section, we discuss the results of *UniCorT* and all possible configurations tested.

5.2.1 Pre-processing Techniques

Recall that we discussed two pre-processing techniques: (i) identification of independent sub-instances and (ii) merging students into clusters.

Identification of independent sub-instances The identification of independent sub-instances allows us to split the problem without loosing solutions. In the end, it is just a question of combining all the solutions. On average, we can split an instance into 3 sub-instances. In most cases, the instances have one large instance and two smaller instances. A detailed description of the sub-instances is shown in Table 1.

[4] *RAPIDXML* is available at `http://rapidxml.sourceforge.net/manual.html`, accessed in February 2019.

Table 1: Number of sub-instances found and the respective average size.

	Instance	# Inst	# Classes	
			Avg.	Median
Early	agh-fis-spr17	2	599.5	599.6
	agh-ggis-spr17	3	617	12
	bet-fal17	4	292.25	127
	iku-fal17	4	649.5	140
	mary-spr17	3	281.3	10
	muni-fi-spr16	1	575	575
	muni-fsps-spr17	3	543.3	10
	muni-pdf-spr16c	4	635.25	9.5
	pu-llr-spr17	3	388.6	165
	tg-fal17	1	711	711
Middle	agh-ggos-spr17	2	620.5	620.5
	agh-h-spr17	1	460	460
	lums-spr18	1	487	487
	muni-fi-spr17	2	515	515
	muni-fsps-spr17c	6	130.1	6.5
	muni-pdf-spr16	5	909	2
	nbi-spr18	3	260	34
	pu-d5-spr17	6	389.7	12
	pu-proj-fal19	4	2207	51
	yach-fal17	3	156.3	165
Late	agh-fal17	4	1876.66	10
	bet-spr18	4	340.75	140
	iku-spr18	5	556.4	56
	lums-fal17	1	502	502
	mary-fal18	3	319.6	5
	muni-fi-fal17	1	535	535
	muni-fspsx-fal17	4	326.4	32
	muni-pdfx-fal17	6	1854.83	2.5
	pu-d9-fal19	7	816.42	8
	tg-spr18	1	676	676

Merging students Merging students with the same curricular plans allows to reduce the number of variables and constraints on the student sectioning part of the problem. Figure 5 shows the percentage of the total number of variables required to model students using different clusters. The *clusters* represent the percentage of the total number of variables required to model students with different curricular plans per instance. However, this type of clusters cannot be applied in practice since they would remove feasible solutions (see Example 3). Alternately, the *GCD clusters* represent the cluster divided using the GCD method discussed above. Recall that the number of variables needed to model students is influenced by the number of classes per enrolled course (see Example 10).

Most instances have a significant bottleneck in the creation of clusters caused by the hard limit on the number of students enrolled in a class. On average the GCD clusters are 40 points worse than a normal cluster. On average, one can reduce the number of variables relating to students up to 23%. Instances *nbi-spr18* and *yach-fal17* have a larger reduction on the number of

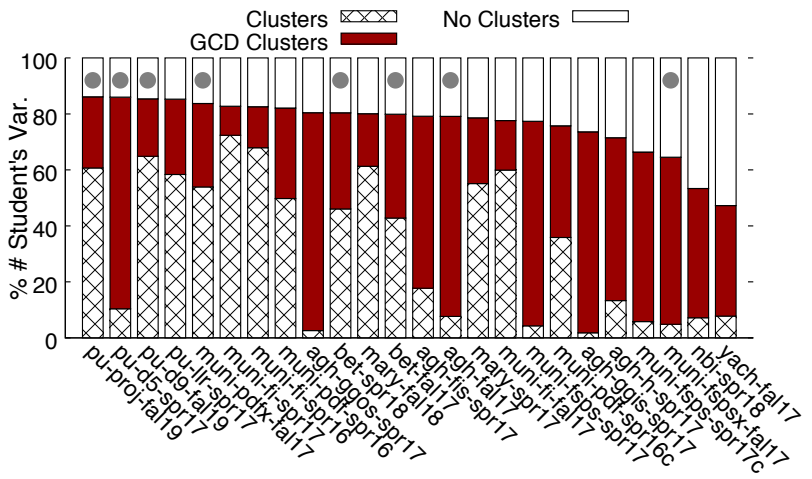


Fig. 5: Percent decrease in the number of variables required to model students increasing clustering strategy. The grey circles represent timed out instances when using GCD clusters.

variables (around 50%). On the other hand, instances *pu** have the smallest reduction (14%).

5.2.2 MaxSAT Solving

In this section, we compare the different CNF encodings and the advantages of solving the course timetabling problem separated from the student sectioning problem.

Figure 6 compares the total number of hard clauses with the number of soft clauses generated by our encodings. It is clear that the number of soft clauses is considerably smaller for all instances. On average, the number of soft clauses is 2% of the global number of clauses. Most instances that timed out have a higher percentage of soft clauses but these instances also have a larger overall number of soft clauses. With this difference in mind, we focused more on the hard constraints as they are dominant.

We can find a solution within the time limit for 20 out of 30 instances using our best approach (see Table 2). However, the solver was not able to prove optimality within the time limit on any of the instances.

Figure 7 compares the number of hard clauses generated by the CNF encoding and the CPU time needed to find the best solution for each instance considering two approaches to encode the problem (*direct* and *linked*). In general, one can see that the instances with a larger number of hard constraints take a larger amount of time. Using the *linked* encoding reduces the number of constraints needed per instance. Therefore, one can solve 9 more instances within the time limit. Most of the unsolved instances actually have two orders

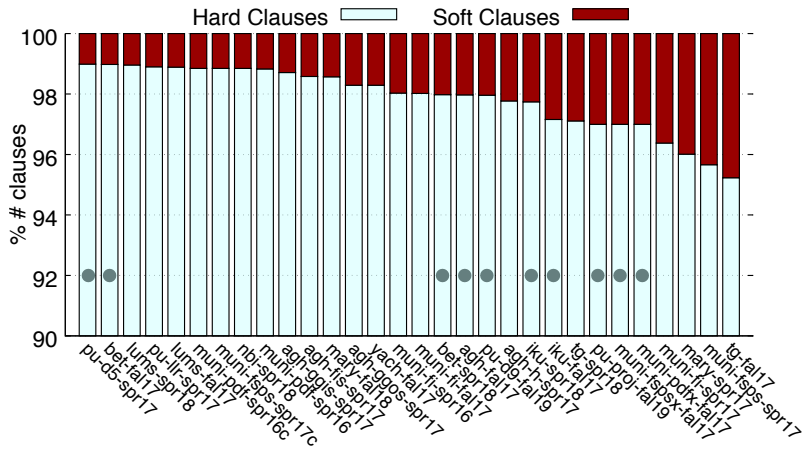


Fig. 6: Percentage of soft clauses for each instance. The grey circles represent timed out instances.

of magnitude more constraints than the other instances (top right corner of Figure 7). Most of these constraints result from the *MaxBlock* and *MaxBreak* constraints (to be discussed further on).

Figure 8 compares the number of hard clauses generated by the CNF encoding for each approach tested. One can see that the *direct* encoding requires much more constraints to encode the same instance. The *direct* encoding requires, on average, 7×10^{10} more constraints than the *linked* encoding. This can be explained by the fact that most constraints are only related either to an hour, day or week. There are few constraints that involve all weeks, days and hours simultaneously. Furthermore, one can reduce the need to combine all these with the usage of auxiliary variables (*e.g. sd*). For this reason, the usage of only one variable for the time allocation problems creates unnecessary constraints.

Figure 9 compares the time spent to find the best solution for each approach tested. One can see that the *direct* encoding requires only a few more seconds to find a solution of the same cost for each instance that both approaches solve within the time limit. The *direct* encoding requires, on average, 200 seconds more than the *linked* encoding. Furthermore, we can clearly see that most instances that timed out with the *direct* encoding are solved in only a few seconds by the *linked* encoding (bottom right of Figure 9). On average, the *linked* encoding requires only 2,000 seconds to solve the timed out instances.

In case of the *linked* encoding, for most instances, the solver requires only a short amount of time to produce the best solution. Figure 10 shows a comparison between the normalized cost of the best found solution and the CPU time in seconds. The figure shows the normalized cost since each instance has its own weights on the optimization criterion and therefore would be impossible

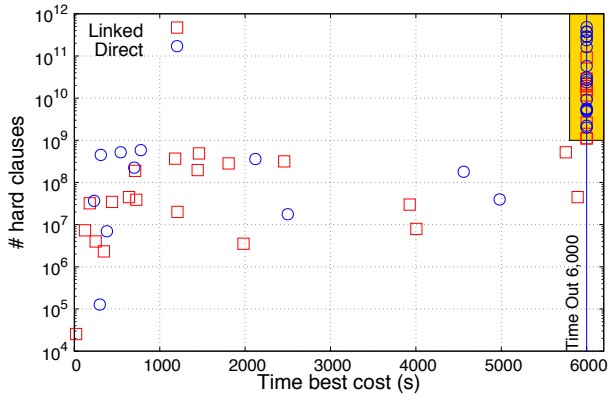


Fig. 7: A comparison between the *linked* and the *direct* encodings in terms of the number of hard constraints (log scale) versus the time spend to find the best solution. The orange square contains the instances that timed out. 33.3% and 63.3% of the instances are in the square for *linked* and the *direct* encodings respectively.

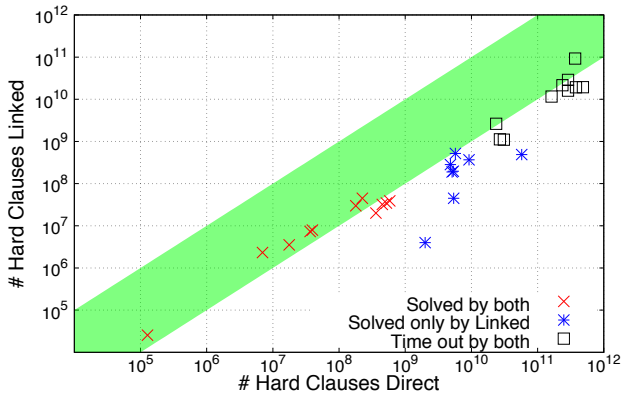


Fig. 8: A comparison between the *linked* and the *direct* encodings in terms of the number of hard constraints (log scale). The blue line represents the time limit.

to compare them in the same graph. One can see that the best solution, for most instances, is found early on (within 2,000 seconds). In fact, only 5 out of 20 instances improve their quality after 2,000 seconds. The quality of the solution does not improve until the time out is reached.

MaxBlocks and MaxBreaks. Figure 11 shows the percentage of clauses generated from *MaxBlocks* and *MaxBreaks* constraints for each instance. One can see that these constraints generate a significant number of additional clauses.

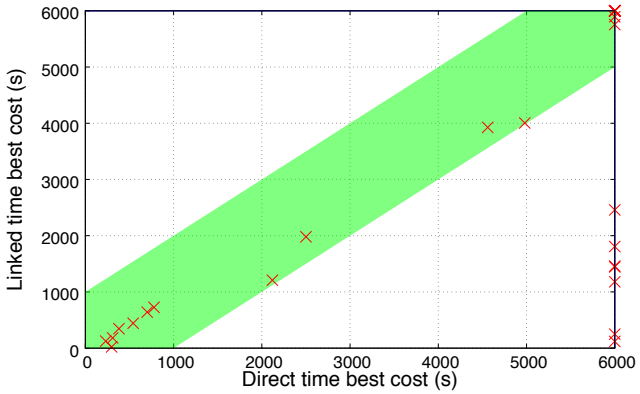


Fig. 9: A comparison between the *linked* and the *direct* encodings in terms of CPU time for each instance.

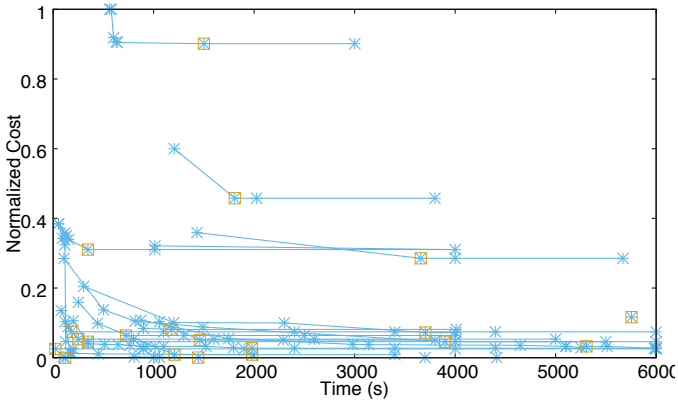


Fig. 10: Normalized cost versus CPU time for each instance with *linked* encoding. The grey square represents the best cost found.

In the worst case, we need to generate over 35% more clauses to deal with these constraints. Note that we cannot solve the 10 instances with a higher number of hard clauses (see Figure 8). In most cases, the high number of constraints is caused by *MaxBlocks* and *MaxBreaks* constraints. The exceptions are the instances from *iku**, which have the largest number of classes. In fact, the size of our *exactly one* constraints is much larger than 26 which is the limit found by Bittner *et al.* [2] for solvable instances.

Decomposing UCTTP. Our best approach decomposes the UCTTP into two sub-problems: (i) course timetabling and (ii) student sectioning. This decomposition may remove the optimal solution. However, it does not remove any feasible solution. The goal of decomposition is to reduce the size of the prob-

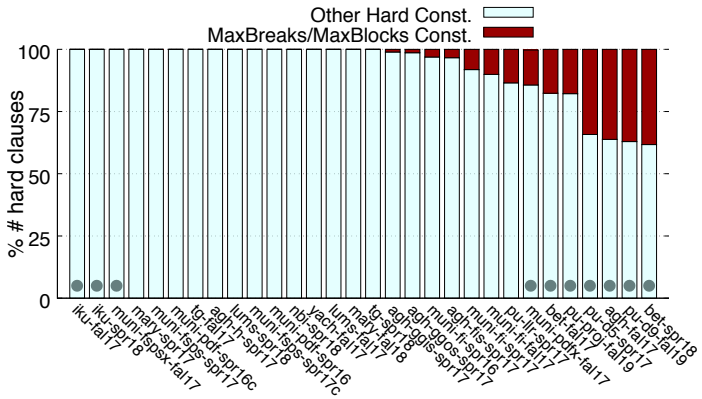


Fig. 11: Percentage of clauses generated by *MaxBlocks* and *MaxBreaks* constraints. The grey circles represent timed out instances.

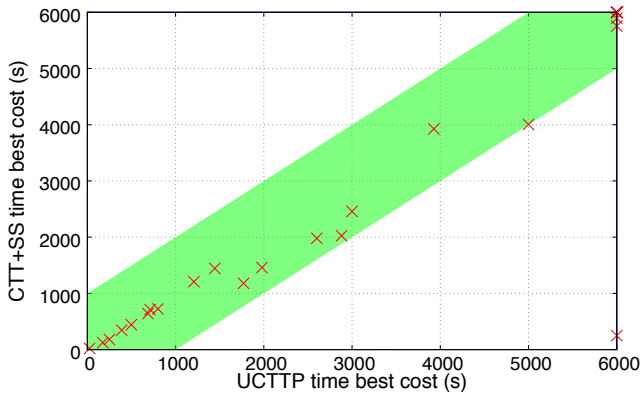


Fig. 12: A comparison of the CPU time, in seconds, when solving the *CTT+SS* problems separated or the *UCTTP* as a whole.

lem, especially for instances with a large number of clusters of students. The decomposition allows us to solve 3 more instances. Figure 12 compares the performance of the solver before and after decomposing the problem, in terms of CPU time.

5.2.3 Local Search

Our straightforward implementation of this method allows to improve the quality of the solution without adding significant overhead. On average, the method requires only 6% of the overall execution time of the approach. Fig-

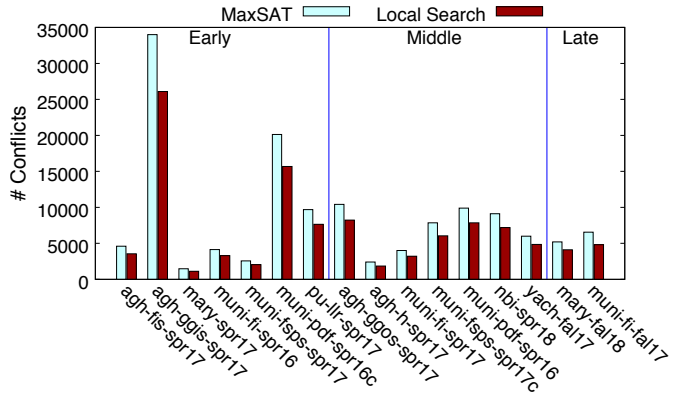


Fig. 13: A comparison of the cost, in terms of students conflicts, before and after applying the LS procedure.

ure 13 compares the number of conflicts, before and after this procedure. On average the procedure reduces the number of conflicts by 22%.

5.3 Final Results

Table 2 shows the best cost found by our best approach per optimization criteria and instance. Note that the penalties associated with the three optimization criteria (student conflicts, allocation penalty, and additional soft constraints) vary from instance to instance. Therefore, it is difficult to compare them. Nevertheless, one can see that the student conflict criteria, overall, is the most costly even with the LS method. The *muni** instances are on average the worst in terms of room allocation penalty. This can be explained by the normal structure of these instances since they have few room options (R_c) and a large penalty associated.

6 Conclusion and Future Work

This paper discusses the results obtained by our approach in ITC 2019. The resulting tool *UniCorT* is able to solve two thirds of the benchmark instances from ITC 2019 within the time limit of 6,000 seconds. This tool placed among the five finalists. *UniCorT* takes advantage of two pre-processing techniques that search for: (i) self-contained sub-instances and (ii) clusters of students. The first method is able to divide, on average, an instance into 3 sub-instances. The clustering of students is able to reduce the number of variables used, on

Table 2: The cost per optimization criteria and instance.

Instance	Cost	Students	Time	Room	Distribution
agh-fis-spr17	35139	3555	2248	2312	404
agh-ggis-spr17	194138	26097	2737	22270	2029
bet-fal17	UKN				
iku-fal17	UKN				
mary-spr17	51147	1114	1376	805	7290
muni-fi-spr16	19314	3286	352	628	120
muni-fsps-spr17	211142	2040	58	292	360
muni-pdf-spr16c	567900	15678	58316	27600	4361
pu-llr-spr17	68003	7642	1169		30
tg-fal17	6774	0	1792	30	158
agh-ggos-spr17	79745	8230	6045	9045	358
agh-h-spr17	55887	1848	1442	1039	2656
lums-spr18	594	0	0	509	17
muni-fi-spr17	18080	3212	284	958	21
muni-fsps-spr17c	618217	6048	411	1027	141
muni-pdf-spr16	310994	7853	38680	27094	900
nbi-spr18	49924	7196	5946	9208	5
pu-d5-spr17	UKN				
pu-proj-fal19	UKN				
yach-fal17	32198	4856	8	1008	687
agh-fal17	UKN				
bet-spr18	UKN				
iku-fal18	UKN				
lums-fal17	1151	0	105	626	63
mary-fal18	44097	4107	596	665	234
muni-fi-fal17	19683	3810	86	289	9
muni-fspsx-fal17	UKN				
muni-pdfx-fal17	UKN				
pu-d9-fal19	UKN				
tg-spr18	31900	0	1942	3996	1201

average, by 23%. The LS method, in the end, is able to reduce the number of conflicts by 22% without adding a significant overhead.

UniCorT solves the course timetabling and student sectioning problems separately in order to reduce the size of the problem and thus the execution time. This decomposition does not remove any feasible solutions. However, it may remove the optimal solution but allows us to solve more instances within the time limit.

The MaxSAT encodings applied in *UniCorT* encode *MaxBlock* and *MaxBreaks* constraints by blocking all invalid assignments. In order to block the invalid assignments, we generate all block combinations possible. However, this method proves inefficient for large instances. For this reason, we plan to work on new ways of encoding these constraints in such a way we avoid enumerating all possible blocks. More precisely, we can take advantage of symmetries in the blocks structure to reduce the clauses generated.

References

1. Müller, T., Rudová, H., Müllerová, Z.: University course timetabling and International Timetabling Competition 2019. In: Burke, E.K., Di Gaspero, L., McCollum, B., Musliu, N., Özcan, E. (eds.) Proceedings of the 12th International Conference on the Practice and Theory of Automated Timetabling (PATAT-2018). pp. 5–31 (2018)
2. Bittner, P.M., Thum, T., Schaefer, I.: SAT encodings of the at-most-k constraint - A case study on configuring university courses. In: Proceedings of the Software Engineering and Formal Methods (SEFM). pp. 127–144 (2019)
3. Asín Achá, R.J., Nieuwenhuis, R.: Curriculum-based course timetabling with SAT and MaxSAT. Annals of Operations Research **218**(1), 71–91 (2014)
4. Biere, A., Heule, M., van Maaren, H.: Handbook of satisfiability, vol. 185. IOS press (2009)
5. Lemos, A., Monteiro, P.T., Lynce, I.: Minimal perturbation in university timetabling with maximum satisfiability. In: Proceedings of 17th International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR) (2020), preprint at `http://web.tecnico.ulisboa.pt/alexandre.lemos/papers/CPAIOR20.pdf`
6. Silva, J.P.M., Sakallah, K.A.: GRASP - a new search algorithm for satisfiability. In: Rutenbar, R.A., Otten, R.H.J.M. (eds.) Proceedings of the International Conference on Computer-Aided Design (ICCAD). pp. 220–227. IEEE Computer Society / ACM (1996)
7. Jr., R.J.B., Schrag, R.: Using CSP look-back techniques to solve real-world SAT instances. In: Kuipers, B., Webber, B.L. (eds.) Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI) and Ninth Innovative Applications of Artificial Intelligence Conference (IAAI). pp. 203–208. AAAI Press / The MIT Press (1997)
8. Davis, M., Putnam, H.: A computing procedure for quantification theory. J. ACM **7**(3), 201–215 (1960)
9. Martins, R., Manquinho, V.M., Lynce, I.: Open-WBO: A modular MaxSAT solver,. In: Theory and Applications of Satisfiability Testing (SAT) - 17th. pp. 438–445 (2014)
10. Nadel, A.: TT-Open-WBO-Inc: Tuning polarity and variable selection for anytime SAT-based optimization. In: Proceedings of the MaxSAT Evaluations (2019)
11. Joshi, S., Kumar, P., Martins, R., Rao, S.: Approximation strategies for incomplete MaxSAT. In: Principles and Practice of Constraint Programming (CP). pp. 219–228 (2018)
12. Eén, N., Sörensson, N.: Translating pseudo-boolean constraints into SAT. Journal on Satisfiability, Boolean Modeling and Computation **2**(1-4), 1–26 (2006)
13. Lovelace, A.L.: On the complexity of scheduling university courses. Master's thesis, California Polytechnic State University, San Luis Obispo

(2010)
14. Herres, B., Schmitz, H.: Decomposition of university course timetabling. Annals of Operations Research (2019)
15. McCollum, B.: University timetabling: Bridging the gap between research and practice. In: 5th International Conference on the Practice and Theory of Automated Timetabling (PATAT). pp. 15–35. Springer (2006)
16. Vrielink, R.A.O., 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)
17. Müller, T.: ITC-2007 solver description: a hybrid approach. Annals of Operations Research **172**(1), 429 (2009)
18. Atsuta, M., Nonobe, K., Ibaraki, T.: ITC-2007 track 2: an approach using a general CSP solver. In: 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT). pp. 19–22 (2008)
19. Banbara, M., Inoue, K., Kaufmann, B., Okimoto, T., Schaub, T., Soh, T., Tamura, N., Wanko, P.: *teaspoon* : Solving the curriculum-based course timetabling problems with Answer Set Programming. Annals of Operations Research **275**(1), 3–37 (2019)
20. Lemos, A., Melo, F.S., Monteiro, P.T., Lynce, I.: Room usage optimization in timetabling: A case study at Universidade de Lisboa. Operations Research Perspectives **6**, 100092 (2019)
21. Lindahl, M., Stidsen, T., Sørensen, M.: Quality recovering of university timetables. European Journal of Operational Research **276**(2), 422 – 435 (2019)
22. Phillips, A.E., Walker, C.G., Ehrgott, M., Ryan, D.M.: Integer programming for minimal perturbation problems in university course timetabling. Annals of Operations Research **252**(2), 283–304 (2017)
23. Gülcü, A., Akkan, C.: Robust university course timetabling problem subject to single and multiple disruptions. European Journal of Operational Research **283**(2), 630 – 646 (2020)
24. Lemos, A., Melo, F.S., Monteiro, P.T., Lynce, I.: Disruptions in Timetables: A Case Study at Universidade de Lisboa. Journal of Scheduling (2020)
25. Carter, M.W.: A comprehensive course timetabling and student scheduling system at the University of Waterloo. In: 3rd International Conference on the Practice and Theory of Automated Timetabling (PATAT). pp. 64–84 (2000)
26. Schindl, D.: Optimal student sectioning on mandatory courses with various sections numbers. Annals of Operations Research **275**(1), 209–221 (2019)
27. Burke, E.K., Mareček, J., Parkes, A.J., Rudová, H.: Penalising patterns in timetables: Novel integer programming formulations. In: Operations Research Proceedings, pp. 409–414. Springer (2008)
28. Nadel, A.: Anytime weighted MaxSAT with improved polarity selection and bit-vector optimization. In: Proceedings of the 19th Conference on Formal Methods in Computer Aided Design (FMCAD) (2019)

29. Warners, J.P.: A linear-time transformation of linear inequalities into conjunctive normal form. Information Processing Letters **68**(2), 63–69 (1998)
30. Ansótegui, C., Manyà, F.: Mapping problems with finite-domain variables into problems with boolean variables. In: Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT). vol. 3542, p. 1–15 (2004)
31. Marques-Silva, J., Argelich, J., Graça, A., Lynce, I.: Boolean lexicographic optimization: algorithms & applications. Annals of Mathematics and Artificial Intelligence **62**(3-4), 317–343 (2011)

An Online Learning Selection Hyper-heuristic for Educational Timetabling

Alexander Steenson · Ender Özcan · Ahmed Kheiri · Barry McCollum · Paul McMullan

Abstract Examination and course timetabling are computationally difficult real-world resource allocation problems. In 2007, an International Timetabling Competition (ITC) consisting of three classes; (i) examination timetabling, (ii) post enrollment-based, and (iii) curriculum-based course timetabling was organised. One of the competing algorithms, referred to as CPSolver, successfully achieved the first place in two out of these three tracks. This study investigates the performance of various multi-stage selection hyper-heuristics sequencing low-level heuristics/operators extending the CPSolver framework which executes hill climbing and two well-known local search metaheuristics in stages. The proposed selection hyper-heuristic is a multi-stage approach making use of a matrix which maintains transitional probabilities between each low-level heuristic to select the next heuristic in the sequence. A second matrix tracks the probabilities of ending the sequence on a given low-level heuristic. The best configuration for the selection hyper-heuristic is explored tailoring the heuristic selection process for the given timetabling problem class. The empirical results on the ITC 2007 problem instances show that the proposed selection hyper-heuristics can reduce the number of soft constraint violations, producing improved solutions over CPSolver as well as some other previously proposed solvers, particularly, in examination and curriculum-based course timetabling.

A. Steenson (corresponding author), E. Özcan
University of Nottingham, Nottingham, UK
E-mail: {psyajs,ender.ozcan}@nottingham.ac.uk

A. Kheiri
Lancaster University, Lancaster, UK
E-mail: a.kheiri@lancaster.ac.uk

B. McCollum, P. McMullan
Queen's University, Belfast, UK
E-mail: {b.mccollum,p.mccmullan}@qub.ac.uk

1 Introduction

Educational timetabling represents a class of challenging real-world problems which are still of interest to many researchers and practitioners. The International Timetabling Competition (ITC) 2007 provided researchers with models of the problems faced, which incorporate an increased number of real-world constraints[1]. In 2007 the ITC was composed of three tracks[2]:

- **Examination Timetabling** where a set of exams must be assigned into a predefined examination time period.
- **Post Enrolment-based Course Timetabling** where a course timetable must be constructed after students have selected and enrolled onto a set of courses.
- **Curriculum-based Course Timetabling** where university lectures, for several courses, must be assigned a room and timeslot within a set of restrictions.

The tracks describe common problems that universities face around the world when scheduling exams or courses. When scheduling these timetables, no hard constraints and as little soft constraints as possible should be violated. Violating no hard constraints is the top priority when creating a timetable, in order to make it feasible and therefore workable. For example, whilst scheduling for an examination timetable, no student can be assigned to take more than one exam at any given time, and no more students than available seats can be assigned to a given room. Soft constraints however, may be violated and still result in a feasible timetable. Soft constraints represent preferences whilst scheduling, to aid in the smooth running and efficiency of the timetable. An example of a soft constraint in the Examination track is 'two exams in a row'; it is preferred if students would not take multiple exams 'back to back'. Therefore, the aim is to minimise soft constraints whilst not violating hard constraints.

The ITC 2007 released a number of data instances to the research community throughout the duration of the competition. This gave researchers the opportunity to develop new and innovative approaches to solving the problems outlined in each track, on real life datasets. As a result, many successful methods have been developed to solve these types of problems. Five finalists were selected based on their submitted results on the released datasets. The solvers developed by the finalists were then judged further using a set of 'hidden' datasets that were not released for testing and development, and again using the previously released datasets. It is important to note that computation time was limited on each solver ensuring a fair outcome. The winner of the Examination track and the Curriculum-based Timetabling track was Thomàš Müller, who developed a hybrid constraint-based solver, known as CPSolver, as part of his PhD to compete in all three competition tracks [23]. This single solution framework was capable of constructing and refining solutions for each

[1] http://www.cs.qub.ac.uk/itc2007/index.htm
[2] http://www.cs.qub.ac.uk/itc2007/index_files/competitiontracks.htm

track, by employing a series of algorithms, including hill climbing and local search metaheuristics [29], namely, Great Deluge and Simulated Annealing, in stages, where each operate over feasible, though not necessarily complete solutions. Those algorithmic components may need to be heavily modified and reconfigured to be applied to another problem [17].

Hyper-heuristics [10] are high-level control methods used in heuristic optimisation that operate over the search space formed by a set of low-level heuristics rather than the search space of solutions, directly. The hyper-heuristics are classified into two as either being generative (generating low-level heuristics) or selective (selecting from a pool of low-level heuristics). This paper will be focusing purely on the latter one. A selection hyper-heuristic attempts to choose the right method or sequence of (meta)heuristics in a given situation at each step or stage during the search process [4,12]. This paper investigates the effects of implementing a sequence-based heuristic selection algorithm within Thomàš Müller's CPSolver framework to uncover Hidden Markov chains and generate heuristic sequences tailored to the problem instance. This paper provides an explanation of the implementation of the sequence-based hyper-heuristic within the CPSolver framework. Finally, results regarding the effectiveness of the implemented hyper-heuristic against Thomàš Müllers original ITC 2007 results, along with the results achieved by the other ITC 2007 finalists and some post ITC 2007 solutions can be found at the end of this paper.

2 Related Work

Many approaches have been taken to solve the problem instances provided as part of the competition. These approaches range from hyper-heuristics capable of solving each instance from each track with minimal domain specific requirements, to domain-specific solvers which are only capable of solving the instances of one given track. A number of different methods have been developed to solve a problem for each track individually. For the Examination track the following approaches have been proposed: graph colouring constructive hyper-heuristic [28], tabu search [11], simulated annealing [13] and cell division [27]. For the Post Enrolment-based Course Timetabling track: local search [6,8,30,31], ant colony [25], simulated annealing [7,20] and tabu search [14]. Finally for the Curriculum-based Course Timetabling: adaptive tabu search [22], iterative local search [22], threshold accepting metaheuristic [16] and repair-based heuristic search [9].

As mentioned above, one hyper-heuristic approach, winning two tracks was Thomàš Müller's three phase constraint-based solver [23]. This single solution framework was capable of constructing and refining solutions for each track, by employing a series of algorithms based on local search techniques that operate over feasible, though not necessarily complete, solutions. The framework only ever operates over a feasible solution space, ensuring all hard constraints are satisfied, by using a series of algorithms operating using local search techniques. The CPSolver framework consists of multiple phases run se-

quentially, namely the construction phase, the Hill Climbing (HC) phase, the Great Deluge (GD) phase and finally the Simulated Annealing (SA) phase. The construction phase generates a complete initial solution by performing an iterative forward search algorithm that uses conflict-based statistics [23]. A local optimum is then obtained using hill climbing and once an improving solution can no longer be found the great deluge phase begins. Oscillations of the bounds within great deluge are used to allow worsening solutions, facilitating the escape of the local optimum. The simulated annealing stage is optional and is not used within the Examination track to increase the speed of convergence. Once simulated annealing has met its termination criteria the hill climbing phase is continued to converge back to a local optimum. As a consequence, the framework constructs a feasible solution and proceeds to improve upon it.

Implemented within each of the perturbative phases are a number of low-level neighbourhood heuristics selected at random. These low-level heuristics are one of the few domain-specific requirements needed within the framework. This paper seeks to improve on this section of the solver by removing the random aspect of selecting low-level heuristics by introducing online learning to create sequences of low-level heuristics to be applied to the current solution. By removing the random selection, it is possible to target low-level heuristics that perform better than others, decreasing the number of iterations needed to achieve a good solution value. This technique has not before been applied to solve the ITC 2007 problem instances over all thee tracks.

Whilst CPSolver method won the Examination track and Curriculum-based Timetabling track, it came fifth in the Post Enrolment-based Course Timetabling track. It is therefore worth noting the techniques capable of producing higher quality solutions.

One notable technique put forward by Atsuta [1] applied a general-purpose solver. Problem data instances were represented as linear 0-1 inequalities, quadratic 0-1 inequalities, and all-different constraints. Using predetermined weights for hard and soft constraints, a tabu search combined with an iterated local search is used to solve the given instance. The constraint weightings are dynamically controlled during the search process to improve the performance of the general-purpose CPS. It is also worth noting that this technique is not domain specific, and to demonstrate the capabilities of the solver it was entered into each track of the competition producing high quality results on all three tracks. The solver was placed third in the Examination and Curriculum-based tracks, and achieved second place in the Post Enrolment track.

Since the end of the second International Timetabling Competition there have been a number of papers released outlining techniques used to solve the problem instances used within the competition. One of the advantages for researchers evaluating techniques post competition is that all the datasets are available, allowing researchers to tailor frameworks to the data without the time pressure of the competition. On six of the Examination datasets, a technique used to provide lower solution values than all other submissions was proposed in [5]. Here a two-stage solver was implemented. The first stage of

the solver employs a construction algorithm using the existing adaptive ordering heuristic [3]. The next stage aims to improve the solution created in the first stage. An extended great deluge algorithm technique is used, applying a reheating mechanism after a set number of non-improving moves. They performed 51 runs for each data instance within the ITC 2007 Examination track to compare with the other competitors' 10 runs of each data instance in the ITC 2007. However, they do acknowledge this in the paper. It is also not clear on the specifications of the computer the instances were solved on, and there is no mention if the benchmarking program issued by the ITC 2007 was used to ensure a fair comparison.

3 A Multi-stage Sequence-based Hyper-heuristic Approach

Müller [23]'s approach to timetabling is a multi-stage approach using the CP-Solver framework, where a different search algorithm is utilised at each fixed stage. Also, this approach used a different set of low-level heuristics/operators implemented for each timetabling problem class in the ITC 2007 competition. At each step in a stage, a random heuristic is selected and applied to the incumbent solution producing a new solution. To maximise the possibility of making even further improvements to the newly created solutions, learning can be utilised guiding the heuristic selection. Hence the random heuristic selection method is altered to incorporate online learning. Online learning techniques allow for learning to take place whilst a problem instance is being solved. Given that there are three tracks presented within the competition, each will have multiple problem instances with different characteristics, hence online learning is likely to improve the performance of a no-learning approach Incorporating an online learning method into the selection hyper-heuristic ordering the execution of low-level heuristics can generate complex sequences (which would correspond to new heuristics/operators) enabling creation of improved solutions in reduced time.

The proposed approach is based on the same CPSolver, hence it is still a multi-stage approach. The selection hyper-heuristics employed at each stage is not changed and assigns a score to each LLH, maintains those scores based on reinforcement learning and then it chooses one of the low-level heuristics based on their scores using the heuristic selection method. The components and variants of the proposed approach is described in the following subsections.

3.1 Low-level Heuristics

The same set of low-level heuristics (LLHs) as suggested in [23] are employed for solving the timetabling problems as summarised below for each ITC 2007 track.

1. Examination timetabling: Exam swap, period swap, room swap, period change, room change, period and room change.

2. Post enrollment-based course timetabling: Time move, room move, event move, event swap, precedence swap (violation oriented move with considering the precedence constraints).
3. Curriculum-based course timetabling: Time move, room move, lecture move, room stability move, min working days move, curriculum compactness move.

3.2 Heuristic Selection

Kheiri and Keedwell [17] provided an effective sequence-based selection hyper-heuristic illustrated its performance across a number of high school timetabling problem instances.

Commonly, sequence-based heuristic selection algorithms work by generating a completed sequence of low-level heuristic (LLH) operators to be sequentially applied to the current solution, creating a new candidate solution. This can lead to the scenario of constructing a sequence of length n, applying each LLH to the current solution and reverting back if the solution value is not accepted by the acceptance strategy, e.g. simulated annealing. In an attempt to eliminate reverting back n neighbourhood changes when a sequence is rejected, when the next LLH in the sequence is selected it is immediately applied to find a neighbouring solution. If the neighbouring solution is accepted, it is officially added to the sequence, otherwise the LLH is not added to the current sequence, and the next LLH is selected. Implementing the sequence construction in this manner could help guide the sequence creation process and discovery of Hidden Markov Chains.

Sequences are constructed by maintaining two matrices. The first matrix is designed to maintain a performance score for each possible heuristic transition, we will refer to this as $TransScore$. We are then capable of selecting the next LLH using these scores. For example, assume n low-level heuristics are implemented. Also, assume the unfinished sequence of $[llh_i, llh_j]$ is constructed and the next LLH selected, using a Roulette Wheel or any other heuristic selector, is llh_k. From this, assume heuristic llh_k is applied and produced an improved solution. The score value at $TransScore[j, k]$ is updated and the current sequence becomes $[llh_i, llh_j, llh_k]$. If llh_k did not produce an improving solution but was still accepted, the score values remain the same and the sequence is again updated to become $[llh_i, llh_j, llh_k]$. The starting scores for each heuristic transition is 1, that is, $TransScore[llh_i, llh_j] = 1, \forall\ i, j$.

The second matrix stores the score values of ending the sequence on the current LLH selected. We will refer to this matrix as $EndScore$. For example, with the same implementation and sequence construction in the example above, when llh_k is selected the scores at $EndScore[llh_k, 0]$ and $EndScore[llh_k, 1]$ are used to determine if the sequence is terminated at that heuristic. If the sequence is to terminate at llh_k then the scores $EndScore[i, 0]$, $EndScore[j, 0]$ and $EndScore[k, 1]$ are all updated. The starting scores for each acceptance strategy for every low-level heuristic is 1, that is, $EndScore[llh_i, j] = 1, \forall\ i, j$.

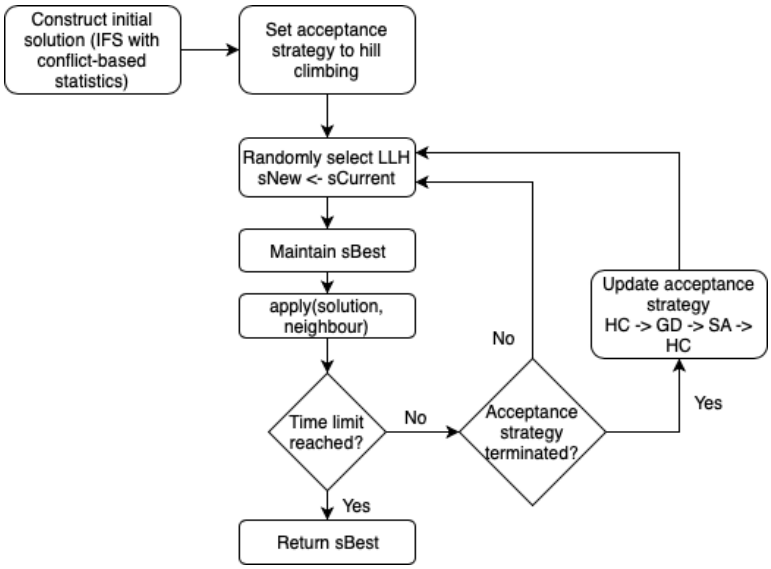


Fig. 1: Flowchart of the CPSolver

This learning method is implemented into the CPSolver framework to select heuristics instead of the current random selection. The figures displayed below compare the stages that the CPSolver takes in the 2007 ITC submission against the stages the CPSolver takes with the new proposed implementation.

Given the two scoring probability matrices described above, a selection strategy is required to select the next LLH in the sequence and to determine if the sequence is terminated. Generally, the selection process can be categorised into proportional and elitist strategies. Proportional strategies take into consideration the probabilities of each heuristic being selected and selects the heuristic accordingly. This allows diversification within the heuristics selected but takes longer to converge. Elitist methods simply take the heuristic with the best score associated to it. This increases convergence but eliminates any diversification throughout the search. The heuristic selection strategies explored in this paper are: Roulette Wheel selection and Tournament selection. Whilst a general aim is to reduce computation time of the solver, and both of these selection methods fall under the proportional strategy, taking the heuristic with the highest score each time does not allow for enough diversification for the problem instances.

Roulette Wheel selection (RW) gives an individual i the probability of being selected $p(i)$ proportional to its fitness $f(i)$ where [15]:

$$p(i) = \frac{f(i)}{\sum_{j=1}^{n} f(j)} \tag{1}$$

Tournament selection (TO) consists of selecting k individuals at random and then ranking them best to worst. The best individual is then selected to

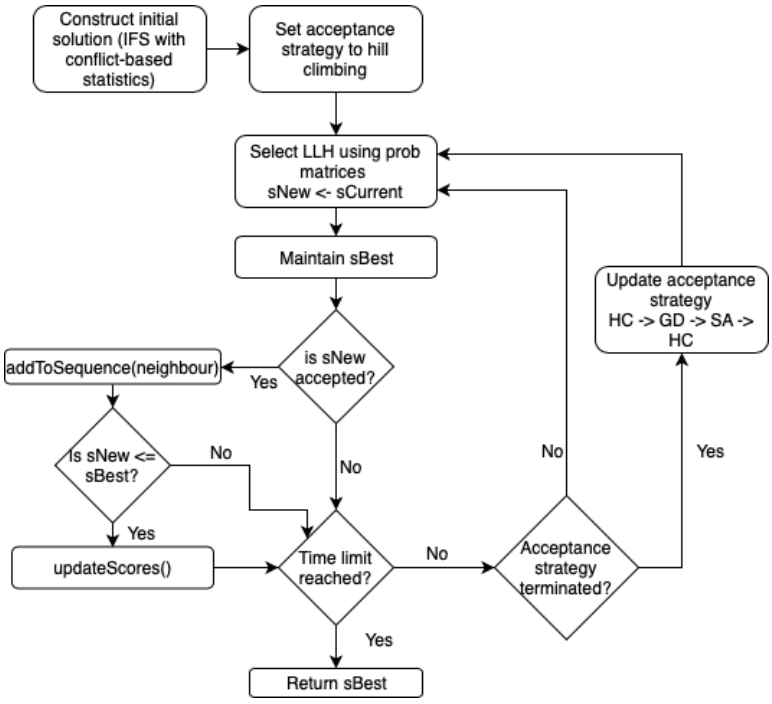


Fig. 2: Flowchart of the new sequence-based hyper-heuristic solver

enter the sequence [15]. Tournament selection has the advantage of maintaining diversification whilst converging faster than Roulette Wheel.

3.3 Reward Scheme

The performance of the sequence-based hyper-heuristic can vary depending on the learning strategy implemented. This paper explores three rewarding schemes as explained below [17].

- Linear reward scheme (LI): The matrices values are incremented linearly with a reward of 1 when an improving solution is obtained. The strategy does not take into consideration the amount to which the solution is improved, but simply acknowledges that an improving solution has been found.
- Non-linear reward scheme (NL): The matrices values are incremented non-linearly by e^t/c, where t is the time elapsed and c is a predetermined constant. This reward system allocates a larger reward to sequences that find an improving move later on in the search.
- Delta reward scheme (DE): The matrices values are incremented according to how much the sequence improved on the overall solution value, giving

larger rewards to sequences that make a bigger impact on the solution value. The heuristic selection can then continue making sequences that target the heuristics that make the biggest difference to the solution value.

3.4 Stage Options

CPSolver is used as a multi-stage approach, constructing a (feasible) solution and then improving upon it using a hill climbing and/or a local search meta-heuristic. During the construction phase, the solver can sometimes run into the issue where it can no longer successfully assign values to variables causing it to be idle. This is very noticeable on data instances 3 and 4 on the Examination track, with Müller being unable to obtain a feasible solution on some of his runs. To combat this issue, if the construction phase does not make an improving move in 200 iterations the current solution will be reset. The construction phase will then restart and build a new solution. This method proved to be reliable as we managed to obtain feasible solutions on all data instances in the Examination track.

After the construction phase, the hyper-heuristic follows a fixed stage structure of Hill Climbing (HC), Great Deluge with re-rising level (GD) and then the optional stage Simulated Annealing with reheating (SA) (Thomàš Müller noted that Simulated Annealing is not used for the Examination timetabling track due to its slow computation time [23]). A selection hyper-heuristic contains two key components: heuristic selection and move acceptance. CPSolver provides a multi-stage selection hyper-heuristic framework where a different move acceptance method can be used at each stage while the heuristic selection is maintained. The move acceptance methods significantly affects the performance as compared to heuristic selection within hyper-heuristics [26]. Since we have only 2 move acceptance methods, we explored the performance of the proposed approach with different options for the stages where we either use HC or not or a move acceptance. Hence every stage ordering using (a set of) two or single move acceptance method(s) is tested along with using HC or not in the first stage, yielding six different options: HC-GA-SA, HC-SA-GD, GD-SA, SA-GD, GD and finally SA.

The same suggested parameter settings for each ITC2007 track provided in [23] for the HC (maximum number of iterations without improvement), GD (upper bound for the level, lower bound for the level, rate of decrease), SA (initial temperature, number of steps spent at each temperature, geometric cooling rate, reheating rate) algorithms are used during the experiments.

4 Configuring the Sequence-based Hyper-heuristic Approach

All configurations of the sequence-based hyper-heuristic approach are evaluated using 4 selected problem instances from each track of the competition. Each experiment is run 10 times for the appropriate amount of time (247 seconds) which was obtained using the benchmarking software supplied by the

ITC 2007. This helps create a level playing field across all competition submissions and allows for the comparison of results without having to factor in computer speed. A total of 132 hours were spent configuring our hyper-heuristic approach, and all the runs were performed on a Windows 10 machine with an Intel Core i5-6400 2.70GHz processor with 8GB of RAM. When testing for the acceptance strategy, heuristic selection strategy and reward mechanism, the data instances used for the Examination timetabling track were 1, 2, 5 and 6. For the Post Enrolment timetabling track the data instances 3, 4, 6 and 8 were used. Finally, for the Curriculum-based timetabling track the data instances 3, 5, 7 and 8 were used. To score the results of each run, a given instance was allocated a rank based on its overall solution value. The solution value with the lowest score was given a ranking score of 1, the next highest will get a ranking score of 2 etc. In the case of a tie an average rank was assigned. For example, if the third highest solution value was 5 and this solution value was obtained 3 times, a ranking score of 4 would be given to each solution. The ranking scores would then continue from 6. The score for a given strategy on a given instance is the mean ranking score for that instance. The overall score for a given strategy is then given by the mean score obtained for each data instance, and the strategy with the lowest score is then taken to be the best.

4.1 Stage Option Experiments

The incremental configuration for the proposed approach first experimented with different stage options: HC-GA-SA, HC-SA-GD, GD-SA, SA-GD, GD and finally SA. For all three tracks the heuristic selection method was set to Roulette Wheel with a linear reward method for each option. Table 1 shows the ranking of each stage option for each selected benchmark instance for each track. Table 2 shows the overall ranking of the acceptance strategies for each track. From the tables, we can see for all three tracks, Great Deluge by itself performed the best. However, Table 2 demonstrates that the second best stage option is HC-GD-SA, GD-SA and SA-GD for the examination, post enrollment-based and curriculum-based course timetabling, respectively. Hence, those methods are fixed for the next set of experiments.

4.2 Testing the Heuristic Selection Methods and Reward Mechanisms

During implementation it was noticed that the heuristic selection method and reward mechanism were closely related with the combination dramatically affecting the scores. This means we were not able to optimise them sequentially, they had to be done simultaneously. We also tested each combination of heuristic selection and reward mechanism for the top two ranking sequence of acceptance strategies obtained from the section above. Table 3 shows the overall ranked scores for each track. We can see from the table that the Examination track performed best by a significant margin, operating with a delta-based

Table 1: Acceptance strategy ordering per instance ranking

Examination Acceptance Strategy Ordering							
Instance 1		Instance 2		Instance 5		Instance 6	
Order	Score	Order	Score	Order	Score	Order	Score
GD	9.30	SA	21.60	GD	5.60	GD	22.20
HC-GD-SA	12.70	GD	23.60	SA	23.00	GD-SA	28.35
SA	35.75	HC-GD-SA	27.15	HC-GD-SA	30.15	SA	30.90
SA-GD	37.10	HC-SA-GD	31.40	GD-SA	40.05	HC-GD-SA	31.40
GD-SA	40.15	SA-GD	37.30	SA-GD	40.60	SA-GD	34.65
HC-SA-GD	48.00	GD-SA	41.95	HC-SA-GD	43.60	HC-SA-GD	35.50
Post Enrolment Acceptance Strategy Ordering							
Instance 3		Instance 4		Instance 6		Instance 8	
Order	Score	Order	Score	Order	Score	Order	Score
GD	17.20	GD	18.05	GD	17.40	GD	12.75
HC-GD-SA	18.45	HC-GD-SA	25.00	GD-SA	22.90	GD-SA	28.00
GD-SA	21.25	GD-SA	26.95	HC-GD-SA	27.30	HC-SA-GD	28.65
SA-GD	35.20	SA-GD	27.40	HC-SA-GD	30.05	SA-GD	29.30
HC-SA-GD	36.00	HC-SA-GD	33.40	SA-GD	36.00	HC-GD-SA	29.70
SA	54.90	SA	52.20	SA	49.35	SA	54.60
Curriculum-based Acceptance Strategy Ordering							
Instance 3		Instance 5		Instance 7		Instance 8	
Order	Score	Order	Score	Order	Score	Order	Score
GD	18.60	GD	17.35	HC-SA-GD	19.65	GD	16.65
GD-SA	22.30	GD-SA	22.40	GD	19.70	SA-GD	16.65
SA-GD	29.55	SA-GD	28.70	SA-GD	22.90	HC-SA-GD	30.45
HC-SA-GD	32.60	HC-GD-SA	31.30	SA	33.30	GD-SA	38.35
HC-GD-SA	34.70	HC-SA-GD	33.05	GD-SA	43.65	SA	39.30
SA	45.25	SA	50.20	HC-GD-SA	43.80	HC-GD-SA	41.60

Table 2: Acceptance strategy ordering overall ranking

Examination		Post Enrolment		Curriculum-based	
Order	Score	Order	Score	Order	Score
GD	15.18	GD	16.35	GD	18.07
HC-GD-SA	25.35	GD-SA	24.77	SA-GD	24.45
SA	27.81	HC-GD-SA	25.11	HC-SA-GD	28.94
SA-GD	37.41	SA-GD	31.98	GD-SA	31.68
GD-SA	37.62	HC-SA-GD	32.02	HC-GD-SA	37.85
HC-SA-GD	39.62	SA	52.76	SA	42.01

reward system with tournament heuristic selection and with Great Deluge as the move acceptance. The Post Enrolment track performed best with a delta-based reward system with Tournament heuristic selection and with Great Deluge followed by Simulated Annealing as the move acceptance ordering. It is interesting to note that Great Deluge followed by Simulated Annealing ranked second best during the move acceptance optimisation stage, running under a linear reward with Roulette Wheel selection. This further indicates that the performance reward mechanism and heuristic selection can be dependent on the move acceptance. Optimising for one feature at a time may not lead to the optimal configuration, however we are confident that testing the top two

Table 3: The overall ranking of heuristic selection (H.S.) and reward scheme (R.S.) pairs combined with top three sequence of low-level (meta)heuristics

Examination			Post Enrolment			Curriculum-based		
R.S.-H.S.	Order	Score	R.S.-H.S.	Order	Score	R.S.-H.S.	Order	Score
DE-TO	GD	24.20	DE-TO	GD-SA	22.61	NL-RW	GD	38.66
NL-TO	GD	37.33	DE-TO	GD	37.48	NL-TO	SA-GD	39.56
LI-RW	GD	42.38	DE-RW	GD	55.17	LI-RW	GD	44.09
NL-RW	GD	42.77	NL-TO	GD	57.34	LI-RW	SA-GD	54.99
LI-TO	GD	52.99	NL-TO	GD-SA	58.36	LI-TO	SA-GD	55.51
DE-RW	GD	57.20	LI-TO	GD-SA	58.42	DE-RW	SA–GD	58.17
LI-RW	HC-GD-SA	59.70	DE-RW	GD-SA	63.59	DE-RW	GD	60.62
DE-RW	HC-GD-SA	79.09	LI-TO	GD	65.94	NL-RW	SA-GD	66.44
NL-TO	HC-GD-SA	79.75	LI-RW	GD	66.71	NL-TO	GD	66.79
NL-RW	HC-GD-SA	80.08	NL-RW	GD	74.34	DE-TO	SA-GD	68.47
DE-TO	HC-GD-SA	80.78	NL-RW	GD-SA	81.55	DE-TO	GD	74.08
LI-TO	HC-GD-SA	89.75	LI-RW	GD-SA	84.49	LI-TO	GD	98.61

ranked move acceptance orderings for each track is sufficient to produce the optimal configurations. Finally, we can see from the table that a non-linear reward mechanism with a Roulette Wheel selection and Great Deluge acceptance is the optimal configuration for the Curriculum-based track.

5 Experimental Results

Evaluation of the sequence-based hyper-heuristic approach was done using the optimised configuration obtained above. We ran each data instance supplied by the ITC for each track, that is: 8 instances for the Examination track, 16 instances for the Post Enrolment track and 21 instances for the Curriculum-based track. To accurately and fairly compare the proposed hyper-heuristic to the ITC competitors results all runs were performed under the same conditions. That is, each instance was evaluated by performing 10 complete runs, with a random seed, for the time allocated by the benchmarking program issued by the ITC. All runs were performed using the same windows machine as described in Section 4. The results described in Tables 4, 5 and 6 present our experimental results; the average and best for the Sequence-based Hyper-heuristic (SBHH). The tables also display the best scores of the competition winners alongside the best solution value produced by Thomàš Müller's code over 100 runs on each instance.

5.1 Examination Timetabling Results

Table 4 displays the results described above along with the best results of two bespoke approaches proposed after the competition finished. McCollum et al. [5] performed their approach for a total of 51 runs per instance. Saber et al. [28] performed their approach for a total of 21 runs per instance. The best scores for each track are displayed in bold. It is worth noting that the approaches in [1, 27, 28] are all single stage approaches, generating a single solution without further

Table 4: Examination track experimental results, where entries in bold style highlight the best performing algorithms

Instance	SBHH		ITC-2007 Finalist						Post ITC-2007	
	Avg.	Best	Müller	[13]	[1]	[11]	[27]	[23] 100 runs	[5] 51 runs	[28] 20 runs
Exam 1	4074.8	**4008**	4370	5905	8006	6670	12035	4356	4633	6234
Exam 2	391	**385**	400	1008	3470	623	3074	390	405	395
Exam 3	10060.9	9347	10049	13862	18622	-	15917	9568	**9064**	13002
Exam 4	19454.6	15870	18141	18674	22559	-	23582	16591	**15663**	17940
Exam 5	2758	**2617**	2988	4139	4714	3847	6860	2941	3042	3900
Exam 6	26867	26195	26950	27640	29155	27815	32250	**25775**	25880	27000
Exam 7	3978.8	**3824**	4213	6683	10473	5420	17666	4088	4037	6214
Exam 8	7228	**7012**	7861	10521	14317	-	16184	7565	7461	8552

improvement. Therefore, it is not completely fair to compare our results with these methods. However, our approach outperformed all other methods on 5 of the data instances. We also managed to provide better solutions than Müller's best over 100 runs on 7 of the data instances and outperformed Müller's ITC 2007 finalist solution values on all 8 instances. This displays the capability of the sequence-based hyper-heuristic for the Examination track. It would be interesting to perform further runs on each data instance to see if our method is capable of outperforming McCollum on instances 3 and 4, and Müller's best solution value achieved on instance 6 over 100 runs.

5.2 Post Enrolment-based Course Timetabling Results

Table 5 presents our experiments results for the Post Enrolment track of the competition. The results are displayed in a tuple, with the first number in each cell being the distance to feasibility (dtf) and the second being the overall solution value. We also compared our results to the best results obtained from state-of-the-art approaches developed after the competition [7, 20, 14, 30, 31]. The best scores for each track are displayed in bold.

The state-of-the-art approaches have made significant improvement within the problem of post enrolment with Ceschia et al. [7] displaying 12 of the best solution values making use of a single-step metaheuristic approach based on simulated annealing. Whilst our proposed technique only manages to obtain the joint best values for two of the instances, we managed to improve the solution value on 9 of Müller's competition runs. This indicates that our proposed method can produce improving solution compared to Müller's original solver, but it is not the best technique for solving Post Enrolment problems.

5.3 Curriculum-based Course Timetabling Results

Table 6 presents our experiments results for the final track, Curriculum-based timetabling. We compared our results to 4 approaches developed after the end of the competition: Tabu search and Iterative Local Search both developed

Table 5: Post Enrolment track experimental results, where entries in bold style highlight the best performing algorithms

Ins.	SBHH		ITC-2007 Finalist							Post ITC-2007			
	Avg.	Best	Müller	[6]	[1]	[8]	[25]	[23] 100 runs	[7] 30 runs	[20]	[14]	[30]	[31]
1	153, 2259	0, 1810	0, 1861	0, 571	0, 61	0, 1482	**0, 15**	0, 1330	0, 59	0, 1166	0, 501	0, 650	0, 630
2	417, 2229	119, 2233	39, 2174	0, 993	0, 547	0, 1635	**0, 0**	0, 2154	**0, 0**	0, 1665	0, 342	0, 470	0, 450
3	0, 292	0, 234	0, 272	0, 164	0, 382	0, 288	0, 391	0, 205	**0, 148**	0, 251	0, 3770	0, 290	0, 300
4	0, 462	0, 386	0, 425	0, 310	0, 529	0, 385	0, 239	0, 394	**0, 25**	0, 424	0, 234	0, 600	0, 602
5	0, 42	0, 9	0, 8	0, 5	0, 5	0, 559	0, 34	**0, 0**	**0, 0**	0, 47	**0, 0**	0, 35	0, 6
6	0, 236	0, 76	0, 28	**0, 0**	**0, 0**	0, 851	0, 87	0, 13	**0, 0**	0, 412	**0, 0**	0, 20	**0, 0**
7	0, 20	0, 5	0, 13	0, 6	**0, 0**	0, 10	**0, 0**	0, 5	**0, 0**	0, 6	**0, 0**	0, 30	**0, 0**
8	0, 20	**0, 0**	0, 6	**0, 0**	**0, 0**	**0, 0**	0, 4	**0, 0**	**0, 0**	0, 65	**0, 0**	**0, 0**	**0, 0**
9	922, 2103	433, 2351	162, 2733	0, 1560	**0, 0**	0, 1947	**0, 0**	0, 1895	**0, 0**	0, 1819	0, 989	0, 630	0, 640
10	773, 2365	490, 2280	161, 2697	0, 2136	**0, 0**	0, 1741	**0, 0**	57, 2440	0, 3	0, 2091	0, 499	0, 2349	0, 663
11	0, 635	0, 437	0, 263	0, 178	0, 548	0, 240	0, 547	0, 347	**0, 142**	0, 288	0, 246	0, 350	0, 344
12	0, 825	0, 698	0, 804	0, 146	0, 869	0, 475	**0, 32**	0, 453	0, 267	0, 474	0, 172	0, 480	0, 198
13	0, 608	0, 302	0, 285	**0, 0**	**0, 0**	0, 675	0, 166	0, 74	0, 1	0, 298	**0, 0**	0, 46	**0, 0**
14	0, 545	0, 82	0, 110	0, 1	**0, 0**	0, 864	**0, 0**	0, 2	**0, 0**	0, 127	**0, 0**	0, 80	0, 35
15	0, 244	**0, 0**	0, 5	**0, 0**	0, 379	**0, 0**	**0, 0**	**0, 0**	**0, 0**	0, 108	**0, 0**	**0, 0**	**0, 0**
16	0, 167	0, 67	0, 132	0, 2	0, 191	0, 1	0, 41	0, 6	**0, 0**	0, 138	**0, 0**	**0, 0**	0, 140

Table 6: Curriculum-based track experimental results, where U indicates undefined and entries in bold style highlight the best performing algorithms

Ins.	SBHH		ITC-2007 Finalist							Post ITC-2007			
	Avg.	Best	Müller	[22]	[1]	[16]	[9]	[23] 100 runs	[16] 30 runs	[22] TS	[22] ILS	[2]	[19]
1	5	**5**	**5**	**5**	**5**	**5**	10	**5**	**5**	**5**	**5**	**5**	13
2	68	48	51	55	50	111	111	**43**	91	55	48	75	**43**
3	86	76	84	**71**	82	128	119	72	108	90	76	93	76
4	42	**35**	37	43	**35**	72	72	**35**	53	45	41	45	38
5	336	309	330	309	312	410	426	298	359	315	**303**	326	314
6	60	**35**	48	53	69	102	130	41	79	58	54	62	41
7	26	**10**	20	28	42	57	110	14	36	33	25	38	19
8	43	**39**	41	49	40	77	83	**39**	63	49	47	50	43
9	108	**101**	109	105	110	150	139	103	128	109	106	119	102
10	25	11	16	21	27	71	85	**9**	49	23	23	27	14
11	0	**0**	**0**	**0**	**0**	**0**	3	**0**	**0**	**0**	**0**	**0**	**0**
12	353	333	333	343	351	442	408	331	389	330	**324**	358	405
13	78	**59**	66	73	68	622	113	66	91	71	68	77	68
14	60	55	59	57	59	90	84	**53**	81	55	**53**	59	54
15	85	75	84	**71**	82	128	119	U	U	U	U	87	U
16	45	38	**34**	39	40	81	84	U	U	U	U	47	U
17	88	**75**	83	91	102	124	152	U	U	U	U	86	U
18	84	75	83	69	**68**	116	110	U	U	U	U	71	U
19	71	64	**62**	65	75	107	111	U	U	U	U	74	U
20	53	34	**27**	47	61	88	144	U	U	U	U	54	U
21	116	**100**	103	106	123	174	169	U	U	U	U	117	U

in [22,2,19]. We were also able to compare our solutions to the best solution values obtained by Geiger [16] over 30 runs for each instance, with each run allowing 100,000,000 evaluations. Solution values are not available for all instances for each approach. The solution values we were not able to obtain are represented with a 'U' for unknown and all the best values for a given instance are displayed in bold. Our approached achieved 10 of the best solution values across all 21 instances. We managed to make improvements on Müller's competition final solutions across 15 of the instances. We also made improvements on 4 instances compared to Müller's 100 runs.

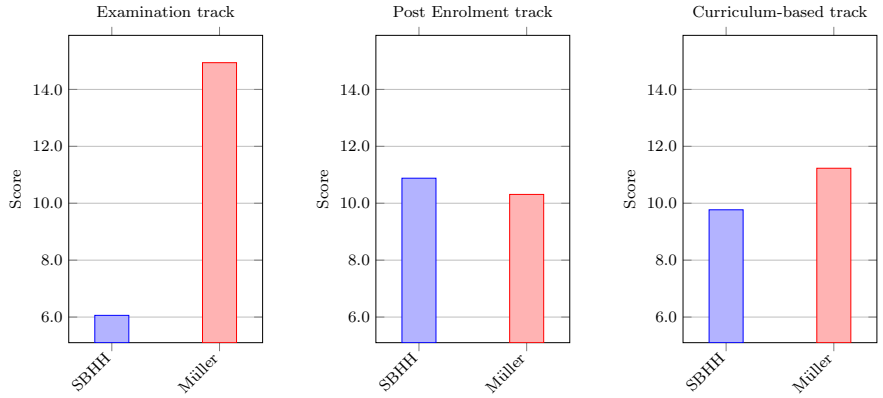


Fig. 3: Overall performance comparison of Müller's approach and SBHH based on average ranking scores for the Examination, Post Enrolment and Curriculum-based tracks of ITC 2007

5.4 Performance Comparison to Müller's Approach

In addition to comparing the best solution values produced by the proposed and various other approaches, we have calculated the average ranking scores for each track based on Müller's and our solutions obtained for ITC 2007. We calculated the ranking scores in the same manner as the ITC calculated the winner, and similarly we ranked the different heuristics in the configuration results section. Figure 3 displays the average ranking scores for our and Müller's approaches for each timetabling track of ITC 2007. The figure shows that our approach outperformed Müller's on the Examination track and Curriculum-based track by a significant margin producing average ranking scores of 6.06 and 9.77 versus 14.94 and 11.23, respectively. This figure also displays that Müller's original solver performs marginally better than with the sequence-based hyper-heuristic implemented with an average ranking score of 10.31 against 10.88. However, we noticed that the proposed implementation had difficulties with data instances that often resulted with a dtf (distance to feasibility) score. If we exclude such Post Enrolment track instances, including 2, 9, 10, the overall average ranking of our approach becomes 10.16 against Müller's approach 11.07, hence we can observe that the sequence-based hyper-heuristic performed better on the remaining 13 instances.

6 Conclusions

There has been a growing body of work on multi-stage selections hyper-heuristics for both single [18] and multi-objective [21] optimisation. This paper presents a tuned multi-stage sequence-based hyper-heuristic approach embedded into the CPSolver framework in an attempt to improve on the solutions

produced by Müller in the ITC 2007 competition. The proposed method is capable of improving an initially created solution. It does this in a more sophisticated manner than the original framework by attempting to adaptively identify and isolate sequences of low-level heuristics that perform well on a given instance. We carried out extensive testing to accurately determine the best configuration setup for our approach to decide the best stage option (acceptance ordering), heuristic selection method and reward scheme. The following stage options are considered: HC-GD-SA, HC-SA-GD, GD-SA, SA-GD, GD and SA. We observed that Great Deluge operating by itself ranked best for all three tracks. However, when testing the heuristic selection method and reward mechanism we found that the performance of the acceptance strategy is not independent of other factors. After configuration tests, we concluded that the best configurations for the Examination, post enrolment, and Curriculum-based tracks were Great Deluge with tournament selection and delta learning, Great Deluge and Simulated Annealing with tournament selection and delta learning, and Great Deluge with Roulette Wheel selection and non-linear learning, respectively.

As illustrated, the proposed approach was effective at improving on Müller's ITC 2007 solution objective values. All the Examination instances were improved upon, with instance 4 improving as much as 12.5%. We improved on the solution values for 9 data instances in the Post Enrolment track. Finally, for the curriculum-based track we made improvements on 15 of the data instances, as much as 50% (instance 7), and found the same solution value on 3 other data instances. We also compared the proposed technique against state-of-the-art approaches for all three tracks. We found improving solution values on 5 Examination data instances, joint lowest solution values on 2 post enrolment data instances and finally, 6 improving and 4 joint lowest solution values on the curriculum-based track. It is stressed that the results obtained by methods after the end of the competition are bespoke solvers designed to only solve problem instances for the given track. Some of the solvers were also allowed to run for more than 10 runs, the allowed number within the competition, leading to an unfair comparison. The educational timetabling is still of interest to many academics as well as practitioners. A trivial future work would be applying the proposed approach to unseen instances, perhaps to those instances provided during the new competition on university course timetabling, ITC 2019 [24].

References

1. Atsuta, M., Nonobe, K., Ibaraki, T.: Itc-2007 track2: An approach using general csp solver (2007)
2. Bonutti, A., De Cesco, F., Di Gaspero, L., Schaerf, A.: Benchmarking curriculum-based course timetabling: Formulations, data formats, instances, validation, visualization, and results. Annals OR **194**, 59–70 (2012)
3. Burke, E., Newall, J.: Solving examination timetabling problems through adaption of heuristic orderings: Models and algorithms for planning and scheduling problems (edi-

tors: Philippe baptiste, jacques carlier, alix munier, andreas s. schulz). Annals of Operations Research **129** (2004)
4. Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Qu, R.: Hyper-heuristics: a survey of the state of the art. Journal of the Operational Research Society **64**(12), 1695–1724 (2013)
5. C Mccollum, G., J Mcmullan, P., Parkes, A., K Burke, E., Abdullah, S., Mccollum, B.: An extended great deluge approach to the examination timetabling problem (2019)
6. Cambazard, H., Hebrard, E., O'Sullivan, B., Papadopoulos, A.: Local search and constraint programming for the post enrolment-based course timetabling problem. Annals of Operations Research **194**, 111–135 (2012)
7. 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** (2011)
8. Chiarandini, M., Fawcett, C., Hoos, H.H.: A modular multiphase heuristic solver for post enrolment course timetabling (2008)
9. Clark, M., Henz, M., Love, B.: Quikfix—a repair-based timetable solver. In: Proceedings of the Seventh International Conference on the Practice and Theory of Automated Timetabling
10. Cowling, P., Kendall, G., Soubeiga, E.: A hyperheuristic approach to scheduling a sales summit. In: International Conference on the Practice and Theory of Automated Timetabling, pp. 176–190. Springer (2000)
11. De Smet, G.: Itc2007 - examination track. p. 22 (2008)
12. Drake, J.H., Kheiri, A., Özcan, E., Burke, E.K.: Recent advances in selection hyper-heuristics. European Journal of Operational Research (2019). DOI https://doi.org/10.1016/j.ejor.2019.07.073
13. Gogos, C., Alefragis, P., Housos, E.: A multi-staged algorithmic process for the solution of the examination timetabling problem, proceedings of the international conference on practice and theory of automated timetabling (patat 2008). p. 22 (2008)
14. Jat, S.N., Yang, S.: A hybrid genetic algorithm and tabu search approach for post enrolment course timetabling. J. Scheduling **14**, 617–637 (2011)
15. Jebari, K.: Selection methods for genetic algorithms. International Journal of Emerging Sciences **3**, 333–344 (2013)
16. Josef Geiger, M.: Applying the threshold accepting metaheuristic to curriculum based course timetabling. Annals of Operations Research **194**, 189–202 (2008)
17. Kheiri, A., Keedwell, E.: A hidden markov model approach to the problem of heuristic selection in hyper-heuristics with a case study in high school timetabling problems. Evol. Comput. **25**(3), 473–501 (2017)
18. Kheiri, A., Özcan, E.: An iterated multi-stage selection hyper-heuristic. European Journal of Operational Research **250**(1), 77 – 90 (2016)
19. Lach, G., Lübbecke, M.: Curriculum based course timetabling: New solutions to udine benchmark instances. Annals of Operations Research **194**, 255–272 (2012)
20. Lewis, R.: A time-dependent metaheuristic algorithm for post enrolment-based course timetabling (2012)
21. Li, W., Özcan, E., John, R.: A learning automata-based multiobjective hyper-heuristic. IEEE Transactions on Evolutionary Computation **23**(1), 59–73 (2019). DOI 10.1109/TEVC.2017.2785346
22. Lü, Z., Hao, J.K.: Solving the course timetabling problem with a hybrid heuristic algorithm. pp. 262–273 (2008)
23. Müller, T.: Itc2007 solver description: A hybrid approach. Annals of Operations Research **172**, 429–446 (2008). DOI 10.1007/s10479-009-0644-y
24. Müller T, R.H., Müllerova, 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). p. 27 (2018)
25. Nothegger, C., Mayer, A., M. Chwatal, A., Raidl, G.: Solving the post enrolment course timetabling problem by ant colony optimization. Annals OR **194**, 325–339 (2012)
26. Özcan, E., Bilgin, B., Korkmaz, E.: A comprehensive analysis of hyper-heuristics. Intelligent Data Analysis **12**, 3–23 (2008). DOI 10.3233/IDA-2008-12102

27. Pillay, N.: A developmental approach to the examination timetabling problem, proceedings of the international conference on practice and theory of automated timetabling (patat 2008). pp. 19–22 (2008)
28. Sabar, N., Ayob, M., Qu, R., Kendall, G.: A graph coloring constructive hyper-heuristic for examination timetabling problems. Applied Intelligence - APIN **37**, 1–11 (2011). DOI 10.1007/s10489-011-0309-9
29. Sörensen, K., Glover, F.W.: Metaheuristics, pp. 960–970. Springer US, Boston, MA (2013)
30. Soria-Alcaraz, J., Ochoa, G., Swan, J., Carpio, M., Soberanes, H., K. Burke, E.: Effective learning hyper-heuristics for the course timetabling problem. European Journal of Operational Research **238**, 77–86 (2014)
31. Soria-Alcaraz, J., Özcan, E., Swan, J., Kendall, G., Carpio, M.: Iterated local search using an add and delete hyper-heuristic for university course timetabling. Applied Soft Computing **40** (2015)

Slack-based Robustness Estimators for the Curriculum-Based Course Timetabling Problem

Can Akkan · Ayla Gülcü · Zeki Kuş

Received: date / Accepted: date

Abstract Developing robust university course timetables is an important practical concern. Due to a variety of possible disruptions, i.e. changes in the input data affecting the constraints, quickly calculating an estimate of robustness to be used within a meta-heuristic, such as Simulated Annealing, would be very useful. In this research, we attempt to develop a set of slack-based estimators of a solution's robustness. To this end, we define 11 different slack measures (period, room and course-based) and use three summary statistics for each measure as an estimator of robustness. Preliminary experimental analysis of the performance of these estimators is done on a sample of 192 solutions for four International Timetabling Competition 2007 instances selected based on their diverse characteristics. The results suggest that a slack-based estimator can be used to identify a Pareto "band" rather than an approximate frontier that strikes a balance between probability of a solution on the true frontier being in the band and one not on the true frontier not being in the band.

Keywords Course timetabling · Robustness · Multi-criteria optimization

This work was supported by TÜBİTAK grant 217M475.

C. Akkan
Sabancı University, Sabancı Business School, Tuzla, 34956 İstanbul, Turkey.
Tel.: +90-216-4839685
E-mail: can.akkan@sabanciuniv.edu

A. Gülcü
Fatih Sultan Mehmet Univ., Dept. of Computer Science, Beyoğlu, 34445 İstanbul, Turkey.
Tel.: +90-212-5218100
E-mail: agulcu@fsm.edu.tr

Zeki Kuş
Fatih Sultan Mehmet Univ., Dept. of Computer Science, Beyoğlu, 34445 İstanbul, Turkey.
Tel.: +90-212-5218100
E-mail: zeki.kus@stu.fsm.edu.tr

1 Introduction

In a typical curriculum-based timetabling process at a university, an initial timetable, Σ_0, is prepared based on a set of constraints, which is then announced to the university staff, giving them some time to submit requirement changes, due to new constraints or data corrections. The timetable is then re-optimized taking these new constraints into account, while ensuring that the changes to Σ_0 is kept to a minimum. This second timetable is announced to the entire university and the students enroll in courses based on this timetable. Several types of disruptions that may affect this process are discussed in the literature (McCollum (2007), Müller et al (2005), Kingston (2013), Phillips et al (2017), Lindahl et al (2019)). As discussed in Phillips et al (2017) many different types of changes in data (disruptions) are possible before enrollment. Some new courses could be added, some others could be removed or canceled (see e.g. Yasari et al (2019)), some new faculty may arrive and some others may leave. Some disruptions could simply change feasibility of certain periods for some lectures. Some disruptions may affect either the availability or the capacity sufficiency of rooms for some lectures.

Problems in which constraints change over time are known as dynamic optimization problems, which fall into the category of optimization in uncertain environments. Meta-heuristics are quite often used for solving these problems (e.g. see Jin and Branke (2005) for a survey of evolutionary algorithms). Recently, there has been increasing interest in modeling and solving dynamic combinatorial optimization problems. One such problem closely related to timetabling is the graph coloring problem. Hardy et al (2018) develop heuristics for the dynamic graph coloring problem where edges are added/removed over time, randomly. They look into how information about the likelihood of future edge changes can be used to produce more robust colorings.

We say a timetable is *robust* if, when disrupted, its feasibility can be restored without significantly lowering its quality in terms of the objective function while keeping it relatively stable. We formulate the problem of identifying a robust timetable as a bi-criteria optimization problem where one objective is the quality of the solution measured as a function of the violated soft constraints (i.e., the penalty function), denoted by P, and the second one is a function that measures the robustness of the timetable, denoted by R.

We assume multiple number and types of disruptions can affect a given timetable. This makes calculating the robustness of a given solution quite time-consuming, since it requires optimally repairing that solution for a reasonably large sample of disruption scenarios. Thus, the main challenge of designing a meta-heuristic, such as Simulated Annealing (SA) to solve such a problem is designing an approximate measure of the robustness of a solution that can be calculated very quickly. In the work reported here, we develop and test some measures based on the degree and distribution of slack in a given timetable. The specific timetabling problem we address is the curriculum-based university course timetabling problem of ITC-2007 (see McCollum et al (2010)).

1.1 Disruption scenarios

Here we use the disruption scenarios that have been first developed and used by Akkan et al (2019). By assuming disruptions that affect two types of limited resources (time and rooms) we believe we introduce disruptions of sufficient variety and complexity. Specifically, we assume the following four types of disruptions may occur:

1. IP: The period to which a lecture of an instructor was assigned is no longer feasible for that instructor. This disruption is specified by a tuple $\langle i, p\rangle$ where i is an instructor, and p is the period to which one lecture of instructor i is assigned in Σ_0. If a disruption $\langle i, p\rangle$ is generated, unavailability constraints for all courses of instructor i at period p are added, unless such a constraint already exists.
2. CP: This disruption is specified by a tuple $\langle c, \mathcal{P}_1, \mathcal{P}_2\rangle$ for course c. Given the set of periods available for course c, denoted by $\mathcal{P}_c^C$, $\mathcal{P}_1 \subseteq \mathcal{P}_c^C$ is a set of consecutive periods on the same day that become infeasible (unavailable) for course c and at least one of these periods is used by course c in Σ_0. $\mathcal{P}_2 \subseteq \mathcal{P} \setminus \mathcal{P}_c^C$ is a set of consecutive periods that become available for course c such that $|\mathcal{P}_2| \leq |\mathcal{P}_1|$.
3. CS: The number of students for a course is increased beyond the capacity of the room assigned to at least one lecture of that course. Note that this does not cause infeasibility, as room capacity is a soft constraint in ITC-2007, but increases the penalty of the initial timetable. This disruption is specified by a tuple $\langle c, s\rangle$, where s is the new number of students for course c. Even if some lectures of the course are currently assigned to rooms with enough capacity, all the lectures of this course are included in the set of room-disrupted lectures.
4. RP: Availability of one room is lost for one or two consecutive periods on the same day. This disruption is specified by $\langle r, p, d\rangle$, where p is the first period that room r becomes unavailable, and d is the number of periods that become unavailable.

A set of disruptions of these types is referred to as a *disruption scenario*. All disruptions in a given scenario are aggregated in two sets of disrupted lectures. The set of lectures e, whose assigned periods in Σ_0 become infeasible due to IP and CP disruptions is denoted by E^P (*period-disrupted* lectures). The set of lectures e, whose assigned rooms in Σ_0 become either infeasible due to RP disruptions or have insufficient capacity due to CS disruptions is denoted by E^R (*room-disrupted* lectures). Then, the set of disrupted lectures, E^D, equals $E^P \cup E^R$, with size δ.

1.2 The robustness measure

The robustness objective is expressed as minimizing $E(R(S, Y_S))$, the expected value of a disruption measure $R(S, Y_S)$, where S is a given solution and Y_S is the random variable representing the disruptions.

Let, $\mathcal{F}(\sigma_i)$ be the set of all solutions that are feasible with respect to a disruption scenario σ_i and $D(S_0, S_1)$ be the Hamming distance between assigned period arrays for all lectures $T_e(S_0)$ and $T_e(S_1)$ of these two solutions. $D(S_0, S_1)$ is equal to the sum, over all courses, of the number of lectures that are assigned to different periods in these two solutions. Then, we define the following neighborhood set for a given solution S_0 and disruption scenario σ_i with δ_i^p period and δ_i^r room-disrupted lectures:

$$\mathcal{N}(S_0, \sigma_i) = \{S : D(S_0, S) \leq f(\delta_i^p, \delta_i^r); S \in \mathcal{F}(\sigma_i)\} \tag{1}$$

Thus, if solution S_0 is disrupted by scenario σ_i, then switching to any solution in $\mathcal{N}(S_0, \sigma_i)$ would restore feasibility by rescheduling at most $f(\delta_i^p, \delta_i^r)$ lectures to a different period, where $f : (\mathbb{N}, \mathbb{N}) \to \mathbb{N}$. Then, we define the robustness measure for a given solution S_0 and a disruption scenario σ_i as,

$$R(S_0, \sigma_i) = \min_{S \in \mathcal{N}(S_0, \sigma_i)} \left(P_{ave} \cdot 1_{D(S,S_0) > \delta_i^p} + (P(S) - P(S_0))^+ \right) \tag{2}$$

where $x^+ := \max(0, x)$ and P_{ave} is the average per lecture penalty for a randomly generated sample of solutions (see Gülcü and Akkan (2020)). P_{ave} is an additional penalty term added so that solutions that only reschedule period-disrupted lectures to different periods are favored. Thus, in addition to quality robustness measured by $(P(S) - P(S_0))^+$, by adding a fixed penalty cost for rescheduling more lectures than the period-disrupted lectures, $R(S_0, \sigma_i)$ incorporates a measure of solution stability. Solution stability is further ensured by adding the constraint $D(S_0, S) \leq f(\delta_i^p, \delta_i^r)$ in defining $\mathcal{N}(S_0, \sigma_i)$. If $\mathcal{N}(S_0, \sigma_i) = \emptyset$, then $R(S_0, \sigma_i)$ is set to a large value, denoted by B.

For solution S, an estimate of $E(R(S, Y_S))$ is calculated as a sample average $\frac{1}{|\mathcal{Y}|} \sum_{y \in \mathcal{Y}} R(S, y)$ for a sampled set of disruption scenarios $\mathcal{Y}$, since a closed-form calculation of $E(R(S, Y_S))$ is not possible. Given the robustness measure, R, and a set of randomly generated sample of disruption scenarios, $\sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_N\}$, $E(R)$ is estimated by $\overline{R}(S, \sigma) = (1/N) \sum_{i=1}^{N} R(S, \sigma_i)$. $\overline{R}(S, \sigma)$ is taken as the true robustness measure, using a reasonably large N. This approach bears some resemblance to the Sample Average Approximation (SAA) method of Kleywegt et al (2002). The algorithm that is used to repair solution S subject to a given disruption scenario is the Simulated Annealing algorithm discussed in detail in Akkan et al (2019), in which $f(\delta_i^p, \delta_i^r) = 2 \times \delta_i^p + 0.25 \times \delta_i^r$, and $B = 1200$.

2 Slack-based Estimators

The tested estimators are summary statistics of some measures of slack in a given timetable. These measures can be classified into three groups. The first group provides measures of slack for each period. Let, $X(p, r)$ equals 1 if room r is used at period p by some lecture, 0 otherwise; and $Rm(p)$ equals the number of rooms used at period p, while the total number of rooms is Rm.

Then, assuming the rooms are indexed in increasing capacity, define $\mathbf{RSA}[p]$, room-based slack at period p, as

$$\mathbf{RSA}[p] = \begin{cases} \frac{1}{Rm(p)} \sum_r \sum_{q>r} 1_{([X(p,r)=1, X(p,q)=0)}, & \text{if } Rm(p) > 0 \\ (Rm - 1), & \text{if } Rm(p) = 0 \end{cases} \quad (3)$$

The summation term in Eqn. 3, $\rho(p,r) = \sum_{q>r} 1_{([X(p,r)=1, X(p,q)=0)}$ gives for every room used at a period, the number of available larger capacity rooms in the same period.

It is quite reasonable to have decreasing marginal benefit in increasing $\rho(p,r)$, so an alternative slack measure could make use of an exponential utility function as $\mathbf{RSU}[p] = \frac{1}{Rm(p)} \sum_r \sum_{j>=1}^{\rho(p,r)} e^{-j}$ for a given period p.

Alternatively, we can assume there is utility in having at least one larger capacity room for a lecture scheduled at a given period, say at (p,r). In that case, we would have $\rho(p,r) > 0$. Then, another slack measure can be defined as $\mathbf{RSB}[p] = \frac{1}{Rm(p)} \sum_r 1_{\rho(p,r)>0}$ for a given period p.

The second group of estimators make use of slack measured for each room. For room r, letting, $\pi(r) = \sum_t 1_{X[t,r]=0}$ denote the the number of available periods in the same room, we define $\mathbf{PSU}[r] = \sum_{j=1}^{\pi(r)} e^{-j}$ as a slack measure for a given room r.

The next measure is a more finely grained version of $\mathbf{PSU}[r]$, where the periods are sub-divided into daily sets. Let the day of a given period p be denoted by $D(p)$. Then, $\xi(d,r) = \sum_{p:D(p)=d} 1_{X[p,r]=0}$ represents the number of available time-slots on day d at room r, and we define $\mathbf{DSU}[d,r] = \sum_{j=1}^{\xi(d,r)} e^{-j}$ as a slack measure for a given room r on day d.

The third group of estimators measure course-specific availability of periods. We first let,

$\mathcal{Y}(p)$ = the set of courses scheduled at period p.
$\mathcal{FR}(p)$ = the set of free rooms in period p
$FR(p)$ = the number of free rooms at period p.
$K(r)$ = the capacity of room r
$K^{max}(p)$ = $\max_{r \in \mathcal{FR}(p)}\{K(r)\}$
$S(c)$ = the number of students planned for course c
$\mathcal{F}(c)$ = the set of feasible periods for course c.
$\mathcal{C}(c)$ = the set of conflicting courses for course c

Note that courses in the same curriculum or taught by the same teacher are referred to as conflicting courses. We then define the following sets,

$\mathcal{AP}(c)$ = $\{p : p \in \mathcal{F}(c), FR(p) > 0\}$
$\mathcal{AP}^+(c)$ = $\{p : p \in \mathcal{AP}(c), K^{max}(p) > S(c)\}$
$\mathcal{CP}(c)$ = $\{p : p \in \mathcal{AP}(c), \mathcal{Y}(p) \cap \{\mathcal{C}(c) \cup c\} = \emptyset\}$
$\mathcal{CP}^+(c)$ = $\{p : p \in \mathcal{AP}^+(c), \mathcal{Y}(p) \cap \{\mathcal{C}(c) \cup c\} = \emptyset\}$

Thus, $\mathcal{AP}(c)$ gives the set of periods available for course c, $\mathcal{CP}(c)$ gives the set of conflict free periods available for course c and $\mathcal{CP}^+(c)$ gives the set of conflict free periods available for course c, having at least one free room with sufficient capacity.

Given these sets, we can define the array $\mathbf{C}[c]$ that contains the number of conflict-free available periods for each course, as $\mathbf{C}[c] = |\mathcal{CP}(c))|$. Similarly, the array of the number of conflict-free available periods with free rooms of sufficient capacity, for each course c, is defined as $\mathbf{R}[c] = |\mathcal{CP}^+(c)|$.

Rather than simply counting the number of available periods, as a basis of the next slack measure, we calculate the utility of the conflict-free available periods using an exponential utility function, that gives decreasing marginal utility with increasing number of conflict-free available periods. The array of these utilities is defined as $\mathbf{UC}[c] = \sum_{j=1}^{|\mathcal{CP}(c)|} e^{-j}$, for each course c. Similarly, we define a utility function that is based on conflict-free available periods with available rooms of sufficient capacity as $\mathbf{UR}[c] = \sum_{j=1}^{|\mathcal{CP}^+(c)|} e^{-j}$, for each course c. Furthermore, one can argue that the value of $\mathbf{C}[c]$ depends on the number of lectures of course c, $L(c)$, so we defined two additional arrays $\mathbf{CED}[c]$ and $\mathbf{CER}[c]$, as $\mathbf{CED}[c] = \mathbf{C}[c] - L(c)$, and $\mathbf{CER}[c] = \mathbf{C}[c]/L(c)$.

Given these eleven slack measuring arrays (3 defined for each period, 2 for each room, and 6 for each course), we calculate the average, standard deviation and the coefficient of variation (standard deviation over average) of each array as estimators of the robustness of the given timetable. These three summary statistics for a given slack measure $\mathbf{S}$ are denoted as $\overline{S}$, SD_S, CV_S, respectively.

3 Computational results

For the computational experiments we selected four ITC-2007 instances, namely ITC1, ITC2, ITC5 and ITC12. They are among the most constrained (thus potentially difficult) instances in terms of conflict intensity, teacher availability, and room occupancy (Bonutti et al (2012)). Then, for each instance, a set of 48 solutions were selected to carry out correlation analysis between the slack metrics and the robustness measure, $\overline{R}$. The purpose of the selection procedure was to obtain a diverse set of solutions, from among the set of solutions accepted through a Simulated Annealing algorithm designed to minimize the penalty. A brief discussion of the selection procedure is provided below, interested readers could find the details in Akkan et al (2020).

3.1 Solutions selected for analysis

The solutions that were found in the SA search process were used to form a network of solutions, where each solution is represented by a node. Selection of solutions for the computational experiments was based on two of their characteristics: the penalty value and the degree of the node. For each ITC instance four networks were generated, and 12 solutions were selected from each network. For the instance ITCi, the network $N_i^{nc,s}$ was generated by collecting *nc thousand* solutions accepted by the SA algorithm (we used 50 and 100 thousand). Starting with the solution accepted in the last iteration of SA, going backwards and skipping every s^{th} accepted solution, a total of

Table 1 Penalty and degree intervals used to select the sample of solutions from $N_2^{50,0}$

P : 75	76	77	78	79	80
[13, 32]	[13, 32]	[13, 32]	[33, 52]	[33, 52]	[53, 72]
[173, 192]	[133, 152]	[113, 132]	[93, 112]	[93, 112]	[73, 92]

nc thousand solutions are collected into the network (we used s equals 0 and 1). Letting $G(N, E)$ denote a solution network, we let the neighbors of node v be the set of nodes w such that $D(v, w) \leq \rho$, where $D(v, w)$ is the number of lectures with different assigned periods in v and w, and ρ is an integer parameter. ρ could be seen as the maximum number of events that would need to be rescheduled to a *different period* in order to respond effectively to a disruption scenario. We calculated ρ for each ITC instance as $2 \times N^p + 0.25 \times N^r$, where N^p and N^r are the maximum number of period-based and room-based disruptions possible for that instance (for ITC2 $N^p = 8$ and $N^r = 16$ events). Thus, the degree of a solution is likely to be correlated with the robustness of the solution, as it indicates the size of the solution space that could be used to repair that solution.

Frequency tables were formed of all solutions in each network based on intervals of penalty and degree, and then a solution was selected from the solutions that fall into selected intervals. The selections were made from among the solutions with penalties that are close to the minimum penalty value, P_{min}, found by the SA algorithm. For each network, six penalty-intervals were selected. For example, for ITC2 $P_{min} = 75$ and all solutions had one of the six penalty values listed in Table 1, so intervals were only defined for the degrees. Given the penalty interval (or the penalty) we randomly sampled one solution from the smallest degree interval, and the second solution from the largest degree interval. In a few cases when a degree interval contained only already selected solutions, we moved to the adjacent degree interval for the same penalty interval.

3.2 Correlation analysis

Pearson correlations coefficients, ρ_i^m, were calculated between each slack metric, say m, and the robustness measure $\overline{R}$ using the set of 48 solutions for each instance $ITCi$. Then the absolute values of these correlations were ranked among those for each instance, in decreasing order so that the largest one is ranked first. Letting ϱ_i^m denote the rank of $|\rho_i^m|$, the average rank of metric m was calculated as, $\overline{\varrho}^m = 1/4 \sum_{i \in \{1,2,5,12\}} \varrho_i^m$. For the nine best ranking metrics, their correlation coefficients and average ranks, $\overline{\varrho}^m$, are reported in Table 2. For each of these correlation coefficients, a test of hypothesis was done where the null hypothesis states the correlation is equal to zero.

Based on the correlations presented in Table 2 we chose three of the metrics for further analysis, namely, CV_{CER}, $\overline{UC}$, and CV_C, which are the best ones for their corresponding arrays. CV_{CER} has the best overall average rank with

Table 2 Pearson correlation coefficients with $\overline{R}$

	CV_{CER}	$\overline{UC}$	$\overline{CER}$	CV_{UC}	SD_{UC}	CV_C	$\overline{RSU}$	$\overline{DSU}$	CV_{RSA}
Ave. Rank	8.5	9.5	10.25	10.75	11.75	12.25	13.25	14	14.25
ITC1	−0.226	−0.227	−0.184	0.211	0.205	0.142	0.106	$0.285^{\flat}$	−0.120
ITC2	$0.292^{\flat}$	$-0.384^{\sharp}$	−0.060	$0.409^{\sharp}$	$0.411^{\sharp}$	$0.303^{\flat}$	$0.429^{\sharp}$	0.088	$-0.426^{\sharp}$
ITC5	$0.296^{\flat}$	−0.169	$-0.321^{\flat}$	0.160	0.150	0.256^{*}	−0.219	0.199	0.152
ITC12	$0.290^{\flat}$	$-0.319^{\flat}$	$-0.375^{\sharp}$	0.271^{*}	0.255^{*}	0.277^{*}	0.176	0.154	−0.205

$\sharp: p < .01$, $\flat: p < .05$, $*: p < .10$

statistically significant correlations for three instances. Correlation between $\overline{UC}$ and $\overline{R}$ is negative for all four instances, although for two of them we have sufficiently low p-values to reject the null hypothesis of 0 correlation. CV_C, on the other hand, is consistently positively correlated with $\overline{R}$ and for three of the instances, the correlations associated with CV_C are different from zero at a statistically significant level. Recall that $\mathbf{C}[c]$ array contains the number of conflict-free available periods for each course. Thus, the positive correlation associated with CV_C suggests that the more evenly such periods are distributed across courses and the larger the average number of conflict-free available periods is, the more robust the solution would be (with smaller $\overline{R}$). CV_{UC}, the utility-adjusted version of CV_C, yields a similar performance to CV_C. The negative correlations associated with $\overline{UC}$ is consistent with this interpretation.

For all instances, the scatter plots of the nine slack metrics with $\overline{R}$ were plotted. We observed for ITC5 that in the plot for $\overline{RSU}$ there is an outlier solution that is increasing the correlation, however the hypothesis testing resulted in not rejecting a zero correlation (see Figure 1). For the other scatter plots, we did not see such a case of a single outlier.

3.3 Accuracy in identifying the Pareto frontier

The planned use of a slack metric is as an estimate of robustness within a multi-objective Simulated Annealing algorithm (MOSA) with two objectives: the penalty of the solution and its robustness. A MOSA algorithm maintains an archive of solutions which comprises the best Pareto frontier at each iteration. A new solution enters the frontier if there is no solution in the current frontier which dominates it. The number of solutions that dominate a given solution s is referred to as the domination count of solution s and the rank of that solution, $r(s)$, is equal to its domination count plus 1. So, a frontier is comprised of solutions of rank 1. Since the MOSA algorithm will be designed to use the slack metric M rather than the robustness measure $\overline{R}$, and M is a estimator, a solution with rank 2 defined by (P, M) might easily be on the Pareto frontier defined by $(P, \overline{R})$. Thus, it would be reasonable to keep in the archive, solutions s with $r(s) \leq K$, where K is an integer cutoff value, rather than $r(s) = 1$.

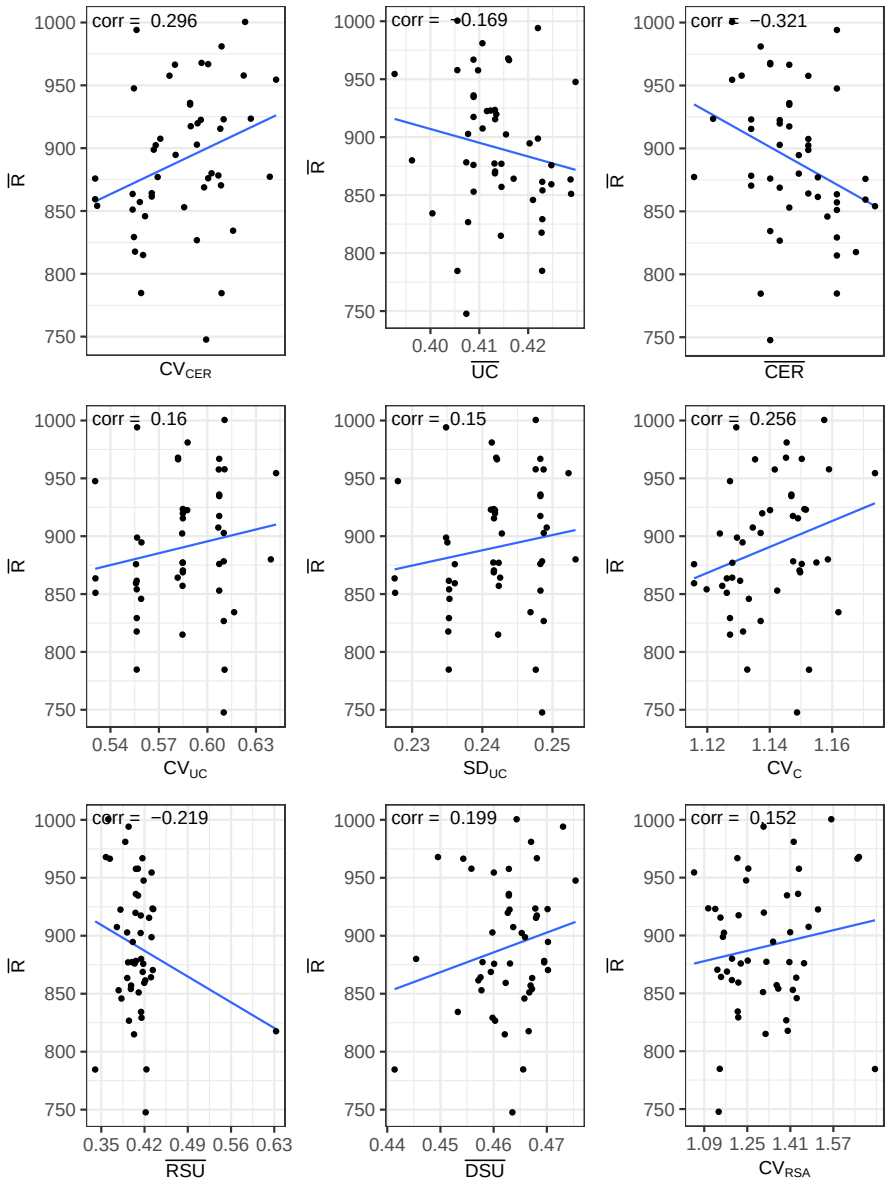


Fig. 1 ITC5: scatter plots of $\overline{R}$ versus selected slack metrics

In this case, the final archive at the end of the MOSA algorithm's run, $\mathcal{A}$, would be a short-list of solutions that are highly likely to contain the solutions forming the Pareto frontier based on $(P, \overline{R})$, $\mathcal{F}$. We then would calculate the robustness $\overline{R}(s)$ for all $s \in \mathcal{A}$ and obtain the approximate Pareto frontier $\widetilde{\mathcal{F}}$, based on $(P, \overline{R})$.

Given the approach discussed above, for a metric M to performs well, we would want a large $P(s \in \mathcal{A} | s \in \mathcal{F})$, and similarly a large $P(s \notin \mathcal{A} | s \notin \mathcal{F})$. To estimate these probabilities, we have done the following experimental analysis. For all 48 solutions of each ITC instance i we calculated their $\overline{R}$ values and determined the corresponding Pareto frontier $\mathcal{F}_i$. Then for each slack metric M, we calculated the rank $r(s)$ of each solution based on (P, M) and determined the solutions that fall into the archive $\mathcal{A}(K)_i$ defined by the cutoff value K. Based on the combined sample of 192 solutions, $\mathcal{S}_i$ for $i = 1, 2, 5, 12$, the estimates of $P(s \in \mathcal{A} | s \in \mathcal{F})$ and $P(s \notin \mathcal{A} | s \notin \mathcal{F})$, denoted by $f^+(K)$ and $f^-(K)$, respectively, are defined as:

$$f^+(K) = \frac{\sum_i |\{s : s \in \mathcal{A}(K)_i \, , s \in \mathcal{F}_i\}|}{\sum_i |\mathcal{F}_i|} \tag{4}$$

$$f^-(K) = \frac{\sum_i |\{s : s \in \mathcal{S}_i \setminus \mathcal{A}(K)_i \, , s \in \mathcal{S}_i \setminus \mathcal{F}_i\}|}{\sum_i |\mathcal{S}_i \setminus \mathcal{F}_i|} \tag{5}$$

Table 3 Accuracy in identifying the solutions on the Pareto frontier

	$\overline{UC}$		CV_{CER}		CV_C	
K	$f^+(K)$	$f^-(K)$	$f^+(K)$	$f^-(K)$	$f^+(K)$	$f^-(K)$
1	0	0.939	0	0.939	0.091	0.917
2	0.182	0.884	0.273	0.884	0.273	0.884
3	0.455	0.840	0.364	0.823	0.364	0.818
4	0.545	0.796	0.455	0.762	0.636	0.768
5	0.636	0.746	0.636	0.729	0.727	0.740
6	0.636	0.713	0.636	0.685	0.727	0.685
7	0.636	0.669	0.727	0.641	0.818	0.646
8	0.727	0.608	0.727	0.613	0.818	0.597

Table 3 presents the above fractions for the three slack metrics selected based on the correlation analysis in Section 3 for the cutoff values $K = 1, \ldots, 8$. For $K \geq 4$, we can see that $f^+(K)$ is consistently larger for CV_C than for the two other metrics. On the other hand, $f^-(K)$ is consistently larger for $\overline{UC}$ than the other two. The chart in Figure 2 suggests that $K = 5$ constitutes a good trade-off between $f^+(K)$ and $f^-(K)$ for CV_C. At $K = 5$, CV_C gives a better performance than CV_{CER} in terms of both $f^+(K)$ and $f^-(K)$. On the other hand, comparing CV_C and $\overline{UC}$ we observe that their $f^-(K)$ values are almost identical but CV_C has a better $f^+(K)$.

4 Concluding remarks

In this work an attempt has been made to develop approximate measures of robustness in the form of slack-based estimators that could be used within a Simulated Annealing algorithm, which would identify the Pareto frontier

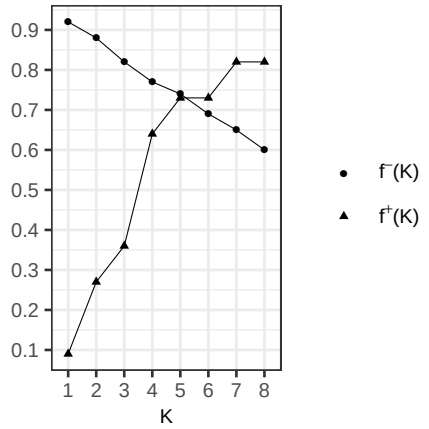


Fig. 2 CV_C accuracy: the tradeoff between $f^+(K)$ and $f^-(K)$

defined by the penalty of a timetable and a measure of its robustness. Finding such fast-to-compute estimators is needed because the calculation of the robustness measure requires too much computational effort when there are multiple disruptions of different types.

The approximate nature of the estimators suggest that they can be used to identify a Pareto "band", comprised of solutions with rank less than or equal to a cutoff value, as opposed to the frontier comprised of solutions with rank equals 1. The performance quality of a band is judged by how accurately it distinguishes solutions that are on the true Pareto frontier (defined by the robustness measure) from those that are not. To this end, for each cutoff value and estimator, we calculated estimates of the probability of a solution on the true frontier being in the band and one not on the true frontier not being in the band. This is done on a sample of 192 solutions (48 solutions for each of four ITC 2007 instances). The results suggest that CV_C, the coefficient of variation of the number of conflict-free available periods for each course, is the best one among the 33 estimators tested.

The experimental analysis presented here should be seen as a preliminary work, as this is currently a work in progress. We are implementing a MOSA (Multi-Objective Simulated Annealing) algorithm that uses a given estimator and maintains the Pareto "band" as opposed to the Pareto frontier in its archive of solutions. Potential extensions of this work could include finding other slack-based estimators, and also investigating different robustness measures for which slack-based estimators can provide a better performance.

References

Akkan C, Gülcü A, Kuş Z (2019) Minimum penalty perturbation heuristics for curriculum-based timetables subject to multiple disruptions, unpublished

Akkan C, Gülcü A, Kuş Z (2020) Search space sampling by simulated annealing for identifying robust solutions in course timetabling. In: Proceedings of the 2020 IEEE Congress on Evolutionary Computation, to appear

Bonutti A, De Cesco F, Di Gaspero L, Schaerf A (2012) Benchmarking curriculum-based course timetabling: formulations, data formats, instances, validation, visualization, and results. Annals of Operations Research (194):59–70, DOI 10.1007/s10479-010-0707-0

Gülcü A, Akkan C (2020) Robust University Course Timetabling Problem Subject to Single and Multiple Disruptions. European Journal of Operational Research 282:630–646, DOI 10.1016/j.ejor.2019.11.024

Hardy B, Lewis R, Thompson J (2018) Tackling the edge dynamic graph colouring problem with and without future adjacency information. Journal of Heuristics 24(3):321–343, DOI 10.1007/s10732-017-9327-z

Jin Y, Branke J (2005) Evolutionary optimization in uncertain environments-a survey. IEEE Transactions on Evolutionary Computation 9(3):303–317, DOI 10.1109/TEVC.2005.846356

Kingston JH (2013) Educational Timetabling. In: Uyar AS, Ozcan E, Urquhart N (eds) Automated Scheduling and Planning: From Theory to Practice, Springer, Berlin, Heidelberg, DOI 10.1007/978-3-642-39304-4

Kleywegt AJ, Shapiro A, Homem-de Mello T (2002) The Sample Average Approximation Method for Stochastic Discrete Optimization. SIAM Journal on Optimization 12(2):479–502, DOI 10.1137/S1052623499363220

Lindahl M, Stidsen T, Sørensen M (2019) Quality recovering of university timetables. European Journal of Operational Research 276(2):422–435, DOI 10.1016/j.ejor.2019.01.026

McCollum B (2007) A perspective on bridging the gap between theory and practice in university timetabling. Practice and Theory of Automated Timetabling, LNCS 3867:3–23

McCollum B, Schaerf A, Paechter B, McMullan P, Lewis R, Parkes AJ, Di Gaspero L, Qu R, Burke EK (2010) Setting the research agenda in automated timetabling: The second international timetabling competition. INFORMS Journal on Computing 22(1):120–130, DOI 10.1287/ijoc.1090.0320

Müller T, Rudová H, Barták R (2005) Minimal Perturbation Problem in Course Timetabling. In: Burke E, Trick M (eds) Practice and Theory of Automated Timetabling V. Lecture notes in computer science (Vol. 3616), vol 3616, pp 126–146, DOI 10.1007/11593577_8

Phillips AE, Walker CG, Ehrgott M, Ryan DM (2017) Integer programming for minimal perturbation problems in university course timetabling. Annals of Operations Research 252(2):283–304, DOI 10.1007/s10479-015-2094-z

Yasari P, Ranjbar M, Jamili N, Shaelaie MH (2019) A two-stage stochastic programming approach for a multi-objective course timetabling problem with courses cancelation risk. Computers & Industrial Engineering 130:650–660, DOI 10.1016/j.cie.2019.02.050

A multiple metaheuristic variable neighborhood search framework for the Uncapacitated Examination Timetabling Problem

Panayiotis Alefragis · Christos Gogos · Christos Valouxis · Efthymios Housos

Received: date / Accepted: date

Abstract The paper describes a framework that was developed to solve the Uncapacitated Examination Timetabling Problem. It also presents a way of reducing problem sizes by removing students and examinations that ultimately have no impact. Moreover, it presents some loose lower bounds that are computed for reference problems. The framework allows the collaboration of multiple metaheuristic algorithms. A common problem and solution representation is created. Multiple evaluators are available and a mechanism to select variable neighborhoods is implemented. A number of simple and complex neighborhoods is created. The application of the framework creates competitive results compared to the best ones available in the literature for the Toronto-b dataset.

Keywords uncapacitated examination timetabling · metaheuristics · framework

1 Introduction

The examination timetabling problem in general involves assigning examinations to a limited number of periods and rooms while respecting a set of hard

Panayiotis Alefragis
Electrical & Computer Engineering Department, University of Peloponnese, Patras, Greece
E-mail: alefrag@uop.gr

Christos Gogos
Informatics & Telecommunications Department, University of Ioannina, Arta, Greece
E-mail: cgogos@uoi.gr

Christos Valouxis
Electrical & Computer Engineering Department, University of Patras, Patras, Greece
E-mail: valouxis@ece.upatras.gr

Efthymios Housos
Electrical & Computer Engineering Department, University of Patras, Patras, Greece
E-mail: housos@ece.upatras.gr

constraints and at the same time trying to minimize the violation of soft constraints. The Uncapacitated Examination Timetabling Problem (UETP) does not consider room capacity requirements. The only hard constraint is that it is not allowed for a student to sit on two examinations at the same period. On the other hand, the only soft constraint is that of spreading examinations as evenly as possible from the student's perspective. A formal description of the problem as an Integer Programming problem follows: In this problem setting, C is the set of courses, P is the set of periods and S is the set of students. Each course c has a number of enrollments which is captured at E_c. In sync, each student s has a set of courses C_s that he is enrolled to. The set CNF contains triplets of the form c_1, c_2, co which represent the situation in which course c_1 and course c_2 have co in common students.

In order to define for each course the period that it will be scheduled the decision variables m_{cp} are used for each $c \in C$ and $p \in 1..P$ which are binary variables assuming value 1 if the course c is scheduled to period p or 0 otherwise. Moreover, the binary variables $y1_{t,p}$ are defined for each $t \in CNF$ and $p \in 1..|P-1|$ assuming value 1 when courses $t.c_1$ and $t.c_2$ are scheduled in consecutive periods and 0 otherwise. Likewise, binary variables $y2_{t,p}, y3_{t,p}, y4_{t,p}, y5_{t,p}$ are defined and assume value 1 when courses $t.c_1$ and $t.c_2$ are scheduled to periods with distance between 2 and 5 periods respectively.

The objective function is presented in Equation 1

$$\begin{aligned} minimize \sum_{t \in CNF} t.co\big(&16 \sum_{p \in 1...|P-1|} y1_{t,p} \\ &+8 \sum_{p \in 1...|P-2|} y2_{t,p} \\ &+4 \sum_{p \in 1...|P-3|} y3_{t,p} \\ &+2 \sum_{p \in 1...|P-4|} y4_{t,p} \\ &+ \sum_{p \in 1...|P-5|} y5_{t,p}\big) \end{aligned} \tag{1}$$

The first constraint states that each course should be scheduled at exactly one period and is captured in equation 2.

$$\sum_{p \in P} m_{c,p} = 1 \quad \forall c \in C \tag{2}$$

The second constraint shown in equation 3 prohibits every possible pair of courses having common students to be scheduled at the same period.

$$m_{t.c_1,p} + m_{t.c_2,p} \leq 1 \quad \forall p \in P, \forall t \in CNF \tag{3}$$

The third set of constraints equations 4 to 8 defines variables $y1_t$ to $y5_t$.

$$\left.\begin{aligned} y1_{t,p} &\geq m_{t.c_1,p} + m_{t.c_2,p+1} - 1 \\ y1_{t.p} &\geq m_{t.c_2,p} + m_{t.c_1,p+1} - 1 \end{aligned}\right\} \forall t \in CNF, \forall p \in 1 \ldots |P| - 1 \tag{4}$$

$$\left.\begin{array}{l} y2_{t,p} \geq m_{t.c_1,p} + m_{t.c_2,p+2} - 1 \\ y2_{t,p} \geq m_{t.c_2,p} + m_{t.c_1,p+2} - 1 \end{array}\right\} \forall t \in CNF, \forall p \in 1 \ldots |P| - 2 \quad (5)$$

$$\left.\begin{array}{l} y3_{t,p} \geq m_{t.c_1,p} + m_{t.c_2,p+3} - 1 \\ y3_{t,p} \geq m_{t.c_2,p} + m_{t.c_1,p+3} - 1 \end{array}\right\} \forall t \in CNF, \forall p \in 1 \ldots |P| - 3 \quad (6)$$

$$\left.\begin{array}{l} y4_{t,p} \geq m_{t.c_1,p} + m_{t.c_2,p+4} - 1 \\ y4_{t,p} \geq m_{t.c_2,p} + m_{t.c_1,p+4} - 1 \end{array}\right\} \forall t \in CNF, \forall p \in 1 \ldots |P| - 4 \quad (7)$$

$$\left.\begin{array}{l} y5_{t,p} \geq m_{t.c_1,p} + m_{t.c_2,p+5} - 1 \\ y5_{t,p} \geq m_{t.c_2,p} + m_{t.c_1,p+5} - 1 \end{array}\right\} \forall t \in CNF, \forall p \in 1 \ldots |P| - 5 \quad (8)$$

When courses c_1 and c_2 have distance 1 either $m_{t.c_1,p}$ and $m_{t.c_2,p+1}$ or $m_{t.c_2,p}$ and $m_{t.c_2,p+1}$ assume value 1. If this is the case variable $y1_t$ will be 1. If at least one of $m_{t.c_1,p}$ or $m_{t.c_2,p+1}$ are 0 the value of $y1_t$ could be 0 or 1 but since it is included in the objective which is minimized it will take the value 0.

This paper describes a framework that was developed to solve the UETP. The most studied dataset was introduced by Carter [14]. Characteristics of the used problem instances are presented in Table 1. The specific dataset is heavily studied, but it is still possible to generate improved solutions compared to the ones published in literature. The specific dataset has the property that while it has very low learning curve and relative simple structure it is still challenging.

Table 1 Dataset Problem characteristics (Toronto-b [22])

Problem	Exams	Students	Admissions	Density	Slots
CAR91	682	16925	56877	0.13	35
CAR92	543	18419	55522	0.14	32
EAR83	190	1125	8109	0.27	24
HEC92	81	2823	10632	0.42	18
KFU93	461	5349	25113	0.06	20
LSE91	381	2726	10918	0.06	18
PUR93	2419	30029	120681	0.03	42
RYE92	486	11483	45051	0.07	23
STA83	139	611	5751	0.14	13
TRE92	261	4360	14901	0.18	23
UTA92	622	21266	58979	0.13	35
UTE92	184	2749	11793	0.08	10
YOR83	181	941	6034	0.29	21

The paper is organized as follows. Section 2 presents related work, section 3 presents some analysis on the dataset instances, section 4 presents a high level framework design and describes the metaheuristic algorithms and the neighborhoods supported by the framework, section 5 presents generated results and finally section 6 presents conclusions and future work.

2 Related Work

Several approaches have been proposed for solving the UETP. The datasets introduced by [14], collectively known as Toronto datasets, have been typically used as a testbed for demonstrating the effectiveness of new approaches. So, a great number of published results exist over those datasets, showing that approximation methods manage to produce better results, when compared to exact methods. This might occur due to the large size of the problem instances and the symmetries among several different schedules that can be produced. Both these factors prohibit Integer Programming solvers, Constraint Programming solvers and SAT solvers to solve the full problem behind each problem instance.

A non-exhaustive list of approaches that have given good results follows. Algorithms based on local search were proposed in [13]. Several hybridizations of metaheuristics with sequential heuristics were used in [24]. The flex-deluge algorithm and the late acceptance strategy for solving UETP where proposed in [8] and [9] respectively. A hybrid variable neighborhood search was used in [12]. In [15] the problem was approached using hyperheuristics. In [4] the problem was solved using a hybrid bee colony optimization method. Another, recent hybrid approach that combines a cellular memetic algorithm with the threshold acceptance metaheuristic is described in [20].

It should be noted that the survey paper in [22] presents many details about the examination timetabling problem in general and in particular about the UETP. A review paper focusing on UETP recently appeared in [2].

3 Some thoughts on the problem and a few loose lower bounds

The simplicity of describing the UETP problem is at odds with the hardness of optimally solving it. This situation strikes resemblance with the iconic combinatorial optimization problem, Traveling Salesman Problem (TSP) [7]. For TSP, large problem instances have proven optimal solutions, while several solvers exist capable of obtaining optimal solutions for TSP instances of moderate to large sizes. In particular, the freely available Concorde TSP solver [5] has the record of obtaining optimal solutions for all TSPLIB [23] instances. It is impressive that Concorde solver can optimally solve the largest TSPLIB instance, counting 85,900 cities, while back in 1962, a problem with only 33 cities was considered very difficult to be solved and a competition was hosted by Procter & Gamble in order to solve it. Moreover, significant theoretical work has been gradually done for TSP in order to provide solutions within a guaranteed distance from the optimal ones or in order to find lower bounds (e.g. Christofides algorithm, 1-tree Lagrangean relaxation). So, new solutions can be quantified according to how close they are to the theoretically optimal ones. For example, the world tour TSP problem with 1,904,711 locations has been solved with total length of only 0.0474% greater than the length of the theoretically optimal tour. Unfortunately, UETP has not achieved the level of

maturity that TSP has reached. There are several approaches that give good results, but this is mostly justified by comparing them with results obtained for the same problem instances by other researchers. The popularity of TSP, the longest time that TSP has been at the center of interest for the scientific community, its adaptation to numerous practical combinatorial optimization problems (e.g. transportation problems, printed circuits, logistics and others) might be among the reasons why research on TSP is much more mature than in UETP.

By simple analyzing the data of the Toronto datasets a few remarks can be made. Firstly, if there are students that participate in only one examination, then their presence is irrelevant to the solution of the problem. We call these students "noise students", since they have no contribution to the cost of the final schedule. Secondly, if there are courses that have only "noise students" then these courses can also be safely removed from the problem, since they can be scheduled at any period without affecting the cost of the schedule. We call these courses "noise courses". It turns out "noise" students and courses exist for the Toronto datasets. Relevant results are displayed in Table 2 and suggest that new equivalent versions of the datasets might be constructed by omitting the "noise" students and courses. The rightmost column of the Table displays the actual identification numbers of the "noise" courses that are used at the corresponding datasets.

Table 2 Carter datasets "Noise"

Problem	#"Noise" Students	#"Noise" Courses	Set of "Noise" Courses
CAR91	3409	4	{0033, 0349, 0440, 0657}
CAR92	3969	1	{0519}
EAR83	1	0	Ø
HEC92	321	0	Ø
KFU93	276	17	{0006, 0016, 0022, 0050, 0095, 0122, 0178, 0204, 0205, 0216, 0285, 0329, 0330, 0355, 0369, 0381, 0443}
LSE91	99	2	{0168, 0256}
PUR93	2627	6	{0153, 0552, 0976, 0983, 1454, 1520}
RYE92	2025	1	{0304}
STA83	0	0	Ø
TRE92	667	1	{0186}
UTA92	6180	0	Ø
UTE92	78	0	Ø
YOR83	1	0	Ø

3.1 Loose lower bounds

Bellow, an idea for calculating some loose lower bounds for the UECP problem instances is described. Each student has a set of courses he is enrolled to. These courses should be scheduled to different periods and each assignment gives an associated cost based on the proximity among the exams. The possible permutations of scheduling E exams over P periods are $\frac{P!}{(P-E)!}$. For each permutation the associated cost is easy to be computed. Since the total number of permutations might be prohibitively large in order to enumerate all permutations and compute its cost, one can notice that the number of examinations that can be scheduled with zero cost in P periods are $L = \lfloor \frac{P-1}{6} \rfloor + 1$. So, when the number of examinations a student is enrolled into is no more than L then the theoretical best cost contribution that can be achieved for this student is zero. Else, a simple optimization problem can be solved that finds the minimum cost of positioning the E exams of a single student in P periods. A lower bound to the problem can be identified when the examinations of each student are scheduled to the exams into the period permutation with the minimum cost. Such an arrangement might not be possible since it does not consider the fact that examinations are common between students and each examination should be scheduled to the same period for all students. Nevertheless, it provides a lower bound for the problem instances. These bounds, alongside with the maximum number of courses that a student is enrolled to are presented in Table 3.

Table 3 Carter dataset lower bounds

Problem	Periods	Courses per Student (max)	Lower Bound
CAR91	35	9	0.01
CAR92	32	7	0.01
EAR83	24	10	17.85
HEC92	18	7	3.49
KFU93	20	8	5.63
LSE91	18	8	2.76
PUR93	42	9	N/A
RYE92	23	10	3.79
STA83	13	11	105.92
TRE92	23	6	0.59
UTA92	35	7	N/A
UTE92	10	6	19.25
YOR83	21	14	18.50

For two of the problem instances, STA83 and UTE92 the lower bounds are relatively close to the best values obtained in the literature. For two other problem instances, PUR93 and UTA92 no lower bound can be found in this way. Following the procedure that was described above the computed value becomes zero. This occurs because there are enough timeslots that allow even

for students that have enrolled to the maximum number of courses to have their examinations spread with no mandatory penalty.

4 The multi metaheuristic VNS framework

4.1 The multi heuristic strategy

The proposed framework which is an extension of Variable Neighborhood Search (VNS) [21] uses a number of basic metaheuristic algorithms that are orchestrated using a promising solution that has already been found from some other algorithm and each one tries to improve it and pass the result to the next one. Each metaheuristic algorithm uses a number of neighborhoods, called moves, to select the next candidate solution. Simple moves are randomly selected, while complex moves are triggered when no progress is achieved using just simple moves. Table presents the names of the used algorithms.

4.2 Supported metaheuristic algorithms

4.2.1 Simulated Annealing

Simulated Annealing (SA) was proposed by Kirkpatrick et al. [18]. It is a stochastic metaheuristic algorithm, which accepts inferior quality candidate solutions with probability $P = e^{(I-C)/T}$, where C and I are the cost of the incumbent and the candidate solutions respectively and T is a control parameter called "temperature". The application of the algorithm to solve the UETP is heavily studied. In our implementation, we have developed a number of alternative cooling schemes and reheating mechanisms.

4.2.2 Late Acceptance Hill Climbing

The Late Acceptance Hill-Climbing (LAHC) [11] is a local search metaheuristic algorithm, which may accept inferior candidate solutions if their cost is better than the solution accepted k iterations before. The algorithm maintains a cyclic buffer of the cost of the last k iterations. The performance of the algorithm is mainly effected by the value of the k parameter. The algorithm implementation in the framework adapts the value of k depending on the progress of the last execution of the LAHC algorithm. At the beginning of the search strategy a relative small value of k is used to promote the algorithm to mainly accept improving solutions. When no significant progress is achieved, the value of k is progressively increased.

4.2.3 Flex Deluge

The Flex Deluge (FD) [10] is a local search metaheuristic algorithm that is a variant of the Great Deluge (GD) algorithm. The GD algorithm may accept

inferior candidate solutions if their cost is better than an upper bound denoted by B. The FD algorithm has an additional k_f parameter that denotes the degree of its flexibility. A k_f value of 0 turns the algorithm to a Hill Climbing algorithm while a value of 1 makes the algorithm identical to the Great Deluge algorithm. The algorithm accepts a new candidate solution with cost Z if $Z < C + k_f(B - C)$ where C is the cost of the current solution.

4.3 Supported neighborhoods

The proposed framework support multiple neighborhoods that the active metaheuristic algorithms select in each iteration to generate a candidate solution. We will call these neighborhoods "moves". The implemented moves are grouped in two groups. The first group contains simple moves that perform small transformations to the incumbent solution. The second group contains more elaborate changes to the incumbent solution and may perform a relatively large local search before proposing a new candidate solution, making them more computationally expensive. The moves are presented in Table 4.

Table 4 Search space exploration

Key	Simple Moves	Key	Complex Moves
S1	BestKempeChain	C1	PenaltyDecreaserMove
S2	RandomKempeChain	C2	RuinAndRecreateMove
S3	MoveWorstExam	C3	SaturationImproverMove
S4	ForceSingleExamAtBest	C4	CyclicExchangeMove
S5	SingleExamAtBestSlot	C5	TimeRelaxerMove
S6	ExchangeExams		
S7	Move Single Exam		

4.3.1 Random and Best Kempe Chain

The first available neighborhood structures uses the concept of Kempe Chains (KCs), initially used to solve the "four color problem". Given the two sets of exams assigned to two periods, a number of chains consisting of exams belonging to either one of the periods is constructed. When an exam of a chain is moved from one period to the other then all other exams of the same chain are also moved to maintain feasibility. Initially, both moves randomly selects two slots and generates all possible KCs between these slots. Then the Random KC move shuffles the generates KCs and sequentially evaluates the difference in the cost, if the move was applied. If it finds a KC that produce a better solution than the incumbent one, it stops the process and returns the selected KC to be applied. If no improving KC is found, a random one is returned. The Best KC move on the other hand always returns the KC that generates the best reduction in the solution cost, among the generated KCs.

4.3.2 Move Worst Exam

The second available neighborhood structure ranks each exam based on the individual contribution to the overall cost. The top k exams with the worst contributions are then put in a squeaky wheel random selection scheme and one of them is selected. The move then tries to find the best legal slot to move the exam.

4.3.3 (Forced) Single Exam at Best Slot

The third available neighborhood structures randomly select an exam and then tries to find the best legal slot to move the exam. The forced version tries all possible slots, even the illegal ones, and calculates the benefit of moving the exam to the specific slot. Then, if the best slot is illegal as exams with conflicting students are already assigned to this slot, it moves the exam to the best slot and then tries to find a new feasible placement for all the conflicting exams.

4.3.4 Move Single / Exchange Exams

The fourth simple available neighborhood structures randomly selects an exam and moves it to the first feasible slot it finds. The exchange move randomly selects a non conflicting exam from the selected slot and moves this exam to the initial slot of the first exam.

4.3.5 Penalty Decreaser

The first of the complex moves either tries to optimize first very costly exams or badly placed exams. For each exam it finds the available legal slots that can be moved. It then calculates for all legal moves the benefit that each move will have to the incumbent solution. Based on the calculated benefit it select f out of all exam moves from the most beneficial to the least one. It then generates up to $k \times f$ combinations out of the selected ones and generate application sets of exam slot pairs that if applied will generate a legal assignment. Finally, it evaluates the generated sets and select to apply the best one. For our experiments, k had a value of 30 and f a value of 5. The move requires a lot of computation time, but is usually able to generate a much better solution compared to the incumbent one.

4.3.6 Ruin & Recreate

The second of the complex moves initially removes a set of exams, the ruin phase. One of the two available methods is randomly used to select exams. The first method selects a fixed percentage of the exams in the problem while the second uses a knapsack inspired method and removes exams that their individual contribution to the incumbent solution cost is used as the weight.

In the second phase, the recreate one, the removed exams are placed in slots sequentially, using a randomly selected method out of the three available . The first method, randomly select one unscheduled exam and places it to the best legal available slot, the second method to the first feasible slot it can find while the last one randomly select one the previous methods for each individual exam. The recreate phase placements may lead to an infeasible solution, as no back tracking is performed when placing unscheduled exams to slots. The move has a very strong effect, if successful, to help algorithms escape from local minimum.

4.3.7 Saturation Improver

The third of the complex moves initially calculates for all exams the available legal slots that each exam can be assigned and then sorts the exams from the most constrained to the least ones. It then tries to find for the most constrained ones if reassigning some of the exams that that have common students to other slots improves the number of the available slots that the currently selected exam can be moved. It is important to note that the reassignments are only allowed if no significant change in the cost of the incumbent solution is performed. For our experiments, we allowed up to 1% total reduction in the quality of the solution. The idea behind the move is to allow exams that due to the placement of other conflicting exams are unable to move to provide new possible slots, helping the algorithms escape from specific areas in the solution space. The move usually generate worse quality but different solutions compared to the incumbent one.

4.3.8 Cyclic Exchange

The fourth of the complex moves randomly selects a slot. It then performs the best Kempe Chain exchange with the next slot and this continues until the last available slot. When we reach the last slot, the exchange is performed between the last slot and the first slot. The exchange sequence is continued with the second slot until the initially selected slot is reached. The idea behind this move is that exams that are placed at the first or the last slots due to the cost structure have a benefit as their individual contribution to cost is calculated with only the exams that are on their right for the exams placed on the first slots and on their left for the exams placed on the last slots. With this move, we promote exams with high cost to move to the edges of the slot periods.

4.3.9 Time Relaxer

The last of the complex moves is inspired by the penalty trader move in [13]. We allow up to d slots for an exam to be moved forward, even if this slot is not available and we calculate the benefit in the cost that this move would have. All exams that have moves that would be beneficial are added to a candidate

list and the list is sorted in an ascending way. We then select the exam with the highest benefit and remove all other exams that are in the candidate list and have common students with the selected exam. The selected exam is added to a new list with non conflicting exams. The process continues until no more exams are available in the candidate list. As the non conflicting list contains exams that are compatible and can be scheduled on the same slot, the best slot for all these exams is selected. The idea behind the move is that it clusters compatible exams that individually may not have a big benefit in the cost but collectively may provide a better candidate solution.

5 Experimental results

Best results achieved using all the metaheuristic algorithms and all the available moves in the framework alongside some of the best results published for the same dataset are presented in Table 5. The results from our framework are in the column MAVN, while results from other heuristic method are from a tabu search method implemented by Di Gaspero and Schaerf[16], a graph coloring method proposed by Pillar and Allende[17], the sequential heuristic construction methods developed by Caramia et al.[13], the Ahuja-Orlin large neighbourhood search by Abdullah et al.[1], the Iterative Restart Variable Neighbourhood Search by Ayob et al.[6] and the Flex-Deluge algorithm developed by Burke and Bykov[10].

Table 5 The best results obtained by MAVN and other optimization methods applied to the Carter datasets

Dataset	MAVN	Di Gaspero & Schaerf	Allende et al.	Caramia et al.	Abdullah et al.	Ayob et al.	Burke & Bykov
CAR91	4.58	6.2	4.39	6.6	5.21	4.90	**4.32**
CAR92	3.80	5.2	3.71	6.0	4.36	4.51	**3.67**
EAR83	32.67	45.7	32.62	**29.3**	34.84	36.28	32.62
HEC92	10.03	12.4	10.05	**9.2**	10.28	11.06	10.06
KFU93	12.87	18.0	12.90	13.8	13.46	14.74	**12.80**
LSE91	10.02	15.5	9.82	**9.6**	10.24	12.08	9.78
PUR93	4.46	-	-	**3.7**	-	4.66	3.88
RYE93	8.08	-	-	**6.8**	8.74	10.67	7.91
STA83	**157.03**	160.8	**157.03**	158.2	159.28	157.32	**157.03**
TRE92	7.87	10.0	7.71	9.4	8.13	8.92	**7.64**
UTA92	3.18	4.2	3.04	3.5	3.63	3.58	**2.98**
UTE92	24.77	29.0	24.77	**24.4**	24.21	26.36	24.78
YOR83	35.11	41.0	34.70	36.2	36.11	38.97	**34.71**

6 Discussion and future work

In this paper, a framework to solve the UETP is presented. Using the framework, it is possible to experiment with the application of different algorithms and different neighborhoods very easily. In addition, the design of the framework allows the application of different evaluation methods that can use hardware accelerators [19]. Algorithmic parameters and the choice of moves that are active have a significant impact in the quality of the generated solutions. An initial experimentation for offline parameter selection can be found in [3]. In the future, we will investigate the use of online machine learning techniques to improve the sequence of the application of the available algorithms and the dynamic selection of neighborhoods.

References

1. Abdullah, S., Ahmadi, S., Burke, E.K., Dror, M.: Investigating Ahuja–Orlin's large neighbourhood search approach for examination timetabling. OR Spectrum **29**(2), 351–372 (2007)
2. Aldeeb, B.A., Al-Betar, M.A., Abdelmajeed, A.O., Younes, M.J., AlKenani, M., Alomoush, W., Alissa, K.A., Alqahtani, M.A.: A comprehensive review of uncapacitated university examination timetabling problem. International Journal of Applied Engineering Research **14**(24), 4524–4547 (2019)
3. Alefragis, P., Sofos, C.: Automated parameter selection of scheduling algorithms using machine learning techniques. In: Proceedings of the 21st Pan-Hellenic Conference on Informatics, pp. 1–5 (2017)
4. Alzaqebah, M., Abdullah, S.: Hybrid bee colony optimization for examination timetabling problems. Computers & Operations Research **54**, 142–154 (2015)
5. Applegate, D., Bixby, R., Chvatal, V., Cook, W.: Concorde TSP solver (2006). URL http://www.tsp.gatech.edu
6. Ayob, M., Burke, E.K., Kendall, G.: An iterative re-start variable neighbourhood search for the examination timetabling problem. Practice and Theory of Automated Timetabling. LNCS **1153**, 336–344 (2006)
7. Bellmore, M., Nemhauser, G.L.: The traveling salesman problem: a survey. Operations Research **16**(3), 538–558 (1968)
8. Burke, E.K., Bykov, Y.: Solving exam timetabling problems with the flex-deluge algorithm. In: Proceedings of PATAT, vol. 2006, pp. 370–372. Citeseer (2006)
9. Burke, E.K., Bykov, Y.: A late acceptance strategy in hill-climbing for exam timetabling problems. In: PATAT 2008 Conference, Montreal, Canada, pp. 1–7 (2008)
10. Burke, E.K., Bykov, Y.: An adaptive flex-deluge approach to university exam timetabling. INFORMS Journal on Computing **28**(4), 781–794 (2016)
11. Burke, E.K., Bykov, Y.: The late acceptance hill-climbing heuristic. European Journal of Operational Research **258**(1), 70–78 (2017)
12. Burke, E.K., Eckersley, A.J., McCollum, B., Petrovic, S., Qu, R.: Hybrid variable neighbourhood approaches to university exam timetabling. European Journal of Operational Research **206**(1), 46–53 (2010)
13. Caramia, M., Dell'Olmo, P., Italiano, G.F.: New algorithms for examination timetabling. In: International Workshop on Algorithm Engineering, pp. 230–241. Springer (2000)
14. Carter, M.W., Laporte, G., Lee, S.Y.: Examination timetabling: Algorithmic strategies and applications. Journal of the Operational Research Society **47**(3), 373–383 (1996)
15. Demeester, P., Bilgin, B., De Causmaecker, P., Berghe, G.V.: A hyperheuristic approach to examination timetabling problems: benchmarks and a new problem from practice. Journal of Scheduling **15**(1), 83–103 (2012)

16. Di Gaspero, L., Schaerf, A.: Tabu search techniques for examination timetabling. In: International Conference on the Practice and Theory of Automated Timetabling, pp. 104–117. Springer (2000)
17. Díaz, P.M., Allende, J.S.: Aproximación al problema de calendarios de exámenes mediante coloreado de grafos. Tecnología y desarrollo **15** (2017)
18. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. science **220**(4598), 671–680 (1983)
19. Kolonias, V., Goulas, G., Gogos, C., Alefragis, P., Housos, E.: Solving the examination timetabling problem in GPUs. Algorithms **7**(3), 295–327 (2014)
20. Leite, N., Fernandes, C.M., Melicio, F., Rosa, A.C.: A cellular memetic algorithm for the examination timetabling problem. Computers & Operations Research **94**, 118–138 (2018)
21. Mladenović, N., Hansen, P.: Variable neighborhood search. Computers & operations research **24**(11), 1097–1100 (1997)
22. 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)
23. Reinelt, G.: TSPLIB — A traveling salesman problem library. ORSA journal on computing **3**(4), 376–384 (1991)
24. Yang, Y., Petrovic, S.: A novel similarity measure for heuristic selection in examination timetabling. In: International Conference on the Practice and Theory of Automated Timetabling, pp. 247–269. Springer (2004)

Generalizing first-break-then-schedule to time-relaxed sports timetabling

David Van Bulck · Dries Goossens

Received: date / Accepted: date

Abstract A popular technique to construct time-constrained timetables is the so-called first-break-then-schedule approach which first determines for each team the time slots on which it plays home or away, after which the opponent of each time slot is determined. Whereas in time-constrained timetables the number of time-slots is just enough to play all games, time-relaxed timetables utilize more time slots than there are games per team. This offers time-relaxed timetables additional flexibility to take into account venue and player availability constraints. Despite their flexibility, time-relaxed timetables have the drawback that the rest period between teams' consecutive games can vary considerably, the difference in rest time between opponents may become unequal, and the difference in the number of games played at any point in the season can become large. In this paper, we explore how to generalize techniques based on first-break-then-schedule to generate time-relaxed timetables that are less prone to these drawbacks.

Keywords Time-relaxed sports timetabling · Availability constraints · Pattern decomposition · First-break-then-schedule · First-off-day-then-schedule · Game-off-day pattern set feasibility

1 Introduction

Every sports competition needs a timetable, also called schedule, which defines who will play whom. We assume that each game has two opponents, and that a game is played in the venue of the home team (the other team plays

D. Van Bulck
Ghent University, Faculty of Economics and Business Administration
E-mail: david.vanbulck@ugent.be

D. Goossens
Ghent University, Faculty of Economics and Business Administration
E-mail: dries.goossens@ugent.be

away). This paper studies the construction of a timetable for a so-called time-relaxed double round-robin tournament (for an overview on sports timetabling, see Van Bulck et al. (2020)). In a double round-robin tournament (2RR), each teams meets every other team once at home, and once away. A closely related variant of the 2RR is the single round-robin tournament (1RR) where each team meets every team exactly once. In contrast to time-constrained timetables, time-relaxed timetables utilize more time slots than minimally needed to schedule all the games. This allows time-relaxed timetables to take into account venue availability constraints that state when a team can play home, and team unavailability constraints that state when a team cannot play at all.

Their flexibility notwithstanding, time-relaxed timetables face three fairness issues that do not occur in time-constrained timetabling. First, the difference in the number of games played per team after each time slot causes tournament rankings to become inaccurate. The *games-played difference index* of a timetable is 'the minimum integer GPDI such that at any point in the schedule, the difference between the number of games played by any two teams is at most GPDI (Suksompong, 2016)'. Second, the rest time between teams' consecutive games can vary substantially, which can result in congested periods or long periods without any game. Since several researchers found a relation between fixture congestion and higher injury rates (e.g. Bengtsson et al. (2013), Dupont et al. (2010)), a popular constraint is to limit the maximal allowed sequence of games and off days. Third, the difference in rest time between two opponents may give an advantage to the most rested team. The *rest difference index* equals 'the minimum integer RDI such that for any game in the timetable, if one team has not played in i_1 consecutive games since its last game and the other team has not played in i_2 consecutive games since its last game, then $|i_1 - i_2| \leqslant$ RDI (Suksompong, 2016)'.

Observing that the difference in rest time is only relevant if the least rested team has not fully recovered from its previous game, this paper proposes to minimize the *sum of truncated rest differences* while at the same time controlling for absolute rest time and difference in games played. The truncated rest time of a team equals its absolute rest time r if $r < \tau$ and equals τ otherwise. For a more profound empirical motivation of this truncation, we refer to Scoppa (2015). Without truncation (i.e., $\tau = \infty$), the objective proposed in this paper is known in the literature as the total rest difference objective (see Çavdaroğlu and Atan (2020)) and as the rest mismatch objective if in addition the magnitude of the difference is ignored (see Atan and Çavdaroğlu (2018)).

Due to the large number and diversity of constraints and objectives that typically need to be considered, the construction of sports timetables is challenging. This difficulty has led to a wide variety of sports timetabling approaches. Many of these methods have in common that they decompose the problem into different subproblems, but differ in the order the subproblems are solved and the methods chosen for each subproblem (e.g., see Trick (2001)). One particular decomposition method is *first-break-then-schedule* which first

fixes for each time slot which teams play home and which play away, after which it determines the opponents in each time slot (see Section 3). First-break-then-schedule has been used to successfully schedule a wide variety of time-constrained competitions, including the Atlantic Coast Conference (Nemhauser and Trick, 1998), the Danish football league (Rasmussen, 2008), and the Belgian football league (Goossens and Spieksma, 2009). Despite its popularity, first-break-then-schedule has not been used so far to schedule time-relaxed competitions. This paper investigates whether first-break-then-schedule can be generalized so as to deal with the above described fairness issues in time-relaxed timetables.

2 Problem description and integer programming formulation

In the time-relaxed availability constrained double round-robin tournament described below, the input consists of a set of arbitrarily many time slots S, a set of teams T with $|T| = n$, for each team $i \in T$ a player availability set $A_i \subseteq S$ and venue availability set $H_i \subseteq A_i$, and four integers GPDI, ρ, σ, and τ. A feasible timetable for this problem assigns each game (i, j) of the double round-robin tournament, with home team $i \in T$ and away team $j \in T \setminus \{i\}$, to a time slot $s \in S$ such that each team plays at most once per time slot and:

- *(C1)* the player availability A_i is respected for all teams (i.e., no game (i, j) or (j, i) is planned on a time slot $s \notin A_i$),
- *(C2)* the venue availability H_i is respected for the home teams (i.e., no game (i, j) is planned on a time slot $s \notin H_i$),
- *(C3)* the games played difference index is at most GPDI,
- *(C4)* each team plays at most two games per ρ consecutive time slots, and
- *(C5)* each team has at most σ consecutive off days.

In addition, the objective is to minimize the sum of truncated rest differences, thereby assuming that teams are fully recovered from their previous game after τ time slots and that they are fully rested at the start of the season.

Availability constraints *(C1)* and *(C2)* have applications in a multitude of time-relaxed tournaments. For example, non-professional teams typically share their venues with other teams and their players need to be able to combine their sport with work and family (e.g., Schönberger et al. (2004), Van Bulck et al. (2019)). A timetable with a low maximal difference in games played *(C3)* is desirable since this results in more accurate tournament rankings and may reduce the opportunities for match fixing. In a 2RR, the games-played difference index may be as high as $2(n-2)$: one team has played twice against every other team except for one team that did not play any game yet. Constraints *(C4)* and *(C5)* respectively limit the maximal and minimal number of games in a series of consecutive time slots so as to avoid fixture congestion and long periods without any game. Since teams might blame the timetable for losing the game if they have less rest than their opponent, the objective is to minimize the sum of truncated rest differences.

Without the objective function and Constraints *(C3)* to *(C5)* (i.e. GPDI = $2(n-2)$, $\rho = 2$, and $\sigma = |S| - 2(n-1)$), the problem is known in the literature as 'RAC-2RR'. From a theoretical point of view, Van Bulck and Goossens (2020a) show that RAC-2RR is $\mathcal{NP}$-complete.

Equations (1)-(17) present an integer programming (IP) formulation for the problem just described. Our main decision variable is $x_{i,j,s}$, which is 1 if team $i \in T$ plays a home game against team $j \in T \setminus \{i\}$ in time slot $s \in S$, and 0 otherwise. Variable $q_{i,s}$ represents the number of games played by team i up to and including time slot $s \in S$, and variable $y_{i,s,t}$ is 1 if team i plays a game in time slot s, followed by its next game in time slot t, for each $s, t \in S$ such that $s < t \leqslant \tau + s$, and 0 otherwise. Finally, variable $d_{i,j}$ contains the truncated rest difference of game (i, j).

$$\textbf{minimize} \sum_{i,j \in T: i \neq j} d_{i,j} \tag{1}$$

subject to

$$\sum_{s \in H_i \cap A_j} x_{i,j,s} = 1 \qquad \forall i, j \in T : i \neq j \tag{2}$$

$$\sum_{j \in T \setminus \{i\}} (x_{i,j,s} + x_{j,i,s}) \leqslant 1 \qquad \forall i \in T, \forall\, s \in S \tag{3}$$

$$q_{i,1} = \sum_{j \in T \setminus \{i\}} (x_{i,j,1} + x_{j,i,1}) \qquad \forall i \in T \tag{4}$$

$$q_{i,s} = q_{i,s-1} + \sum_{j \in T \setminus \{i\}} (x_{i,j,s} + x_{j,i,s}) \qquad \forall i \in T, s \in S \setminus \{1\} \tag{5}$$

$$q_{i,s} - q_{j,s} \leqslant \text{GPDI} \qquad \forall i, j \in T : i \neq j, \forall s \in S \tag{6}$$

$$\sum_{j \in T \setminus \{i\}} \sum_{p=s}^{s+\rho-1} (x_{i,j,p} + x_{j,i,p}) \leqslant 2 \qquad \forall i \in T, \forall\, s \in S : s + \rho - 1 \leq |S| \tag{7}$$

$$\sum_{j \in T \setminus \{i\}} \sum_{p=s}^{s+\sigma} (x_{i,j,p} + x_{j,i,p}) \geqslant 1 \qquad \forall i \in T, \forall\, s \in S : s + \sigma \leq |S| \tag{8}$$

$$\sum_{j \in T \setminus \{i\}} \Big(x_{i,j,s} + x_{j,i,s} + x_{i,j,t} + x_{j,i,t} - \sum_{k=s+1}^{t-1} (x_{i,j,k} - x_{j,i,k}) \Big) - 1 \leqslant y_{i,s,t} \qquad \forall i \in T, s, t \in S : s < t \leqslant \tau + s \tag{9}$$

$$y_{i,s,t} \leqslant \sum_{j \in T \setminus \{i\}} (x_{i,j,s} + x_{j,i,s}) \qquad \forall i \in T, s, t \in S : s < t \leqslant \tau + s \tag{10}$$

$$y_{i,s,t} \leqslant \sum_{j \in T \setminus \{i\}} (x_{i,j,t} + x_{j,i,t}) \qquad \forall i \in T, s, t \in S : s < t \leqslant \tau + s \tag{11}$$

$$|t - u|(y_{i,t,s} + y_{j,u,s} + x_{i,j,s} - 2) \leqslant d_{i,j} \qquad \forall i, j \in T : i \neq j, s, t, u \in S : s - \tau \leqslant t, u < s \tag{12}$$

$$(\tau - (s - t - 1)) \Big(x_{i,j,s} + y_{j,t,s} - 1 - \sum_{\substack{u \in S: \\ s - \tau \leqslant u < s}} \sum_{\substack{k \in T: \\ k \neq i}} (x_{i,k,u} + x_{k,i,u}) \Big) \leqslant d_{i,j} \qquad \forall i, j \in T : i \neq j, \forall s, t \in S : s - \tau \leqslant t < s \tag{13}$$

$$(\tau-(s-t-1))\Big(x_{i,j,s}+y_{i,t,s}-1- \sum_{\substack{u\in S:\\ s-\tau\leqslant u<s}}\sum_{\substack{k\in T:\\ k\neq j}}(x_{j,k,u}+x_{k,j,u})\Big)\leqslant d_{i,j} \quad \forall i,j\in T: i\neq j, \forall s,t\in S: s-\tau\leqslant t<s \quad (14)$$

$$x_{i,j,s}=0 \qquad \forall i,j\in T: i\neq j, s\notin H_i\cap A_j \quad (15)$$

$$x_{i,j,s}\in\{0,1\}, d_{i,j}\geqslant 0 \qquad \forall i,j\in T: i\neq j, s\in H_i\cap A_j \quad (16)$$

$$0\leqslant y_{i,s,t}\leqslant 1 \qquad \forall i\in T, s,t\in S: s<t\leqslant\tau+s \quad (17)$$

The objective function (1) minimizes the sum of truncated rest differences. The first set of constraints ensures that each team plays the required number of home games against every other team, while respecting the player and venue availability constraints *(C1)* and *(C2)*. The next set of constraints enforces that a team plays at most one game per time slot. Constraints (4) and (5) recursively model the number of games a team played up to and including time slot $s\in S$, and constraints (6) limit the maximal difference in games played *(C3)*. The next two sets of constraints respectively model *(C4)* and *(C5)*. Constraints (9) to (11) regulate the value of the $y_{i,s,t}$ variables by considering the number of time slots between two consecutive games of the same team. Constraints (12) to (14) respectively model the difference in rest time when neither of the teams is fully rested, i is fully rested but j is not, and j is fully rested but i is not. Constraints (15) reduce the number of variables in the system; when implementing this formulation, these variables need not be created. Constraints (16) are the binary constraints on the x-variables and the non-negativity constraints for the d variables. Note that the integrality of $q_{i,s}$ follows from (4), (5), and (16), and that the integrality of $d_{i,j}$ and $y_{i,s,t}$ follows from the objective function and constraints (9), (10), (11), (16), and (17).

3 Pattern-based decomposition methods

A time-relaxed sports timetable can be seen as a combination of a game-off-day pattern set, a home-away pattern set, and an opponent schedule. The game-off-day pattern (GOP) of team i is a function $g_i: S\to\{G,O\}$ such that $g_i(s)=G$ if i plays a game and $g_i(s)=O$ if i has an off day (also called bye) on time slot s (see also Bao (2009)). The home-away pattern (HAP) of team i is a function $h_i: S\to\{H,A,O\}$ such that $h_i(s)=H$ if i plays a home game, $h_i(s)=A$ if i plays an away game, and $h_i(s)=O$ if i has an off day on time slot s. An assignment of one GOP to each team is known as a GOP set, and an assignment of one HAP to each team as an HAP set. The opponent schedule determines which opponent each team faces for each of the time slots. Note that the GOPs are fully defined once the HAP set or the assignment of opponents is known. Furthermore, we observe that every two patterns in an HAP set must be different, since otherwise two teams can never play against each other, while this need not be the case in a GOP set. Clearly, the assignment of opponents must be compatible with the GOP and HAP set before they can merge into a timetable: for each pair of opponents, the

corresponding home-away patterns need to give one team the home advantage, and designate an away game for the other team. Moreover, if a team does not play against any opponent, it must have an off day in its game-off day pattern. We call a GOP and HAP set feasible if there exists a compatible timetable (see also Van Bulck and Goossens (2020b)).

Perhaps the most popular decomposition method in sports timetabling is the so-called first-break-then-schedule approach that breaks down timetabling by first enumerating all possible HAPs, then constructing an HAP set and checking whether it is feasible, and finally determining the opponent schedule (see Nemhauser and Trick (1998)). Despite the popularity of first-break-then-schedule, there are two issues that make it less straightforward to construct time-relaxed timetables using this approach: the combinatorial explosion of possible HAPs, and the feasibility of HAP sets.

Since in a double round-robin each team plays $(n-1)$ home games and $(n-1)$ away games, the total number of HAPs is given by $\binom{|S|}{n-1}\binom{|S|-(n-1)}{n-1}$. For 8 teams there are therefore 3.432 time-constrained HAPs, while there are 923.780 HAPs with one off day and 181.416.306.202.560 HAPs when the number of time slots is twice more than minimally needed. Hence, where the total number of time-constrained patterns is already substantial, the total number of time-relaxed HAPs quickly becomes intractable.

The efficiency of the first-break-then-schedule method heavily depends on its ability to avoid infeasible HAP sets early on (e.g., Miyashiro et al. (2003)). While it is conjectured that feasibility of a 1RR time-constrained HAP set can be verified in polynomial time (see Briskorn (2008)), it is known that verifying feasibility of a 1RR time-relaxed HAP set is $\mathcal{NP}$-complete (see Van Bulck and Goossens (2020b)).

Motivated by the following three observations, an attractive alternative to the first-break-then-schedule method seems to construct the game-off-day patterns first.

Observation 1 *Once the GOP of a team is known, it is known whether its compatible timetables respect Constraints (C1).*

Observation 2 *Once the GOP of a team is known, it is known whether its compatible timetables respect Constraints (C4) and (C5).*

Observation 3 *Once the GOP set is known, it is known whether its compatible timetables respect Constraint (C3).*

We refer to the method of first determining the GOP set and then the timetable as the *first-off-day-then-schedule* method. When constructing time-relaxed timetables, first-off-day-then-schedule seems to have several advantages over first-break-then-schedule.

First, since each team plays $2(n-1)$ games in a double round-robin tournament, there are 'only' $\binom{|S|}{2(n-1)}$ GOPs. Unfortunately, this is still a considerably large number making it impractical to enumerate all GOPs when there are many more time slots than games per team. However, the construction

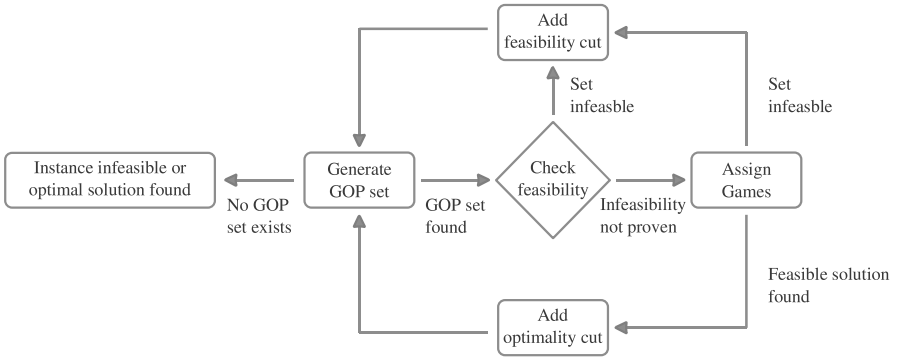


Fig. 1 General structure of the first-off-day-then-schedule algorithm. The generation of GOP sets is discussed in Section 4.1, the feasibility of GOP sets in Section 4.2, and the assignment of games in Section 4.3.

of a GOP set may be simple enough so that there is no need to explicitly enumerate all patterns first.

Second, GOP sets provide more flexibility to schedule games later on since they do not impose restrictions on the home advantage of games. Nevertheless, determining whether a given GOP set is feasible remains an $\mathcal{NP}$-complete problem, unless there is at most one off day per team (see Van Bulck and Goossens (2020b)).

While the conceptual idea of first-off-day-then-schedule was to some extent already outlined by Bao (2009), Bao does not provide any computational experiments with regard to the performance of the first-off-day-then-schedule method. Moreover, Bao does not provide any information on how to backtrack when one of the subproblems turns out to be infeasible.

4 Implementing first-off-day-then-schedule

In order to implement a first-off-day-then-schedule approach for the problem outlined in Section 2, we implement a GOP set generating decomposition method (see Figure 1). Our approach decomposes the problem into the following three components: generate a GOP set (Section 4.1), check feasibility of the GOP set (Section 4.2), and find a timetable compatible with the GOP set (Section 4.3). If one of the subproblems turns out to be infeasible, we implement backtracking. If the master problem is infeasible, then the last solution found is optimal, or in case no solution was found the problem instance is infeasible.

4.1 Generating GOP sets

In order to generate GOP sets with constraint programming (CP), we formulate the GOP set generation model (18)-(27). Our main decision variable is

$g_{i,s}$ which is 1 if team $i \in T$ has a 'G' on time slot $s \in S$, and 0 otherwise. The global constraint sequence$(nbMin, nbMax, width, vars, values, card)$ takes six arguments: $values$ and $card$ are arrays of integers and must have the same index set I, $vars$ is an array of decision variables, and the remaining arguments are integers. For each $i \in I$ the constraint requires that $card[i]$ elements of $vars$ take value $value[i]$, and that any subsequence of size $width$ must contain at least $nbMin$ and at most $nbMax$ values from $values$ (see CP Optimizer (2014)).

$$\text{sequence}\big(0, 2, \rho, (g_{i,1}, \ldots, g_{i,|S|}), (1), (2n-2)\big)) \qquad \forall i \in T \quad (18)$$

$$\text{sequence}\big(0, \sigma, \sigma+1, (g_{i,1}, \ldots, g_{i,|S|}), (0), (|S|-2n+2)\big)) \qquad \forall i \in T \quad (19)$$

$$\sum_{s \in H_i} g_{i,s} \geqslant n-1 \qquad \forall i \in T \quad (20)$$

$$\sum_{s \in H_i \cap A_j} (g_{i,s} = 1 \wedge g_{j,s} = 1) \geqslant 1 \qquad \forall i, j \in T : i \neq j \quad (21)$$

$$\sum_{\substack{s \in (H_i \cap A_j) \\ \cup (H_j \cap A_i)}} (g_{i,s} = 1 \wedge g_{j,s} = 1) \geqslant 2 \qquad \forall i, j \in T : i < j \quad (22)$$

$$\Big|\sum_{p=1}^{s} g_{i,p} - \sum_{p=1}^{s} g_{j,p}\Big| \leqslant \text{GPDI} \qquad \forall i, j \in T : i < j, \forall s \in S \quad (23)$$

$$\big(\sum_{i \in T} g_{i,s} \bmod 2\big) = 0 \qquad \forall s \in S \quad (24)$$

$$\sum_{i \in T : s \in H_i} g_{i,s} \geqslant \sum_{i \in T : s \in A_i \setminus H_i} g_{i,s} \qquad \forall s \in S \quad (25)$$

$$g_{i,s} = 0 \qquad \forall i \in T, s \in S \setminus A_i \quad (26)$$

$$g_{i,s} \in \{0, 1\} \qquad \forall i \in T, s \in S \quad (27)$$

Constraints (18) model that each GOP contains $2(n-1)$ 'G's, and that no GOP contains more than 2 games in any sequence of ρ time slots *(C4)*. Constraints (19) model that each GOP contains $|S| - 2(n-1)$ off days, and that no GOP contains more than σ consecutive off days *(C5)*. Since each team has to play $(n-1)$ home games, the next set of constraints requires that each GOP contains $(n-1)$ 'G's during time slots on which the venue of the team is available. Constraints (21) enforce that each game (i, j) can be scheduled, and Constraints (22) enforce that game (i, j) and (j, i) can be scheduled simultaneously (at least when ignoring all other games). Constraints (23) limit the maximal difference in games played *(C3)*. The next set of constraints enforces that the sum of 'G's is even on each time slot, a necessary condition since each game involves two teams. Its simplicity notwithstanding, this necessary condition is sufficient for the feasibility of a 1RR GOP set if each team has at most one off day (see Van Bulck and Goossens (2020b)). Since each game involves one home team and one away team, Constraints (25) enforce that the total number of teams that can play home and have a 'G' on time slot $s \in S$ must be larger than or equal to the number of teams that have a 'G' and

can only play away on s. Finally, Constraints (26) enforce that a team has an off day when its players are unavailable, and Constraints (27) are the binary constraints.

In order to increase the probability that there exists a compatible timetable for the generated GOP set, we may replace the right-hand side of Constraints (21) by $1+\epsilon$ where ϵ is a positive integer parameter which is reduced by one if no feasible solution for formulation (18)-(27) is found. Note that Constraints (22) become redundant if $\epsilon \geq 1$.

4.2 Feasibility checks

A GOP set is feasible if all games can be assigned to time slots on which the opposing teams have a 'G' in their pattern and the venue of the home team is available. In case a GOP set turns out to be infeasible, we add a constraint that prevents from finding the infeasible GOP set again. We thereby try to reduce the total number of solutions that need to be enumerated in the master problem by cutting-off as many infeasible or sub-optimal solutions as possible (see also Rasmussen and Trick (2007)).

4.2.1 Game possibilities

Denote with $c_G(T', s)$ the number of 'G's in the GOPs of a subset of teams $T' \subseteq T$, $|T'| = m$, on time slot $s \in S$. The number of games between teams in T' on s is at most $\lfloor \frac{c_G(T',s)}{2} \rfloor$. Hence, Condition 1 is a necessary condition that requires to check $\mathcal{O}(2^n)$ constraints.

Condition 1 (Bao (2009)) $\sum_{s \in S} \left\lfloor \frac{c_G(T',s)}{2} \right\rfloor \geqslant m(m-1)$ *for each* $T' \subseteq T$.

Instead of explicitly checking Condition 1 for each subset, we formulate an IP model to find a minimal subset of teams for which Condition 1 is violated, or which proves that no such subset exists. The formulation of this IP is based on Rasmussen and Trick (2007), with the main difference that it checks the feasibility of a GOP set instead of an HAP set and that it requires to be solved only once instead of once for each cardinality of T'. Parameter $\mathrm{UB}_{\mathrm{GP}}$ gives an upper bound on the cardinality of the subsets to be checked, and parameters $g'_{i,s}$ define the GOP set found by model (18)-(27). The main purpose of parameter $\mathrm{UB}_{\mathrm{GP}}$ is to control for the expected computation time to solve the IP formulation. Variable z_c, $2 \leqslant c \leqslant \mathrm{UB}_{\mathrm{GP}}$, determines whether the cardinality of the subset is c ($z_c = 1$) or not ($z_c = 0$), α_i determines whether team i is in the subset ($\alpha_i = 1$) or not ($\alpha_i = 0$), and variable β_s, $s \in S$, calculates an upper bound on the total number of games the teams in the subset can play.

$$\textbf{minimize} \quad \sum_{i \in T} \alpha_i \tag{28}$$

$$\sum_{i \in T} \alpha_i = \sum_{c=2}^{\mathrm{UB_{GP}}} c z_c \tag{29}$$

$$\sum_{c=2}^{\mathrm{UB_{GP}}} z_c = 1 \tag{30}$$

$$\beta_s \geqslant \big(\sum_{i \in T} g'_{i,s} \alpha_i - 1\big)/2 \qquad \forall s \in S \tag{31}$$

$$\sum_{s \in S} \beta_s \leqslant c(c-1) - 1 + \mathrm{UB_{GP}}(\mathrm{UB_{GP}} - 1)(1 - z_c) \qquad \forall c \in \mathbb{N} : 2 \leqslant c \leqslant \mathrm{UB_{GP}} \tag{32}$$

$$\alpha_i, z_c \in \{0,1\}, \beta_s \in \mathbb{N} \qquad \forall i \in T, s \in S, c \in \mathbb{N} : 2 \leqslant c \leqslant \mathrm{UB_{GP}} \tag{33}$$

The objective function minimizes the total number of teams chosen such that infeasibility of the GOP set can be traced back to as few patterns as possible, and Constraints (29)-(30) model the value of z_c. Constraints (31) calculate the upper bound on the total number of games between teams in the subset on time slot s, and Constraints (32) ensure that Condition 1 is violated. Finally, Constraints (33) are the binary and integrality constraints.

If a violating set of teams T' defined by the α_i's is found, the following constraint is added to the GOP set generation model.

$$\mathrm{forbiddenAssignment}\big((g_{i,1}, \ldots, g_{i,|S|}\, \forall i \in T'), (g'_{i,1}, \ldots, g'_{i,|S|}\, \forall i \in T')\big) \tag{34}$$

The global constraint forbiddenAssignment($vars, values$) takes two arguments: an array of decision variables $vars$ and an ordered set $values$ that both have index set I. The constraint enforces there is at least one $i \in I$ such that $vars[i]$ is not equal to $values[i]$. Intuitively this constraint forbids any GOP set in which the teams in T' play according to the GOPs currently assigned. Clearly, the smaller is $|T'|$, the stronger is the reduction in the search space.

4.2.2 Isolated slots

Define with $S_{T'} \subseteq S$ the subset of time slots on which at least two teams in $T' \subseteq T$, $|T'| = m$, have a game and all teams not in T' have an off day, i.e. $\sum_{i \in T'} g'_{i,s} \geqslant 2$ and $\sum_{i \notin T'} g'_{i,s} = 0$ for all $s \in S_{T'}$. We refer to the subset of time slots $S_{T'}$ as isolated slots for the subset of teams T'. Note that it follows from the definition that S_T contains all time slots on which at least two teams have a 'G' in their pattern.

Condition 2 *For each subset of teams $T' \subseteq T$, the sum of G's during isolated slots is smaller than twice the total number of mutual games in T', i.e. $\sum_{i \in T'} \sum_{s \in S_{T'}} g'_{i,s} \leqslant 2m(m-1)$.*

Condition 2 is a necessary condition since teams in T' can only play against other teams in T' during isolated slots, and the total number of games between teams in T' is limited by $m(m-1)$.

Instead of explicitly checking each subset, we formulate an IP model to find a minimal subset of teams for which Condition 2 is violated, or which proves that no such subset exists. In this formulation, parameter $\mathrm{UB}_{\mathrm{IS}}$ denotes the maximal subset to be checked. Furthermore, let γ_s, $s \in S_T$, denote the number of 'G's if s is an isolated slot for the subset of teams defined by α_i.

$$\textbf{minimize} \quad \sum_{i \in T} \alpha_i \tag{35}$$

$$\sum_{i \in T} \alpha_i = \sum_{c=2}^{\mathrm{UB}_{\mathrm{IS}}} c z_c \tag{36}$$

$$\sum_{c=2}^{\mathrm{UB}_{\mathrm{IS}}} z_c = 1 \tag{37}$$

$$\gamma_s \leqslant \alpha_i \sum_{j \in T} g'_{j,s} \qquad \forall s \in S_T, i \in T : g'_{i,s} = 1 \tag{38}$$

$$\sum_{s \in S_T} \gamma_s \geqslant 2c(c-1) + 1 - \big(\mathrm{UB}_{\mathrm{IS}}(\mathrm{UB}_{\mathrm{IS}} - 1) + 1\big)(1 - z_c) \qquad \forall c \in \mathbb{N} : 2 \leqslant c \leqslant \mathrm{UB}_{\mathrm{IS}} \tag{39}$$

$$\alpha_i, z_c \in \{0,1\}, \gamma_s \geq 0 \qquad \forall i \in T, s \in S_T, c \in \mathbb{N} : 2 \leqslant c \leqslant \mathrm{UB}_{\mathrm{IS}} \tag{40}$$

The objective function minimizes the total number of teams in the subset, while Constraints (36) and (37) model the z_c variables. If all teams that have a 'G' on time slot $s \in S_T$ are in the subset, s is an isolated slot and Constraints (38) counts the total number of 'G's. Constraints (39) ensure that Condition 2 is violated, and Constraints (40) are the binary and non-negativity constraints.

If a violating set of teams T' defined by the α_i's is found, the following constraint is added to model (18)-(27).

$$\mathrm{forbiddenAssignment}\big((g_{1,s}, \ldots, g_{|T|,s}\, \forall s \in S'), (g'_{1,s}, \ldots, g'_{|T|,s}\, \forall s \in S')\big) \tag{41}$$

Constraint (41) forbids any GOP set in which the teams have a 'G' according to the current columns in S'.

4.3 Assign games

Given a GOP set, a third and final step is to construct a compatible timetable. This section first provides two more feasibility checks based on the construction of a compatible timetable, and then shows how to construct a compatible timetable that minimizes the sum of truncated rest differences.

Consider first the following condition for the feasibility of a GOP set.

Condition 3 *For each subset of time slots $S' \subseteq S$, an assignment of games to time slots in S' exists such that for each $s \in S'$ team $i \in T$ plays exactly one*

game (i,j) *or* (j,i) *if* $g'_{i,s}=1$ *and* $s \in H_i$*, exactly one game* (i,j) *if* $g'_{i,s}=1$ *and* $s \in A_i \setminus H_i$*, and no game if* $g'_{i,s}=0$*.*

By the definition of a feasible GOP set, Condition 3 is a necessary condition for all $S' \subseteq S$ and a sufficient condition if $S'=S$. In essence, Condition 3 checks feasibility for a subset of columns in the GOP set. We use the linear relaxation of formulation (42)-(45) to check Condition 4 for each S' with cardinality UB_{AGC} or lower, and add a constraint of type (41) to the GOP set generation model if a violating subset of time slots S' is found. The main motivation for the use of the linear relaxation is the expected decrease in computation time, while (hopefully) still detecting violations of Condition 3 reasonably well.

$$\sum_{s \in S' \cap H_i} x_{i,j,s} \leq 1 \qquad \forall i,j \in T : i \neq j \quad (42)$$

$$\sum_{j \in T \setminus \{i\}} (x_{i,j,s} + x_{j,i,s}) = g'_{i,s} \qquad \forall i \in T, \forall\, s \in S' \quad (43)$$

$$x_{i,j,s} = 0 \qquad \forall i \in T, \forall s \in S' : s \notin H_i \vee g'_{i,s} = 0 \vee g'_{j,s} = 0 \quad (44)$$

$$x_{i,j,s} \in \{0,1\} \qquad \forall i,j \in T : i \neq j, \forall\, s \in S' \quad (45)$$

The first set of constraints restricts each game (i,j) to be scheduled at most once, and the second set of constraints enforces the GOP set for the given subset of time slots. Finally, Constraints (44) reduce the number of variables in the system, and Constraints (45) are the binary constraints.

Instead of checking the columns of a GOP set, we may also check feasibility for a subset of rows in the GOP set (see also Rasmussen and Trick (2007)).

Condition 4 *For each subset of teams* $T' \subseteq T$*, an assignment of the mutual games between teams in* T' *to time slots in* S *exists in which the opposing teams have a 'G' in their pattern and the venue of the home team is available.*

By the definition of a feasible GOP set, Condition 4 is a necessary condition for all $T' \subseteq T$ and a sufficient condition if $T'=T$. We use the linear relaxation of formulation (46)-(50) to check Condition 4 for each subset of teams with cardinality UB_{AGR} or lower, and add a constraint of type (34) to the GOP set generation model if a violating subset of teams T' is found. Observing that the rest time of teams is known once the GOP set is known, parameter $r_{i,s}$ gives the truncated rest time of team $i \in T'$ in time slot $s \in S$. Parameter obj*, initially equal to infinity, gives the objective value of the current best found solution.

$$\sum_{s \in H_i} x_{i,j,s} = 1 \qquad \forall i, j \in T' : i \neq j \quad (46)$$

$$\sum_{j \in T' \setminus \{i\}} (x_{i,j,s} + x_{j,i,s}) \leqslant 1 \qquad \forall i \in T', \forall\, s \in S : g'_{i,s} = 1 \quad (47)$$

$$\sum_{\substack{i,j \in T': \\ i \neq j}} \sum_{s \in S} |r_{i,s} - r_{j,s}| x_{i,j,s} \leqslant \text{obj}^* - 1 \quad (48)$$

$$x_{i,j,s} = 0 \qquad \forall i \in T', \forall s \in S : s \notin H_i \vee g'_{i,s} = 0 \vee g'_{j,s} = 0 \quad (49)$$

$$x_{i,j,s} \in \{0, 1\} \qquad \forall i, j \in T' : i \neq j, \forall\, s \in S \quad (50)$$

The first set of constraints enforces that each team in T' plays exactly once at home against every other team in T', and the second set of constraints enforces that each team in T' plays at most one game per time slot. Constraint (48) states that the sum of truncated rest differences must be better than the current best found solution. Finally, Constraints (49) reduce the number of variables in the system, and Constraints (50) are the integrality constraints.

In case no subset of time slots or teams is found that respectively violates Condition 3 or Condition 4, we enable all integrality constraints and we solve formulation (46)-(50) for $T' = T$. In addition, we minimize the following objective.

$$\textbf{minimize} \quad \sum_{\substack{i,j \in T': \\ i \neq j}} \sum_{s \in S} |r_{i,s} - r_{j,s}| x_{i,j,s} \quad (51)$$

If a solution exists, we have found a strictly better solution and we update obj* . Moreover, regardless of the feasibility, we add a constraint of type (34) to the GOP set generation model to prevent that the same solution is found again.

5 Computational Results

This section experimentally evaluates IP formulation (1)-(17) and the first-off-day-then-schedule approach respectively proposed in Section 2 and Section 4.

Our benchmark of problem instances consists of 9 artificial double round-robin problem instances. A problem instance in this set is of type (n, o, h, a) if it contains n teams and $2(n-1) + o$ time slots (i.e., each team has o off days), and for each team $i \in T$ it holds that $|H_i| = h$ and $|S \setminus A_i| = a$ (see also Van Bulck and Goossens (2020a)). We consider n in the set $\{8, 12, 16\}$, o in $\{(n-1), 2(n-1), 3(n-1)\}$, set $h = o/2$ and $a = o/4$, and assume that a team is fully rested after five time slots (i.e. $\tau = 5$). Furthermore, we require that the games-played difference index is not larger than 2 (i.e. GPDI $= 2$), and that a team has at most six consecutive off days (i.e. $\sigma = 6$). Finally, if the total number of time slots is at least twice the minimal number needed we

				Algorithms	
Teams n	o	h	a	IP	FOTS
8	21	3	1	8	16
	28	7	3	8	29
	35	10	5	38	42
12	33	5	2	71	45
	44	11	5	133	98
	55	16	8	205	143
16	45	7	3	185	76
	60	15	7	/	155
	75	22	11	320	267

Table 1 Results for the artificial problem instances. The first four columns respectively refer to the number of teams (n), the number of off days per team (o), the average venue availability (h), and the average player unavailability (a). Algorithm 'IP' refers to solving integer programming formulation (1)-(17), and algorithm 'FOTS' refers to the first-off-day-then-schedule approach outlined in Section 4. Finally, '/' means that no solution was found within the given computation time.

require that no team plays more than 2 games within 3 consecutive time slots (i.e. $\rho = 3$) and set $\rho = 2$ otherwise.

In order to define the parameters of the first-off-day-then-schedule algorithm, we require that each game can be scheduled during at least four time slots (i.e. $\epsilon = 3$), check feasibility of Condition 1 and Condition 2 for up to 6 teams (i.e. $\text{UB}_{\text{GP}} = 6$ and $\text{UB}_{\text{IS}} = 6$), and check feasibility of Condition 3 and Condition 4 for up to 4 time slots and 4 teams (i.e. $\text{UB}_{\text{AGC}} = 4$ and $\text{UB}_{\text{AGR}} = 4$).

All IP formulations are solved with ILOG CPLEX version 12.10, and the CP formulation is solved with ILOG CPLEX CP OPTIMIZER 12.10. The IP formulation (1)-(17) was granted 3600 seconds of computation time, whereas the first-off-day-then-schedule approach was granted only 600 seconds of computation time. All models were run on a CentOS 7.4 GNU/Linux based system with an Intel E5-2680 processor, running at 2.5 GHz and provided with 16 GB of RAM and 8 cores.

Table 1 presents preliminary results for the best found solution by each algorithm. The first four columns provide the total number of teams (n), the number of off days per team (o), the average team availability (h), and the average player unavailability (a). The next column gives the best found solution within the given computation time using IP formulation (1)-(17); none of the instances was solved to optimality and the best lower bound found was equal to 0 for all problem instances. The final column shows the best found solution by our first-break-then-schedule algorithm. Despite being given 6 times less computation time, the first-off-day-then-schedule finds solutions that are only slightly worse when there are 8 teams in the competition, and finds even better solutions for all problem instances that have more than 8 teams.

6 Conclusion

Decomposition methods are common in sports timetabling and break down the timetabling process into different subproblems. Perhaps the most popular decomposition method is first-break-then-schedule where the first subproblem is to determine the home-away pattern (HAP) set which defines when teams must play at home or play away. Observing that existing decomposition methods focus exclusively on time-constrained timetables where the number of time slots is the minimally needed, this paper investigates how to generalize first-break-then-schedule so as to construct time-relaxed timetables where there are more time slots than games per team. In particular, this paper proposes a first-off-day-then-schedule method that first determines the game-off-day pattern (GOP) set defining when teams play (home or away) or have an off day, after which it constructs a compatible timetable.

The main advantage of our method is that the construction of GOP sets turns out to be simple enough so that there is no need to explicitly enumerate all patterns first. This avoids a combinatorial explosion by only implicitly enumerating all patterns, and allows to check feasibility of GOP sets both on the level of rows (representing patterns of teams) and columns (representing the teams that play on a particular time slot). Nevertheless, our approach still generates many infeasible GOP sets and would therefore profit from further research on the level of GOP set feasibility.

We use our approach to generate a number of time-relaxed double round-robin timetables where availability constraints and fairness issues play a prominent role. Indeed, while the structure of a round-robin tournament already adds a substantial level of fairness to any timetable designed for it, there are many other fairness issues that need to be considered. In this paper, we minimize the sum of rest differences while controlling for the rest period between teams' consecutive games and the maximal difference in games played.

References

Atan T, Çavdaroğlu B (2018) Minimization of rest mismatches in round robin tournaments. Comput Oper Res 99:78 – 89

Bao R (2009) Time relaxed round robin tournament and the NBA scheduling problem. PhD thesis, Cleveland State University

Bengtsson H, Ekstrand J, Hägglund M (2013) Muscle injury rates in professional football increase with fixture congestion: an 11-year follow-up of the UEFA Champions League injury study. Br J Sports Med 47:743–747

Briskorn D (2008) Feasibility of home-away-pattern sets for round robin tournaments. Oper Res Lett 36:283 – 284

Çavdaroğlu B, Atan T (2020) Determining matchdays in sports league schedules to minimize rest differences. Oper Res Lett 48:209 – 216

CP Optimizer II (2014) 12.6, IBM ILOG CPLEX Optimization Studio CP Optimizer user's manual, 2014

Dupont G, Nedelec M, McCall A, McCormack D, Berthoin S, Wisløff U (2010) Effect of 2 soccer matches in a week on physical performance and injury rate. Am J Sport Med 38:1752–1758

Goossens D, Spieksma F (2009) Scheduling the Belgian soccer league. Interfaces 39:109–118

Miyashiro R, Iwasaki H, Matsui T (2003) Characterizing feasible pattern sets with a minimum number of breaks. In: Burke E, De Causmaecker P (eds) Practice and Theory of Automated Timetabling IV, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 78–99

Nemhauser GL, Trick MA (1998) Scheduling a major college basketball conference. Oper Res 46:1–8

Rasmussen RV (2008) Scheduling a triple round robin tournament for the best Danish soccer league. Eur J Oper Res 185:795 – 810

Rasmussen RV, Trick MA (2007) A benders approach for the constrained minimum break problem. Eur J Oper Res 177:198 – 213

Schönberger J, Mattfeld DC, Kopfer H (2004) Memetic algorithm timetabling for non-commercial sport leagues. Eur J Oper Res 153:102–116

Scoppa V (2015) Fatigue and team performance in soccer: Evidence from the FIFA World Cup and the UEFA European Championship. J Sport Econ 16:482–507

Suksompong W (2016) Scheduling asynchronous round-robin tournaments. Oper Res Lett 44:96–100

Trick MA (2001) A schedule-then-break approach to sports timetabling. In: Burke E, Erben W (eds) Practice and Theory of Automated Timetabling III, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 242–253

Van Bulck D, Goossens D (2020a) Handling fairness issues in time-relaxed tournaments with availability constraints. Comput Oper Res 115:104856

Van Bulck D, Goossens D (2020b) On the complexity of pattern feasibility problems in time-relaxed sports timetabling. Oper Res Lett 48:452 – 459

Van Bulck D, Goossens DR, Spieksma FCR (2019) Scheduling a non-professional indoor football league: a tabu search based approach. Ann Oper Res 275:715–730

Van Bulck D, Goossens D, Schönberger J, Guajardo M (2020) RobinX: A three-field classification and unified data format for round-robin sports timetabling. Eur J Oper Res 280:568 – 580

Hybridizing Constraint Programming and Meta-Heuristics for Multi-Mode Resource-Constrained Multiple Projects Scheduling Problem

Arben Ahmeti · Nysret Musliu

the date of receipt and acceptance should be inserted later

Abstract The Multi-Mode Resource-Constrained Multiple Projects Scheduling Problem (MMRCMPSP) is an important real-life problem. The aim is to schedule activities belonging to multiple project instances respecting different shared resources, precedence, and time constraints. To solve this problem we propose a new hybrid approach combining constraint programming (CP) and a meta-heuristic based algorithm. To this aim, we propose and evaluate a CP model that includes all constraints of MMRCMPSP. The hybrid approach takes advantage of the complementary features of CP and meta-heuristics. Our method outperforms state-of-the-art methods for this class of problems by generating new upper bounds for several instances. Moreover, we evaluate our method on the existing well-studied benchmark instances for multiple-mode resource constrained single project scheduling problems and provide new upper bounds for many instances.

Keywords Meta-Heuristics and Constraint Programming · Hybrid approach · Iterated Local Search · Project Scheduling · Min Conflicts

1 Introduction

Scheduling problems sum up a class of various combinatorial optimization problems of high interest for academics and industry. In particular, project scheduling is an important representative of this type of problems that usually has to deal with scheduling of activities of project/s under different types of

Arben Ahmeti
Institute of Logic and Computation, DBAI, TU Wien
E-mail: aahmeti@dbai.tuwien.ac.at

Nysret Musliu
Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling, Institute of Logic and Computation, DBAI, TU Wien
E-mail: musliu@dbai.tuwien.ac.at

constraints (resource constraints, precedence constraints, time horizons, etc.) and different types of objectives. One of the most studied problems of this class, Resource Constrained Project Scheduling Problem (RCPSP), have to deal with scheduling of activities of a project that require a certain amount of resources and are subject to precedence constraints among them. According to a review article [1], the RCPSP belongs to the group of NP-hard optimization problems. In many cases, the objective of this problem is the minimization of the project makespan. One of the most prominent benchmark libraries for RCPSP instances, PSPLIB, is provided by Kolisch and Sprecher [2]. Various enhanced versions of the RCPSP of academic and engineering interest have been introduced over time. A general variant (Multi-Mode Resource-Constrained Multiple Projects Scheduling Problem (MMRCMPSP)) of RCPSP is presented at the MISTA 2013 challenge.

In this paper, we focus on the MMRCMPSP problem. It extends RCPSP problem from two perspectives [3]: activities of a project may have more than one mode of execution (MRCPSP problem) and multiple instances of projects sharing scarce resources need to be scheduled simultaneously (RCMPSP problem). MMRCMPSP is a closer representation of the real-world scheduling problems. It consists on simultaneous scheduling of multiple project instances in an optimal schedule while their activities can be executed in more than one of modes and are subject to different types of resources, time, and precedence constraints. Moreover, the main objective in case of this problem is the minimization of the total project delay (TPD) and the total makespan (TMS) of projects serves as a tie-breaker.

MMRCMPSP as a more generalized form of project scheduling problem attracted the attention of many researchers. In the review article [4] a combination of Monte-Carlo Tree Search (MCTS) and hyper-heuristics were proposed. This was the winning approach of MISTA 2013 challenge. Geiger [5] introduced an iterated variable neighborhood search with four neighborhood structures. In [6] a multi-neighborhood, parallel local search approach was proposed. Another meta-heuristic approach based on iterated local search was introduced in [7]. [8] implemented a genetic algorithm for a multi-project environment with projects with assigned due dates and a resource dedication policy. A stochastic local search procedure with two neighborhoods was implemented by [9]. In [10] a genetic algorithm for mode assignment and a priority rules heuristic for job selection in MMRCMPSP was implemented. Other proposed methods for this problem may be found in [11], [12], [13] and [14]. Regardless of the contributions of many researchers to MMRCMPSP, optimal solutions of many existing instances are not known yet. In this paper, we introduce a new hybrid approach that combines a constraint programming approach with a meta-heuristic that extends the previous method proposed in [7]. Our approach outperforms the best existing algorithm [4] (to our best knowledge) for MMRCMPSP, by providing new upper bounds to many benchmark problems. Additionally, our approach provides new upper bounds for fifty benchmark instances (MMLIB library) for the MRCPSP problem [15]. The main contributions of this paper are:

- We provide a constraint programming model for MMRCMPSP and apply a state of the art CP solver to solve existing problem instances. Although constraint programming has been used previously for related project scheduling problems, to the best of our knowledge this is the first time that CP is applied for MMRCMPSP, which includes several extensions. Our model was tested with thirty benchmark instances from MISTA 2013 challenge. Long runs of algorithm resulted in new upper bounds for six instances. Execution of algorithm under time restriction conditions (5 minutes run per instance) gave the best results for three instances compared to other best solvers.
- The improvement of a local search based algorithm proposed in [7]. The improved version includes a new neighborhood operator that improved results of [7] for most benchmark instances. Additionally, it provides new upper bounds for five multiple-mode resource constrained single project scheduling problem instances.
- Proposal of a hybrid algorithm that combines the CP model and the improved meta-heuristic approach. The hybrid algorithm improved further the results and outperformed best state-of-the-art solver ([4]) for MMRCMPSP in many instances. We provide new upper bounds for almost half of the benchmark instances (fourteen new upper bounds and four equal upper bounds out of thirty) and fifty new upper bounds for multiple-mode resource constrained single project scheduling problem benchmark instances.

2 Problem Description

The MMRCMPSP problem [3] is comprised of a set of n projects: $P = \{1, 2, .., n\}$ and every project $i \in P$ is comprised of activities J_i that are executed in more than one of the modes taking into account different shared resources, time, and precedence constraints. Also, every project has a release date r_i, i.e. the earliest time when its activities could start. Every activity $j \in \{1, 2, .., |J_i|\}$ of every project has to be scheduled, i.e. its starting time s_{ij} has to be defined considering all constraints. The first and last activities of projects are dummy activities with only one execution mode, duration equal to zero and no resource requirements. There are sets of renewable and non-renewable resources $L_i \in \{1, ..., |L_i^\rho|, |L_i^\rho| + 1, ..., |L_i^\rho| + |L_i^v|\}$, where $L_i^\rho \in \{1, ..., |L_i^\rho|\}$ indicates renewable resources and $L_i^v \in \{|L_i^\rho| + 1, ..., |L_i^\rho| + |L_i^v|\}$ non-renewable resources. All non-renewable resources have fixed capacities for the whole project duration, for every project. Renewable resources have a fixed capacity per time unit. There are local renewable resources dedicated to a specific project only and global renewable resources (G^ρ) that are shared among all the projects. The availability of global renewable resources is limited by c_g^ρ, $g \in G^\rho$. There are no global non-renewable resources. Every activity has more than one available execution mode. An execution mode of an activity defines time duration required to complete the activity and its specific resource re-

quirements. Execution modes of activities are defined by $M_{ij} \in \{1, ..., |M_{ij}|\}$ and d_{ijm} (duration of activity $j \in J_i, i \in P$ in mode $m \in M_{ij}$). Moreover, r^{ρ}_{ijml}, r^{v}_{ijml} and r^{ρ}_{ijmg} determine the requirements for local renewable, non-renewable and global renewable resources, respectively, when activity $j \in J_i, i \in P$ is processed in mode $m \in M_{ij}$. Feasible projects schedules must always satisfy following hard constraints:

- For every local non-renewable resource $l \in L^v_i$ dedicated to every project $i \in P$, its total consumption cannot exceed its capacity $l \leq c_{il}$.
- For every local renewable resource $l \in L^{\rho}_i$ dedicated to every project $i \in P$, its total consumption at time unit t cannot exceed its capacity $l \leq c_{il}$.
- For every global renewable resource $g \in G^{\rho}$, its total resource consumption at time unit t cannot exceed its capacity $l \leq c^{\rho}_g$.
- Particular activities may require the completion of other activities before they start. In that case, feasible schedules must fulfill all precedence constraints between such activities, i.e. if activity $j \in J_i$, which is executed in mode m, must precede activity $j' \in J_i$.
- Release time of every project is respected, i.e. for each activity $j \in \{1, 2, .., |J_i|\}$ of every project $i \in P$, its start time $s_{ij} \geq 0$ and $s_{ij} \geq r_i$.

The objective is to find a feasible schedule with minimum total project delays (TPD) and projects total makespan (TMS). TPD is the primary objective and TMS is used as a tie-breaker. Project delay of a project i is defined as the difference between Critical Path Duration (CPD), a theoretical lower bound on the earliest finish time of the project, and the actual project duration (makespan):

$$PD_i = MS_i - CPD_i \tag{1}$$

MS_i - makespan of project i is calculated as difference:

$$MS_i = f_i - r_i \tag{2}$$

f_i - finish time of project i,
r_i - the release date of project i,
Total project delay is calculated as:

$$TPD = \sum_{i=1}^{n} PD_i \tag{3}$$

n - the number of projects.

Total makespan is the duration of the complete multi-project schedule:

$$TMS = \max_{i \in P} (f_i) - \min_{i \in P} (r_i) \tag{4}$$

Both soft constraints are combined into a single objective function as follows:

$$F = a * TPD + TMS \tag{5}$$

where value a in the MISTA 2013 challenge was $a = 100,000$.

3 The constraint programming model for MMRCMPSP

As stated earlier, MMRCMPSP generalizes other types of scheduling problems and is closer to real-world problems representations. Different models for different scheduling problems have been presented in the literature. We built our model using several features of the model for MRCPSP scheduling problems presented in [16].
Main extensions to our MMRCMPSP model are related to modeling of:

- Constraints related to a set of global renewable resources (G^{ρ}) that are shared among all the projects and their availability is limited by c^{ρ}_{g}, $g \in G^{\rho}$ (equation 18). There are no global non-renewable resources.
- Release dates constraints and dummy activities for every project (equations 7-9).
- Implementation of an objective function that consists on finding a feasible schedule that fulfills constraints while minimizing the total project delay (TPD) and the total makespan (TMS). Project delay is defined as the difference between Critical Path Duration (CPD) and the actual project duration (makespan). TPD is the primary objective and TMS as the duration of the whole multi-project schedule is used as a tie-breaker (equation 20).
- Multi-dimensional data structures to model multi-project instances execution modes, local renewable and non-renewable resources and add global resources.

We further describe the main components and constraints of the CP model. We have used the IBM CP Optimizer to implement it. According to the problem definition, every project $i \in P = \{1, 2, .., n\}$ is comprised of a set of non-preemptive activities or jobs J_i and has a release date r_i, i.e. the earliest time when the activities of the project i can start.
Activities are modelled as decision variables:

$$\text{interval}\, J_i \qquad \forall i \in P \tag{6}$$

and projects' release dates constraints and dummy activities:

$$\text{startOf}(s_i) = r_i \qquad \forall i \in P, s_i \in P_i \tag{7}$$

$$\text{startOf}(s_i) \leq \text{startOf}(j) \qquad \forall i \in P, j \in P_i \tag{8}$$

$$\text{endOf}(j) \leq \text{startOf}(e_i) \qquad \forall i \in P, e_i, j \in P_i \tag{9}$$

Every activity $j \in J_i$, of every project $i \in P$, has one or more available execution modes $m \in M_{ij}$. The execution mode of an activity determines duration d_{ijm} required to complete the activity and its specific resource requirements. Execution modes and processing time in every mode for every activity are modeled as decision variables in our model as well:

$$\text{interval}\, m_{ij}\, \text{optional} \qquad \forall i \in P, j \in J_i, m_{ij} \in M_{ij} \tag{10}$$

$$\text{interval}\, d_{ijm}\, \text{optional} \qquad \forall i \in P, j \in J_i, m \in M_{ij} \tag{11}$$

Since, every activity $j \in J_i$, can be executed in only one of its modes, then:

$$\text{alternative}(j, [m_{ij}]_{\in Mij}) \qquad \forall i \in P, j \in J_i \tag{12}$$

Resources are expressed as cumul-function expressions:

$$\text{cumulFunction}\, c_{il} \qquad \forall i \in P, l \in L_i^{\rho} \tag{13}$$

$$\text{cumulFunction}\, c_g^{\rho} \qquad \forall g \in G^{\rho} \tag{14}$$

$$\text{intExpr}\, h_{il} \qquad \forall i \in P, l \in L_i^{\nu} \tag{15}$$

Feasible schedules of projects must always fulfill following hard constraints:

- For each project $i \in P$ and each local non-renewable resource associated to that $l \in L_i^{\nu}$, total resource consumption does not exceed its capacity $l \leq h_{il}$. Since non-renewable resources have fixed capacities for the whole project duration we modeled them in our model as scalar expressions:

$$\sum_{j \in J_i} \sum_{m_{ij} \in M_{ij}} \text{presenceOf}(m_{ij})({r_{ijm}}_l^{\nu}) \leq h_{il} \quad \forall i \in P, l \in L_i^{\nu} \tag{16}$$

  In MMRCMPSP model every project has its own list of non-renewable resources.
- For each project $i \in P$ and each local renewable resource associated to that $l \in L_i^{\rho}$, total resource consumption does not exceed its capacity $l \leq c_{il}$. As the name implies local renewable resources are dedicated to a specific project and have a fixed capacity per time unit, meaning their capacity constraints have a temporal dimension. Therefore, they are modeled as cumulative functions:

$$\sum_{j \in J_i} \sum_{m_{ij} \in M_{ij}} \text{pulse}(m_{ij}, {r_{ijm}}_l^{\rho}) \leq c_{il} \quad \forall i \in P, l \in L_i^{\rho} \tag{17}$$

  In MMRCMPSP model every project has its own list of renewable resources.
- For each time unit t and each global renewable resource $g \in G^{\rho}$, total resource consumption at t does not exceed its capacity $l \leq c_g$. Global renewable resources are modeled similarly as local renewable resources as cumulative function. There is only one global resources list common to all projects:

$$\sum_{i \in P} \sum_{j \in J_i} \sum_{m_{ij} \in M_{ij}} \text{pulse}(m_{ij}, {r_{ijm}}_g^{\rho}) \leq c_g \qquad \forall g \in G^{\rho} \tag{18}$$

  There is no global non-renewable resources list.
- Feasible schedules must fulfill all precedence constraints between activities:

$$\text{endBeforeStart}(a, j) \qquad \forall i \in P, a, j \in P_i \tag{19}$$

According to definition the main objective function of MMRCMPSP problem consists of minimizing total project delay of projects and the projects total makespan as tie-breaker. It is defined as:

$$f = \min(\max(\text{endOf}(j) + \sum_{i \in P}(\alpha * \max(\text{endOf}(P_i))))), \forall j \in J_i \tag{20}$$

α – is a constant.

4 Description of extended meta-heuristic

Our main aim regarding the hybrid algorithms for MMRCMPSP is to combine two complementary search strategies. Various classifications and taxonomies for hybrid approaches can be found in the literature [17], such as combining meta-heuristics with constraint programming, combining meta-heuristics with exact methods from mathematical programming, combining meta-heuristics with other meta-heuristics or machine learning techniques. We opted for combination of an exact approach, a CP model, with a local search based algorithm that extends the algorithm in [7], which combines min conflicts and tabu search heuristics embedded in an iterated local search framework.

4.1 Solution representation

In our extended meta-heuristic solutions are represented as pair of vectors: $\overrightarrow{S} = \{\overrightarrow{\pi}, \overrightarrow{M}\}$, where $\overrightarrow{\pi}$ represents the vector of all activities from all projects and $\overrightarrow{M}$ is its corresponding modes vector:

$$\overrightarrow{\pi} = \{1, 2, .., |J_1|, |J_1| + 1, .., |J_1| + |J_2| + .. + |J_\text{n}|\} \tag{21}$$

Analogous representations were employed by [4], [6], [5] and [7]. Construction of a solution alternative is done by sequentially assigning activities into a schedule as early as possible.

Algorithm 1 SwapAndModeCh(s) neighborhood operator

```
repeat
  Improved ← false
  act_i ← random.select.from(J_i)
  while not IsSuccessor(s, act_{i+1}) do
    s' ← Swap(s, act_i, act_{i+1})
    for all modes_of_act_{i+1} do
      s'' ← OneModeChange(s', act_{i+1})
      if (eval(s'') < eval(s)) then
        s ← s''
        Improved ← true
      end if
    end for
  end while
until not Improved
```

First, we added a complex neighborhood operator, *SwapAndModeCh*, comprised of two existing simple ones *SwapActivity* and *OneModeChange* noted in [4], [6], [5]. As our CP solver (IBM CP optimizer) also includes a large neighborhood search, we excluded from the implementation of local search four-mode-change (*MinConFMC*) neighborhood operator due to the fact that it generates also large neighborhoods. *SwapAndModeCh* applies swap between an activity and its successor along the schedule until the precedence constraint is not violated. After each swap, activity is assigned each of its modes in turn and if there is an improvement the solution is accepted (algorithm 1).
Figure 1 illustrates the generated neighborhood by this operator assuming that $act_i = 3$, $SuccessorOf(act_i) = 5$ in a given input solution s. Activity 3 swaps with its descendants in the schedule all the way up to its successor, activity 5, with whom it has a precedence constraint.

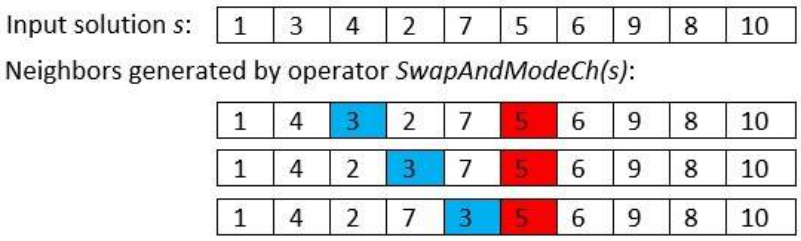


Fig. 1 SwapAndModeCh(s) neighborhood operator

The implementation of improved meta-heuristic is designed for multi-threaded execution environment (algorithm 2).

Algorithm 2 Improved algorithm ILS_Min_Con(s, Time)

for all $\overrightarrow{S}_i$ **do**
  $\overrightarrow{S}_i \leftarrow s$
end for
$\overrightarrow{S}_{best} \leftarrow \emptyset, \quad \overrightarrow{S}_{bestlocal} \leftarrow \emptyset$
$NoImprovement \leftarrow true, \quad LocalImprovement \leftarrow true$
repeat
  repeat
    for all $\overrightarrow{S}_i$ **do**
      $CloneProj(), CloneProjPart(), ComE(), ComF()$
    end for
    if $(BroadcastBestLocal())$ **then**
      $NoImprovement \leftarrow false$
    else
      $NoImprovement \leftarrow true$
    end if
  until NoImprovement
  $Improve \leftarrow true$
  while $Improve$ **do**
    for all $\overrightarrow{S}_i$ **do**
      $MinConOMC()$
    end for
    if $(BroadcastBestLocal())$ **then**
      $LocalImprovement \leftarrow true$
    else
      $Improve \leftarrow false$
    end if
  end while
  if $LocalImprovement$ **then**
    for all $\overrightarrow{S}_i$ **do**
      $PCom(), MinConTMC()$
    end for
  else
    for all $\overrightarrow{S}_i$ **do**
      $SwapAndModeCh(\overrightarrow{S}_i), INVS(), MinConSJL(), MinConSJR()$
    end for
    $BroadcastBestLocal()$
  end if
  if $(\overrightarrow{S}_{bestlocal} < \overrightarrow{S}_{best})$ **then**
    $\overrightarrow{S}_{best} = \overrightarrow{S}_{local}, \quad NoImprovement \leftarrow false$
  else
    $NoImprovement \leftarrow true$
  end if
  $LocalImprovement \leftarrow NoImprovement$
  if $(not \quad LocalImprovement)$ **then**
    $LocalImprovement \leftarrow not \quad LocalImprovement$
    $PerturbationSize \leftarrow PerturbationSize + 1$
  else
    $PerturbationSize \leftarrow 1$
  end if
  for all $\overrightarrow{S}_i$ **do**
    $Perturbate(\overrightarrow{S}_{bestlocal}, PerturbationSize), Reset(\overrightarrow{S}_{bestlocal})$
  end for
until Time

Other neighborhood operators depicted in algorithm 2, one mode change (MinConOMC), two-mode change (MinConTMC), shift an activity to its last predecessor (MinConSJL), shift an activity to its first successor (MinConSJR), invert subsequence of activities (INVS), compress project and move to the end (ComE), compress project and move to the front (ComF), clone a project (CloneProj), clone a project partially (CloneProjPart) and clone a sequence from a project (CloneSeq) are implemented similar to the implementation in [7]. Definition of BroadcastBestLocal() method is depicted in algorithm 3.

Algorithm 3 BroadcastBestLocal()

$minLocal \leftarrow \min_{i \in \{1,..,4\}} eval(\overrightarrow{S}_i)$
$s_{temp} \leftarrow \emptyset$
if $(eval(\overrightarrow{S}_{bestlocal}) > minLocal)$ **then**
  $minLocal = \min_{i \in \{1,..,4\}} eval(\overrightarrow{S}_i)$
  $s_{temp} \leftarrow \overrightarrow{S}_{arg \min_{i \in \{1,..,4\}} \overrightarrow{S}_i}$
  for all $\overrightarrow{S}_i$ **do**
    $\overrightarrow{S}_i \leftarrow s_{temp}$
  end for
  $\overrightarrow{S}_{bestlocal} \leftarrow s_{temp}$
  $return \quad true$
else
  $return \quad false$
end if

4.2 Acceptance criteria and perturbation

Similar to the implementation in [7], we accept only better solutions and implemented adaptive perturbation strategy in relation to instance size. The perturbation consists of changing modes of up to 10 % of randomly selected activities from the schedule.

4.3 Parameter tuning

In this algorithm, two adaptive tabu lists are implemented: one tabu list for operators that manipulate modes and one for operators that manipulate positions of activities in the schedule. The dimensions of these tabu lists are parameterized and experimentally determined as a percentage of the total number of activities in a given instance. Other parameters we fine tuned are: size of variable set for MinConOMC and MinConTMC operators and perturbation size threshold. Parameter values are fine tuned using the SMAC tool ([18], [19]) and obtained are depicted in Table 1.

Table 1 Parameters used for tests

Parameter	Value	Domain of values
ModeTBLength	30%	{10%, 15%, 20%, 25%, 30%, 35%, 40%}
SeqTBLength	20%	{10%, 15%, 20%, 25%, 30%, 35%, 40%}
VarSetSize	11	{5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15}
PertSizeThreshold	10%	{3%, 5%, 7%, 10%, 13%, 15%, 20%}

5 Hybrid Method

Our hybrid implementation consists of sequential execution of constraint programming model and extended meta-heuristic explained in the previous paragraph. According to [17] it could be classified as high-level relay hybrid (HRH) approach. The output solution of one method serves as initial solution for the other. A perturbation of the best incumbent solution is performed if there was no improvement between every sequential call of algorithms. The perturbation strategy is based on a mode change of an activity and inversion of a short sequence of activities randomly selected from the schedule. It always produces feasible solutions. The perturbed solution serves as a starting point for the CP model. We experimented with allotted execution time for every algorithm in ratio two to one and three to one in favor of CP model within an execution sequence. Slightly better results were obtained with ratio three to one (algorithm 4).
In our hybrid method data reading technicalities for MMRCMPSP and MRCPSP problems are implemented. Proper data structures (interval variable arrays, multi-dimensional cumulative functions arrays, scalar expression arrays, constraints, etc.) for the proper type of problems are generated and populated.

6 Computational results

We performed experiments on thirty benchmark MMRCMPSP problem instances provided in MISTA 2013 challenge. Algorithms presented in this challenge were executed in a computer with a 64-bit Intel Core i7 processor (3.4 GHz) CPU of eight cores, 8 GB RAM. Every algorithm was executed ten times for each instance and five minutes for every run. All of our tests were executed on a machine with 64-bit Intel Core i5 processor (3.3 GHz) CPU of four cores, 8GB RAM. Algorithms are implemented in the programming language C# (Visual Studio 2019 development environment). In order to perform experiments in approximately identical conditions, we determined running time in our machine according to benchmark program (the 64-bit version) of the 2011 International Timetabling Competition. [1] The execution of this program on our computer lasted 690 seconds, while on the computer used in the competition 645 seconds. Therefore, we set the running time on our computer

[1] https://www.utwente.nl/ctit/hstt/itc2011/benchmarking/

Algorithm 4 Hybrid approach for MMRCMPSP combining a CP model and an extended meta-heuristic

```
s ← ∅
s' ← ∅
CP.Add(Projects, Activities, Modes, Resources)
CP.Add(ConstraintsTypes : (7), (8), (9), (16), (17), (18)and(19))
{See equations (7), (8), (9), (16), (17), (18) and (19)}
CP.Add(ObjectiveFunction : equation(20))
CP.SetParameters(Time * 3, SearchType, RandomSeed)
while not Termination_Condition do
   CP.Solve()
   s ← CP.ExtractSolution()
   s'' ← ILS_Min_Con(s, Time)
   if (eval(s'') < eval(s')) then
      s' ← s''
   else
      act_i ← random.select.from(J_i)
      s ← RandomOneModeChange(s', act_i)
      s ← RandomInvertSubSequence(s)
   end if
   sp ← CP.Solution()
   sp.Set(s)
   CP.SetStartingPoint(sp)
end while
```

to 321 seconds, accordingly. Additional experiments were accomplished under different settings with and without time restrictions.

6.1 Benchmark instances

Every set of benchmark instances (A, B and X) is comprised of ten instances. Sizes of instances vary from smallest one comprised of 2 projects and 20 activities up to the biggest one with 20 projects and 600 activities. Even though, we were focused on solving MMRCMPSP problem instances, we tested our algorithms on solving multi-mode resource-constrained project scheduling problem instances introduced in [15] as well. New upper bounds were obtained for many instances from each set of problems.

6.2 Evaluation of the hybrid method

Our hybrid approach outperformed the best solver implemented in [4] for MMRCMPSP problem when running algorithms under no-restriction conditions. It generated new upper bounds for the majority of instances of this group, (Table 2). Solver presented in [4] ran 2500 times for each instance and 5 minutes for every run. We executed our solver for 24 hours only once for each instance. Many of the instances converged much earlier, within a few minutes. Comparison of results of our hybrid method with those of the solver imple-

Table 2 Comparison of our hybrid approach's results (TPD/TMS) with the best solvers results so far for MMRCMPSP problem

Inst.	[4]	[5]	[6]	Our algorithm	New upper bounds
A1	1/23	1/23	1/23	1/23	Equal
A2	2/41	2/41	2/41	2/41	Equal
A3	0/50	0/50	0/50	0/50	Equal
A4	65/42	65/42	68/50	65/42	Equal
A5	150/103	153/104	154/104	151/104	
A6	133/99	144/94	151/94	**132/90**	Yes
A7	590/190	601/206	626/194	595/189	
A8	272/148	319/162	281/147	**257/147**	Yes
A9	197/122	225/128	212/127	**186/122**	Yes
A10	836/303	920/313	983/309	854/307	
B1	345/124	349/130	358/131	348/127	
B2	431/158	481/171	431/159	**404/160**	Yes
B3	526/200	604/214	585/196	**515/204**	Yes
B4	1252/275	1283/287	1435/294	1296/283	
B5	807/245	866/252	867/254	813/250	
B6	905/225	1067/246	970/224	**888/219**	Yes
B7	782/225	827/232	876/234	800/233	
B8	3048/523	3618/565	3001/520	**2871/525**	Yes
B9	4062/738	4606/783	4753/741	4093/736	
B10	3140/436	3541/473	3123/430	**3057/437**	Yes
X1	386/137	-	392/142	**385/139**	Yes
X2	345/158	-	416/167	**342/163**	Yes
X3	310/187	-	332/177	**287/183**	Yes
X4	907/201	-	980/209	**896/204**	Yes
X5	1727/362	-	1904/369	1757/370	
X6	690/226	-	821/237	700/232	
X7	831/220	-	909/232	854/224	
X8	1201/279	-	1389/281	**1188/279**	Yes
X9	3155/632	-	3945/639	3269/641	
X10	1573/373	-	1718/377	**1572/374**	Yes

mented by [4] are also given in Figure 2, where differences between results for every benchmark instance of both solvers are visually presented.

We also evaluated our solver on MMRCSPSP instances and experiments resulted with new upper bounds for fifty instances and equal results for many more compared to results of state-of-the-art solvers (Table 3). In experiments run under time restriction conditions, i.e. 10 runs per instance and 5 minutes per each run, our hybrid solver generated best results for nine and equal results for four benchmark instances (Table 4). Average results and standard deviations are reported, too. Differences between results of our solver and best solver [4] under these restrictions are presented graphically in Figure 3. It can be noticed that for very large instances, e.g. B9 and X9, hyper-heuristic approach performs better compared to our hybrid method. This is due to the nature of the CP model that is a constituent part of the hybrid method, it performs very well for small and medium instances.

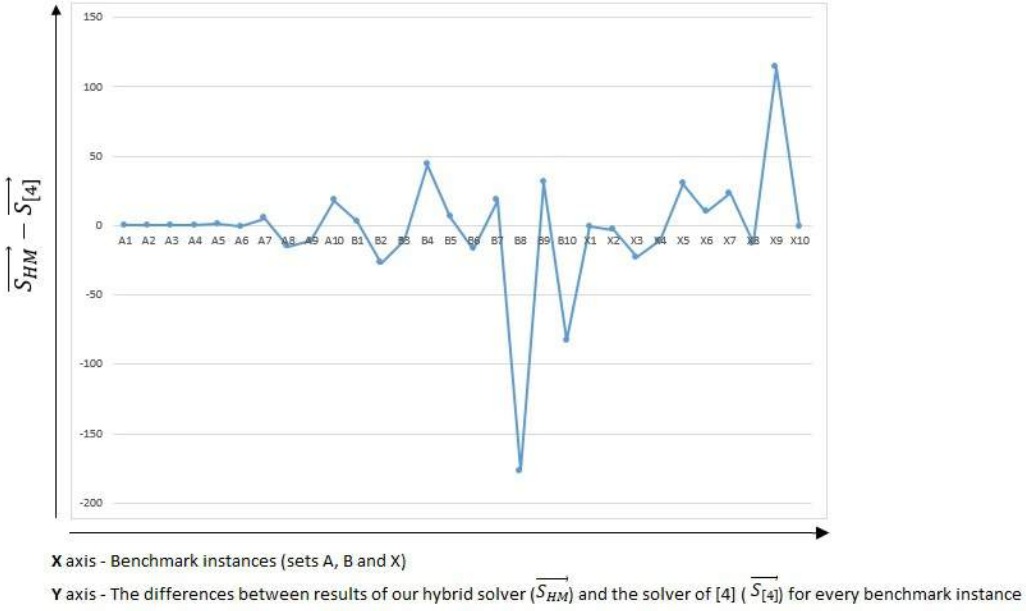


Fig. 2 Comparison of our results (TPD/TMS) with the best solver's results ([4]), under no time restriction conditions for MMRCMPSP problem

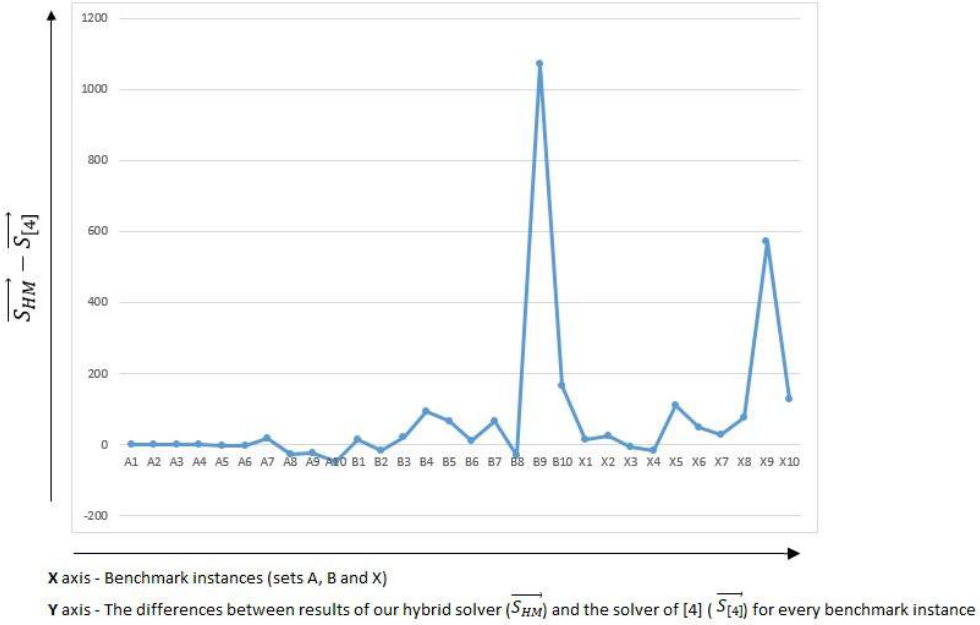


Fig. 3 Comparison of our results (TPD/TMS) with the best solver's results ([4]), under time restriction conditions for MMRCMPSP problem

Table 3 New upper bounds achieved by the hybrid method for MRCPSP problem

MMLIB Instances					
mmlib50		**mmlib100**		**mmlibPlus**	
Inst.	**Makespan**	**Inst.**	**Makespan**	**Inst.**	**Makespan**
J507_1.mm	43	J1008_5.mm	49	Jall146_1.mm	64
J507_5.mm	40	J10045_3.mm	53	Jall185_3.mm	107
J5043_5.mm	63	J10074_4.mm	75	Jall193_3.mm	75
J5045_4.mm	35	J10076_3.mm	58	Jall254_3.mm	81
J5046_5.mm	38	J10079_1.mm	128	Jall256_3.mm	113
J5047_5.mm	38	J10080_5.mm	83	Jall302_1.mm	47
J5048_2.mm	39	J10081_1.mm	85	Jall344_4.mm	83
J5080_5.mm	69	J10081_2.mm	74	Jall346_2.mm	78
-	-	J10081_3.mm	70	Jall347_3.mm	57
-	-	J10082_3.mm	78	Jall363_1.mm	57
-	-	J10082_4.mm	94	Jall371_1.mm	84
-	-	J10083_2.mm	79	Jall372_3.mm	72
-	-	J10083_3.mm	71	Jall374_4.mm	66
-	-	J10084_3.mm	78	Jall394_2.mm	102
-	-	J10084_5.mm	73	Jall399_1.mm	237
-	-	J10092_2.mm	80	Jall401_2.mm	193
-	-	J10092_5.mm	71	Jall410_5.mm	62
-	-	J10094_1.mm	54	Jall458_3.mm	79
-	-	-	-	Jall509_1.mm	138
-	-	-	-	Jall512_1.mm	132
-	-	-	-	Jall537_3.mm	133
-	-	-	-	Jall537_4.mm	104
-	-	-	-	Jall556_2.mm	99
-	-	-	-	Jall566_5.mm	124

6.3 Evaluation of the extended meta-heuristic

We have accomplished separate tests with extended meta-heuristic and our CP model under time restriction conditions. It turned out that the extended meta-heuristic slightly improved most of the MMRCMPSP results (the best results, average and standard deviation) in comparison with the implementation in [7] (Table 6). According to [7] their solver is competitive to the third ranked solver under time restrictions. Meta-heuristic approach provided better results in nine and equal in three instances compared to our CP model and none compared to hybrid method.
Additionally, we tested our extended meta-heuristic with MMRCPSP problem benchmark instances. Tests resulted in the achievement of new upper bounds for five instances (Table 5).

Table 4 Comparison of our results (TPD/TMS) with the best solver's results ([4]), under time restriction conditions for MMRCMPSP problem

	[4]			Hybrid approach		
Inst.	**Best**	**Avg**	**Std**	**Best**	**Avg**	**Std**
A1	1/23	-	-	1/23	1/23	0
A2	2/41	-	-	2/41	2/41	0
A3	0/50	-	-	0/50	0/50	0
A4	65/42	-	-	65/42	65/44	0/2
A5	153/105	-	-	**152/104**	165/106	8/2
A6	147/96	-	-	**144/92**	160/100	5/3
A7	596/196	-	-	615/205	639/201	2/5
A8	302/155	-	-	**276/150**	291/150	11/3
A9	223/119	-	-	**200/121**	215/128	11/3
A10	969/314	-	-	**921/313**	982/325	41/7
B1	349/127	352/128	-	364/126	379/128	11/2
B2	434/160	454/168	-	**419/162**	461/164	21/2
B3	545/210	554/211	-	566/212	601/212	19/1
B4	1274/289	1305/284	-	1368/291	1464/291	65/6
B5	820/254	833/254	-	887/262	923/262	18/4
B6	912/227	953/232	-	925/233	992/236	38/4
B7	792/228	801/232	-	859/239	921/244	49/5
B8	3176/533	3314/548	-	**3145/561**	3511/568	179/14
B9	4192/746	4264/755	-	5262/884	5740/922	320/30
B10	3249/456	3338/460	-	3415/473	3583/469	137/6
X1	392/142	405/142	-	408/142	432/145	21/2
X2	349/163	357/164	-	375/167	402/170	20/4
X3	324/192	330/193	-	**318/187**	346/190	19/4
X4	955/213	971/212	-	**939/210**	1033/209	53/2
X5	1768/374	1785/373	-	1878/386	1956/388	43/5
X6	719/232	738/241	-	768/240	840/249	72/8
X7	861/237	868/236	-	890/235	925/239	24/5
X8	1233/283	1257/289	-	1310/287	1452/300	92/8
X9	3268/643	3303/647	-	3840/718	4134/750	163/15
X10	1600/381	1614/382	-	1727/403	1777/403	36/5

Table 5 New upper bounds achieved by meta-heuristic for MMRCPSP problem

	Project total makespan new upper bounds	
Instance	**Our algorithm**	**Different authors**
Jall127_3.mm	142	143
Jall128_5.mm	94	95
Jall184_1.mm	177	179
Jall263_4.mm	146	147
Jall289_5.mm	196	197

6.4 Evaluation of CP Model

We also performed tests with our CP model under time restriction conditions. According to [16] the search in a CP model can be directed through configuration of several parameters, e.g. search type, random seed, time limit, etc. Results of the CP model depicted in Table 6 for MMRCMPSP instances

are performed with *Search Type = Restart*, *Random Seed = 1292619981* and *Time Limit = 300 s*. In restart search mode the algorithm restarts and executes depth first search after a parameterized number of failures. Another search type we have experimented with is automatic search that employs large neighborhood search and failure directed search [16]. The former one tries to converge quickly to a good quality solution and the latter one tries to prove that no better solution exists than the existing one when the search space is too small or LNS cannot improve further the solution. CP model uses random seed parameter for tie-breaking situations only. In Table 6 it can be seen that the CP model provided slightly better results for three instances compared to hybrid method and three compared to [4].
The hybrid approach seems to be far more successful than executing the constituent algorithms separately. In the case of very large instances, under short time restriction conditions, e.g. MISTA challenge conditions, meta-heuristic approaches appear to be slightly better. Under no time restriction conditions, the hybrid approach outperforms all solvers.

7 Conclusions and future work

In this paper we investigated three approaches for MMRCMPSP: a CP model, an extended meta-heuristic and a hybrid approach combining the first two solvers. We performed separate experiments with all solvers on existing MMRCMPSP problem benchmark instances and compared to the state-of-the-art solver for this problem. It turned out that the meta-heuristic approach provided better results in nine instances compared to our CP model and none compared to the hybrid method. CP model provided better results for three instances compared to the hybrid method and three compared to one of the best solvers ([4]) for restricted time conditions. Regarding the hybrid approach, we performed extensive tests in various experimental settings. It outperformed best existing solver for MMRCMPSP in several instances and achieved new upper bounds for fourteen out of thirty instances under no time restrictions conditions. Under time restriction, the model generated best results for nine and equal results for four benchmark instances. For some very large instances, the best existing solver ([4]) provided better results. Additionally, we tested our approach with multiple-mode resource constrained single project scheduling problems well-known instances and experiments resulted with new upper bounds for fifty instances.
Our main objective was to evaluate a hybrid approach that combines two complementary search strategies. In this study, we introduce a successful combination of an exact model with a meta-heuristic approach and a certain perturbation mechanism. We consider that it is of high interest to investigate further hybrid approaches that combine different search strategies in order to create a robust and efficient method. Especially, the role of meta-heuristics in a hybrid method should be further studied. In our case, we noticed that for very large instances (B9 and X9), under short time restrictions, hyper-

Table 6 Comparison of extended meta-heuristics results (TPD/TMS) with the best solvers results for MMRCMPSP problem under time restriction conditions

Ins.	**Extended meta heuristic**	[7]	CP Model	hybrid method	[4]	[5]	[6]
A1	1/23	1/23	1/23	1/23	1/23	-	-
A2	2/41	2/41	2/41	2/41	2/41	-	-
A3	0/50	0/50	0/50	0/50	0/50	-	-
A4	65/42	65/45	65/45	65/42	65/42	-	-
A5	162/107	163/108	173/110	**152/104**	153/105	-	-
A6	156/94	152/96	162/104	**144/92**	147/96	-	-
A7	644/203	652/208	655/197	615/205	596/196	-	-
A8	337/166	335/163	279/148	**276/150**	302/155	-	-
A9	245/139	253/137	212/124	**200/121**	223/119	-	-
A10	969/338	980/331	1017/317	**921/313**	969/314	-	-
B1	355/129	361/129	375/130	364/126	349/127	353/125	363/132
B2	498/177	502/180	438/167	**419/162**	434/160	490/176	434/160
B3	639/230	637/224	586/206	566/212	545/210	598/215	660/207
B4	1386/302	1415/290	1493/286	1368/291	1274/289	1274/289	1548/295
B5	955/275	927/268	922/267	887/262	820/254	866/254	919/254
B6	1139/261	1146/253	936/227	925/233	912/227	1044/242	1128/232
B7	890/251	864/249	1003/252	859/239	792/228	834/234	908/246
B8	3687/628	3836/626	3113/544	**3145/561**	3176/533	3585/568	3276/529
B9	5858/948	5757/926	5253/833	5262/884	4192/746	4674/796	5373/769
B10	3636/456	3654/514	3295/455	3415/473	3249/456	3518/469	3325/447
X1	435/148	427/150	443/144	408/142	398/142	394/142	392/142
X2	419/175	408/174	405/167	375/167	349/163	368/165	418/165
X3	382/202	407/206	346/194	**318/187**	324/192	372/195	326/188
X4	1035/221	1081/221	996/208	**939/210**	955/213	970/215	986/207
X5	2083/393	2089/428	1940/376	1878/386	1768/374	1938386	2043/375
X6	967/281	953/284	799/243	768/240	719/232	844/253	880/240
X7	951/245	968/248	902/233	890/235	861/237	879231	944/234
X8	1584/329	1515/324	1366/288	1310/287	1233/283	1380/296	1478/289
X9	4374/790	4167/776	4320/760	3840/718	3268/643	3645/688	4169/662
X10	1938/437	1934/427	1739/396	1727/403	1600/381	1669/402	1851/385

heuristic approach still performed better than our hybrid approach. Since the exact model relies on large neighborhood search and performs very well on small and medium instances than it is up to meta-heuristics to perform efficient search within very short time conditions in order to provide excellent results even in the case of very large instances. Furthermore, improving the complementary search nature of algorithms that comprise a potential hybrid method would be of valuable interest.

References

1. Jacek Blazewicz, Jan Karel Lenstra, and Alexander H.G. Rinnooy Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5(1):11–24, 1983.

2. Rainer Kolisch and Arno Sprecher. Psplib-a project scheduling problem library: Or software-orsep operations research software exchange program. *European journal of operational research*, 96(1):205–216, 1997.
3. Tony Wauters, Joris Kinable, Pieter Smet, Wim Vancroonenburg, Greet Vanden Berghe, and Jannes Verstichel. The multi-mode resource-constrained multi-project scheduling problem. *Journal of Scheduling*, 19(3):271–283, 2016.
4. Shahriar Asta, Daniel Karapetyan, Ahmed Kheiri, Ender Özcan, and Andrew J. Parkes. Combining monte-carlo and hyper-heuristic methods for the multi-mode resource-constrained multi-project scheduling problem. *Information Sciences*, 373:476–498, 2016.
5. Martin Josef Geiger. A multi-threaded local search algorithm and computer implementation for the multi-mode, resource-constrained multi-project scheduling problem. *European Journal of Operational Research*, 256(3):729–741, 2017.
6. Túlio A.M. Toffolo, Haroldo G. Santos, Marco A.M. Carvalho, and Janniele A. Soares. An integer programming approach to the multimode resource-constrained multiproject scheduling problem. *Journal of Scheduling*, 19(3):295–307, 2016.
7. Arben Ahmeti and Nysret Musliu. Min-conflicts heuristic for multi-mode resource-constrained projects scheduling. In Hernán E. Aguirre and Keiki Takadama, editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2018, Kyoto, Japan, July 15-19, 2018*, pages 237–244. ACM, 2018.
8. Umut Beşikci, Ümit Bilge, and Gündüz Ulusoy. Multi-mode resource constrained multi-project scheduling and resource portfolio problem. *European Journal of Operational Research*, 240(1):22–31, 2015.
9. Leonardo M. Borba, Alexander J. Benavides, Tadeu Zubaran, Germano C. Carniel, and Marcus Ritt. A simple stochastic local search for multi-mode resource-constrained multi-project scheduling. In *Proceedings of the 6th Multidisciplinary International Scheduling Conference*, pages 826–830, 2013.
10. Mathias Kühn, Taiba Zahid, Michael Völker, Zhugen Zhou, and Oliver Rose. Investigation of genetic operators and priority heuristics for simulation based optimization of multi-mode resource constrained multi-project scheduling problems (mmrcmpsp). In *ECMS*, 2016.
11. Emmanuel Hebrard Christian Artigues. Mip relaxation and large neighborhood search for a multi-mode resource-constrained multi-project scheduling problem. In *Proceedings of the 6th Multidisciplinary International Scheduling Conference*, pages 815–819, 2013.
12. Federico Alonso-Pecina, Johnatan E. Pecero, and David Romero. A three-phases based algorithm for the multi-mode resource-constrained multi-project scheduling problem. In *Proceedings of the 6th Multidisciplinary International Scheduling Conference*, pages 812–814, 2013.
13. Hermann Bouly, Duc-Cuong Dang, Aziz Moukrim, and Huang Xu. Evolutionary algorithm and post-processing to minimize the total delay in multi-project scheduling. In *Proceedings of the 6th Multidisciplinary International Scheduling Conference*, pages 831–835, 2013.
14. Manuel López-Ibánez, Franco Mascia, Marie-Eléonore Marmion, and Thomas Stützle. Automatic design of a hybrid iterated local search for the multi-mode resource-constrained multi-project scheduling problem. In *Proceedings of the 6th Multidisciplinary International Scheduling Conference*, pages 820–825, 2013.
15. Vincent Van Peteghem and Mario Vanhoucke. An experimental investigation of metaheuristics for the multi-mode resource-constrained project scheduling problem on new dataset instances. *European Journal of Operational Research*, 235(1):62–72, 2014.
16. Philippe Laborie, Jérôme Rogerie, Paul Shaw, and Petr Vilím. Ibm ilog cp optimizer for scheduling. *Constraints*, 23(2):210–250, April 2018.
17. El-Ghazali Talbi. *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons, 2009.
18. Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization - 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers*, pages 507–523, 2011.
19. Marius Lindauer, Katharina Eggensperger, Matthias Feurer, Stefan Falkner, Andre Biedenkapp, and Frank Hutter. Smac v3: Algorithm configuration in python. `https://github.com/automl/SMAC3`, 2017.

Solving Vehicle Routing and Scheduling with Delivery and Installation of Machines using ILS

Valon Kastrati · Arben Ahmeti · Nysret Musliu

the date of receipt and acceptance should be inserted later

Abstract We propose a new method based on iterated local search to solve a Vehicle Routing and Scheduling problem that was recently introduced in the VeRoLog Solver Challenge 2019. Our algorithm includes several neighborhood search operators and destroy/repair heuristics. We propose two new neighborhood operators aiming to address conflicts between requests and trips for better allocation on days, vehicles and routes. Our algorithm was one of the competitors of the VeRoLog Solver Challenge where 13 different teams participated by submitting solutions for the instances from the all-time-best challenge. Results on twenty-five instances presented by the organizer of the challenge show that our approach provided second best result for one of the instances, for six instances third best and for most of the rest fourth best result.

Keywords Vehicle Routing · Scheduling · Metaheuristics

1 Introduction

The class of Vehicle Routing Problems is a family of combinatorial optimization problems that arise in the field of logistics. Capacitated Vehicle Routing Problem (CVRP) is the simplest and most well-known variant, whose objective

Valon Kastrati
Vienna University of Technology, TU Wien
E-mail: e1634370@student.tuwien.ac.at

Arben Ahmeti
Institute of Logic and Computation, DBAI, TU Wien
E-mail: aahmeti@dbai.tuwien.ac.at

Nysret Musliu
Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling, Institute of Logic and Computation, DBAI, TU Wien
E-mail: musliu@dbai.tuwien.ac.at

is to find a sequence of visits for each truck and minimize the total distance travelled, where all the customers are served, and the sum of demands for each truck does not exceed the maximum capacity of a truck. CVRP instance that uses only one truck with unlimited capacity is equivalent to the Travelling Salesman Problem (TSP) which serves as proof that CVRP is also NP-Hard problem. In CVRP all the routes start and end at a single depot, and the number of trucks is not important. Different variations and extensions of VRP have been defined and proposed as a result of many practical applications resulting in several types of constraints and objectives. For example, there are variations of VRP that include pickup and delivery requests (VRPPD), soft or hard time window for the requests (VRPTW), multiple depot (MDVRP). Vehicles might have different capacities (Heterogeneous VRP) and they are allowed to make multiple trips (Multitrip VRP). For a more comprehensive overview of the classification of VRP problems see [1].

As the number of models of VRP grows continually, so does the number of methods and approaches proposed by researchers. Classical methods include several construction heuristics, for example, the Clarke and Wright savings algorithm. The Clarke and Wright heuristic, proposed at [2], repeatedly merges two routes (starting from single request routes) by maximizing their savings. Other important construction heuristics are the *nearest neighbor* and the *cheapest insertion*, proposed by [21], that work by inserting requests into routes following a strategy such as nearest neighbor or cheapest insertion.

Well-known classical algorithms include inter-route improvement methods (*k-opt* moves based on TSP problem [11]), intra-route improvements like *λ-interchanges* [15], *Or-opt* [14] and other move operators that basically exchange or relocate a number of customers between different routes or vehicles.

Further improvement of solutions was only possible with the use of metaheuristics based on a single solution and a population of solutions. Simulated annealing was proposed by [15], tabu search was implemented in the algorithm proposed by [3] to solve vehicle routing problems with time windows. Large neighborhood search is used by many authors in general and one of the most successful algorithms was Adaptive LNS, proposed by [16], which was able to solve five different variants of VRP: VRPTW, VRPMD, OVRP, SDVRP, and CVRP. The approach proposed in [7] was to use Variable Neighborhood Search with operators such as 2-Opt, Or-Opt, inter-route relocate, or inter-route exchange embedded in Adaptive Large Neighborhood Search. A heuristic search method that only uses a single neighborhood was proposed by [6]. It focuses on techniques of fast manipulation of memory data during the preprocessing, checks, and different speedups. In [10] the problem is divided into the technicians and the deliveries subproblems, where the first one is treated as a Set Covering Problem and the second part is treated and solved as a Bin Packing Problem.

In this paper, we focus on the problem of Routing and Scheduling of Delivery and subsequent Installations of machines. This variant of VRP was

introduced on the VeRoLog Solver Challenge 2019 [1], organized by EURO Working Group on Vehicle Routing and Logistics Optimization (VeRoLog [2]) and ORTEC [8].

The challenge was organized in two parts: the first one was called *all-time-best challenge* and the second *restricted resources challenge*. In *all-time-best challenge* there were no restrictions in time or technology that could be used to solve the problem, and there was a set of 25 instances called "Early" that were made available in the challenge. For the second part of the challenge, the participants had to submit an algorithm that was used by organizers to solve instances from the "Hidden" set. In the restricted resources challenge there was a time limit for an instance based on formula $t = f * (10 + n)$, where f is the factor of the computing machine, n is the number of requests in the instance and t is time in seconds.

VeRoLog Solver Challenge 2019 attracted the attention of several researchers and 13 different teams participated by submitting solutions for instances from all-time-best challenge. After this stage of the competition was closed, 8 best teams were selected to participate in the restricted resources challenge. Organizers determined the final ranking for restricted resources challenge after running the solvers submitted by each finalist for the hidden set of instances.

Main contributions presented in this paper are:

- A new approach for Vehicle Routing and Scheduling with Deliveries and Installation of Machines. We present a local search comprised of min-conflicts, nearest neighbor and destroy and repair heuristics embedded in an iterated local search framework.
- Introduction of several neighborhood operators including two new neighborhoods for the VRP problem.
- Evaluation of our approach with three sets of benchmark instances proposed at different stages of the VeRoLog Solver Challenge 2019.

The final results of restricted resources challenge were announced in VeRoLog Meeting in Seville on 4 June 2019 [3], with top three teams giving their presentations at the conference.

2 Problem Description

Vehicle Routing Problem with Delivery and Installation of the Machines (DIM) is one of the newest problems in VRP, that was introduced by the EURO Working Group on Vehicle Routing and Logistics Optimization (VeRoLog) and ORTEC (see [8]). DIM is an extension of CVRP that combines routing and scheduling aspects, which optimizes the distribution and installation of vending machines on specific days. There is a planning horizon spanning over a given number of days, different kinds of machines, and a set of technicians

[1] https://verolog2019.ortec.com/

[2] http://www.verolog.eu/

[3] https://verolog2019.sciencesconf.org

with different skills and different starting locations. Requests from customers contain details such as the first and the last day within which a machine must be delivered, the number of machines and type of machine. If a customer requires different kinds of machines, then a request exists for each kind. Machines have different sizes which must be taken into account when loaded into trucks since trucks have a limited capacity. Delivery of machines is performed by a fleet of identical trucks and each of them starts and ends the day in the depot. Trucks can return many times in depot during a day but they also have a limited maximum daily distance. After deliveries are made, installations must be performed as soon as possible starting from the following day, otherwise, there is a penalty for each day the machine installation is delayed. There is a set of technicians that have different skills, meaning they can only install specified kinds of machines. Each technician may have a different home location, where they start and end a route. They have a limited daily distance, a limited number of locations to visit and they have to take two days off after five consecutive days of work. For each kind of machine, there is a cost associated with each day between delivery and installation. The objective is to minimize the total cost which includes the weighted sum of these components: total distance travelled by trucks, number of total trucks, truck days (a day when a truck is used), total distance travelled by technicians, number of technicians, technicians' days, and waiting times of machines for installations. Distances between locations of the depot, customers, and technicians are calculated as the ceiling of the Euclidean Distance.

In DIM problem, we are given a set of requests $R = \{1, 2, ..., n\}$, and a set of days $D = \{1, 2, ..., d_{max}\}$. Each $r \in R$ has its attributes including location, first and last day between which machines must be delivered, type of machines, and the number of them. The set $T = \{1, 2, ..., t_{max}\}$ contains all the technicians where each of them has a home location and a flag for each kind of machine that shows if the technician is able to perform the installation of that kind of machine. Each location is given as a pair of coordinates (x, y), and the distance between any two locations is calculated with this formula $d = \lceil \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \rceil$

The aim is to design a set of vehicle routes $DR_{dv} = \{dr_1, dr_2, dr_3, ...\}$ for the deliveries, and a set of technician routes for the installations $IR_{dt} = \{ir_1, ir_2, ir_3, ...\}$ for each day $d \in D$, each vehicle, and each technician $t \in T$.

Each route $dr_i \in DR_{dv}$ starts from the depot, delivers the machines to the customers, and ends at the depot. Also, each route $ir_i \in IR_{dt}$ starts from the location of the technician t, performs the installation of the machines, and ends at the starting location. We are also given a weight parameter for each component of the cost, including: (1) vehicle distance, (2) vehicle days, (3) vehicles' number, (4) technicians' distance, (5) technician days and (6) number of technician employed. And at the end, there are the cost parameters for each kind of machine, for the number of days a machine waits for the installation (7). The goal is to minimize the total cost which is a sum of seven components mentioned above and fulfill all hard constraints which are: for each request, delivery must be done between the first and last day, and installation

can be performed starting from the following day of its delivery. A truck has a limited capacity of machines it can deliver during one trip, and maximum distance it can travel during a day. Technicians have a set of skills that specifies which kinds of machines they can install, a maximum distance they can travel during a day, and a maximum number of requests they can execute during a day. Technicians are only allowed to work 5 consecutive days at maximum, and if they do, then they are forced to take two days off.

3 General Description of our Approach

Our approach introduces several innovative ideas such as applying several new neighborhood operators that rely on min-conflicts, destroy-repair, and nearest neighbors heuristics for delivery and installation of machines (DIM) problem. The proposed components operate within the framework of Iterated Local Search (ILS) algorithm (algorithm 1).

ILS framework includes four main components: initial solution, embedded local search, acceptance criteria and perturbation. In this section we describe our approach in details.

3.1 Solution representation

Solution candidates are represented using a data structure that includes several entities: days, vehicles, vehicle trips and routes, delivery requests, technicians, technician routes and installation requests. Since the delivery of the machines is done separately from the installation of the machines, then we refer to them as delivery requests and installation requests.

These entities are organized in a hierarchical way so that they reflect the relationship between each of them. A solution contains a vector of days and each day has two lists: the vehicle routes and technician routes. The vehicle routes contain the list of delivery requests assigned to a vehicle, including the returns of the vehicle to the depot during the day. We use the concept of the trip which represents a part of the route that starts and ends at the depot, to allow the trips to be assigned to different vehicles or even different days. The technician routes contain the installation requests. This approach is known in the literature as 'cluster first-route second' paradigm, where first we create the groups of the request that are assigned to days and vehicles (cluster), and then every cluster is treated as a TSP problem. This hierarchical decomposition into subproblems was proposed by several authors such as [5], [4] and [19]. In the case of our problem, we can perform another decomposition by dividing the deliveries and installation schedules, because they can be treated as two dependent VRP problems.

Algorithm 1 ILS for DIM problem

```
s ← InitialSolution()
s'' ← s
gap ← 0
repeat
   s1 ← MoveTrip(s)  or
   s1 ← SwapTrips(s)
   Through roulette wheel set order of :
   s2 ← RemoveInsertDeliveriesStack(s1)
   s3 ← RemoveInsertInstallationsStack(s2)
   s4 ← RegionGroupingNN(s3)
   s5 ← LoopSwap(s4)
   s6 ← SwapInstallations(s5)
   s ← s6
   if (s < s'') then
      s'' ← s
      gap ← 0
   else
      gap ← gap + 1
      if (gap ≥ k) then
         gap ← 0
         Intensify search :
         s7 ← ReInsertDeliveriesSerial(s)
         s8 ← ReInsertInstallationsSerial(s7)
         s9 ← BestSwapWithNNDays(s8)
         s10 ← BestSwapWithNNDaysInst(s9)
         s ← s10
         if (s < s'') then
            s'' ← s
         end if
         Perturbate s :
         sp1 ← PerturbSwapTrip(s)
         sp2 ← PerturbReInsertDeliveriesStack(sp1)
         s ← sp2
      end if
   end if
until timeExpired
```

3.2 Initial solution

After the empty solution is populated with empty routes for each technician, we start the generation of an initial solution by adding the installations in the first feasible position in available technician routes in the given order of technicians and requests. If no feasible position is found for a request, then we perform backtracking by removing the previous requests and inserting the current one (this approach has been successful for all the instances we experimented with). In the next step, we add the deliveries in the same way, by adding each of them in the first available truck. The insertion of installations is more critical because of the hard constraints such as technician skills, technician days-off, a limited number of visits per day, and limited daily distance. This is not the case for deliveries, because the number of trucks is not limited.

It is worth noting that in this phase we only take care of the feasibility of the solution.

3.3 Local search and neighborhood structures

Local search is based on several neighborhood structures and is conducted in two stages. The first one is executed during every cycle of the local search. The second stage contains other operators and they are executed only after a specific number of cycles of local search don't find a better solution. The operators from the first stage are faster compared to those from the second stage, thus they are given more time as long as they improve the solution. After the first stage fails to find improvement for a certain number of cycles, then the algorithm calls the operators from the second stage.

3.3.1 First stage of local search

The first stage contains these operators:

- MoveTrip,
- SwapTrips,
- ReInsertDeliverStack and ReInsertInstallStack,
- RegionGroupingNN,
- LoopSwap,
- SwapInstallations.

The order of the execution of operators is determined in a probabilistic manner, i.e. in each iteration, a new order of operators is calculated based on probabilities that are determined by an algorithm configurator (we will give more information in the section about experimenting).

MoveTrip - moves a trip from one truck to another within a permitted time window. It uses the idea of min-conflicts heuristic ([13]) to generate the neighborhood in the way that all the trips from a day are treated as variables that are in conflict for a better allocation in one of the available trucks. This neighborhood operator is expected to improve the solution by decreasing the number of trucks or truck days and indirectly changing the configuration of the installations. It also may generate solutions with equal cost but a different structure.

SwapTrips - swaps two trips from different trucks. It also uses min-conflicts heuristic to generate the neighborhood by treating all the possible pairs of trips as variables and finding the best swapping pair. *SwapTrips* and *MoveTrip* are executed alternately, i.e. the former is executed in current loop and the latter in the next. These operators are very efficient when it comes to decreasing the number of vehicles because they treat the problem as a bin packing problem, where the vehicles and their trips are treated as bins and items respectively. In order to evaluate the role of these neighborhood operators, we ran our algorithm with and without operators *MoveTrip* and *SwapTrips*, with a set

of instances comprised of 15 instances from the early set (E01 to E15) and 9 from the late set (L01 to L09). The algorithm was run for each instance, five times in each mode for 500 seconds. We measured the relative improvement of the average costs for each instance. The results are depicted in figure 1. As the chart shows, when the algorithm uses *MoveTrip* and *SwapTrips*, it generates better solutions for most of the instances. For example, we have an improvement ratio of over 8% for the problem instance L09.

The pseudocode of an operator that uses min-conflicts heuristic is given in the algorithm 2. In this procedure, we have an outer loop that is executed as long as there are global improvements. Inside the outer loop, we first set the variable $BestLocalCost$ to infinity. Then we start a loop that performs the moves for as long as there is an improvement compared to $BestLocalCost$. At the end of the inner loop, we check if the resulting solution s is better than $IncumbentSolution$. If we have a global improvement we repeat the outer loop. As it can be noticed, the inner loop will always execute at least two times, because first, we compare the $Cost1$ against ∞. In this way, the first move may result in a worse solution, and in the next cycles only improving moves are accepted. The method *SelectAMove* selects a move depending on the operator, while the method *EvaluateMove(s, Move)* returns the cost of solution s if $Move$ is executed. In the case of *MoveTrip* and *SwapTrips* the method $SelectAMove$ selects a trip to move or a pair of trips to be swapped respectively.

Algorithm 2 Procedure Min-Conflicts

```
s ← CopyOf(IncumbentSolution)
repeat
    LocalImprovement ← false
    BestLocalCost ← ∞
    repeat
        Move ← SelectAMove(s)
        Cost1 ← EvaluateMove(s, Move)
        if Cost1 < BestLocalCost then
            ExecuteMove(s, Move)
            LocalImprovement ← true
            BestLocalCost ← Cost1
        else
            LocalImprovement ← false
        end if
    until LocalImprovement = false
    if s.Cost < IncumbentSolution.Cost then
        IncumbentSolution ← CopyOf(s)
        GlobalImprovement ← true
    else
        GlobalImprovement ← false
    end if
until GlobalImprovement = false
```

ReInsertDeliverStack and ReInsertInstallStack - use destroy and repair heuristic and are similar except for the fact that the former starts destroy-

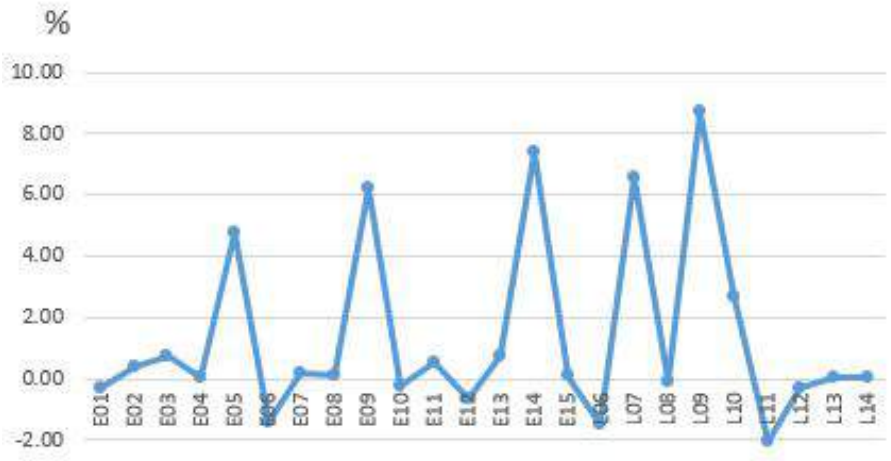


Fig. 1 The ratio between with and without *MoveTrip* and *SwapTrips*

repair on deliveries and the latter starts on installations. We implement several strategies for the destroy part and they are described later in this section. It is worth noting that when we select a request for removal, we remove both the delivery and installation from the schedule. Reinsertion is done in a greedy way for each pair of the delivery and installation, by finding the best positions in the entire feasible time window. This increases the search space because we now consider the best insertion for both the delivery and installation. We have experimented with the order of the insertion, but the procedure was very slow and it did not improve the results significantly. Insertion of deliveries and installations is committed in the order they are removed from the schedule.

RegionGroupingNN - performs removal and insertion of requests by selecting a list of Nearest Neighbors. After a set of requests are selected, they are grouped in a new trip that is assigned to a vehicle. This strategy has the potential to improve the solution because it removes from the schedule the requests that are assigned to different trips and regroups them in an optimized trip.

The three operators mentioned previously are based on the idea of different heuristics for destroy and repair, an approach that has been used by [20], [16] and [18]. This operator implements the min-conflicts heuristic in the same way as algorithm 2.

LoopSwap - performs a sequential swap of requests (nodes) noted as r_i and their positions $p_i, i \in \{1, ..., l\}$, where request r_1 is moved from position p_1 to p_2, r_2 to p_3, ..., r_l to p_1, creating a closed cycle of swapping. This operator affects only deliveries from a certain day. The idea of this operator comes from the classical move called Ejection Chain which was proposed by authors in [17]. The potential of this move relies on the fact that while an improving

single swap of two nodes becomes less likely to happen, increasing the number of swaps in a move increases the possibility to find improvements.

SwapInstallations - swaps two installation requests. The neighborhood is generated so that the first request is selected randomly and then the best swap with all possible installations is executed. In order to extend the selected neighborhood, we also remove the delivery part of the request. The procedure is executed as long as there are improvements and its main target is the subproblem regarding the installations and technicians.

3.3.2 Second stage of local search

The second stage of local search is executed after a parameterized (k) gap of iterations of the first stage without improvement and it uses these operators:

- *ReInsertDeliverSerial* and *ReInsertInstallSerial*
- *BestSwapWithNNDays* and *BestSwapWithNNDaysInst*

ReInsertDeliverSerial and ReInsertInstallSerial - take an input list of requests (delivery and installation) and then relocate each request, including the delivery and installation. This operator, in fact, performs a series of relocations, where the evaluation of the move is done only after all the requests are relocated. The idea behind this is to accept temporarily worse solutions in order to move to more promising search space regions.

BestSwapWithNNDays and BestSwapWithNNDaysInst - These two operators search for the best pair of requests to swap, considering the complete set of requests assigned on a certain day. This procedure is repeated for as long as it finds improvements. Since this generated neighborhood is relatively large, we put these operators in the second stage of local search, which is executed only after the first stage cannot make improvements for a given number of cycles.

Destroy - Remove Heuristics - As mentioned previously, several operators use a method that selects a set of requests for removal so that different heuristics for insertion are used afterward. The simplest way is to select a set of requests from a random day, a random truck, a random trip, or a random technician, depending on the number of requests. The idea behind this is to destroy a particular region of the solution. Another approach is to select randomly a request and then appending the request with the nearest location and with overlapping time windows. In this way, we select a set of requests that have a higher probability to be arranged in a single trip or assigned to a day or truck. Another way to select a set of requests is to find the requests whose partial cost in the total solution cost is highest. This can be done using a function $P(s, r) = E(s) - E'(s, r)$ that calculates the difference of current evaluation $E(s)$ of a solution s, and the cost when the request r is removed from the solution $E'(s, r)$. We refer to this heuristic as $RemoveWorst$.

3.4 Acceptance criteria

In our approach, we accept only better solutions. This logic is implemented inside the min-conflicts algorithm which is used by the mentioned operators.

3.5 Perturbation

Perturbation is implemented through two moves: *PerturbSwapTrip* and *PerturbReInsertDeliverStack*. They are the same operators as *SwapTrip* and *ReInsertDeliverStack* respectively, but modified to accept solution of a worse quality. Perturbation is executed with a parameterized (k) gap of iterations, i.e. for successive k iterations no perturbation takes place. We compensate sparse perturbations with the application of several perturbations in a row and selecting the best current solution as the incumbent solution for the next iteration.

3.6 Parameter tuning

Our algorithm contains a set of operators that are executed in a certain order, and this order appeared to be important. While performing experiments we noticed that for different instances, different orders of operators gave different results. For this reason, we used a method based on roulette wheel to determine the order of operators for each cycle, namely the operators: *ReInsertDeliverStack*, *ReInsertInstallStack*, *RegionGroupingNN*, *LoopSwap* and *SwapInstallations* (respectively $O_1, ..., O_5$). The roulette wheel system takes five parameters $p_i, i \in [1,5], \sum p_i = 1$ as input and produces an order of operators. Each parameter p_i is the probability that operator O_i is selected as j-th operator for $j \in [1,5]$. For example, if all the parameters have the same value, $p_i = 0.2, i \in [1,5]$, then the operators will have the same probability to be chosen at each step in the sequence, which means that all the possible permutations of sequences will have the same probability to be chosen. Or, if parameter $p_1 = 1, p_i = 0, i \in [2,5]$, then the operator O_1 will always be the first one in the sequence, and the four remaining operators will be chosen as the next in the sequence with the same probability. We used the SMAC tool ([9,12]) to determine the best configuration of parameters for our algorithm. The SMAC tool performs intensive experiments using one set of training instances, and one set of testing instances to find the optimal values for each parameter. We ran SMAC tool with these parameters:

- Total time limit was set to 2 days,
- Maximum runtime for instance: 500 seconds
- Training Instances: 8 instances chosen randomly from 'Early' and 'Hidden' sets
- Testing Instances: 27 instances chosen randomly from 'Early' and 'Hidden' sets
- The list of parameters to be tuned: $p_i, i \in [1,4]$, (since $p_5 = 1 - \sum_{i=1}^{4} p_i$)

– The possible values for parameters:

$$\{0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.7, 0.8\}$$

The execution of SMAC tool with this setup resulted in parameter settings as shown in table 1. From the table, we see that the operator O_4 (*SwapInstallations*) should be selected as the first parameter in sequence 40% of the time. Operators O_1, O_2, O_5 have equal probability to be selected, and that the O_3 will always be the last operator in sequence.

Table 1 Parameter setting found by SMAC

Parameter	Value
p_1	0.2
p_2	0.2
p_3	0
p_4	0.4
p_5	0.2

4 Computational results

We compared the results of our approach with the results submitted in the challenge website by all the participating teams. In the *all-time-best* challenge took part 13 teams who submitted their solutions for 25 instances. Eight teams, including our team, were selected for the final phase with the *restricted resources*. The organizers of the challenge had made available three sets with 25 instances each:

– Early set: 25 instances for *all-time-best* mode
– Late set: 25 instances used to train the algorithm for *restricted resources* mode
– Hidden set: 25 instances that were used in *restricted resources* mode. This set was made available after the finalists for the second stage were chosen and after the submission of solvers was done.

For both challenges, the participants were allowed to use external state of the art solvers such as: FICO [4], GUROBI [5] or CPLEX [6]

4.1 All-time-best challenge mode

At this stage of the challenge, all the participating teams were able to submit their solutions for each instance from the Early set, and the results were ranked

[4] `https://community.fico.com/s/academic-programs`
[5] `http://www.gurobi.com/academia/academia-center`
[6] `http://ibm.biz/AI_CPLEX128`

for each instance. Since at this stage there was no limitation regarding the run time, the participating teams only submitted their best solution. The challenge website [7] reported the best solutions from all the participating teams for each instance from the Early set. From the table 2 we can see the best result for each instance and the results of our approach. The column *Ratio* expresses the relative difference of the cost of our approach compared to the best result, and the *Rank* column shows the ranking of our algorithm (out of 13 teams) for each instance. We can see that we reached the second place for the first instance, the third place for 6 instances, the fourth place for 13 instances, the fifth place for 4 instances and sixth place for one instance. For 7 instances the ratio is less than 1 percent, which shows that the results of our solver are very close to the best results. A more detailed comparison of all-time-best

Table 2 Best solutions submitted by participants for All-Time-Best challenge

Instance	Best	Our Approach	Ratio	Rank
Early01	3,487,969,810	3,487,996,910	0.0008%	2
Early02	11,149,038,115	11,225,768,410	0.6882%	5
Early03	179,700,885	184,117,395	2.4577%	3
Early04	284,205,965	289,436,220	1.8403%	3
Early05	2,223,814,105	2,479,106,500	11.4799%	4
Early06	24,160,989,040	24,732,913,745	2.3671%	3
Early07	45,815,700	45,853,150	0.0817%	3
Early08	109,798,470	110,016,310	0.1984%	3
Early09	18,075,485	19,762,065	9.3308%	4
Early10	18,500,638,020	18,761,510,110	1.4101%	5
Early11	28,549,460	31,781,875	11.3222%	4
Early12	23,933,097,895	24,447,956,620	2.1512%	5
Early13	582,708,670	584,976,550	0.3892%	4
Early14	94,780,375	99,699,795	5.1903%	4
Early15	1,772,831,110	1,784,630,020	0.6655%	5
Early16	3,287,392,325	3,446,044,045	4.8261%	4
Early17	3,018,108,020	3,035,277,510	0.5689%	6
Early18	5,129,752,375	5,405,790,030	5.3811%	4
Early19	9,290,203	9,457,608	1.8020%	3
Early20	4,764,640	5,296,100	11.1543%	4
Early21	1,292,914,150	1,360,412,960	5.2207%	4
Early22	203,485,635	217,759,862	7.0149%	4
Early23	55,207,660	58,177,435	5.3793%	4
Early24	17,337,730	17,960,220	3.5904%	4
Early25	66,769,325	71,241,680	6.6982%	4

VeRoLog Solver Challenge top solvers results is given in table 3. For each solver/team an each instance the relative gap to to the best is presented. The first column represents the instance, the second shows the number of requests, and the rest of columns show the results of five best teams including our team (last column: AAVK). It can be seen that difference of results for most of the instances among solvers is very narrow.

[7] `https://verolog2019.ortec.com/`

Table 3 Comparison of All-time-best VeRoLog Solver Challenge top solvers results

Inst.	n	MJG	UOS	TCS	COKA	AAVK
E01	150	0.0000	0.0008	0.0009	0.0096	0.0008
E02	300	0.0000	0.1086	0.4372	0.5217	0.6882
E03	450	0.0000	0.3570	4.2765	3.7151	2.4577
E04	600	0.0000	0.7426	2.4175	3.8037	1.8403
E05	750	0.0000	1.0570	7.2300	20.2542	11.4799
E06	900	0.0000	0.6966	3.6447	6.9432	2.3671
E07	150	0.0000	0.0159	0.5373	2.4136	0.0817
E08	300	0.0000	0.0083	0.3387	9.8424	0.1984
E09	450	0.0000	0.7916	5.1190	22.8997	9.3308
E10	600	0.0000	0.0822	1.7774	0.2722	1.4101
E11	750	0.0000	1.7460	3.0933	76.2355	11.3222
E12	900	0.0000	0.8513	2.3508	1.4309	2.1512
E13	150	0.0000	0.0410	0.7328	0.3682	0.3892
E14	300	0.0000	0.4298	4.6534	8.7505	5.1903
E15	450	0.0000	0.0704	0.1740	2.9431	0.6655
E16	600	0.0000	0.8654	2.5741	37.7519	4.8261
E17	750	0.0000	0.1128	0.3475	0.2998	0.5689
E18	900	0.0000	1.0690	3.3212	9.5738	5.3811
E19	150	0.0000	0.2566	1.9767	42.7723	1.8020
E20	300	0.7238	0.0000	11.0304	111.3894	11.1543
E21	450	0.0000	0.7025	3.3241	51.1948	5.2207
E22	600	0.0000	0.6228	3.1012	23.3063	7.0149
E23	750	0.0000	1.9225	2.1815	17.7000	5.3793
E24	900	0.0000	2.2040	2.3669	45.7606	3.5904
E25	150	0.0000	0.4744	4.9185	7.0527	6.6982

4.2 Restricted resources challenge

The late set of 25 instances was used for the second stage of the challenge. Participants had to submit the algorithm and the solution for each instance from this set. Then the organizers ran the solver for these instances in their servers and selected a limited number of best performing teams. Solvers from this short list then were used with the hidden set of instances to further determine the complete ranking of submitted solvers. From 9 runs, the best two and worst two solutions were dropped, and the remaining 5 were used to determine the average and standard deviation. Table 4 shows our results for each instance, including the best cost, average cost, and standard deviation.

For each team, the organizers calculated the rank per instance, and then the final ranking was determined as average of all ranks. The table 5 shows the results of the winning team, and the relative results from fourth, fifth and our solver (teams: orlab, TCS and AAVK). From this table we can see that our solver provided better results than fourth place for seven problem instances.

Table 4 The results of our approach for the hidden set in the restricted resources challenge

Instance	Best	Average	StDev
H01	68,049,230	68,497,477	502,829
H02	1,470,177,245	1,654,579,976	164,820,353
H03	1,388,162,220	1,394,947,955	5,454,085
H04	6,053,945	6,122,060	42,006
H05	4,687,014,190	4,946,153,397	207,877,012
H06	33,537,475	33,642,695	62,003
H07	128,584,875	138,686,310	6,575,414
H08	743,077,790	750,319,371	4,230,922
H09	2,925,297,281	3,087,641,280	120,454,552
H10	47,813,445	49,105,137	848,275
H11	5,050,880,606	5,176,963,402	98,658,731
H12	3,082,704,895	3,095,575,436	8,427,309
H13	5,341,187,251	5,368,906,565	16,223,058
H14	1,394,884,865	1,403,511,603	5,094,711
H15	166,559,140	167,109,473	410,510
H16	61,626,085	63,624,954	1,218,645
H17	27,870,689,964	27,950,302,615	51,793,696
H18	54,778,115	55,313,427	290,138
H19	4,425,251,625	4,450,177,592	20,478,291
H20	208,729,450	231,061,473	14,678,813
H21	38,119,685	38,845,432	477,952
H22	7,473,115	7,575,488	96,316
H23	22,946,775,355	23,021,877,311	48,690,845
H24	32,129,246,375	32,216,184,262	74,165,994
H25	817,352,060	988,807,257	118,314,588

Table 5 Comparison of results in restricted resources challenge

Inst.	UOS	orlab	TCS	AAVK
H01	67,347,510	1.757	1.963	1.708
H02	884,328,838	46.944	9.308	87.100
H03	1,356,206,683	1.245	5.431	2.857
H04	5,470,823	17.165	9.190	11.904
H05	2,438,943,861	46.875	18.939	102.799
H06	33,122,942	0.565	2.041	1.569
H07	102,289,614	29.817	7.889	35.582
H08	729,462,821	1.043	7.975	2.859
H09	1,713,909,634	50.785	18.646	80.152
H10	31,615,173	36.361	14.808	55.321
H11	4,143,464,703	47.843	10.383	24.943
H12	2,988,919,325	1.398	6.547	3.568
H13	5,239,090,235	1.164	3.510	2.478
H14	1,380,531,069	0.843	4.536	1.665
H15	163,861,015	1.025	3.069	1.982
H16	53,561,471	33.553	10.688	18.789
H17	27,322,051,463	0.744	3.341	2.299
H18	53,040,623	7.058	3.951	4.285
H19	4,379,503,211	0.402	2.368	1.614
H20	127,117,759	31.932	11.306	81.770
H21	33,217,241	42.250	8.522	16.944
H22	6,750,354	16.978	3.996	12.224
H23	22,368,503,381	1.233	5.690	2.921
H24	31,373,781,566	0.577	4.962	2.685
H25	549,854,756	23.468	13.028	79.831

5 Conclusions and Future Work

In this work, we presented a new approach for Vehicle Routing and Scheduling with Deliveries and Installation of Machines. Our algorithm includes min-conflicts heuristic, nearest neighbors, and destroy and repair heuristics in the iterated local search framework. Several neighborhood operators are evaluated and additionally, we introduced two new neighborhood operators. The experiments showed that the use of new neighborhoods improved the results for most instances. Overall our approach has been able to provide good solutions for a complex problem that includes scheduling and routing.

We participated in the VeRoLog Solver Challenge 2019 where organizers introduced three sets of problem instances for two different experimental settings: all-time-best and restricted resources challenge. Thirteen teams participated in all-time-best challenge, and eight of them were qualified for the restricted resources challenge, including our team. The approach we proposed provided promising results for instances that were used in the competition. In the future, we plan to extend our metaheuristic approach and investigate its hybridization with exact methods.

References

1. Braekers, K., Ramaekers, K., Van Nieuwenhuyse, I.: The vehicle routing problem: State of the art classification and review. Computers & Industrial Engineering **99**, 300–313 (2016)
2. Clarke, G., Wright, J.W.: Scheduling of vehicles from a central depot to a number of delivery points. Operations research **12**(4), 568–581 (1964)
3. Cordeau, J.F., Laporte, G., Mercier, A.: A unified tabu search heuristic for vehicle routing problems with time windows. Journal of the Operational research society **52**(8), 928–936 (2001)
4. De Franceschi, R., Fischetti, M., Toth, P.: A new ilp-based refinement heuristic for vehicle routing problems. Mathematical Programming **105**(2-3), 471–499 (2006)
5. Fisher, M.L., Jaikumar, R.: A generalized assignment heuristic for vehicle routing. Networks **11**(2), 109–124 (1981)
6. Geiger, M.J.: A contribution to the verolog solver challenge 2019 (2019). URL `https://verolog2019.sciencesconf.org/244024`. Workshop of the EURO Working Group on Vehicle Routing and Logistics optimization (VeRoLog)
7. Graf, B.: An adaptive large variable neighborhood search for a combined vehicle routing and scheduling problem (2019). URL `https://verolog2019.sciencesconf.org/249822`. Workshop of the EURO Working Group on Vehicle Routing and Logistics optimization (VeRoLog)
8. Gromicho, J., van't Hof, P., Vigo, D.: The verolog solver challenge 2019. Journal on Vehicle Routing Algorithms **2**(1-4), 109–111 (2019)
9. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: International conference on learning and intelligent optimization, pp. 507–523. Springer (2011)
10. Jagtenberg, C., Raith, A., Sundvick, M., Shen, K., Maclaren, O., Mason, A.: Matheuristics for the 2019 verolog solver challenge: Mips and bits (2019). URL `https://verolog2019.sciencesconf.org/250600`. Workshop of the EURO Working Group on Vehicle Routing and Logistics optimization (VeRoLog)
11. Lin, S., Kernighan, B.W.: An effective heuristic algorithm for the traveling-salesman problem. Operations research **21**(2), 498–516 (1973)

12. Lindauer, M., Eggensperger, K., Feurer, M., Falkner, S., Biedenkapp, A., Hutter, F.: Smac v3: Algorithm configuration in python. 2017 (2017)
13. Minton, S., Johnston, M.D., Philips, A.B., Laird, P.: Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. Artif. Intell. **58**(1-3), 161–205 (1992). DOI 10.1016/0004-3702(92)90007-K. URL `https://doi.org/10.1016/0004-3702(92)90007-K`
14. Or, I.: Traveling salesman type combinatorial problems and their relation to the logistics of regional blood banking. (1977)
15. Osman, I.H.: Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. Annals of operations research **41**(4), 421–451 (1993)
16. Pisinger, D., Ropke, S.: A general heuristic for vehicle routing problems. Computers & operations research **34**(8), 2403–2435 (2007)
17. Rego, C., Roucairol, C.: A parallel tabu search algorithm using ejection chains for the vehicle routing problem. In: Meta-Heuristics, pp. 661–675. Springer (1996)
18. Ropke, S., Pisinger, D.: An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation science **40**(4), 455–472 (2006)
19. Salari, M., Toth, P., Tramontani, A.: An ilp improvement procedure for the open vehicle routing problem. Computers & Operations Research **37**(12), 2106–2120 (2010)
20. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: International conference on principles and practice of constraint programming, pp. 417–431. Springer (1998)
21. Solomon, M.M.: Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations research **35**(2), 254–265 (1987)

Effective Pruning Heuristics for the Fixed Route Dial-a-Ride Problem

Tal Grinshpoun · Elad Shufan · Hagai Ilani · Vadim Levit · Haya Brama

Abstract The fixed route dial-a-ride problem (FRDARP) is a variant of the famous dial-a-ride problem, in which all the requests are chosen between terminals that are located along a fixed route. A reduction to the shortest path problem has been proposed for finding an FRDARP optimal solution. However, the basic graph construction ends up with a huge graph, which makes the reduction impractical due to its memory consumption. To this end, we propose several pruning heuristics that enable us to considerably reduce the size of the graph through its dynamic construction. Our experiments show that each of the proposed heuristics on its own improves the practical solvability of FRDARP. Moreover, using them together is considerably more efficient than any single heuristic.

Keywords Timetabling in Transport · DARP · Pruning Heuristics

Tal Grinshpoun
Department of Industrial Engineering and Management, Ariel University, Ariel, Israel
E-mail: talgr@ariel.ac.il

Elad Shufan
Physics Department, SCE – Shamoon College of Engineering, Ashdod, Israel
E-mail: elads@sce.ac.il

Hagai Ilani
Department of Industrial Engineering and Management, SCE – Shamoon College of Engineering, Ashdod, Israel
E-mail: hagai@sce.ac.il

Vadim Levit
Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel
E-mail: vadim@bgu.ac.il

Haya Brama
Department of Industrial Engineering and Management, Ariel University, Ariel, Israel
E-mail: hayahartuv@gmail.com

1 Introduction

Dial-a ride is a flexible demand responsive transportation solution, in which a fleet of vehicles fulfills a set of transport requests [9,2]. The flexibility is usually in both the adjustable departure and pick-up times, as well as in the vehicle routes. A usual customer request in all the variants of the Dial-a-Ride Problem (DARP) contains a pick-up location, a destination station, and a desired pick-up or arrival time. The scheduler groups customers into specific vehicles, and determines the vehicle routes and timetable. The objective function is usually related to minimizing the solution cost, and may also include the aim to increase customer satisfaction. The problem of route determination in DARP makes the problem hard. Vehicle routing problems, for example the traveling salesman problem, are known to be NP-hard [5].

In a previous work, we have presented a DARP transportation solution that neutralizes the difficulty in finding the vehicle routes by determining the route in advance [7]. The customers can be picked-up from stations that are located along a known route. We termed this DARP variant a Fixed Route DARP (FRDARP). With a fixed route, the remaining problem is of grouping customers together and scheduling the timetable. The DARP solutions, and in particular the solution we offer, are suitable for a variety of transportation needs, including dedicated solutions for low-populated areas, and also for customers with special needs, such as children or elderly [2,4].

In the FRDARP problem, each customer requests to be transported between two terminals along the fixed route, at requested times. There are two types of requests, called s-type and r-type. S-type requests have a deadline. An s-type customer who requested to leave at a certain time cannot leave later than that time, but could leave earlier. In the latter case, the customer will reach the destination sooner than ideally wanted. R-type requests have a release time. An r-type customer who requested to leave at a certain time cannot leave before the requested time, but can leave at a later time. The (positive) difference between the requested time and the real departure time is called the waiting time of the passenger. Naturally, the passengers expect it to be as minimal as possible. The aim in the presented FRDARP is to minimize the sum of all the waiting times, for a given operational cost. The cost is determined by the number of times a vehicle leaves the depot for its round trip. The scheduler receives as input the requests, the vehicles' working hours, and the number of round transports. She is then expected to find a schedule that will increase customer satisfaction by minimizing the total waiting time.

In [7] we have presented a polynomial algorithm to solve the FRDARP by a reduction to the shortest path problem [8]. 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. In Section 2 we fully describe the FRDARP and the reduction.

Though the presented method for solving the FRDARP is polynomial, its implementation involves construction of huge graphs. As elaborated in Section 3, this makes it difficult to find an optimal solution when the number

of requests becomes large. The main contribution of this work is in obtaining heuristic algorithms for improving the implementation of the solution method. Using these heuristics considerably reduces the graph size, without compromising the optimality of the solution. Improvement is done by setting pruning rules that avoid creating duplicate representations of transports, or pruning rules that do not develop an arc, when it is clear that it will not participate in an optimal path. In Section 4 we present the pruning heuristics. Section 5 is dedicated to experiments showing the improvement that is obtained by using these heuristics.

2 FRDARP: the model and its relation to the shortest path problem

2.1 Problem input

The problem input consists of:

- A vehicle fleet of size M. Vehicle m has capacity C_m ($m = 1, 2, \ldots, M$). It starts working at time a_m and finishes working at time b_m.
- A circular route with L terminals, including the depot. In each round, which is also termed transport, a vehicle starts at a depot A_0, goes through terminals $A_1, A_2, \ldots, A_{L-1}$ in this order, and finally returns back to the depot, which is also denoted A_L. The ride time from A_i to A_j is denoted D_{ij}. The total transport time is $D = D_{0L}$.
- A declared number of transports, which is denoted K. The cost of a transport is assumed to be vehicle-independent.
- Ride requests. A request is determined by its type, either s-type or r-type, a pick-up terminal A_i, a destination terminal A_j, and a desired pick-up time, which is denoted s^{ij} or r^{ij}, for s-type and r-type requests, respectively. The total number of requests is called N.

The ride requests are collected into sets. The set of all the s-type requests from terminal A_i to terminal A_j is denoted S^{ij}. The size of the set is denoted $N_S^{ij} = |S^{ij}|$. An element $s^{ij} \in S^{ij}$ has a value, which is the desired pick-up time of this request. We also define a normalized value $\bar{s}^{ij} = s^{ij} - D_{0i}$; in order to pick a customer from A_i at time s^{ij} the vehicle should depart from the depot at time $\bar{s}^{ij}$. The normalized values form the set $\bar{S}^{ij}$. Note that the elements in the described sets are the requests; two different requests are considered as two different elements, even when they have the same value. The elements of each set are indexed according to their value in a non-increasing order, e.g., if $S^{ij} = \{s_1^{ij}, s_2^{ij}, \ldots, s_{N_S^{ij}}^{ij}\}$ then $s_1^{ij} \le s_2^{ij} \le \cdots \le s_{N_S^{ij}}^{ij}$. The letter l is usually used for the index of an s-type request: s_l^{ij}. We similarly define R^{ij}, $\bar{R}^{ij}$, N_R^{ij}, and the indexed elements r_h^{ij} and $\bar{r}_h^{ij}$, with $h = 1, 2, \ldots, N_R^{ij}$. Finally, we define the sets $R = \cup_{i,j} R^{ij}$, $S = \cup_{i,j} S^{ij}$, and $Q = S \cup R$.

2.2 A feasible solution for FRDARP

An FRDARP solution $\mathcal{S}$ is a triplet $(\mathcal{P}, \mathcal{V}, \mathcal{T})$, where:

- $\mathcal{P} = (P_1, P_2, \ldots, P_K)$ is a partition of the set of Q into K disjoint sets. The partition part P_i represents the requests that are fulfilled by the ith transport. For simplicity, a partition part is also called a transport.
- $\mathcal{V} = (v_1, v_2, \ldots, v_K)$ is a list of vehicle indices. Each vehicle is numbered from 1 to M and, according to $\mathcal{V}$, the ith transport is operated by vehicle number v_i.
- $\mathcal{T} = (t_1, t_2, \ldots, t_K)$ is a list of departure times. The ith transport departs from A_0 at t_i.

A solution $\mathcal{S} = (\mathcal{P}, \mathcal{V}, \mathcal{T})$ is feasible if it satisfies the following capacity, partitioning, and time constraints:

1. Capacity constraint: The number of passengers on vehicle v_i that operates transport P_i, at any part of the ride, cannot exceed C_{v_i}, the vehicle's capacity. Note that $|P_i|$, the total number of passengers that share a transport, may be larger than C_{v_i} because passengers are using only a part of the circular route.
2. SR constraint: A request $\bar{s}^{ij}$ can share a ride with a request $\bar{r}^{i'j'}$ only if $\bar{r}^{i'j'} \leq \bar{s}^{ij}$. This leads to the following partitioning constraint, for a mixed transport, i.e., one that contains both s-type and r-type requests. Consider a mixed transport P_i. Denote the latest normalized r-type request in P_i by $r_{\texttt{last}} = \max_{\bar{r} \in P_i}\{\bar{r}\}$, and the earliest s-type normalized request by $s_{\texttt{first}} = \min_{\bar{s} \in P_i}\{\bar{s}\}$. The transport is feasible only if $r_{\texttt{last}} \leq s_{\texttt{first}}$.
3. Time constraints: The departure time t_i of a feasible transport P_i must satisfy $r_{\texttt{last}} \leq t_i \leq s_{\texttt{first}}$. An additional time constraint regards the working hours of the vehicle, i.e., $a_{v_i} \leq t_i \leq b_{v_i} - D$. Another time constraint regards two transports that are operated by the same vehicle. If $v_i = v_j$ (with $i \neq j$), then the respective departure times must satisfy $|t_i - t_j| \geq D$.

In the next section we define the objective function and discuss properties of an optimal solution. We close this section by an example.

Example 1 Consider an FRDARP problem with $M = 2$ vehicles of capacity $C_1 = C_2 = 2$, which are available from $a_1 = a_2 = \texttt{8:00}$ until $b_1 = b_2 = \texttt{16:00}$. The route consists of $L = 3$ terminals (including the depot), $A_0 \rightarrow A_1 \rightarrow A_2 \rightarrow A_3$, and takes 90 minutes to complete. The number of transports is set to $K = 3$. The normalized requests are shown in Table 1.

$\bar{s}_1^{01}$	$\bar{s}_1^{02}$	$\bar{s}_1^{13}$	$\bar{r}_1^{01}$	$\bar{r}_1^{12}$	$\bar{s}_1^{23}$	$\bar{r}_1^{23}$	$\bar{s}_2^{13}$	$\bar{r}_1^{13}$	$\bar{r}_2^{23}$
8:45	9:45	10:15	10:15	10:30	10:45	12:00	12:45	12:45	13:15

Table 1 Ride requests normalized to the respective departure times from the depot

The transport $\{s_1^{01}, s_1^{02}, s_1^{13}\}$ is an example of a feasible transport. Though the capacity of the vehicle is two, it can accommodate these three requests,

because s_1^{01} frees her seat at terminal A_1, which allows s_1^{13} to join the ride. This transport should leave the depot not later than `8:45` in order to meet the deadline requirement of $s_{\texttt{first}} = \bar{s}_1^{01}$. The transport $\{s_1^{13}, s_1^{23}, s_2^{13}\}$ is not feasible, as it violates the capacity constraint. The transport $\{s_1^{13}, r_1^{01}, r_1^{12}\}$ is an example of a feasible mixed transport that must depart exactly at `10:15`. The transport $\{s_1^{23}, r_1^{23}\}$ is an example of a non-feasible transport, because it violates the SR constraint. An example of a feasible solution is $\mathcal{S} = (\mathcal{P}, \mathcal{V}, \mathcal{T})$, with $\mathcal{P} = (\{s_1^{01}, s_1^{02}, s_1^{13}, s_1^{23}\}, \{r_1^{01}, r_1^{12}, r_1^{23}, s_2^{13}\}, \{r_1^{13}, r_2^{23}\})$, $\mathcal{V} = \{1, 2, 1\}$, and $\mathcal{T} = \{\texttt{8:45}, \texttt{12:00}, \texttt{13:15}\}$.

2.3 The objective function and optimal solution properties

A given solution defines for each passenger, either s^{ij} or r^{ij}, an actual departure time, denoted $\texttt{adt}(q^{ij})$, where q may be either s or r. The waiting time of a request is the difference between the desired and the actual departure times: $w(q^{ij}) = |q^{ij} - \texttt{adt}(q^{ij})|$. Our aim is to minimize the sum over the waiting times of all the passengers, i.e., the objective function is $W = \sum_{q \in Q} w(q)$.

Definition 1 A solution is called S-ordered if for any two terminals A_i and A_j, and any two requests $s_l^{ij}, s_{l'}^{ij} \in S^{ij}$ between these two terminals, $s_l^{ij} < s_{l'}^{ij} \Rightarrow \texttt{adt}(s_l^{ij}) \leq \texttt{adt}(s_{l'}^{ij})$.

We similarly define an R-ordered solution.

Definition 2 Given an FRDARP problem, we define *the set of optional departure times* (ODT) as a set whose elements are the following values:

1. $a_m + nD$, with $m = 1, 2, \ldots, M$.
2. $\bar{s}_l^{ij} \pm nD$, with $0 \leq i < j \leq L$, $1 \leq l \leq N_S^{ij}$.
3. $\bar{r}_h^{ij} \pm nD$, with $0 \leq i < j \leq L$, $1 \leq h \leq N_R^{ij}$.
4. $b_m - (n+1)D$, with $m = 1, 2, \ldots, M$.

In all of the above $n = 0, 1, \ldots, (K-1)$. In addition, the value in ODT must fit the working hours of the vehicles, i.e., they have to be larger than $\min_{1 \leq m \leq M}\{a_m\}$, and smaller than $\max_{1 \leq m \leq M}\{b_m\} - D$. In case one of the above values is not, it is deleted from the set.

In [7] we have shown that if FRDARP has a feasible solution, then it is always possible to find an optimal solution with the following properties:

1. It is both S-ordered and R-ordered.
2. For every $t \in \mathcal{T}$, $t \in ODT$, i.e, every vehicle leaves the depot in a departure time, which is an element of ODT.

The solution presented in Example 1 of Section 2.2 is an optimal solution with these two properties. We have reached it by a reduction of the FRDARP to the shortest path problem. The reduction procedure, which is based on these two properties, is described in the following section.

2.4 Reduction of FRDARP to a shortest path problem

In this section we show how to solve the FRDARP by a reduction to the problem of finding a shortest path in a directed and weighted graph G. The reduction has been previously introduced [7], and is inspired by a similar approach that we applied to an FRDARP with two terminals [8]. The graph is constructed dynamically, starting from a source node. Two nodes that are connected by an arc correspond to a possible transport. A path from the source node to one of several goal nodes corresponds to a feasible schedule of the respective FRDARP. The path that has the lowest total weight corresponds to an optimal schedule. The algorithm is polynomial in the number of requests, for a given number of terminals, vehicles and transports.

A node in G has $(L-1)(L+2)+M$ coordinates. The first $(L-1)(L+2)$ coordinates are indices, which are related to the $(L-1)(L+2)$ sets: R^{ij} and S^{ij}, where $0 \leq i < j \leq L$. The other M coordinates are related to the M vehicles. A node is a sequence $(\mathbf{h}^\frown\mathbf{l};\boldsymbol{\tau})$. The symbol $^\frown$ is used as a concatenation sign. $\mathbf{h} = (h^{ij})$ is a vector of indices that regard to r-type requests. For example, if $L = 3$ then $\mathbf{h} = (h^{01}, h^{02}, h^{12}, h^{13}, h^{23})$. Similarly, $\mathbf{l} = (l^{ij})$ is vector of indices that regard to the s-type requests. Finally, $\boldsymbol{\tau} = (\tau_1, \tau_2, \ldots, \tau_M)$ is vector of the availability times of the vehicles. A node $(\mathbf{h}^\frown\mathbf{l};\boldsymbol{\tau})$ indicates that all requests $\{r_1^{ij}, r_2^{ij}, \ldots, r_{h^{ij}}^{ij}\}$ and $\{s_1^{ij}, s_2^{ij}, \ldots, s_{l^{ij}}^{ij}\}$, for the respective i and j, have been handled, and that the vehicle m $(m = 1, 2, \ldots, M)$ is available for the next transport at time τ_m.

An arc connecting $(\mathbf{h}^\frown\mathbf{l};\boldsymbol{\tau})$ and $((\mathbf{h}+\boldsymbol{\Delta}\mathbf{h})^\frown(\mathbf{l}+\boldsymbol{\Delta}\mathbf{l});\boldsymbol{\tau}')$ represents a possible transport shared by $\{r_{h^{ij}+1}^{ij}, \ldots, r_{h^{ij}+\Delta h^{ij}}^{ij}\}$ and $\{s_{l^{ij}+1}^{ij}, \ldots, s_{l^{ij}+\Delta l^{ij}}^{ij}\}$, for all relevant pairs of i and j. The vector $\boldsymbol{\tau}'$ is different from $\boldsymbol{\tau}$ by only one component, the one that represents the available time of the vehicle, which operates this transport. The weight of the arc is its cost, i.e., the total waiting time of the passengers in the transport.

In order to describe the structure and the construction of the graph, we return to the FRDARP problem of Example 1. The source node in the constructed graph is $\texttt{node}_0 = (0,0,0,0,0,0,0,0,0,0;\texttt{8:00},\texttt{8:00})$, which means that none of the requests has been handled yet, and that both vehicles are available at `8:00`. Note that the first five (zero) coordinates are related to r-type requests, for the following ordered pairs of terminals: (01,02,12,13,23). This order is kept also for the other five coordinates, which are related to s-type requests. A goal node is of the form $(1,0,1,1,2,1,1,0,2,1;\tau^*,\tau^{**})$, where τ^* and τ^{**} might have any valid value smaller than b_1 and b_2, respectively. Arcs and nodes are created, starting from the source node, by two steps: (1) Choose a vehicle, create all the feasible transports, and represent each by the index coordinates; (2) Consider all relevant departure times and represent each possibility by a weighted arc leading to a node (with updated available time for the operating vehicle). For example, let us choose vehicle v_1 as a first operating vehicle, and consider a transport handling $\{s_1^{01}, s_1^{02}, s_1^{13}, s_1^{23}\}$. The representative 10 index coordinates are $(0,0,0,0,0,1,1,0,1,1)$. This transport should leave the depot between `8:00` to `8:45`. If there are more s-type requests

in the transport (as in this transport), than there is a dilemma. On one hand, the weight of the arc will be minimum if the transport will leave at the latest possible time. On the other hand, leaving earlier might be beneficial for the next considered transports. So, in the case of a majority of s-type requests, we need to generate nodes with all possible departure times from the ODT set, in the relevant time slot, which is, in the example, between `8:00` and `8:45`. When the vehicle departs at the latter time, as an example, it will complete its 90 minute route at `10:15`. Passengers s_1^{02}, s_1^{13} and s_1^{23} arrive at their destinations 60, 90 and 120 minutes ahead of time, respectively, thus their accumulated waiting time is 270 minutes. The node in the graph that represents the state after handling this transport is $\texttt{node}_1 = (0, 0, 0, 0, 0, 1, 1, 0, 1, 1; \texttt{10:15}, \texttt{8:00})$ and the cost of the arc $\texttt{node}_0 \rightarrow \texttt{node}_1$ is 270.

The described generating process (steps 1 and 2 above) continues from each node, which is connected to the source node by less than K arcs. When the graph is completed, a path of minimal weighted length from the source node to any goal node, using at most K arcs, represents an optimal schedule.

We continue the example by considering a possible second transport that accommodates requests r_1^{01}, r_1^{12}, r_1^{23} and s_2^{13} using vehicle v_2. This transport should leave the depot between `12:00` and `12:45` in order to meet the respective release and deadline requirements of $r_{\texttt{last}} = r_1^{23}$ and $s_{\texttt{first}} = s_2^{13}$. In this example, there are more r-type than s-type requests. When this is the case, there is no dilemma regarding the departure time of the vehicle. It should depart as early as possible, i.e., at `12:00` and finish at `13:30`. Passengers r_1^{01} and r_1^{12} will wait at their source terminals for 105 and 90 minutes, respectively, whereas s_2^{13} will arrive at her destination 45 minutes ahead of time. The node that represents the state after handling this ride is $\texttt{node}_2 = (1, 0, 1, 0, 1, 1, 1, 0, 2, 1; \texttt{10:15}, \texttt{13:30})$ and the cost of the arc $\texttt{node}_1 \rightarrow \texttt{node}_2$ is 240.

The careful reader might have noticed that the above described transports belong to the optimal schedule, which was presented in Section 2.2. The third transport accommodates the remaining requests r_1^{13} and r_2^{23} using vehicle v_1. It will depart from the depot at `13:15` (as early as possible). The node that represents the state after handling this ride is a goal node $\texttt{node}_3 = (1, 0, 1, 1, 2, 1, 1, 0, 2, 1; \texttt{14:45}, \texttt{13:30})$ and the cost of the arc $\texttt{node}_2 \rightarrow \texttt{node}_3$ is 30. The presented path $\texttt{node}_0 \rightarrow \texttt{node}_1 \rightarrow \texttt{node}_2 \rightarrow \texttt{node}_3$ is optimal for the FRDARP problem of Example 1 and represents a solution of cost 540.

3 Graph traversal

The original idea for solving FRDARP was to use the constructed graph of Section 2.4, and to run on it K iterations of the Bellman-Ford algorithm [1] in order to find the shortest path [7]. The Bellman-Ford algorithm traverses the graph similarly to *breadth-first search* (BFS) [3], and needs to maintain all the graph nodes in memory at all times.

We have recently attempted to implement the above solution method for a new transportation solution for the elderly that is based on FRDARP. We have found out that elderly people living in nursing facilities have transportation needs of rather limited dispersity, e.g., medical clinic, community center, and supermarket [4]. Therefore, a transportation solution with a flexible schedule and a circular route going through these places, starting and ending in the nursing facility (depot), may very well fit their needs. However, even though the FRDARPs to be solved were of rather limited size ($N \leq 20$ daily requests), the resulting graphs (on which the Bellman-Ford algorithm should run) turned out to be very large and the solver ran out of memory.

To this end, we propose to traverse the graph in *depth-first search* (DFS) [3] manner by using a classical Branch & Bound procedure. A major advantage of DFS traversal lies in its memory usage – in DFS there is no need to maintain the graph nodes in memory. Moreover, the Branch & Bound procedure reaches very quickly a first valid solution (not necessarily optimal) and can consequently use this solution as an initial bound. In order to understand the merits of holding such a bound, let us, for the sake of clarity, term a path consisting of less than K arcs that has not reached a goal node, as a *partial* path. Now, every time the cost of a partial path reaches the bound, the remainder of this path can be pruned. The bound is updated every time a new better solution is discovered.

The above basic pruning of the Branch & Bound procedure helps in some extent to reduce the search space, and serves as a good first step for reducing the actual graph size. In the next section, we propose several domain-specific heuristic that considerably enhance the effectiveness of pruning.

4 Pruning heuristics

We propose herein various heuristics for pruning the FRDARP search space. We separate the heuristics into two types according to their main motif of pruning – redundancy-removal heuristics and path-removal heuristics. Broadly speaking, the redundancy-removal heuristics (Sections 4.1 and 4.2) attempt to a priori identify multiple paths in the graph that ultimately lead to equivalent solutions, and consequently remove redundant paths. On the other hand, the path-removal heuristics (Sections 4.3 and 4.4) aim to a priori identify paths that lead to either infeasible solutions or solutions that are not optimal, and consequently completely remove them.

4.1 SR redundancy

Consider $\texttt{node}_{SR}$ that represents a state in which there are both s-type and r-type unhandled requests, and the earliest normalized unhandled request is an s-type request, denoted $s_{\texttt{first}}$. In this situation a single transport cannot possibly handle all the remaining requests, because $s_{\texttt{first}}$ prevents a mixed

transport with the r-type requests (SR constraint in Section 2.2). The *SR redundancy heuristic* prevents at $\texttt{node}_{SR}$ the development of any arc (transport) P_r that contains only r-type requests. It can do so, without compromising the completeness (optimality) of the solution, since developing transport P_r at this stage leads to redundant paths and can therefore be pruned.

Lemma 1 *Developing an arc for transport P_r at $\texttt{node}_{SR}$ is redundant.*

Proof Denote by P_s any arc (transport) that fulfills the early s-type request $s_{\texttt{first}}$. It may potentially include additional s-type requests, but not r-type requests, due to the SR constraint. So clearly any two potential transports P_s and P_r do not share any mutual requests. We consider two cases according to the vehicles v_s and v_r that operate the respective transports P_s and P_r:

- $v_s = v_r$: In case the same vehicle operates both P_s and P_r it will have to perform the P_s transport before P_r. This is because P_s must depart from the depot not later than $s_{\texttt{first}}$, and the r-type passengers in P_r must all depart later than $s_{\texttt{first}}$. Consequently, P_s should be developed before P_r.
- $v_s \neq v_r$: In case different vehicles operate P_s and P_r then the two transports have absolutely no effect on each other. Consequently, developing P_s at $\texttt{node}_{SR}$ followed by the development of P_r is exactly the same as developing P_r at $\texttt{node}_{SR}$ followed by the development of P_s.

Considering both cases, it is redundant to develop arc P_r at $\texttt{node}_{SR}$. □

Corollary 1 *The SR redundancy heuristic does not compromise the optimality of the obtained solution.*

To exemplify the SR redundancy heuristic, consider $\texttt{node}_0$, the source node in Example 1. $\texttt{node}_0$ is an example of $\texttt{node}_{SR}$, since it has unhandled requests of both types and the earliest is an s-type request ($s_{\texttt{first}} = s_1^{01}$). Now consider two possible transports, $P_s = \{s_1^{01}, s_1^{02}, s_1^{13}, s_1^{23}\}$ to be operated by v_1 and $P_r = \{r_1^{01}, r_1^{12}, r_1^{23}\}$ to be operated by v_2. In case we handle ride P_s first we reach $\texttt{node}_1 = (0,0,0,0,0,1,1,0,1,1;\texttt{10:15},\texttt{8:00})$, and after also handling ride P_r we reach $\texttt{node}_{2'} = (1,0,1,0,1,1,1,0,1,1;\texttt{10:15},\texttt{13:30})$. Conversely, if we start with ride P_r we reach $\texttt{node}_{1'} = (1,0,1,0,1,0,0,0,0,0;\texttt{8:00},\texttt{13:30})$, and after handling ride P_r we reach exactly the same $\texttt{node}_{2'}$ with exactly the same costs as in the first case. This redundancy is illustrated in Figure 1.

4.2 Equal capacity redundancy

Consider $\texttt{node}_{EC}$ in the graph and a potential transport P that can be operated by two or more vehicles of equal capacity C_{EC}. Denote by v_{first} the earliest available vehicle of capacity C_{EC}, with ties broken according to the vehicle index. The *equal capacity redundancy heuristic* prevents at $\texttt{node}_{EC}$ the development of more than a single arc for transport P with equal capacity vehicles, i.e., it considers at most a single vehicle v_{first} of each capacity

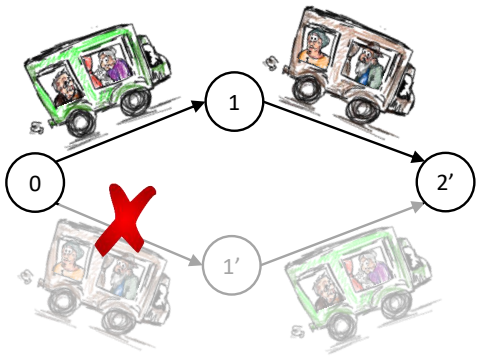


Fig. 1 SR redundancy illustration

C_{EC} to operate transport P. It can do so, without compromising the completeness (optimality) of the solution, since developing at this stage additional arcs of the same transport, operated by vehicles of the same capacity, leads to redundant paths and can therefore be pruned.

Lemma 2 *Developing an arc for transport P operated by a vehicle $v_{other} \neq v_{first}$ with capacity C_{EC} at $\texttt{node}_{EC}$ is redundant.*

Proof We consider two cases according to the availability times τ_{other} and τ_{first} of the respective vehicles v_{other} and v_{first}:

- $\tau_{other} = \tau_{first}$: The redundancy in this case is trivial, because there is absolutely no difference between vehicles v_{other} and v_{first},
- $\tau_{other} > \tau_{first}$: The only reason to prefer the later-available v_{other} over v_{first} is to free up vehicle v_{first} for operating some other transport P' that needs to departure earlier. However, this possibility would be accounted for anyway, because when the arc for transport P' will be considered it would be operated by vehicle v_{first} (for exactly the same reason that transport P is operated by v_{first}). As a consequence, transport P would be considered to be operated by vehicle v_{other} later on along that path. Thus, considering v_{other} for operating transport P at $\texttt{node}_{EC}$ is redundant.

According to the way that v_{first} was selected it holds that $\tau_{first} \leq \tau_{other}$, thus considering both above cases, it is redundant to develop an arc operated by v_{other} for transport P at $\texttt{node}_{EC}$. □

Corollary 2 *The equal capacity redundancy heuristic does not compromise the optimality of the obtained solution.*

To exemplify the equal capacity redundancy heuristic, consider $\texttt{node}_0$, the source node in Example 1. $\texttt{node}_0$ is an example of $\texttt{node}_{EC}$, since it has two vehicles of equal capacity ($C_1 = C_2 = 2$), both available ($\tau_1 = \tau_2 = \texttt{8:00}$). Now consider a possible transport $P = \{s_1^{01}, s_1^{02}, s_1^{13}, s_1^{23}\}$ to be operated by either v_1 (v_{first}) or v_2 (v_{other}). In case vehicle v_1 operates transport P we

reach $\texttt{node}_1 = (0,0,0,0,0,1,1,0,1,1;\texttt{10:15},\texttt{8:00})$. In case vehicle v_2 operates transport P we reach $\texttt{node}_{1'} = (0,0,0,0,0,1,1,0,1,1;\texttt{8:00},\texttt{10:15})$. Nodes $\texttt{node}_1$ and $\texttt{node}_{1'}$ are clearly equivalent. This redundancy is illustrated in Figure 2.

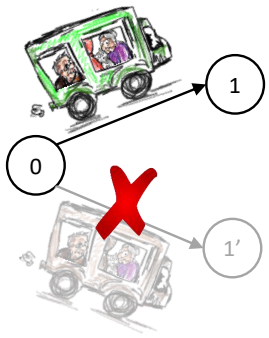


Fig. 2 Equal capacity redundancy illustration

4.3 Infeasible capacity path removal

Consider an arc $\texttt{node}_A \rightarrow \texttt{node}_B$, which is developed during the DFS traversal. We assume that the arc represents a feasible transport, i.e., it satisfies the SR and capacity constraints, and that it is not a goal node. The removal criterion that is described in this section accounts for arcs that, though feasible, will necessarily not be a part of a feasible solution due to capacity considerations.

The idea behind the *infeasible capacity path removal heuristic* is that for each developed arc there is an upper bound for the number of requests that can be fulfilled, in between any two consecutive terminals, by the later developed arcs. The heuristic prevents the development of arcs for which this bound exceeded. More specifically, the bound is determined by the number of remaining transports, and by the maximal capacity $C^* = \max\{C_1, C_2, \ldots, C_M\}$. When considering an arc $\texttt{node}_A \rightarrow \texttt{node}_B$ during DFS construction, we know the number of arcs leading to $\texttt{node}_B$ from the source node in the considered path. Denote this number by K_{path}. This defines the number of remaining transports $\tilde{K} = K - K_{path}$. The maximal number of requests that can be fulfilled in any particular segment of the route is restricted by $N_{max} = \tilde{K} \cdot C^*$.

Given a developed arc, we also know the number of remaining requests that should be fulfilled. Denote the integer coordinates of $\texttt{node}_B$ by $(\mathbf{h}^\frown\mathbf{l})$, where $\mathbf{h} = (h^{01}, h^{02}, \ldots, h^{L-1,L})$ and similarly $\mathbf{l} = (l^{01}, l^{02}, \ldots, l^{L-1,L})$. Then the number of *remaining* requests from terminal A_i to terminal A_j is given by $\tilde{N}^{ij} = N_S^{ij} + N_R^{ij} - (h^{ij} + l^{ij})$. Finally, the number of remaining requests between two consecutive terminals A_k and A_{k+1} is given by $\tilde{N}_k = \sum_{i=0}^{k} \sum_{j=k+1}^{L} \tilde{N}^{ij}$. If

the considered arc is part of a feasible path, then it must hold that $\tilde{N}_k \leq N_{max}$, for each $k = 0, 1, 2, \ldots, L-1$. This proves the following lemma.

Lemma 3 *An arc for which* $\max\{\tilde{N}_0, \tilde{N}_1, \ldots, \tilde{N}_{L-1}\} > N_{max}$ *can be removed from the graph that represents the respective FRDARP.*

Corollary 3 *The infeasible capacity path removal heuristic does not compromise the optimality of the obtained solution.*

To exemplify the infeasible capacity path removal heuristic, consider $\texttt{node}_0$, the source node in Example 1. $\texttt{node}_0$ is an example of $\texttt{node}_A$. Now consider $\texttt{node}_B$ whose integer coordinates are $(0, 0, 0, 0, 0, 1, 1, 0, 1, 0)$. The arc $\texttt{node}_A \rightarrow \texttt{node}_B$ represents the transport of $\{s_1^{01}, s_1^{02}, s_1^{13}\}$. The number of remaining transports is $\tilde{K} = 2$ and the maximal capacity is $C^* = 2$. Hence, the maximal number of requests that could be treated in a given route segment is $N_{max} = 4$. However, among the remaining seven requests, there are five requests that use the route segment between terminals A_2 and A_3, i.e., $\tilde{N}_2 = 5$. These are $s_1^{23}, r_1^{23}, s_2^{13}, r_1^{13}$, and r_2^{23}. Since $\tilde{N}_2 > N_{max}$ this arc can be removed from the graph.

4.4 A*-based path removal

The A* algorithm [6] is a classical graph traversal algorithm, often used for minimal path search. The classical A* traverses the graph in a *best-first search* [10] manner, which we do not use herein due to its space requirements (see Section 3). However, we can adopt the node expansion strategy of A* as an additional pruning heuristic.

When the A* algorithm needs to decide whether to expand a node in the graph, say $\texttt{node}_i$, it considers two values – the cost $g(\texttt{node}_i)$ of the path from $\texttt{node}_0$ (the source node) to $\texttt{node}_i$, and a heuristic estimate $h(\texttt{node}_i)$ of the cost of the optimal path from $\texttt{node}_i$ to any goal node. A heuristic $h(\cdot)$ is called *admissible* if its estimate is never larger than the real cost of the optimal path from $\texttt{node}_i$ to a goal. Therefore, when $h(\cdot)$ is admissible, the sum $g(\texttt{node}_i) + h(\texttt{node}_i)$ can serve as an optimistic view for the potential of the current partial path going through $\texttt{node}_i$. When $\texttt{node}_i$ is expanded, $g(\texttt{node}_i)$ is already known. Consequently, the challenge in achieving efficient pruning lies in the development of an admissible heuristic $h(\cdot)$ that is both fast to compute and tight, in the sense that its estimate is as close to the actual optimal value as possible.

We propose herein a heuristic $h(\cdot)$ that focuses on the set of yet unhandled requests $\tilde{Q} \subseteq Q$. The main idea is to look at the differences between the request times in $\tilde{Q}$. The differences between requests correspond to costs that would definitely be applied in case these requests share the same ride. However, considering all combinations of unhandled requests, including taking into account their types and pick-up/destination terminals, is computationally heavy. Therefore, we completely disregard the request types and pick-up/destination

terminals, and consider just a sequence of the unhandled ride requests $\tilde{Q}$, ordered according to their requested times. Relating only to consecutive elements is optimistic, as is required for the admissibility of the heuristic, for the following reasons. On one hand, such pairs of elements (passengers sharing a ride) are generally "cheaper" than non-consecutive elements. On the other hand, they may turn out to be infeasible, due to the disregard of request types (e.g., SR constraint) and terminals (e.g., capacity constraint).

```
   Input: set of unhandled requests Q̃, number of remaining transports K̃
   Output: value of h(·)
 1 sort Q̃ according to the request times
 2 Δ = multiset of time differences between consecutive elements in Q̃
 3 sort Δ in reverse order
 4 remove the first (highest) K̃ − 1 elements from Δ
 5 initialize h_val and factor to zero
 6 for index = 0 . . . (|Δ| − 1) do
 7 |   if index mod 2 · K̃ = 0 then
 8 |   |   increment factor by 1
 9 |   end
10 |   increment h_val by Δ(index) · factor
11 end
12 return h_val
```

Algorithm 1: Computation of $h(\cdot)$

The computation of our proposed $h(\cdot)$ is given in Algorithm 1. It starts with the sorting of $\tilde{Q}$ (line 1). Then, in line 2, a multiset Δ is generated, consisting of the request time differences between every pair of consecutive elements in $\tilde{Q}$. (Δ is a multiset because it may contain several repetitions of the same time difference.) Next, Δ is sorted in reverse order (line 3) followed by the removal of its first (highest) $\tilde{K} - 1$ elements (line 4). $\tilde{K}$ remaining transports mean that $\tilde{K} - 1$ time-request differences can be removed without cost, because their respective passengers can be separated to different transports. For the sake of admissibility we remove the most costly $\tilde{K} - 1$ differences.

Two variables are introduced and initialized in line 5 – h_{val}, the value of $h(\cdot)$ that needs to be computed, and $factor$ that represents the minimal number of times a cost difference should be accounted for in the computation of h_{val}. To understand the concept of $factor$ consider an `adt` of a ride that is earlier than the ride time requests of three (s-type) passengers on that ride. The difference between the `adt` and the latest of these three request times will be accounted for three times, once for the cost of each passenger, i.e., the $factor$ should be 3 in this case. However, the difference between the request times of the first and second passengers will only be accounted for once for the cost of the first passenger, i.e., the $factor$ should be 1 in this case.

Now we continue to the main loop (lines 6-11), which visits all the remaining elements of Δ (after the removal in line 4). In $index = 0$ and later in every $2 \cdot \tilde{K}$ iterations the $factor$ increases by 1 (lines 7-8). Consider again the time

difference between the requests of the first and second passengers from the last example. It has a $factor$ of 1 because it is in the "border" of the ride (no passenger with a later request than the first). However, there may also be a similar border for r-type requests of earlier time than `adt`. Moreover, any remaining transport of the $\tilde{K}$ can have two such borders. Thus, for sake of admissibility we consider the maximal number of possible time differences accounted for before the $factor$ grows. Next, in line 10, each time-difference value in Δ is multiplied by the corresponding $factor$ at that time of the computation, before being added to h_{val}. Naturally, for the optimistic scenario needed herein, we must consider that the high differences (high costs) are multiplied by low $factor$ values, and vice versa, and this is indeed achieved by starting with the highest values in Δ (recall the reverse ordering in line 3). Finally, in line 12, the accumulated h_{val} is returned as the output of the computation.

Lemma 4 *The $h(\cdot)$ heuristic of Algorithm 1 is admissible.*

Proof In Algorithm 1 we relate to the unhandled requests $\tilde{Q}$ without regarding their types and terminals. By disregarding the terminals, the problem becomes a simple case of the *two-campus transport problem* (TCTP) [8]. It has been proven that there always exists an optimal solution that groups together in a ride only consecutive request times (according to each type) [8, Theorem 1]. In $\tilde{Q}$ we also disregard the types, so now grouping together only consecutive request times is definitely optimistic (though probably infeasible). As a consequence, we can relate only to the costs of consecutive ride requests, i.e., their time differences (Δ). A best-case optimal solution may (i) potentially rule-out the most costly differences, and (ii) repeat as few times as possible the costs of the remaining costly differences. This is exactly what happens in line 4, and in the loop of lines 6-11, respectively. Consequently, the returned h_{val} serves as an optimistic lower bound to the actual cost for the remaining requests $\tilde{Q}$ and transports $\tilde{K}$. □

Following the admissibility of $h(\cdot)$, the A*-*based path removal heuristic* does not expand any $\texttt{node}_i$ for which $g(\texttt{node}_i) + h(\texttt{node}_i) \geq UB$, where UB is the cost of the best solution found so far. The upper bound UB is maintained as part of the Branch & Bound procedure (Section 3).

Corollary 4 *The* A*-*based path removal heuristic does not compromise the optimality of the obtained solution.*

To exemplify the computation of $h(\cdot)$ in the A*-based path removal heuristic, consider $\texttt{node}_0$, the source node in Example 1. Given that $\texttt{node}_0$ is a source node, we have $\tilde{Q} = Q$ and $\tilde{K} = K = 3$. The ride requests Q are already ordered (line 1) in the presentation of Table 1, so we continue to present in Table 2 the corresponding Δ and ordered Δ (lines 2 and 3).

After the execution of line 4, the two $(\tilde{K} - 1)$ highest costs (75 and 60) are removed from Δ. Their remain 7 elements in Δ to be considered in the main loop (lines 6-11). The $factor$ is incremented every six $(2 \cdot \tilde{K})$ iterations,

Δ	60	30	0	15	15	75	45	0	30
Ordered Δ	75	60	45	30	30	15	15	0	0

Table 2 Δ and ordered Δ in the computation of $h(\texttt{node}_0)$

so only the last element, which is zero anyway, is multiplied by $factor = 2$. So the computed h_{val} is $(45 + 30 + 30 + 15 + 15 + 0) \cdot 1 + 0 \cdot 2$, resulting in $h(\texttt{node}_0) = 135$. Note that in case this was not a source node, the cost $g(\texttt{node})$ of the path until `node` would have been added for the comparison with the current UB.

5 Experimental evaluation

We have implemented an FRDARP solver based on Branch & Bound (Section 3) and all the presented pruning heuristics (Section 4) in Java. In the experiments we evaluate six versions of the algorithm – a basic version without any pruning heuristic, four versions that apply a single heuristic each, and a full version that incorporates all four heuristics. By that we can learn about the effectiveness of each of the heuristics, and also examine their combined effect. The experiments were executed on a hardware comprised of an Intel i7-6820HQ processor and 32GB memory.

In order to evaluate the efficiency of the proposed pruning heuristics we chose a setting that is based on a transportation solution for elderly people living in nursing facilities [4], see Section 3. The *basic setting* includes $N = 15$ ride requests, $M = 2$ vehicles of capacity $C_1 = C_2 = 5$, which are available from $a_1 = a_2 =$ `7:00` until $b_1 = b_2 =$ `19:00`. The route consists of $L = 4$ terminals (including the depot), and takes 60 minutes to complete. The number of transports is set to $K = 3$.

We create 50 instances for each setting in each of the experiments, and present the mean values of these 50 instances. The instances differ in the ride requests. For each ride request in an instance we randomly choose its type (s-type or r-type), its two terminals (pick-up and destination) and its desired pick-up time (every 15 minutes within the working hours of the vehicles).

Our experiments examine the percentage of problems that can be (optimally) solved within a short period of time, where the timeout is set to one minute. In each experiment we focus on some parameter in order to observe its effect.

We begin with the number of ride requests parameter, which varies in the range $9 \leq N \leq 21$, while all other parameters remain fixed according to the basic setting. The results are shown in Figure 3.

Next, we focus on the vehicles' capacity parameter, which we vary in the range $3 \leq C \leq 7$. However, changing only the capacity may result in unsatisfiable problems (when C is too small) or in rather easy problems (when C is too large). To this end, we would like to maintain a balance between the number of

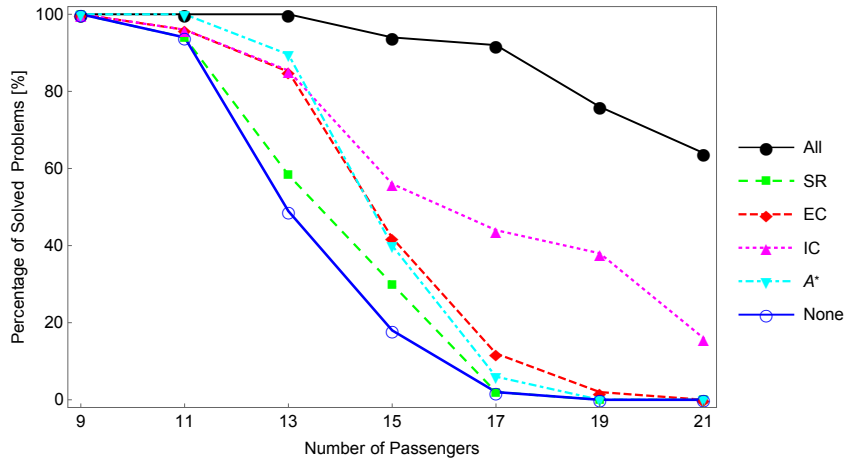


Fig. 3 Percentage of 1-minute solved problems with varying number of ride requests N

ride requests and the overall capacity (considering both the number of transports and the vehicles' capacity), according to the formula $N = K \cdot C$. Note that since passengers free up places in intermediate terminals, this initially tight-looking ratio usually corresponds to satisfiable problems. Thus, in this experiment, shown in Figure 4, we also accordingly vary the number of ride requests in the range $9 \leq N \leq 21$, where the number of transports remains $K = 3$ as in the basic setting.

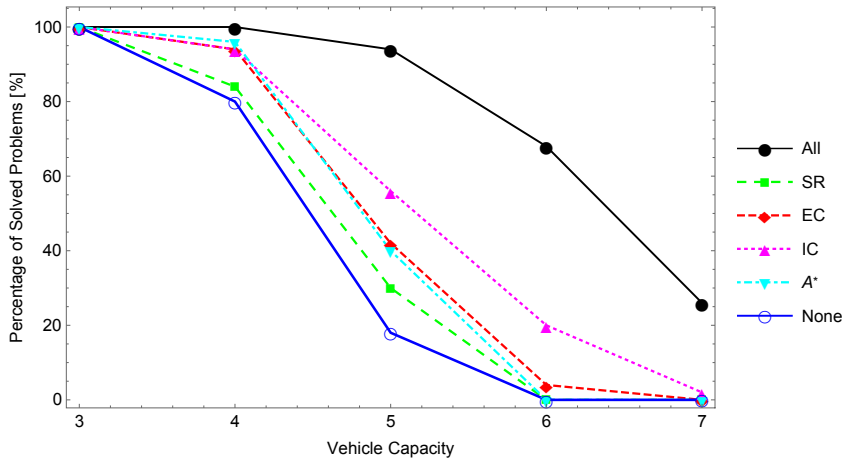


Fig. 4 Percentage of 1-minute solved problems with varying vehicles' capacity C

The last parameter we study is the number of transports, which we vary in the range $2 \leq K \leq 4$. Again, we maintain the balance $N = K \cdot C$. Consequently, in this experiment, depicted in Figure 5, we also vary the number of ride

requests in the range $10 \leq N \leq 20$, where the vehicles' capacity remains $C = 5$ as in the basic setting.

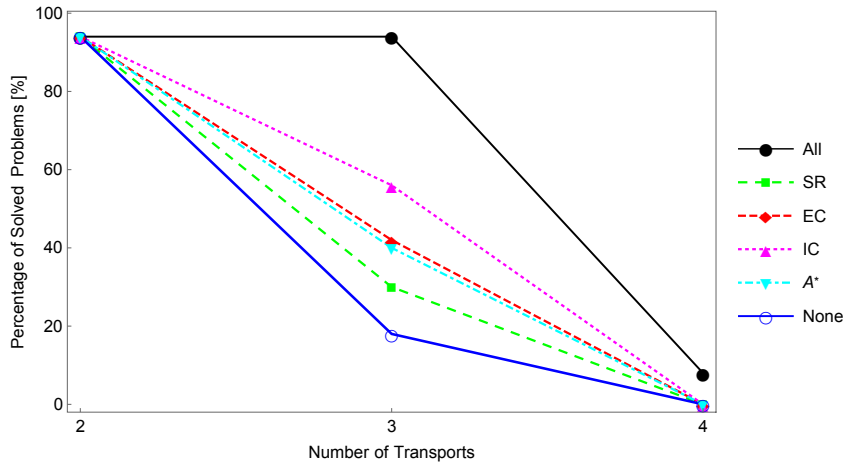


Fig. 5 Percentage of 1-minute solved problems with varying number of transports K

The presented results shed light regarding the performance of the proposed heuristics. Each of the heuristics on its own improves the practical solvability of FRDARP, but interestingly their combination is considerably more efficient than any single heuristic. This phenomenon suggests that the heuristics do not overlap much and are able to prune different parts of the constructed graph. This comes as no surprise given that the four heuristics use completely different ideas that lead their pruning strategies. Further insight regarding the combined effectiveness of the four heuristics can be gained by observing the sizes of obtained graphs without and with the heuristics. For example, the number of developed nodes for instances of the basic setting, which were solved within the 1-minute timeout, reduced from $\sim 300K$ to $\sim 5.5K$ on average.

Out of the four heuristics, the infeasible capacity heuristic performs best in most settings. It is actually only outperformed by the A*-based heuristic in rather easy settings ($N = 11$,$N = 13$ in Figure 3 and $C = 4$ in Figure 4).

Turning the spotlight to the A*-based heuristic, its performance seems to deteriorate as the problems become harder (in terms of number of passengers or vehicle capacity, see Figures 3 and 4). This suggests that although the basic idea of the heuristic has potential (as witnessed in its performance in the easy settings), the value of $h(\cdot)$ for hard instances is not tight enough, i.e., not close enough to the actual cost. Indeed, in the computation of $h(\cdot)$ we completely disregard the types and terminals of unhandled ride requests (Algorithm 1, line 1). The idea was to make the computation fast. However, for harder problems we may consider performing a more complex and tighter computation of $h(\cdot)$ that may result in increased overheads, but at the same time significantly reduce the size of the constructed graph.

6 Conclusion

The main contribution of this paper is in presenting and investigating pruning heuristics for the FRDARP model. In Section 2 we have presented the FRDARP model, including the reduction to the shortest path problem. A side contribution of this paper lies in the compactness of this presentation. We also believe it is more fluent and accessible, and in addition, it includes an illustrating example, which was missing in the first paper on this subject [7].

Regarding the pruning heuristics, we have presented four heuristics. Two of the heuristics attempt to identify and remove redundant paths, whereas the objective of the other two is to rule-out paths that are provably infeasible or non-optimal. According to our experimental evaluation, all the heuristics perform well and improve the practical solvability of FRDARP. However, the most interesting phenomenon is that their combined effort is considerably more efficient than any single heuristic, probably due to their completely different pruning strategies.

With the help of these pruning heuristics, the model is now more applicable for use in real-life problems, as well as in more complex variants of the model. For example, one can regard adding the option for vehicles to wait at intermediate terminals. There are some real-life scenarios in which such waiting is acceptable by the passengers, e.g., train connections in major stations. This option may reduce the value of the objective function, because new scheduling possibilities arise. For example, a vehicle that previously could not peak an r-type passenger from an intermediate station, can now wait in order to pick up this passenger. This can reduce the waiting time of passengers and may also lead to better vehicle availability. However, the option to wait at intermediate terminals is expected to increase the graph size, because of these new possibilities. Therefore, pruning heuristics are crucial for the applicability of such a complex variant.

Acknowledgements This research was supported by the Ministry of Science & Technology, Israel.

References

1. Richard E Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.
2. Jean-François Cordeau and Gilbert Laporte. The dial-a-ride problem: models and algorithms. *Annals of Operations Research*, 153(1):29–46, 2007.
3. Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT Press, 3rd edition, 2009.
4. Svetlana Daichman, Hillel Bar-Gera, and Tal Grinshpoun. Mobility habits and transportation needs of inhabitants of elderly homes in Israel. In *14th International Conference on Industrial Logistics (ICIL)*, 2018.
5. Willem E de Paepe, Jan Karel Lenstra, Jiri Sgall, René A Sitters, and Leen Stougie. Computer-aided complexity classification of dial-a-ride problems. *INFORMS Journal on Computing*, 16(2):120–132, 2004.

6. Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.
7. Hagai Ilani, Elad Shufan, and Tal Grinshpoun. A fixed route dial-a-ride problem. In *Proceedings of the 7th Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA)*, pages 313–324, 2015.
8. Hagai Ilani, Elad Shufan, Tal Grinshpoun, Aviad Belulu, and Alex Fainberg. A reduction approach to the two-campus transport problem. *Journal of Scheduling*, 17(6):587–599, 2014.
9. Yves Molenbruch, Kris Braekers, and An Caris. Typology and literature review for dial-a-ride problems. *Annals of Operations Research*, 259(1-2):295–325, 2017.
10. Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited, 3rd edition, 2016.

Robustness of periodic reoptimization policies for the dynamic PDPTW

Farzaneh Karami · Wim Vancroonenburg · Greet Vanden Berghe

the date of receipt and acceptance should be inserted later

Abstract In the dynamic pickup and delivery problem with time windows (PDPTW), there exists uncertainty concerning the time windows and locations associated with requests. Periodic reoptimization policies are suitable for dealing with such uncertainty. The key question is to what extent these policies are robust for the dynamic PDPTW. We analyze robustness by taking into account both the reoptimization period and the dynamism degree. We first evaluate the robustness locally after each reoptimization and then evaluate it globally at the end of the scheduling horizon. Results indicate that robustness increases both locally and globally when either the reoptimization period lengthens or the dynamism degree increases, and vice versa. Finally, we conclude that periodic reoptimization policies for the dynamic PDPTW handle uncertainty best when the reoptimization period matches the requests' urgency.

Keywords Dynamic pickup and delivery with time windows · Periodic reoptimization policy · Robustness analysis

1 Introduction

How should one handle the inherent uncertainty present in dynamic problems? This is not a straightforward question to answer given that there is little agreement concerning how exactly one should even quantify performance in dynamic problems, let alone how best to maintain high quality solutions under such conditions. In a problem such as the dynamic pickup and delivery problem with time windows (PDPTW), uncertainty can arise in many forms: request locations, request time windows, vehicle availability and so forth. If an approach for the dynamic PDPTW is able to consistently produce high quality solutions despite these unknowns, then

F. Karami[1] (corresponding author), W. Vancroonenburg[1], G. Vanden Berghe[1]
[1] KU Leuven, Department of Computer Science, CODeS, Gebroeders De Smetstraat 1, 9000 Gent, Belgium
Tel.: +32 9 265 87 04
E-mail: {farzaneh.karami, wim.vancroonenburg, greet.vandenberghe}@cs.kuleuven.be

it is referred to as robust.

The more uncertainty a method can handle, the more robust it is said to be. Two possibilities for addressing uncertainty in the dynamic PDPTW are (i) those which reserve unscheduled vehicles to deal with unforeseen request arrivals and (ii) reoptimization-based approaches. Despite their widespread use, reservation-based approaches are inefficient for two reasons. First, surplus vehicles are not profitable during non-peak periods. Second, if none of the reserved vehicles are located close to the location(s) associated with unpredictable requests, then it is a waste of resources to reserve additional vehicles during the peak periods.
As Psaraftis [5] have highlighted, there is usually very little flexibility when it comes to varying fleet size in reoptimization-based approaches. There is usually some timespan before the schedules are executed. Consequently, solving the dynamic PDPTW as a static scheduling problem using robust optimization affords greater flexibility in terms of including additional vehicles in order to meet demand.
Dynamism and urgency [7] are two important parameters associated with a dynamic PDPTW instance. The dynamism degree corresponds to the frequency with which requests arrive. Meanwhile, urgency levels convey how quickly those requests must be serviced. The combination of these two parameters represents an instance's degree of uncertainty. A thorough understanding of the issues stemming from the various forms of uncertainty is helpful for solving the PDPTW. This is especially the case when there is no prior knowledge available concerning request arrivals, their locations and their time windows.
The time interval associated with reoptimization is what determines optimization runtime. One possibility is to react to each and every information update, a policy referred to as reactive reoptimization. Another possibility is to only reoptimize once a set of predefined criteria is met, a policy referred to as periodic reoptimization. Reactive reoptimization is likely to generate huge computational burdens when solving large-scale problems and demands more resources than periodic reoptimization policies to maintain solution quality [3]. On the other hand, the periodic reoptimization policy introduced by Karami et al. [2] takes urgency levels into account and defines a buffering time interval between consecutive calls to the solver. As a result, periodic reoptimization policies are capable of controlling the amount of time available for reoptimization. Thus, they call for a trade-off between solution quality and robustness.
Figure 1 illustrates how utilizing a reactive reoptimization policy differs from using a periodic policy. This example concerns the assignment of two requests to a single vehicle before noon. The two requests are announced during the workday at 9:35 and 9:40 with their pickup and delivery time windows being [9:35-11:00] and [9:40-9:50], respectively. Rejecting requests is not permitted. The vehicle traverses a Manhattan-style grid where each edge requires five minutes of travel time. The objective is to minimize the sum of travel times, driver overtime, and lateness of requests. While the best solution the reactive reoptimization policy can achieve takes 150 minutes, the periodic policy generates a solution of 135 minutes when its reoptimization period is set to 5 minutes, despite the fact it begins servicing at a later point in time. If we increase the reoptimization period from 5 to 20 minutes, the best attainable objective value then becomes 190 minutes. This example simply demonstrates the potential benefit of a periodic reoptimization policy and the cru-

cial role of the reoptimization period. Now, however, one crucial question arises: what reoptimization period will produce high quality solutions that are also robust with respect to unpredictable request arrivals?

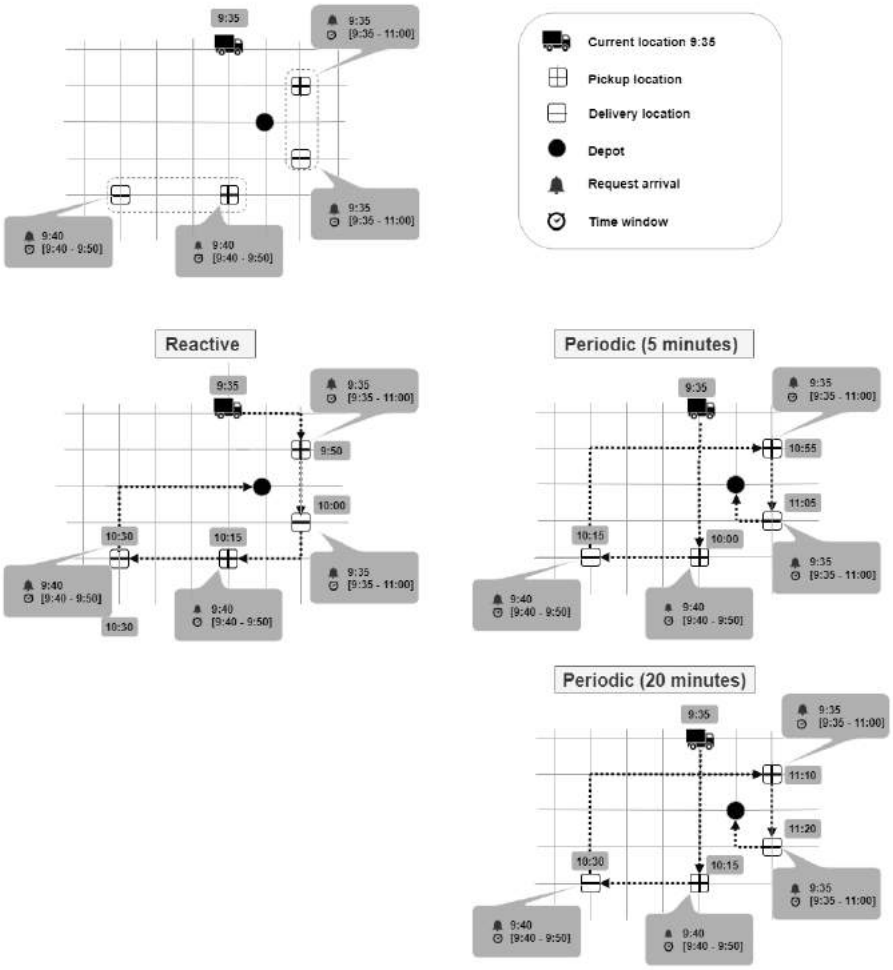


Figure 1: Dynamic PDPTW addressed by one reactive and two periodic reoptimization policies

Two possible metrics for robustness are the solution itself and its quality. In the former, robustness concerns preserving the assignments constituting a solution. Meanwhile, the latter metric only considers solution quality, regardless of the number of changes made to the schedule. Since solutions are frequently updated during a periodic reoptimization policy, the solution quality perspective of robustness

makes most sense. Therefore, for the remainder of this paper, robustness refers to solution quality robustness.
Investigating a periodic reoptimization policy's robustness involves exploring the range of different degrees of uncertainty under which a given dynamic PDPTW performs 'well', 'as intended' or 'as required'. The dynamic PDPTW with *request time windows' uncertainties* has been defined by Srour et al. [6] with the assumption that exact pickup and delivery locations are known while the time windows are uncertain. Morlok and Chang [4], however, consider request location uncertainty for a transportation system.

Nevertheless, the academic literature lacks a means of quantifying robustness when it comes to periodic reoptimization policies for the dynamic PDPTW, when considering uncertainty for both timing (arrivals and time windows) and locations. This research is dedicated to addressing this issue. We analyze the robustness of periodic reoptimization in the dynamic PDPTW with respect to varying reoptimization periods and degrees of dynamism. Our experiments are conducted on generated instances which exhibit a variety of dynamism degrees and urgency levels.

2 Uncertainty metric

2.1 Urgency and dynamism

A dynamic PDPTW instance [1] is a triple (τ, ε, V) where τ denotes the scheduling horizon, V the fleet of vehicles, and where ε consists of all request arrivals. The time at which a request r becomes known is referred to as its arrival time a_r. The continuity of request arrivals corresponds to the PDPTW's degree of dynamism, while urgency is treated as a distinct characteristic and defined as the length of time from a request's arrival until the end of either its pickup or delivery time window. The information regarding both the pickup and delivery tasks' time windows is available upon request arrival. Given that the pickup should be finished first, we define urgency based on the pickup time window.

To generate instances with different degrees of dynamism using the generator provided by van Lon et al. [7] consider $\triangle := \{\delta_0, \delta_1, \ldots, \delta_{|\varepsilon|-2}\} = \{a_{rj} - a_{ri} \mid j = i+1 \ \wedge \ \forall r_i, r_j \in \varepsilon\}$, which represents the sequence of inter-arrival times for requests, where $|\triangle| := |\varepsilon| - 1$.

The inter-arrival time required for uniform distribution, in other words 100 percent dynamism, is $\frac{\tau}{|\varepsilon|}$. Based on the definition provided by van Lon et al. [7], dynamism is measured by

$$Dy = 1 - \frac{\sum_{i=0}^{|\triangle|} \sigma_i}{\sum_{i=0}^{|\triangle|} \hat{\sigma_i}},$$

for which the numerator is the sum of all deviations of inter-arrival times (σ_i) relative to the 100 percent case and where the denominator is the maximum deviation for the scenario.

One possible example sequence of inter-arrival times for requests is $\triangle_a = \{0.1, 1, 0.1, 1, 0.1, 1, 0.1, 1, 0.1\}$, which corresponds to five small bursts with intervals of 0.1 and four of 1 unit, as shown in Figure 2(a). By changing this order, another possible sequence for request inter-arrival times is $\triangle_b = \{1, 1, 0.1, 0.1, 0.1, 0.1, 0.1, 1, 1\}$, which has one large burst of request arrivals in the middle and two individual request arrivals at both the beginning and end, as shown in Figure 2(b). Thus, the degree of dynamism of Figure 2(a) is 55.5%, whereas that of Figure 2(b) is 35.1%.

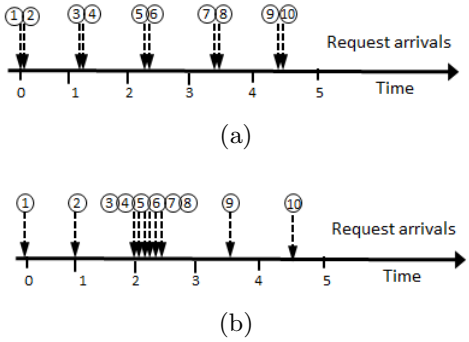


Figure 2: Two possible examples of request arrival events with five inter-arrival times of 0.1 and four inter-arrival times of 1 unit.

In order to conduct our computational study we created a set of instances whose characteristics exhibit a variety of dynamism and urgency combinations. The generator was employed to produce instances with three different degrees of dynamism (20%, 50% and 90%) and five different levels of urgency (5, 15, 25, 35 and 45 minutes). Five instances were produced for each of the 15 possible combinations, resulting in a total of 75 instances.

2.2 An uncertainty metric for periodic reoptimization policy

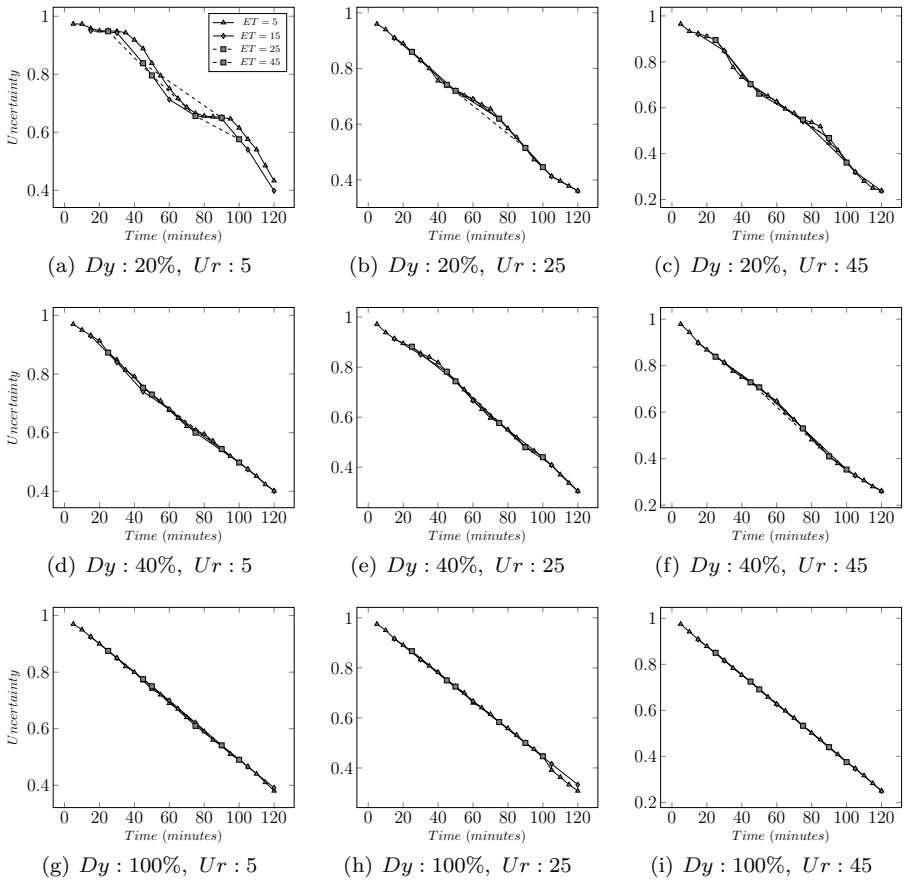
The *degree of certainty* of a periodic reoptimization policy refers to the proportion of observed requests it has integrated into the reoptimization at a particular time t. Let $|\varepsilon|$ be the total number of requests for an instance and n_t the number of requests which have arrived before t. The degree of certainty at time t is then defined as $\frac{n_t}{|\varepsilon|}$.

The length of time between two consecutive optimizations is denoted by ET [2]. Meanwhile, the degree of certainty C_i of reoptimization period $i \geq 1$ is equal to $C_{i-1} + \Gamma_i$, where $C_0 = 0$ and Γ_i represents the additional amount of certainty a periodic reoptimization policy gains during period i, which can be calculated as follows:

$$\Gamma_i = \frac{n_{iET} - n_{(i-1)ET}}{|\varepsilon|}$$

We define the degree of uncertainty at optimization period i as $U_i = 1 - C_i$. We consider instances with a variety of dynamism and urgency configurations in the

first two hours of the scheduling horizon in Figure 3. The graphs show the uncertainty degree determined by a periodic reoptimization policy. The scheduling horizon's length τ is four hours. Dy and Ur denote each instance's dynamism degree and urgency level, respectively. The degree of uncertainty is calculated for various ETs. One can observe nonlinear variations concerning the uncertainty degree of instances with lower degrees of dynamism. However, given that the uncertainty degree increases relative to the reoptimization frequency, one question logically arises: does solution quality deteriorate as uncertainty increases? And if the answer to this question is affirmative, then by how much does it deteriorate?



(a) Dy : 20%, Ur : 5 (b) Dy : 20%, Ur : 25 (c) Dy : 20%, Ur : 45

(d) Dy : 40%, Ur : 5 (e) Dy : 40%, Ur : 25 (f) Dy : 40%, Ur : 45

(g) Dy : 100%, Ur : 5 (h) Dy : 100%, Ur : 25 (i) Dy : 100%, Ur : 45

Figure 3: Uncertainty degrees of periodic reoptimization policy in the first two hours of the scheduling horizon.

3 Periodic reoptimization policies robustness

In this section we analyze the robustness of the periodic reoptimization policy proposed by Karami et al. [2] where additional requests are inserted into the

existing schedule with the objective of minimizing the sum of travel time, vehicle overtime and request lateness. We employed the iterative scheduling algorithm introduced by Vancroonenburg et al. [8]. At each iteration, a random request is eliminated from its associated transporter's route. This request is then reinserted into the schedule using the cheapest insertion procedure. If the resulting solution is better than the current solution, it is accepted. The search stops if a predefined stopping criterion as a maximum number of non-improvements is met. This study will compare the periodic policy's objective values against those of the optimum static solutions obtained by a MILP, which were generated for both the local and the global problems.

Robustness metric A challenge present in periodic reoptimization is how to measure the impact of the degree of uncertainty on an existing schedule, both locally after each optimization and globally at the end of the scheduling horizon. There is no metric available for calculating local or global robustness. However, such a metric is needed to assess the impact of request arrivals on the solution's quality. First, consider the solution quality associated with each of the scheduling horizon's reoptimization periods. Let us then define L_{nr} as the percentage difference between the average solution quality across the entire horizon and the worst quality observed. We also define G_{nr} as the gap between the objective value incurred by the periodic reoptimization policy over the entire scheduling horizon with respect to the optimal objective value over the same period. Therefore, *local and global robustness* can be defined as 100-L_{nr} and 100-G_{nr}, respectively.

Local robustness Figure 4 provides the local robustness values for instances with urgency levels of {5, 15, 25, 35 and 45 minutes} and dynamism degrees of {low-20%, medium-50% and high-90%}. Each node in Figure 4(a) corresponds to the local robustness for instances with five different urgency levels and identical dynamism degrees. Similarly, each node in Figure 4(b) corresponds to the local robustness for instances with three different dynamism degrees and identical urgency levels. Note that progressing along the x-axis involves the level of urgency decreasing.

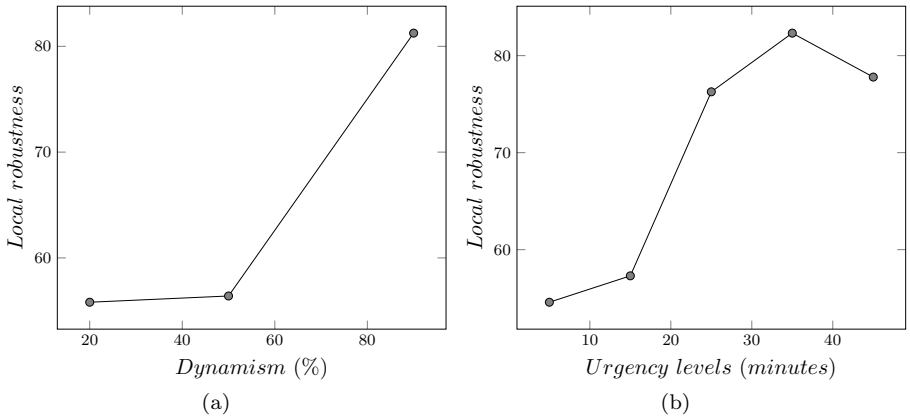


Figure 4: Local robustness of the periodic reoptimization policy.

Figure 4 shows how better local robustness is obtained for instances which are highly dynamic or which experience low urgency. The figure also illustrates the small decrease of 4% in the local robustness value when transitioning from urgency level 35 to 45. This may be attributed to the decreased flexibility of the scheduling algorithm during the very long reoptimization period when the large number of buffered requests demands more vehicles to achieve a higher robustness. Meanwhile, lower robustness levels are anticipated as the urgency increases further. The computational results fully support this prediction, as evidenced by the lack of a peak on the left-hand side of Figure 4(b). This effect is due to the existence of scenarios which incorporate queues of highly urgent requests that are larger than the fleet of available vehicles, which implies less flexibility and thus less robustness. Non-urgent scenarios with any dynamism degree appear the most flexible scenarios for the scheduling algorithms to address. This is possibly due to the balanced number of requests associated with such scenarios, a demand pattern which does not require a large fleet of vehicles. An average local robustness of 70% is achieved across all instances.

Global robustness Figure 5 provides the global robustness results and illustrates how global robustness for all instances remains in the range of 66-78%. Figure 5(b) shows that the global robustness decreases in a nonlinear fashion when the reoptimization frequency is increased. Nevertheless, global robustness across all instances is relatively high, averaging 73%.

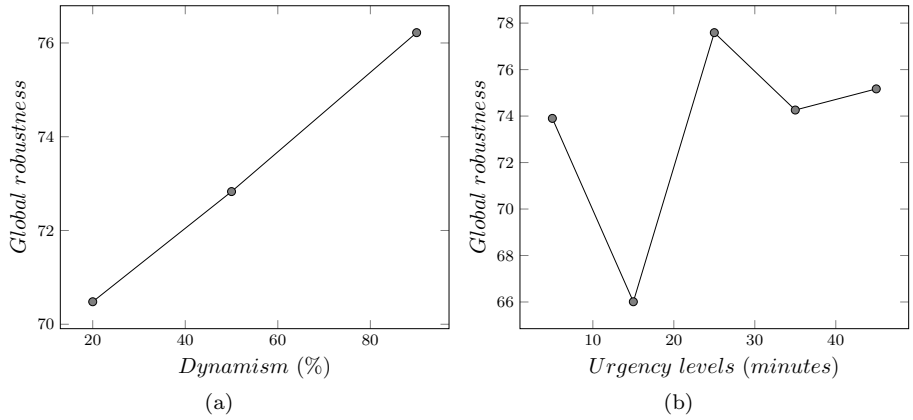


Figure 5: Global robustness of the periodic reoptimization policy.

4 Conclusions

This paper evaluated the robustness of a periodic reoptimization policy for the dynamic PDPTW with regard to varying reoptimization periods and dynamism degrees. The analysis indicates that robustness increases globally at the end of an instance's horizon or locally after each optimization when either the reoptimization period or the dynamism degree increases.

Shortening the reoptimization period may increase the frequency of disruptions, which incurs nervous decision-making without improving the schedule's quality. Another way of putting this is that a periodic reoptimization policy's response may be perfectly sufficient to deal with the disruptions faced and that rescheduling more frequently can prove counterproductive, at least from a robustness perspective. Therefore, the reoptimization frequency should be low enough to ensure a high degree of robustness. As soon as robustness begins to decrease the reoptimization frequency must be increased. If this simple rule is applied then the robustness of a periodic reoptimization policy will increase.

Acknowledgements

This research is funded by FWO as part of ORDinL (Operational Research and Data science in Logistics) project under the Strategic Basic Research (SBO) program. We also acknowledge the support provided by KU Leuven C24/17/012 "Dynamic Combinatorial Optimization". Editorial consultation provided by Luke Connolly (KU Leuven).

References

Gendreau, M., Guertin, F., Potvin, J.-Y., and Séguin, R. (2006). Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries. *Transportation Research Part C: Emerging Technologies*, 14(3):157–174.

Karami, F., Vancroonenburg, W., and Vanden Berghe, G. (2018). A buffering-strategy-based solution method for the dynamic pickup and delivery problem with time windows. In *12th International Conference on the Practice and Theory of Automated Timetabling, Vienna, Austria, 28-31 August*, pages 256–272.

Karami, F., Vancroonenburg, W., and Vanden Berghe, G. (2019). Periodic vs. reactive scheduling for internal hospital logistics. *TR, KU Leuven*.

Morlok, E. K. and Chang, D. J. (2004). Measuring capacity flexibility of a transportation system. *Transportation Research Part A: Policy and Practice*, 38(6):405–420.

Psaraftis, H. N. (1988). Dynamic vehicle routing problems. *Vehicle routing: Methods and studies*, 16:223–248.

Srour, F. J., Agatz, N., and Oppen, J. (2016). Strategies for handling temporal uncertainty in pickup and delivery problems with time windows. *Transportation Science*, 52(1):3–19.

van Lon, R. R., Ferrante, E., Turgut, A. E., Wenseleers, T., Vanden Berghe, G., and Holvoet, T. (2016). Measures of dynamism and urgency in logistics. *European Journal of Operational Research*, 253(3):614–624.

Vancroonenburg, W., Esprit, E., Smet, P., and Vanden Berghe, G. (2016). Optimizing internal logistic flows in hospitals by dynamic pick-up and delivery models. In *Proceedings of the 11th international conference on the practice and theory of automated timetabling, Udine, Italy, 23-26 August*, pages 371–383.

A metaheuristic approach for an intermittent traveling salesperson problem extension

Pieter Leyman · Patrick De Causmaecker

Received: date / Accepted: date

Abstract We discuss the intermittent traveling salesperson problem (ITSP), an extension to the well-known traveling salesperson problem. Similar to the classical problem, the ITSP requires that a tour visits all nodes in the network, but each node now has a required processing time as well. Furthermore, the allowable consecutive processing time of a node is limited, which results in the introduction of waiting time and/or multiple visits. As a result, a valid tour for the ITSP may no longer constitute a Hamiltonian cycle, but could include loops as well as use edges multiple times.

We omit assumptions made in earlier work on the ITSP, to allow for a broader and more realistic discussion of the problem. Specifically, we generalize the underlying model that determines the maximum consecutive node processing time. The contribution of the research can be summarized as follows. First, we propose a metaheuristic algorithm for this extended ITSP. We focus on algorithm components and specifically propose several options for the decoding procedure from solution representation to an ITSP tour. Second, we perform computational experiments, to allow for meaningful insights into each algorithm component's performance. We generate sufficiently diverse test instances, to warrant generalizable conclusions.

Keywords Routing · Temperature functions · Metaheuristics · Solution representation

Pieter Leyman is a Postdoctoral Fellow of the Flemish Research Foundation with contract number 12P9419N.

P. Leyman
Etienne Sabbelaan 53, 8500 Kortrijk, Belgium
E-mail: pieter.leyman@kuleuven.be

P. De Causmaecker
Etienne Sabbelaan 53, 8500 Kortrijk, Belgium
E-mail: patrick.decausmaecker@kuleuven.be

1 Introduction

We consider the intermittent traveling salesperson problem (ITSP) as a variant of the traveling salesperson problem (TSP), in which each node may need to be visited more than once. Each node has a required processing time, and a maximum temperature is imposed above which overheating would occur (or, alternatively, a maximum node capacity would be exceeded). An important trade-off to be made is whether to wait for a node to cool down sufficiently to allow for further processing, or to move to another node, process (a part of) that one, and move back to the first node. Note that there is no restriction which states that one should go back to an earlier abandoned node as soon as possible. Depending on among others, the underlying graph structure, it may prove beneficial to only finish processing some nodes at a much later time.

A major assumption of earlier work by Pham et al. (2020) concerns the temperature increase and decrease functions. Whereas in earlier work, the node temperature is considered to increase and decrease in a linear fashion, we now wish to relax this assumption, and propose generalized temperature increase and decrease functions (Section 2).

Consider that, while we retain the original naming, specifically regarding temperature (functions), the ITSP can potentially be applied in many different fields. Examples are the processing of a metal surface with a laser, in which case the metal cannot be allowed to overheat, and the delivery of fuel to different locations, with both a fill and usage rate.

2 Problem definition

2.1 General

An undirected graph $G(V, E)$ can be used to model the ITSP, with V the set of vertices or nodes (including the depot), and E the edges between the nodes. Each node i has a required processing time p_i ($p_0 = 0$ for the depot) and a temperature τ_i which would be achieved if node i would be processed for p_i consecutive time units. The specific values for both p_i and τ_i are determined independently. The travel time between each pair of nodes (i, j) is d_{ij} ($d_{ij} \geq 0$). We assume that $d_{ij} = d_{ji}$ and that the triangle inequality holds, which means that the direct travel time between any pair of nodes is never larger than the travel time between both nodes via a detour to any other node. Furthermore, a maximum temperature of T_{max} is imposed. The goal of the ITSP is to minimize the time at which the "salesperson" returns to the depot and all nodes have been fully processed. As a result, the objective function value consists of the total processing time ($\sum_{i=1}^{|V|} p_i$), the times traveled between the nodes and any waiting time incurred. Consider that the latter two depend on the route selected through the network, whereas the former is independent of the route taken. Finally, we wish to point out the use of some terminology. For the ITSP, each *node* i may be *visited* multiple times, with a different visit implying

that at least one other node has been (partially) processed and traveled to in between. Additionally, each *visit* may consist of multiple *processing steps*, which are separated by waiting times. In this case, however, we do not leave node i, but wait for it to cool down, before we continue with processing.

2.2 Temperature functions

To model the temperature $T_i(t)$ of node i after t_i consecutive time units of processing we use the following general polynomial function:

$$T_i(t_i) = \tau_i \cdot (t_i/p_i)^a,\ t_i \leq p_i \tag{1}$$

with a a constant which determines the increase rate ($a \in]0; +\infty[$). A value of a between 0 and 1 (neither included) leads to a concave temperature increase function (fast initial increase), whereas a value larger than 1 leads to a convex function (slow initial increase). A value of 1 corresponds with a linear increase. Figure 1 shows an example of how the node temperature increase function can differ based on the selected value for a. Furthermore, a can be seen as a feature of the problem studied and the nature of processing, which determines how the temperature of the material increases.

Allow us to illustrate our reasoning with a simple example. Assume that for a given node i ($p_i = 10, \tau_i = 10$), we have a value for a of 0.5 (which corresponds with the $a = 0.5$ curve in Figure 1). If we furthermore assume a T_{max} value of 7, this leads to a maximum consecutive processing time c_i for node i of 4 time units ($= \lfloor p_i \cdot \exp(\ln(T_{max}/\tau_i)/a) \rfloor$), by rounding down the inverse of Function (1) given a temperature $T_i(t)$ of 7. Note that we round the result down to the nearest integer, since we assume integer processing times.

So far, however, we have only explained how the temperature increase is modeled, but not the temperature decrease. Provided that the increase function, given a node i, only depends on the heating parameter a, we choose to employ the same function but with a different cooldown parameter b:

$$T_i(t_i) = \tau_i \cdot (t_i/p_i)^b,\ t_i \leq p_i \tag{2}$$

The way we want to use this decrease function is, however, different from the manner in which we use the increase function. Recall from the earlier example, that for node i we had determined that the maximum number of consecutive time units of processing c_i was 4. The cooldown function now allows us to determine how long it takes to once again reach the node start temperature of 0, assuming no further processing occurs in the meantime. From Function (1) we calculate the actual temperature after processing 4 time units, which yields 6.32. This actual temperature will never be higher than T_{max}, since c_i is determined by rounding down the inverse of Function (1), given T_{max}. We now calculate the inverse of Function (2), assume $b = 2$, with a temperature of 6.32. We round up the resulting number of time units of 8.37 to 9, to ensure that node i fully cools down.

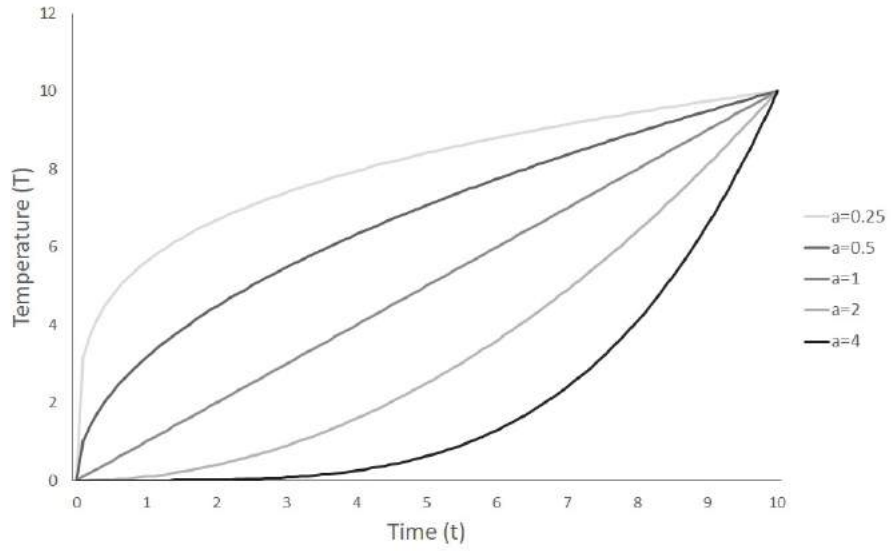


Fig. 1 Example temperature functions: the higher (lower) the value for a, the slower (faster) the initial T increase.

Table 1 Example of how a visit to node i can be split into multiple processing steps (PS), with waiting time in between the different steps to ensure node i cools down sufficiently.

PS	1	2	3
t_i	4	4	2
w_i	9	3	0

Recall, however, that the total p_i value of node i in the example was 10. One way to finish work on node i entirely is to process another 4 time units, cool down such that the final 2 time units can be processed, and then finish processing. For the example in Table 1, this leads to another waiting time of 3 time units, based on functions (1) and (2) (details are omitted for the sake of conciseness).

A summary of the way in which node i is processed in the example is given in Table 1, with t_i the selected processing time for the current processing step PS and w_i the waiting time after processing. Terminology wise, it is worth mentioning that in this example we use three *processing steps* for a single *visit* to *node* i.

The manner in which Functions (1) and (2), based on given a and b values, are employed, can then be summarized as follows:

1. Function (1) is used to model the temperature increase of a node i.
2. The maximum number of consecutive time units being processed (c_i) is calculated, based on the inverse of Function (1) and a given value for T_{max}.

3. Based on a processing time t_i, which is never larger than c_i, the resulting temperature is determined by Function (1).
4. This temperature value is then employed to calculate the required cooldown time to reach a node temperature of 0, based on the inverse of Function (2).
5. The interplay between Functions (1)-(2), and hence between a and b, determines what the best way is to minimize waiting time, for a given visit of a node i and a required processing time for that visit.
6. Different values for a and b allow for vastly different types of increase and decrease functions, and hence different types of applications.

3 Methodology

3.1 Metaheuristic framework

We employ the metaheuristic framework of Vidal et al. (2012, 2013) as starting point for our own algorithm. The hybrid genetic search with adaptive diversity control (HGSADC) metaheuristic is a hybrid metaheuristic framework, which combines the exploration elements of a classical genetic algorithm with both population diversity management and with efficient local search procedures. Given the intricate nature of the ITSP as discussed in Section 2, we have selected the HGSADC as metaheuristic framework, since it allows for a focus on local search techniques and deals with infeasible solutions, both of which we feel are crucial for the ITSP. Furthermore, the HGSADC achieved excellent results for the vehicle routing problems discussed in Vidal et al. (2012, 2013). Finally, in this abstract we focus, albeit briefly, on the components that we designed specifically for the ITSP, namely the solution representations and decoding procedure, since we have not deviated from the general structure of the HGSADC. An overview of the decoding procedure is provided in Figure 2.

3.2 Decoding procedure

Since a metaheuristic algorithm operates on a solution representation and not on a solution itself, this is a crucial algorithm component. Leyman et al. (2019) unambiguously showed that, albeit for a specific project scheduling problem, the choice of a solution representation deserves more attention than it in general receives in literature. The following two types of solution representations are used:

- Node list + processing time list (NL+PTL): A NL holds the order in which the nodes are to be visited, similar to a representation for the TSP, with the major difference that nodes may occur more than once, signifying multiple visits. The PTL holds the time processed during the current visit to the corresponding node in the NL (both the NL and PTL have the same length). A major downside of this PTL is that the sum of all values for a

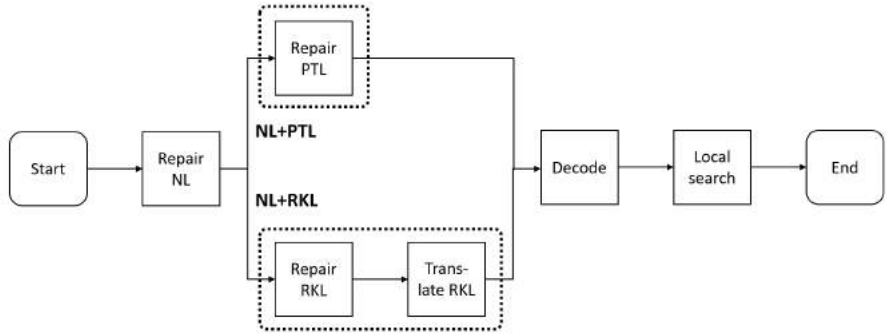


Fig. 2 Overview different steps decoding procedure.

given node i not only has to equal p_i, but also that each PTL value has to lie in the interval $[1; p_i]$. As a result, a repair operator is required as well.
- Node list + random key list (NL+RKL): The NL functions in the same manner as discussed earlier, but the general logic behind the RKL is that it contains only a floating point number between zero and one for each visit. Specifically, each value in the RKL corresponds with the fraction of the node's processing time to be completed during the current visit. This way, we only need to make sure that the sum of the RKL values for the same node equals one. We do, however, need a translation procedure when constructing a tour, to convert the RKL values to actual processing times.

4 Computational study

4.1 Test data

We generate test data based on the features in Table 2. Aside from the temperature constrainedness (TC), all features have been discussed in Section 2. The value for TC determines how tight the maximum temperature restriction is, given τ_i values for individual nodes. The value for T_{max} is calculated as follows: $T_{max} = \sum_{i=1}^{|V|} \tau_i \cdot$ TC $/|V|$. As a result, a low (high) value for TC leads to a low (high) T_{max} value as well. For each combination of instance features, 10 instances are generated, which results in a total dataset of 4320 instances.

4.2 Computational results

Table 3 shows a summary of the results for both the NL+PTL and NL+RKL combinations in terms of total time, total travel time and total waiting time. Each time, a stopping criterion of 5000 tours was used, to allow for a computer and code independent comparison (Leyman and De Causmaecker, 2017). Each cell in the table contains the average value over all 4320 test instances used.

Table 2 Instance features and feature values of test data.

Feature	Values
$\|V\|$	50, 100
p_i	$[1;20]$, $[1;100]$
τ_i	$[1;20]$, $[1;100]$
d_{ij}	$[1;20]$, $[1;100]$
TC	0.25, 0.5, 0.75
a	0.5, 1.0, 2.0
b	0.5, 1.0, 2.0

Table 3 Overview results for different solution representations (average).

	Total time	Total travel time	Total waiting time
NL+PTL	5030.93	1155.27	1567.95
NL+RKL	5036.09	1160.86	1567.51
Pham et al. (2020)	5647.83	1227.98	2112.13

Table 4 Temperature function results for different solution representations (average total time).

		NL+PTL	NL+RKL	Pham et al. (2020)
a	0.5	**7234.18**	7245.81	8587.31
	1	**4147.95**	4158.06	4467.74
	2	3710.66	**3704.40**	3888.44
b	0.5	3976.68	**3967.76**	4123.68
	1	**5050.12**	5057.89	5484.14
	2	**6066.00**	6082.62	7335.66

We also compare with the results of Pham et al. (2020), by including their decoding procedure in the HGSADC. This way, we can show that there is a clear contribution of taking the temperature functions into account in the proposed decoding procedures, since Pham et al. (2020) assume linear functions and do not include any mechanism to account for different types of functions.

Based on the results in Table 3, we can conclude that both NL+PTL and NL+RKL outperform the other case, but that the difference between themselves is small. The large improvement in average waiting time compared with Pham et al. (2020) shows that explicitly considering the temperature functions is worth its salt.

Table 4 provides more detailed results of the three alternatives, by splitting results based on the different a and b values used in the test design (Table 2). In general, the results are in line with those of Table 3, but we can see that regardless of a and b values there is an improvement compared to the decoding procedure of Pham et al. (2020). Hence, we can argue that our decoding procedure performs better in general, since linear functions are included in the results of Table 4 as well ($a = 1$, $b = 1$).

4.3 Critical remarks

Due to the results discussed above, we should more closely study the impact of travel versus processing times of a visit, since currently it seems that the algorithm prefers waiting time over travel time. We believe this is due to the travel times in the test data on average being of similar size as the processing times. This implies that if the average travel times are smaller than or similar to the average processing times, it is preferential to wait instead of moving to another node. We assume that as the travel times grow comparatively smaller, moving to another node instead will become the better choice, on average. But in order to be able to verify (or reject) this assumption, further testing with a more diverse test set is required.

5 Conclusions & future work

In this paper, we have discussed the intermittent traveling salesperson problem (ITSP) with generalized temperature functions. The ITSP is an extension of the well-known traveling salesperson problem (TSP), but in which each node has to be processed for a given duration and not just visited. Processing a node increases the node temperature, which should not exceed a maximum value. As a result, the ITSP may require multiple visits to nodes, unlike the TSP, where a single visit is imposed. Additionally, we have proposed general polynomial functions, which determine the manner in which node temperature increases and decreases.

To solve the ITSP with generalized temperature functions, we have employed the hybrid genetic search with adaptive diversity control of Vidal et al. (2012, 2013) as metaheuristic framework, and focused our contribution on solution representation alternatives and a decoding procedure.

In the future, we will study specific applications of the ITSP, closely linked to real-world problems to demonstrate the ITSP's practical relevance. Alternatively, we aim to consider heat transfer functions from physics and thermodynamics, which differ from the presently modeled cooling schemes for e.g. the metal surface processing application discussed earlier.

From a methodology and testing point of view, we will further investigate the decoding procedures, to better grasp the ITSP's intricacy, and to include additional solution representations. The test design should also be expanded to allow for networks of different (and larger) sizes, alongside a greater degree of variation in the heating and cooldown feature values. The algorithm's parameters should be tuned using an automatic algorithm configurator (e.g., SMAC, ParamILS, irace) rather than using the default values of Vidal et al. (2012, 2013) and ad-hoc values for additional parameters. Finally, the results should be analyzed in more detail, to allow for a clear evaluation of the contribution of each algorithm component, and to determine for which (type of) instances which solution representation leads to the best results.

References

Leyman P, De Causmaecker P (2017) Evaluation of metaheuristics: Is computation time worth the time? In: Proceedings of the 31st annual meeting of the Belgian Operational Research Society (ORBEL), pp 89–90

Leyman P, Van Driessche N, Vanhoucke M, De Causmaecker P (2019) The impact of solution representations on heuristic net present value optimization in discrete time/cost trade-off project scheduling with multiple cash flow and payment models. Computers and Operations Research 103:184–197

Pham TS, Leyman P, De Causmaecker P (2020) The intermittent travelling salesman problem. International Transactions in Operational Research 27:525–548

Vidal T, Crainic T, Gendreau M, Lahrichi N, Rei W (2012) A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. Operations Research 60(3):611–624

Vidal T, Crainic T, Gendreau M, Prins C (2013) A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. Computers and Operations Research 40:475–489

Solving the Production Leveling Problem with Order-Splitting and Resource Constraints

Johannes Vass · Nysret Musliu · Felix Winter

Abstract We investigate an extended problem formulation of the Production Leveling Problem (PLP), which was recently introduced in the literature. For the PLP problem the task is to assign orders to production periods such that the load is balanced, capacity limits are not exceeded and the order's priorities are considered. The extended problem (PLP-OSRC) introduced in this paper additionally includes order-splitting, resource constraints and due dates. We provide a mixed integer programming formulation for the PLP-OSRC based on the existing model for the PLP and evaluate it with a state-of-the-art MIP solver. To solve practically sized instances we apply a local search approach based on simulated annealing and propose two innovative neighborhood moves. We compare our approaches on two sets of randomly generated instances and show that the simulated annealing approach provides competitive results to MIP for the smaller instances. Moreover, it provides good solutions for very large instances that could not be solved by our MIP model in a reasonable amount of time.

Keywords Production Leveling, Simulated Annealing, MIP

1 Introduction

As many modern-day factories in the area of industrial manufacturing migrate towards full automation, a strong need for efficient automated production planning systems becomes more and more apparent. Although many known practical planning problems deal with short-term scheduling and long-term planning tasks, there is also an important need for mid-term planning systems

Johannes Vass, Nysret Musliu and Felix Winter
Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling
DBAI, TU Wien
E-mail: {jvass,musliu,winter}@dbai.tuwien.ac.at

that aim to efficiently distribute orders created by long-term planning systems into smaller short-term scheduling problems.

Recently, we have introduced such a mid-term planning problem called the production leveling problem (PLP) in [11,10]. The PLP takes a number of jobs as its input and aims to evenly distribute the production orders in an optimized plan over a given planning horizon. Finding a balanced distribution of the workload over the set of production periods is mainly motivated by the idea that solutions to the PLP will lead to improved short-term schedules that efficiently utilize the production capacities in each period and to encourage just-in-time manufacturing.

Applications for the PLP arise in different areas of the industry. For example, a practical application of the PLP has been deployed by our industry partners in electronic component manufacturing, where it is desired to assign a well balanced product mix to each production period. As setup costs that arise between the manufacturing of different product families are not too high in this case, a well balanced product mix leads to increased capacity utilization as well as decreased storage- and transport costs as just-in-time production in the sense of heijunka [3] is encouraged. Several practical instances for this application have been introduced and are available[1].

Leveling problems similar to the PLP have been studied in other application domains in the past like for example the balanced academic curriculum problem [2,4], nurse scheduling [7,9,8] or assembly line balancing [1]. However, in contrast to other leveling problems the PLP includes the consideration of order priorities in its objective function, which is of high importance for practical production planning problems.

Previously, we have shown that the PLP is NP-hard and have proposed metaheuristic approaches as well as an exact approach based on mixed integer programming in [11]. Although these existing solution methods can be used to approach practically sized instances of the problem, some real life mid-term planning scenarios cannot be tackled with the standard PLP problem formulation, as it does not consider the availability of resources used during production. Furthermore, the original specification of the PLP does not allow to split given orders into multiple sub orders, which can in some scenarios improve the quality of solutions.

In this paper, we therefore introduce an extended problem formulation of the PLP that supports order-splitting and the consideration of resource constraints called the production levelling problem with order-splitting and resource constraints (PLP-OSRC). In addition to providing a formal specification to the problem, we extend the mixed integer programming formulation for the PLP to model the PLP-OSRC, which allows us to approach instances of the extended problem with state-of-the-art mixed integer programming solvers. Furthermore, we propose two innovative neighborhood moves that we utilize within a metaheuristic approach based on simulated annealing to solve very large realistically sized instances of the PLP-OSRC. Finally, we implement

[1] `https://www.dbai.tuwien.ac.at/staff/jvass/production-leveling/`

all of our proposed methods and perform a large number of experiments to empirically evaluate the performance of the investigated solution approaches.

The remainder of the paper is structured as follows: Section 2 briefly reviews the formal problem specification of the PLP and then gives a detailed formal specification of the PLP-OSRC. Afterwards, we propose the integer programming formulation for the extended problem and describe the details absout the metaheuristic approach and the novel neighborhood moves in Section 3. In Section 4 we give an overview of all conducted experiments and discuss computational results before we give concluding remarks in Section 5.

2 Problem Statement

In this section, we provide a description of the production leveling problem with resource constraints and order-splitting (PLP-OSRC), which is a real-life industrial planning problem that is concerned with evenly distributing a set of orders over a planning horizon.

Recently, we introduced the production leveling problem (PLP) in [11]. The main differences between the PLP and the extended problem, that we describe in this paper, are that given orders are allowed to be split into multiple parts and that additional resource constraints can be defined for the PLP-OSRC.

In the following, we first review the problem description for the PLP in Section 2.1, before we later in Section 2.2 introduce the notion of order splits and additional constraints of the extended problem.

2.1 The Production Leveling Problem

The input to the PLP contains a list of orders to be distributed over the planning horizon, where each order defines a number of demanded items of a particular product type that need to be produced. Furthermore, the importance of each order is defined by a given priority value.

The goal of the problem is to assign each order to a single period in the given production horizon. A feasible solution needs to make sure that given maximum production volumes for each of the periods are not exceeded, where each period defines an overall maximum production volume and product type specific maximum production volumes.

The following lists the formal parameters and decision variables to an instance of the PLP:

Input Parameters

$K = \{1, \ldots, k\}$	Set of orders, where k is the number of orders
$M = \{1, \ldots, m\}$	Set of product types, where m is the number of product types
$N = \{1, \ldots, n\}$	Set of periods, where n is the number of periods
$c \in \mathbb{R}^+$	the maximum overall production volume per period
$c_t \in \mathbb{R}^+$	for each product type $t \in M$ the maximum production volume per period
$d_j \in \mathbb{Z}^+$	for each order $j \in K$ its associated demand
$p_j \in \mathbb{Z}^+$	for each order $j \in K$ its associated priority
$t_j \in \mathbb{Z}^+$	for each order $j \in K$ the product type
$d^* \in \mathbb{Z}^+$	the target production volume per period, i.e. $\frac{1}{n} \sum_{j \in K} d_j$
$d_t^* \in \mathbb{Z}^+$	the target production volume per period for each product type $t \in M$, i.e. $\frac{1}{n} \sum_{j \in K \mid t_j = t} d_j$

Variables

- A variable y_j for each order $j \in K$ determines in which period the order shall be produced:

$$y_j \in N \quad \forall j \in K$$

- The total production volume for each period is stored in auxiliary variables $w_i \quad \forall i \in N$:

$$w_i = \sum_{\substack{j \in K: \\ y_j = i}} d_j \quad \forall i \in N$$

- The total production volume for each product type and period is stored in auxiliary variables $w_{i,t} \quad \forall i \in N, t \in M$:

$$w_{i,t} = \sum_{\substack{j \in K: \\ y_j = i \wedge t_j = t}} d_j \quad \forall i \in N, t \in M$$

Hard Constraints

The following hard constraints impose restrictions on the maximum production volumes for each period in the planning horizon:

- H1: The limit for the overall production volume is satisfied for each period:

$$\forall i \in N \quad w_i \leq c$$

- H2: The limit for the production volume of each product type is satisfied for each period:

$$\forall i \in N, t \in M \quad w_{i,t} \leq c_p$$

Objective Function

A multi-objective function which includes three objectives determines the quality of solutions to the PLP. Intuitively, an optimized production plan should assign orders to periods such that the production volume is balanced between the periods while trying to adhere to the production sequence which is implied by the order's priorities.

Therefore, the objectives for the PLP are defined as follows:

1. Minimize the sum of deviations of the planned production volume to the average demand (i.e. the target value d^*) for each period, ignoring the product types.

$$f_1 = \sum_{i \in N} |d^* - w_i| \tag{1}$$

$$\tag{2}$$

2. Minimize the sum of deviations of the production volume of each product type to its respective mean target value d_t^*, making sure that the production of each product type is being leveled.

$$f_2 = \sum_{t \in M} \left(\frac{1}{d_t^*} \cdot \sum_{i \in N} |d_t^* - w_{i,t}| \right) \tag{3}$$

$$\tag{4}$$

3. Minimize the number of times a higher prioritized order is planned for a later period than a lower prioritized order, which we call a priority inversion. This objective makes sure that it costs less to plan the production of more important orders for earlier periods.

$$f_3 = \left| \left\{ (i,j) \in K^2 : y_i > y_j \text{ and } p_i > p_j \right\} \right| \tag{5}$$

In order to combine the objectives (f_1, f_2, f_3) into a single objective function each of the individual cost components is normalized as follows:

$$g_1 = \frac{1}{n \cdot d^*} \cdot f_1 \tag{6}$$

$$g_2 = \frac{1}{n \cdot m} \cdot f_2 \tag{7}$$

$$g_3 = \frac{2}{k \cdot (k-1)} \cdot f_3 \tag{8}$$

The normalization ensures that g_1 and g_2 stay between 0 and 1 with a high probability. Only for degenerated instances, where even in good solutions the target is exceeded by factors ≥ 2 higher values are possible for g_1 and g_2. The value of g_3 is guaranteed to be ≤ 1 because the maximum number of inversions in a permutation of length k is $k \cdot (k-1)/2$.

The final objective function of the PLP is a weighted sum of the three normalized objective functions, where user defined weights a_{1--3} determine the relative importance of each objective.

$$\textbf{minimize } g = a_1 \cdot g_1 \; + \; a_2 \cdot g_2 \; + \; a_3 \cdot g_3 \tag{9}$$

Figure 1 shows a small example instance of the PLP with five orders, which are shown as boxes, where the box height corresponds to the order size. The orders should be distributed over three production periods such that the distances of the stacks of orders and the dashed target line is minimized and no stack crosses the red line which represents the overall maximum production volume.

Similar to the first example, Figure 2 shows a solution for a small example instance with five orders. The numbers inside the orders this time determine the order priorities, where a larger number indicates a higher importance. In this case, it is undesirable that the red order is assigned to an earlier period than the yellow or blue order. Whenever a pair of two orders is not planned so that their priority values are descending over time they cause a priority inversion in the production plan. The third objective of the PLP aims to minimize the total number of priority inversions. In the example, a better solution could be obtained by swapping the red order with the yellow one, because it would eliminate both priority inversions.

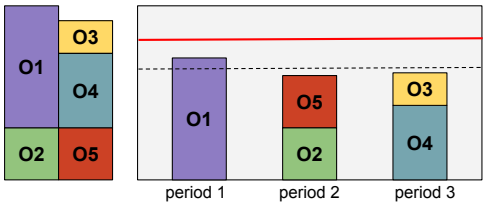


Fig. 1: Example visualizing the effects of the first objective f_1 that aims to minimize the total deviation to the target value (i.e. the dashed line). The solution shown in this example is optimal w.r.t. this objective.

2.2 The Production Leveling Problem with Order-Splitting and Resource Constraints

The PLP as we described it in the previous section appears in many real-life industrial applications where production volumes are ought to be leveled evenly over the planning horizon to keep the production process robust and efficient. However, as it assumes that each order is indivisible, it cannot be used in any practical context where single customer orders can actually be distributed over

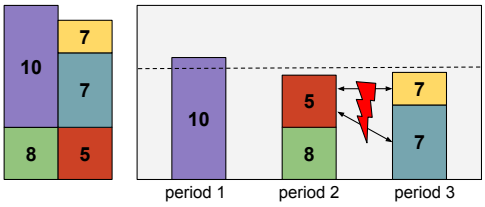


Fig. 2: Example solution that contains two priority inversions. The numbers in each box determine the priority value of the associated order.

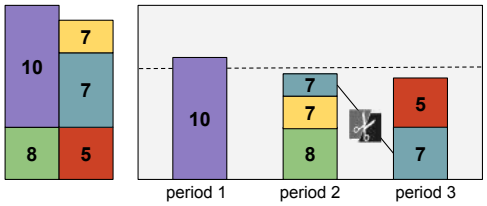


Fig. 3: Example solution, where the blue order (with priority 7) is split between period 2 and 3. This way, we do not introduce any priority inversion and nevertheless get a similarly good leveling as in the example before.

multiple periods in the planning horizon. Figure 3 shows an example why splitting orders can be sometimes useful: The additional flexibility that is obtained by allowing orders to be split can help to create solutions that are good both in terms of levelness and prioritization. Another shortcoming of the basic PLP in practical environments is the absence of resource constraints, which prevents us, for example, to take staffing and the availability of tools or machines into account while distributing orders over production periods.

In this section, we therefore introduce a novel extension of the production leveling problem that we call the production leveling problem with order-splitting and resource constraints (PLP-OSRC). Instances to the PLP-OSRC define the same parameters as specified for the PLP in Section 2.1, but include additional parameters that determine how many splits per order are feasible and provide the parameters about the additional constraints. The following lists the additional input parameters for the PLP-OSRC:

Input Parameters

$R = \{1, \ldots, o\}$	Set of secondary resources where o is the number of secondary resources
$pd_j^{\min} \in N$	the earliest period to which an order $j \in K$ can be assigned without a penalty
$pd_j^{\max} \in N$	the latest period to which an order $j \in K$ can be assigned without a penalty
$ps_j^{\min} \in \{1, \ldots, d_j\}$	the minimum size of any partition of order $j \in K$
$pc_j^{\max} \in \mathbb{Z}^+$	the maximum number of partitions of order $j \in K$
$ru_{j,r} \in \mathbb{R}_0^+$	the amount of usage of secondary resource $r \in R$ by order $j \in K$
$ru_r^{\min} \in \mathbb{R}^+$	the minimum penalty-free usage of resource $r \in R$ in each period
$ru_r^{\max} \in \mathbb{R}^+$	the maximum penalty-free usage of resource $r \in R$ in each period

Since a single order can be split and planned into multiple periods, the decision variables of the PLP-OSRC have to capture additional information than for the PLP. The following list defines the variables for the PLP-OSRC:

Variables

- Variables $x_{i,j}$ determine the amount of order j which is planned to be produced in period i. If a variable $x_{i,j} > 0$, we say that a partition of order j is planned in period i.

$$x_{i,j} \in \mathbb{Z}_0^+ \quad \forall i \in N, j \in K$$

- Auxiliary variables y_j^{start} and y_j^{end} determine the periods where the first and last partition of an order j are planned:

$$y_j^{start} = \min(\{i \in N \mid x_{i,j} > 0\}) \quad \forall j \in K$$

$$y_j^{end} = \max(\{i \in N \mid x_{i,j} > 0\}) \quad \forall j \in K$$

- The production volume for each period is stored in auxiliary variables w_i:

$$w_i = \sum_{j \in K} x_{i,j} \quad \forall i \in N$$

- The production volume for each product type and period is stored in auxiliary variables $w_{i,t}$:

$$w_{i,t} = \sum_{\substack{j \in K: \\ t_j = t}} x_{i,j} \quad \forall i \in N, \forall t \in M$$

- Auxiliary variables $u_{i,r}$ capture the total usage of secondary resources in each of the planning periods, where the resource usage of a single order partition is determined relative to the total order usage.

$$u_{i,r} \in \mathbb{R}_0^+ = \sum_{j \in K} ru_{j,r} \cdot \frac{x_{i,j}}{d_j} \quad \forall i \in N, r \in R$$

Hard Constraints

In addition to the two production volume hard constraints from the PLP, the PLP-OSRC defines another two hard constraints that restrict the minimum partition size and the maximum partition count of each order:

- H3: The minimum partition size is reached for every partition of every order:

$$\forall i \in N, j \in K \quad x_{i,j} = 0 \ \vee \ x_{i,j} \geq ps_j^{\min}$$

- H4: The maximum partition count is not exceeded for any order:

$$j \in K \quad |\{i \in N \mid x_{i,j} > 0\}| \leq pc_j^{\max}$$

Objective Function

The three objectives f_1, f_2, f_3 defined in Section 2.1 are also used in the multi-objective function of the PLP-OSRC. However, f_3 has to be slightly reformulated to be compatible with the novel variable definitions. Furthermore, two new objectives f_4 and f_5 influence the quality of solutions to the PLP-OSRC depending on the earliness/lateness of orders and the over- and under-utilization of resources.

The following defines objectives f_3, f_4, f_5 for the PLP-OSRC:

- Function f_3 counts the number of priority inversions in the assignment where they are redefined to handle order splits. That is, f_3 counts the number of order-pairs (i, j) for which i has a higher priority than j but i finishes only after j starts.

$$f_3 = \left| \left\{ (i,j) \in K^2 : p_i > p_j \text{ and } y_i^{end} > y_j^{start} \right\} \right| \tag{10}$$

$$\tag{11}$$

- The objective function f_4 calculates a penalty for every order whose first partition is planned before the order's minimum period or whose last partition is planned after the order's maximum period.

$$f_4 = \sum_{j \in K} \left(\max(pd_j^{\min} - y_j^{start}, 0) \ + \ \max(y_j^{end} - pd_j^{\max}, 0) \right) \tag{12}$$

$$\tag{13}$$

- Objective f_5 calculates a penalty for over-usage and under-usage of secondary resources.

$$f_5 = \sum_{r \in R} \sum_{i \in N} \begin{cases} 1 - \frac{u_{i,r}}{ru_r^{\min}} & \text{if } u_{i,r} < ru_r^{\min} \\ \frac{u_{i,r}}{ru_r^{\max}} - 1 & \text{if } u_{i,r} > ru_r^{\max} \\ 0 & \text{otherwise.} \end{cases} \tag{14}$$

Whereas objectives f_1, f_2, f_3 can be normalized as specified in Section 2.1, objectives f_4 and f_5 are normalized as follows:

$$g_4 = \frac{1}{n \cdot k} \cdot f_4 \tag{15}$$

$$g_5 = \frac{1}{n \cdot o} \cdot f_5 \tag{16}$$

$$\tag{17}$$

Objective g_4 applies normalization through a division by the number of orders times the number of periods. As the penalty for each order is at most k, the normalized objective is also guaranteed to be ≤ 1. The resource objective g_5 is normalized by the number of periods and resources which normally also leads to values between 0 and 1, however the upper bound is not strict. When looking at instances without secondary resources, g_5 must not be considered in the objective function because it would yield a division by zero.

The final objective function for the PLP-OSRC is the following weighted sum (with user defined weights $a_1 - a_5$).

$$\textbf{minimize } g = a_1 \cdot g_1 \ + \ a_2 \cdot g_2 \ + \ a_3 \cdot g_3 + \ a_4 \cdot g_4 + \ a_5 \cdot g_5 \tag{18}$$

3 Solution Approaches

In the previous section we have provided an in depth problem definition of the PLP-OSRC. In this section, we first propose an integer programming formulation of the problem in Section 3.1 before we later describe a metaheuristic local search based solution approach in Section 3.2

3.1 Integer Programming Model

Previously, we proposed an integer programming model for the PLP [11]. In this section we extend that model for the PLP-OSRC, based on the formal problem description that we specified in Section 2. We carry over the input parameters without changes, which is why they are not repeated in this section. In contrast to the integer programming model for the PLP, th e The former binary decision variables x_{ij} are converted to an integer domain in order to model the planned production amount for order j in period i and thus account for order splits. Furthermore, the model for the PLP-OSRC The variables and detailed formulation are as follows:

Variables

$x_{ij} \in \mathbb{Z}^+$	for each $i \in N$, $j \in K$ stating how much demand of order j is planned in period i
$\hat{x}_{ij} \in \{0,1\}$	for each $i \in N$, $j \in K$ stating whether a partition exists in period i. A partition exists for order j in period i iff $x_{ij} > 0$.
$y_j^{start} \in N$	for each order $j \in K$ the period assignment of its first partition
$y_j^{end} \in N$	for each order $j \in K$ the period assignment of its last partition
$z_{ij} \in \{0,1\}$	for orders $i, j \in K$ where $p_i > p_j$, existence of a priority inversion between i and j
$s_i^+ \in \mathbb{R}^+$	for each $i \in N$ the surplus production volume for period i
$s_i^- \in \mathbb{R}^+$	for each $i \in N$ the missing production volume for period i
$s_{it}^+ \in \mathbb{R}^+$	for each $i \in N$, $t \in M$ the surplus production volume for period i and product type t
$s_{it}^- \in \mathbb{R}^+$	for each $i \in N$, $t \in M$ the missing production volume for period i and product type t
$u_{ir}^+ \in \mathbb{R}^+$	for each $i \in N$, $r \in R$ the amount of over-usage of resource r in period i
$u_{ir}^- \in \mathbb{R}^+$	for each $i \in N$, $r \in R$ the amount of under-usage of resource r in period i
$v_j^{start} \in \mathbb{Z}^+$	for each $j \in K$ the amount of violation of the earliest period soft constraint
$v_j^{end} \in \mathbb{Z}^+$	for each $j \in K$ the amount of violation of the latest period ($\widehat{=}$ due date) soft constraint

Formulation

$$\min \quad a_1 g_1 + a_2 g_2 + a_3 g_3 + a_4 g_4 + a_5 g_5 \tag{19}$$

s.t.

$$\sum_{i \in N} x_{ij} = d_j \qquad j \in K \tag{20}$$

$$x_{ij} \leq d_j \cdot \hat{x}_{ij} \qquad j \in K \tag{21}$$

$$y_j^{start} \leq i + (n-1) \cdot (1 - \hat{x}_{ij}) \qquad j \in K, i \in N \tag{22}$$

$$y_j^{end} \geq i \cdot \hat{x}_{ij} \qquad j \in K, i \in N \tag{23}$$

$$\sum_{j \in K} x_{ij} + s_i^+ - s_i^- = d^* \qquad i \in N \tag{24}$$

$$\sum_{j \in K | t_j = t} x_{ij} + s_{it}^+ - s_{it}^- = d_t^* \qquad i \in N,\ t \in M \tag{25}$$

$$d^* + s_i^+ \leq c \qquad i \in N \tag{26}$$

$$d_t^* + s_{it}^+ \leq c_t \qquad i \in N,\ t \in M \tag{27}$$

$$x_{ij} \geq ps_j^{\min} \cdot \hat{x}_{ij} \qquad j \in K \tag{28}$$

$$\sum_{i \in N} \hat{x}_{ij} \leq pc_j^{\max} \qquad j \in K \tag{29}$$

$$y_i^{end} - y_j^{start} \leq (n-1) z_{ij} \qquad i, j \in K \mid p_i > p_j \tag{30}$$

$$v_j^{start} \geq pd_j^{\min} - y_j^{start} \qquad j \in K \tag{31}$$

$$v_j^{end} \geq y_j^{end} - pd_j^{\max} \qquad j \in K \tag{32}$$

$$\sum_{j \in K} ru_{j,r} \cdot \frac{x_{ij}}{d_j} - u_{ir}^+ \leq ru_r^{\max} \qquad i \in N, r \in R \tag{33}$$

$$\sum_{j \in K} ru_{j,r} \cdot \frac{x_{ij}}{d_j} + u_{ir}^- \geq ru_r^{\min} \qquad i \in N, r \in R \tag{34}$$

$$y_i^{end} \leq y_j^{start} \qquad i, j \in S, S \subseteq K \mid p_i \geq p_j, d_i = d_j, t_i = t_j \tag{35}$$

$$\sum_{t \in M} (s_{it}^- - s_{it}^+) = s_i^- - s_i^+ \qquad i \in N \tag{36}$$

$$g_1 = \frac{1}{n \cdot d^*} \cdot \sum_{i \in N} (s_i^+ + s_i^-) \tag{37}$$

$$g_2 = \frac{1}{n \cdot m} \cdot \sum_{t \in M} \left(\frac{1}{d_t^*} \cdot \sum_{i \in N} (s_{it}^+ + s_{it}^-) \right) \tag{38}$$

$$g_3 = \frac{2}{k \cdot (k-1)} \cdot \sum_{i,j \in K} z_{i,j} \tag{39}$$

$$g_4 = \frac{1}{2k} \cdot \sum_{j \in K} \left(v_j^{start} + v_j^{end} \right) \tag{40}$$

$$g_5 = \frac{1}{n \cdot o} \cdot \sum_{r \in R} \sum_{i \in N} \left(\frac{u_{ir}^-}{ru_r^{\min}} + \frac{u_{ir}^+}{ru_r^{\max}} \right) \tag{41}$$

Constraints (20) to (25) link auxiliary variables to the decision variables. Constraint (20) ensures that the total demand of all partitions of j equals the order's demand d_j. Constraint (21) links the binary variables $\hat{x}_{ij}$ to the decision variables x_{ij} such that $x_{ij} > 0 \rightarrow \hat{x}_{ij} = 1$. Constraints (22) and (23) link the x_{ij} to the y_i^{start} and y_i^{end} variables so that they always hold the period assignment of the first and last period of any partition of an order, respectively. Constraint (24) states for each period that the total planned production volume plus the surplus minus the slack equals the target d^*. As both variables have positive domains and they are subject to minimization, at most one of them will be non-zero in any optimal solution. Constraint (25) repeats this relationship over the variables s_{it}^+ and s_{it}^- for each product type t.

The block of constraints between (26) and (29) models the problem's hard constraints. Constraint (26) ensures that the capacity bound per period is satisfied by enforcing that the sum of target demand d^* and the surplus variable s^+ does not exceed the threshold. Analogously, Constraint (27) enforces the capacity limit per period and product type. Constraint (28) enforces the minimum partition size and (29) the maximum number of partitions into which an order may be split.

Constraints (30) to (34) populate penalty variables for the objective function. Constraint (30) links the y_i^{start} and y_i^{end} to the $z_{i,j}$ variables which track the number of priority inversions. It makes sure that for every pair of orders i, j where i has a higher priority than j, z_{ij} is 1 (representing a priority inversion) if order i finishes after order j starts. The constraints (31) and (32) force the variables v_j^{start} and v_j^{end} to keep track of the violations of the allowed assignment range. Constraints (33) and (34) force u_{ir}^+ and u_{ir}^- to contain the amount of over-usage and under-usage of resource r in period i. That is achieved by comparing with the amount of planned resource usage which is given by the first summand.

Finally, there are two redundant constraints for strengthening the formulation: Constraint (35) enforces a dominance relation for all pairs of orders which have the same product type and demand value. The constraint requires that the higher prioritized order ends not later than the lower prioritized one starts, which is sensible because otherwise we could swap the two orders to obtain a better solution. This cuts off parts of the search space where the optimal solution cannot reside. Constraint (36) links the $s_i^{\{+,-\}}$ and $s_{it}^{\{+,-\}}$ variables together, which also leads to improvements in the average run-time.

The objective function is equivalent to the one presented in Section 2.2 but here it is stated on the variable set of the MIP formulation. In g_1, the sum of the slack and surplus variable $(s_i^+ + s_i^-)$ is equivalent to the absolute difference between planned demand and target demand $|d^* - w_i|$, because at least one of s_i^+ and s_i^- will be 0 in any optimal solution and the other one holds the absolute difference. The same holds true for the analogous variables in g_2. Similarly, instead of using one variable per period to track the resource usage, the two separate variables for over-usage and under-usage u_{ir}^+ and u_{ir}^- are used to compute the cost component g_5.

3.2 Metaheuristic Approach

The IP formulation we proposed in the previous section can be used to solve instances of the PLP-OSRC to optimality. However, solving realistically sized instances with a large number of variables may often not be possible for an IP solver within reasonable run-time and memory limitations. Therefore, we additionally propose a local search based metaheuristic solution approach to the PLP-OSRC in this section.

Previously, we have investigated a construction heuristic and a local search approach using simulated annealing for the PLP in [11] and proposed basic neighborhood moves that can reposition single orders and swap pairs of orders.

In this section, we extend the two neighborhoods previously proposed for the PLP and furthermore propose two innovative search neighborhoods for the PLP-OSRC

3.2.1 Neighborhood Relations

We propose four neighborhood relations when approaching the PLP-OSRC with local search: The first two are extensions of the search neighborhoods to the PLP that have been previously proposed in [11]. Basically, instead of moving and swapping complete orders, the new versions move and swap single order partitions. The third neighborhood splits and merges order partitions and is able to exchange demands between partitions of an order. Finally, the fourth neighborhood shifts all partitions of a single order at once.

In the following, we describe the neighborhood operators in detail:

Move-Partition Neighborhood The move-partition neighborhood of a solution s consists of all solutions s' whose only difference to s is that one partition of some order has been moved to a different period. When splits are disallowed and thus every order has exactly one partition, this is equivalent to the move-order neighborhood of the basic PLP version. When generating random moves for this neighborhood, we uniformly sample the order and the partition as well as the target period.

Swap-Partitions Neighborhood The swap-partitions neighborhood of a solution s consists of all solutions s' whose only difference to s is that two order partitions of different orders, which are not assigned to the same period in s appear with swapped period assignments in s'. In random neighborhood generation, the partitions to be swapped are chosen uniformly at random among all pairs of partitions of different orders, which are not already assigned to the same period.

Split Neighborhood The split neighborhood of a solution s consists of all solutions s' which differ from s only with respect to one order, where the possible changes are:

- One partition of that order is split into two parts and the new part is moved to some other period.
- Two partitions of that order are merged.
- The sizes of two partitions of that order are changed such that the total size of the two partitions together stays the same and both partitions remain non-empty. In other words, demands are moved from one partition to another, which is potentially in a different period.

When sampling split moves randomly, first any single order is chosen randomly. Then, the move is generated such that the three options described above are equally likely.

Shift-order neighborhood The shift-order neighborhood of a solution s consists of all solutions s' which differ from s only with respect to a single order, whose partitions all have been shifted i periods to the left or to the right. If a partition cannot be shifted any more because it is already in the first or last period, it remains in that period. When sampling random shift-order moves, an order is chosen uniformly at random and i is chosen from $\{-1, 1\}$.

3.2.2 Simulated Annealing

Similar as previously described for the PLP in [11], we use a simulated annealing metaheuristic to approach the PLP-OSRC in this paper. Simulated annealing was first introduced in [5] and resembles the physical process of annealing in metallurgy. The basic idea is to iteratively apply randomly generated neighborhood moves to an initial solution to the problem. Whether a neighborhood move is accepted, depends on the resulting solution quality and a temperature parameter which is cooled down over the course of the search process.

Algorithm 1 shows the detailed simulated annealing procedure:

The acceptance function uses the metropolis criterion [5], where the probability $P(i \Rightarrow j)$ to accept a move from solution i to solution j is defined as follows ($f(x)$ is the objective function):

$$P(i \Rightarrow j) = \begin{cases} 1, & \text{if } f(j) \leq f(i). \\ \exp\left(\frac{f(i)-f(j)}{t}\right), & \text{otherwise.} \end{cases} \tag{42}$$

We further use a geometric cool down scheme:

$$t_i = \alpha \cdot t_{i-1} \tag{43}$$

The simulated annealing procedure shown in 1 relies on a number of parameters and an initial solution. We use the construction heuristic that was previously proposed in [11] to produce an initial solution, and set $\mathcal{N}_{1-4}$ to the search neighborhoods we proposed in this section. The remaining parameters have to be carefully selected depending on the computational environment. We describe the tuning of these parameters in Section 4.

Algorithm 1: Simulated Annealing

Data: *initialSolution*, neighborhoods $\mathcal{N}_i$ with probabilities p_i, $t_{\max}$, $t_{\min}$, iterations per temperature w, *timeLimit*, *iterationLimit*
Result: a solution at least as good as *initialSolution*

```
currentSolution ← initialSolution;
bestSolution ← currentSolution;
t ← t_max;
while t ≥ t_min and ¬ time limit reached and ¬ iteration limit reached do
    foreach j ∈ 1, ..., w do
        N ← choose one of neighborhoods N_i according to probabilities p_i;
        m ← select a random move out of N(currentSolution);
        if Accept(m, t) then
            currentSolution ← Apply(m, currentSolution);
            if currentSolution is better than bestSolution then
                bestSolution ← currentSolution;
            end
        end
    end
    t ← Cool-Down(t);
end
return bestSolution;
```

4 Experimental Evaluation

In this section, we provide a detailed description of our experimental environment and give an overview of our conducted experiments. We first describe an instance generator for the PLP-OSRC in 4.1, that we use to generate a large number of benchmark instances for our experiments. Later in Section 4.2, we provide the details on how parameters for the simulated annealing algorithm have been selected. Finally, we describe our computational environment and discuss the final experimental results in sections 4.3 and 4.4.

4.1 Instance Generation

To generate instances for the PLP-OSRC, we extend the random instance generator we previously proposed in [11] for the PLP with additional input parameters regarding partitioning, resources and due dates.

Given a number of orders k, periods n, product types m and resources o, the heuristic constructs an initial solution with the following steps:

1. Partition the number of orders k into m parts $o_1 \dots o_m$, one for each product type.
2. Randomly choose the maximum priority of all orders $p_{\max} \in \{1, \dots, 10\}$
3. For each product $t \in M$, create a set of demands D_t and randomly select the size of the set between 1 to 50. Then, insert the corresponding number of items into the set, where each item is a randomly selected value $d \in \{1, \dots, random(1000-5000)\}$ (the upper bound is a random value between 1000–5000 which is recalculated for each product type).

4. To determine resource usages for each product type, first randomly select a probability between 0.0 and 1.0. Then, based on this probability decide for each resource whether or not it is used for the product type. If it turns out a resource is used for the product type, randomly select a usage value between 0.0 and 1.0. This usage value then determines the resource usage per unit of demand for the particular product type.
5. For each product $t \in M$, generate o_t orders, where each order is generated as follows:
   (a) Randomly choose the demand d_j from the set D_t.
   (b) Randomly choose the priority from $\{1, \ldots, p_{\max}\}$.
   (c) The usage of each resource is calculated by multiplying the order demand d_j with the previously chosen usage factor. The resulting value is rounded up.
   (d) Randomly choose the earliest start $pd_j^{\min}$ from $\{1, \ldots, \lfloor 3/4 \cdot n \rfloor\}$.
   (e) Randomly choose the latest end $pd_j^{\max}$ from $\{pd_j^{\min} + 1, \ldots, n\}$.
   (f) Randomly choose the maximum partition count $pc_j^{\max}$ from $\{1, \ldots, 10\}$.
   (g) Randomly choose the minimum partition size $ps_j^{\min}$ from $\{1, \ldots, \lceil 1/2 \cdot d_j \rceil\}$[2]
6. In order to set the capacity limit c, we first calculate the target demand d^* as $\sum_{j \in K} {}^{d_j}/_n$. The capacity limit c is then derived from the d^* by multiplying with a random value from the normal distribution $\sigma(1.1, 0.02)$. Hence, c is in the expected case 10% larger than d^*. The capacity limits c_t for $t \in M$ are chosen analogously.
7. The minimum and maximum resource usages per period are calculated similarly: First, for each resource r, the average usage per period $\bar{u}_r$ is calculated. Then, the maximum deviation percentage $d_{\max}$ is drawn from the normal distribution $\sigma(0.1, 0.02)$, i.e. 10% on average. Finally, the minimum resource usage $ru_r^{\min}$ is set to $(1 - d_{\max}) \cdot \bar{u}_r$ and the maximum resource usage $ru_r^{\max}$ is set to $(1 + d_{\max}) \cdot \bar{u}_r$.

Using the instance generation procedure, we generated a total of 986 realistically sized large instances for our experiments. The following parameters were sampled uniformly at random: The number of orders k is chosen from $100 \ldots 4000$, the number of periods n from $2 \ldots 80$, the number of products m from $1 \ldots 20$ and the number of resources o from $1 \ldots 5$.

Additionally, we generated another set of 20 smaller instances, using the following random parameters: The number of orders k is chosen from $10 \ldots 100$, the number of periods n from $5 \ldots 10$, the number of products m from $1 \ldots 3$ and the number of resources o from $0 \ldots 3$.

[2] Note that an order can only be split in two parts if the demand is at least twice as large as the minimum partition size. Therefore, $ps_j^{\min}$ is chosen so that splitting is possible.

4.2 Parameter Tuning

As previously mentioned in Section 3.2, the simulated annealing algorithm we use in this paper depends on a number of parameters whose setting has an influence on the algorithm's efficiency and effectiveness. We configure the parameters for our experiments using SMAC, an automatic algorithm configuration tool that relies on Bayesian Optimization in combination with an aggressive racing mechanism in order to efficiently search through parameter configuration spaces [6].

We applied SMAC in parallel mode using 24 cores and a total time limit of 96 hours on the large set including 986 instances. For each problem instance, we specified a time limit of five minutes per run and no iteration limit. The cooling rate was not tuned but set to a value of 0.95, which however does not restrict the parameterization of the simulated annealing procedure as the number of iterations per temperature is still being tuned.

We tuned the initial temperature $t_{\max}$, the minimum temperature $t_{\min}$, the number of iterations per temperature w and a weight for each of the four neighborhood relations (p_{1-4}) which determines how often it is selected for the next move. The detailed configuration space with minimum and maximum values as well as the defaults and the tuning result is shown in Table 1.

Table 1: Configuration space of Simulated Annealing for parameter tuning

Parameter	**Type**	**Minimum**	**Maximum**	**Default**	**Tuned**
Initial Temperature	real	0.01	10.0	1	6.2
Minimum Temperature	real	10^{-9}	10^{-3}	10^{-6}	$7.6 \cdot 10^{-4}$
Iterations Per Temperature	integer	10^3	10^6	10^3	$1.54788 \cdot 10^5$
Move Partition Neighborhood Weight	integer	0	10	1	0
Swap Partitions Neighborhood Weight	integer	0	10	1	7
Split Order Neighborhood Weight	integer	0	10	1	3
Shift Order Neighborhood Weight	integer	0	10	1	0
Cooling Rate	real (fixed)	0.95	0.95	0.95	0.95

Note that the automatically tuned parameters set the weights for the move partition and shift order neighborhoods to 0, which disables these two neighborhood operators. We therefore evaluated in addition to the automatically tuned algorithm parameters (C_1) also two manually selected parameters configuration that include all neighborhood operators. One that we selected based on manual tuning with a number of conducted benchmarks (C_2) and another one that uses an equal weight for all four neighborhood operators (C_3). Table 2

shows the details about the three parameter configurations evaluated in our experiments.

Table 2: Overview of the algorithm parameter configurations used for experimental evaluation

Parameter	C_1**: auto tuned**	C_2**: manually tuned**	C_3**: equal weights**
Initial temperature	6.2	0.01	0.1
Minimum temperature	0.00076	10^{-9}	10^{-9}
Iterations per temperature	154788	300000	300000
Move Partition weight	0	4	2.5
Swap Partitions weight	7	2	2.5
Split Order weight	3	3	2.5
Shift Order weight	0	1	2.5
Cooling Rate	real (fixed)	0.95	0.95

4.3 Computational Environment

We conducted all experiments for this paper (including parameter tuning) on 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 of memory, running Ubuntu 16.04.1 LTS. Experiments with the proposed MIP model have been conducted using Gurobi 8.1.

4.4 Computational Results

In a first series of experiments we evaluated the performance of all investigated methods on the instance set which contains 20 small randomly generated instances. To give the MIP approach sufficient time to prove optimal solutions, we set the time limit for all experiments to 1 hour. The simulated annealing algorithm was run under the same time limit with each of the three parameter configurations (C_1, C_2, C_3) on the instances. We performed 10 repeated runs with every configuration on each instance, and used the median objective value from the 10 runs to compare the final results between the different methods.

Table 3 gives an overview of the experimental results with the small instance set. The first row of the table shows the number of instances where the evaluated methods could produce feasible solutions within the time limit, whereas the second row counts the number of overall best upper bounds achieved by each method. Finally, the third row displays the number of optimal solutions found. We can see that all methods were able to produce feasible solutions for every instance. The exact approach using the Gurobi solver produced the best results for the majority of the instances, followed by simulated

annealing with the manually tuned and automatically tuned parameter configurations. Gurobi was able to prove optimal solutions for four of the instances, while simulated annealing was able to reach one optimal solution.

	Gurobi	SA C_1	SA C_2	SA C_3
# solved	20	20	20	20
# best	13	3	4	0
# optimal	4	0	1	1

Table 3: Summarized results for the experiments with the set of smaller instances. The rows of the table display, from top to bottom, the number of feasible solutions found, the number of best upper bounds produced, and the number of optimal solutions achieved by each method.

Detailed results for experiments with the small instances are visualized in Figure 4. In addition to the achieved results by Gurobi and the simulated annealing approach the figure also displays the best lower bounds found by the mixed integer programming approach. All objective values are shown relative to the overall best found objective value, and therefore costs of 1 denote the overall best found solution costs (results with a lower bound value of 1 denote proven optimal solutions).

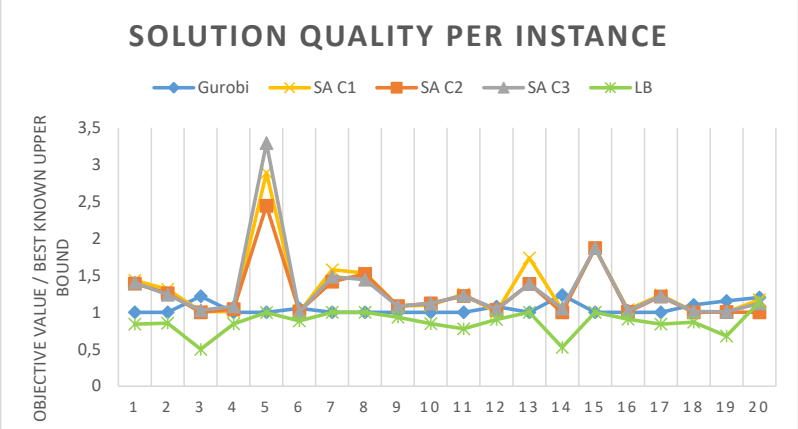


Fig. 4: A visualization of the experimental results for the 20 small instances. The horizontal axis represents the 20 evaluated instances, whereas the vertical axis measures the achieved relative objective values (solution cost produced by each method divided by the overall best found solution cost).

We can see that for the majority of the instances, the exact approach produces the best results. The simulated annealing approach produces similar

outcomes with all three evaluated parameter configurations, however the best results are produced with the manually tuned parameter configuration for the set of smaller instances in a few cases. Compared to the exact method, the simulated annealing algorithm can provide a similar solution quality on the majority of the instances, except for instances 5, 7, 8, 13, and 15, where Gurobi is able to produce the best results.

Figure 5 further visualizes the summarized results as box plots. We can see that Gurobi produces the overall best results for the set of small instances. All three parameter configurations for simulated annealing give similar results, with configurations C_2 and C_3 producing slightly better results than configuration $C1$.

Fig. 5: Box plots comparing the overall results achieved on the set of small instances. The vertical axis measures the achieved relative objective values (solution cost produced by each method divided by the overall best found solution cost).

In a second series of experiments we evaluated the performance of the proposed methods on the instance set which contains 986 large randomly generated instances. Similar as with the first series of experiments we conducted 10 repeated runs for each simulated annealing parameter configuration per instance and used the median objective value to compare the results between the evaluated methods. We used a five minute time limit for the set of larger instances.

The results for the experiments with the set of large instances are summarized in Table 4. Similar as in Table 3, the first row of the table shows the number of instances where a feasible solution could be found, the second row counts the number of best upper bounds found by each method, and the third row displays the number of proven optimal solutions.

	Gurobi	**SA** C_1	**SA** C_2	**SA** C_3
# solved	30	927	936	939
# best	3	758	95	95
# optimal	0	0	0	0

Table 4: Summarized results for the experiments with the set of larger instances. The rows of the table display, from top to bottom, the number of feasible solutions found, the number of best upper bounds produced, and the number of optimal solutions achieved by each method.

The results show that no approach is able to produce feasible solutions for all 986 instances within the time limit[3]. The exact method using Gurobi could only obtain 30 feasible solutions and three best upper bounds, whereas the simulated annealing approach is able to solve the large majority of instances in our experiments. We can see that simulated annealing with parameter configurations $C2$ and $C3$ was able to obtain slightly larger number of feasible solutions as $C1$. However, most best solutions was produced using parameter configuration $C1$. No optimality proofs could be achieved within the given time limits.

Figure 6 visually compares the produced solution qualities achieved by simulated annealing with parameter configurations C_1, C_2 and C_3 for the instances that could be solved by all three configurations. One can see that overall that the automatically tuned algorithm configuration C_1 overall produces the best results in our experiments whereas configurations C_2 and C_3 produce solutions of similar quality.

In summary, our experiments show that the exact approach obtains the best results for most of the small instances. However, in experiments with the larger instances the integer programming solver turned out to be not competitive compared to the simulated annealing approach. Overall, the three evaluated parameter configurations for simulated annealing produced a very similar number of feasible solutions, but the automatically tuned configuration that only uses the swap and split neighborhood operators produced the best results for the majority of the larger instances. However, we observe that configurations which make use of all four investigated neighborhood operators produced slightly better results on the set of smaller instances in our experiments.

5 Conclusion

In this paper we have investigated an extended problem formulation of the PLP that allows production orders to be split during the planning process and additionally considers the resource constraints. We provided a detailed formal specification of the extended production leveling problem and further

[3] There is no guarantee, though, that every instance actually has a feasible solution.

Fig. 6: Box plots comparing the overall results achieved with the simulated annealing approach on the set of larger instances. The vertical axis measures the achieved relative objective values (solution cost produced by each method divided by the overall best found solution cost). Note that some outliers have been excluded for an improved visual comparison.

proposed a mixed integer programming formulation that can be used to approach the problem with state-of-the-art solver technology. Additionally, we described a metaheuristic approach using simulated annealing that can be used to tackle realistically sized problem instances and investigated several local search neighborhood relations for the problem.

We empirically evaluated all proposed methods by performing experiments using a large number of instances that have been randomly generated by an instance generation routine that we proposed in this paper. Experimental results show that the exact approach using integer programming formulation was able to prove optimal results on several of the considered smaller instances and overall produced the best results for the experiments with small sized instances. However, results obtained by experiments with larger problem instances revealed that the exact approach was not competitive compared to the evaluated metaheuristics on realistically sized instances in our experiments. The simulated annealing based approach finds feasible solutions for most of the instances within a reasonable time limit and can be used to solve instances of realistic size. Based on the configuration provided by the automated parameter tuner SMAC, we can conclude that the most important neighborhoods are the swap partitions and split moves.

In future work, we plan to investigate an approach that hybridizes the proposed exact and metaheuristic techniques within the framework of large neighborhood search.

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

References

1. Azizoğlu, M., İmat, S.: Workload smoothing in simple assembly line balancing. Computers & Operations Research **89**, 51–57 (2018)
2. Chiarandini, M., Di Gaspero, L., Gualandi, S., Schaerf, A.: The balanced academic curriculum problem revisited. Journal of Heuristics **18**, 119–148 (2012)
3. Coleman, B.J., Vaghefi, M.R.: Heijunka (?): A key to the toyota production system. Production and Inventory Management Journal **35**(4), 31 (1994)
4. Di Gaspero, L., Schaerf, A.: Hybrid Local Search Techniques for the Generalized Balanced Academic Curriculum Problem. In: M.J. Blesa, C. Blum, C. Cotta, A.J. Fernández, J.E. Gallardo, A. Roli, M. Sampels (eds.) Hybrid Metaheuristics, 5th International Workshop, HM 2008, Málaga, Spain, October 8-9, 2008. Proceedings, *Lecture Notes in Computer Science*, vol. 5296, pp. 146–157. Springer (2008)
5. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by Simulated Annealing. Science **220**(4598), 671–680 (1983). DOI 10.1126/science.220.4598.671
6. Lindauer, M., Eggensperger, K., Feurer, M., Falkner, S., Biedenkapp, A., Hutter, F.: Smac v3: Algorithm configuration in python. `https://github.com/automl/SMAC3` (2017)
7. Mullinax, C., Lawley, M.: Assigning patients to nurses in neonatal intensive care. Journal of the Operational Research Society **53**(1), 25–35 (2002)
8. Punnakitikashem, P., Rosenberber, J.M., Buckley-Behan, D.F.: A stochastic programming approach for integrated nurse staffing and assignment. IIE Transactions **45**(10), 1059–1076 (2013)
9. Schaus, P., Hentenryck, P.V., Régin, J.C.: Scalable Load Balancing in Nurse to Patient Assignment Problems. In: W.J.v. Hoeve, J.N. Hooker (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 6th International Conference, CPAIOR 2009, Pittsburgh, PA, USA, May 27-31, 2009, Proceedings, *Lecture Notes in Computer Science*, vol. 5547, pp. 248–262. Springer (2009)
10. Vass, J.: Exact and metaheuristic approaches for the production leveling problem. Master's thesis, TU Wien, Vienna, Austria (2019). `https://dbai.tuwien.ac.at/staff/jvass/publications/plp_thesis_final_corrected_20191202.pdf`
11. Vass, J., Lackner, M.L., Musliu, N.: Exact and Metaheuristic Approaches for the Production Leveling Problem. submitted to Computers & Operations Research (2019)

A constructive matheuristic approach for the vertex colouring problem

Reshma Chirayil Chandrasekharan · Tony Wauters

Received: date / Accepted: date

Abstract The vertex colouring problem is one of the most widely studied and popular problems in graph theory. In light of the recent interest in hybrid methods involving mathematical programming, this paper presents an attempt to design a matheuristic approach for the problem. A decomposition-based approach is introduced which utilizes an integer programming formulation to solve subproblems to optimality. A detailed study of two different decomposition strategies, vertex-based and colour-based, is discussed. In addition, the impact of algorithm design parameters on the particular decompositions used and their influence on final solution quality is also explored.

Keywords Vertex colouring · matheuristic · decomposition

1 Introduction

The vertex colouring problem (VCP) seeks to assign colours to vertices of a graph such that no two adjacent vertices are assigned the same colour. Initially

Reshma Chirayil Chandrasekharan
KU Leuven
Department of Computer Science
CODeS
Gebroeders De Smetstraat 1
9000 Gent
Belgium
E-mail: ReshmaCC@kuleuven.be

Tony Wauters
KU Leuven
Department of Computer Science
CODeS
Gebroeders De Smetstraat 1
9000 Gent
Belgium
E-mail: tony.wauters@kuleuven.be

studied as a problem on planar graphs, the problem has been generalized over general graphs and represents a large share of the graph theory literature given its widespread applications. Problems which can be modeled as the assignment of conflicting elements of a set to distinct subsets such as scheduling (Leighton, 1979), timetabling (Babaei, Karimpour, and Hadidi, 2015), frequency assignment (Aardal, Van Hoesel, Koster, Mannino, and Sassano, 2007) and register allocation (Chow and Hennessy, 1990), are some of the major areas where there exist practical applications of the VCP.

The VCP is an example of a problem which is easy to define, yet difficult to solve. Determining the smallest number of colours required to colour a graph is an NP-hard problem (Garey and Johnson, 1979). This inherent difficulty of the problem means that only certain kinds of graph are capable of being solved by the best mathematical models formulated for the VCP (Méndez-Díaz and Zabala (2006), Méndez-Díaz and Zabala (2008), Malaguti, Monaci, and Toth (2011)), thereby motivating the need for efficient heuristic strategies. Decades of research have contributed multiple models and performance guarantees for the VCP. Despite these achievements, the problem continues to fascinate researchers in this area due to its theoretical complexity and the constantly growing number of practical applications that demand colouring larger graphs.

Malaguti and Toth (2010) provide a useful survey on the various exact and heuristic algorithms developed for the VCP. Several high performing algorithms for the VCP such as Malaguti, Monaci, and Toth (2008), Funabiki and Higashino (2000), Galinier and Hao (1999) suggest that hybrid methods are efficient in colouring some of the very large random graphs. This paper presents some preliminary experiments conducted in order to test a matheuristic approach for the VCP. Matheuristics are methods which hybridize mathematical programming and heuristics. The recent success of matheuristic strategies in scheduling applications which are, at their most fundamental level, graph colouring problems has motivated this study.

The outline of the paper is as follows. Section 2 briefly introduces the problem and the terminology. The matheuristic strategy proposed for the VCP is introduced in Section 3, while the related experiments are summarized in Section 4. Section 5 then ends this paper by concluding and offering future research possibilities.

2 The vertex colouring problem

Let $G = (V, E)$ denote a graph on a finite vertex set V and edge set E, whose cardinalities are denoted by n and m respectively. In this paper, E is assumed to be the collection of unordered pairs $E = \{\{v, v'\} | v, v' \in V, v \neq v'\}$, thereby limiting the problem to finite simple graphs (no loops or multiple edges). A k-colouring of G is the assignment of k colours to elements of V such that no two adjacent vertices share the same colour. The smallest k for which a k-colouring exists for G is defined to be the chromatic number of G, denoted

by χ_G. The saturation degree of a vertex is defined as the number of colours to which it is adjacent.

3 Constructive Matheuristics

The present work applies a decomposition-based approach which utilizes powerful exact techniques to solve subproblems to optimality and thus can be called a matheuristic (Maniezzo, Stützle, and Voß, 2010). More specifically, this approach adapts the constructive matheuristic (CMH) strategy introduced by Smet, Wauters, Mihaylov, and Vanden Berghe (2014). This constructive heuristic sequentially solves subproblems and utilizes these optimal solutions to construct a solution for the entire original problem. The subproblems of a CMH strategy are called blocks.

The following CMH design parameters introduced in Chandrasekharan, Toffolo, and Wauters (2019) are also tested.

1. Block size (η): This parameter defines the size of subproblems and often significantly influences algorithmic runtime.
2. Overlap (θ): This parameter allows blocks to share some constraints instead of being completely disjoint. θ denotes the extent of overlap between consecutive blocks.
3. Relaxed future (ρ): This feature allows the CMH to have larger subproblems by solving a part of the block in a relaxed fashion, which tends to have less of an impact on algorithmic runtime than increasing block size. The size of the relaxed part is expressed as percentage (ρ) of the size of the original block. The relaxed part must be later solved again with the original formulation to ensure feasibility.

A CMH configuration is therefore represented by the three tuple (η, θ, ρ). Figure 1 illustrates the overall CMH strategy utilized in this paper and the design parameters. The CMH approach utilized relaxes one or more constraints of an IP formulation for the VCP in order to define the decomposition. The subproblems generated are then sequentially solved by a MIP solver. Depending on the solutions of previously solved subproblems, additional constraints may need to be added to the blocks or objective functions utilized in the blocks modified to ensure feasibility of the final solution. Given a block b, its definition and the precise optimization problem it solves is realized by means of its *block objective function* $Z_b(\eta, \theta, \rho)$.

The CMH strategy utilizes the simple assignment-based IP formulation (VCP-ASS) of the VCP. Let H denote the set of colours. The algorithm starts with a total of n colours, $|H| = n$. Variables x_{ih} decide whether colour h is assigned to vertex i while variables y_h decide whether colour $h \in H$ is utilized

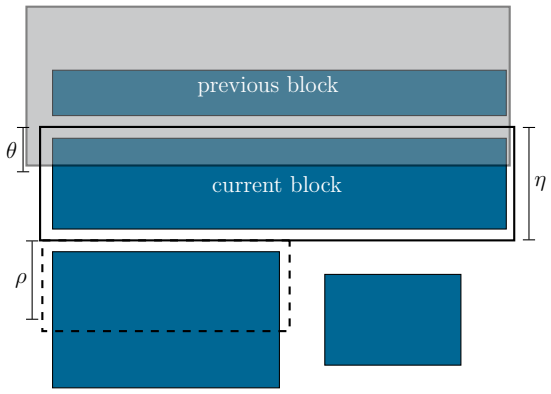


Fig. 1: An overview of the general CMH strategy. Blue rectangles represent subproblems (blocks) in the CMH strategy and the solid window represents current block. The gray rectangle represents the block previously solved and the dotted rectangle represents the part of a future unsolved block solved in a relaxed fashion.

or not.

$$x_{ih} = \begin{cases} 1 & \text{if vertex } i \in V \text{ is assigned to colour } h \in H \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$y_h = \begin{cases} 1 & \text{if colour } h \in H \text{ is used} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The model can then be formulated as follows:

$$\text{minimize:} \quad \sum_{h=1}^{n} y_h \tag{3}$$

$$\text{subject to:} \quad \sum_{h=1}^{n} x_{ih} = 1 \quad \forall i \in V \tag{4}$$

$$x_{ih} + x_{jh} \leq y_h \quad \forall (i,j) \in E, h = 1, \ldots, n \tag{5}$$

$$y_{h+1} \leq y_h \quad \forall i = 0, \ldots, n-1 \tag{6}$$

$$x_{ih} \in \{0,1\} \quad \forall i \in V, h = 1, \ldots, n \tag{7}$$

$$y_h \in \{0,1\} \quad \forall h = 1, \ldots, n \tag{8}$$

Constraints 4 ensure that vertices are assigned exactly one colour, while Constraints 5 prevent adjacent vertices from being assigned the same colour. Constraints 6 are introduced to break the symmetry incurred by the interchangeability of colours.

The aim is to develop a CMH algorithm for the VCP inspired by the recent success of CMH techniques for task scheduling problems, as discussed in both Smet et al. (2014) and Chandrasekharan, Smet, and Wauters (2020). The CMH

strategy aims to optimally colour subgraphs of G in a sequential fashion by utilizing their MIP formulations, and then eventually combining them together into a solution for the entire graph. The two obvious candidates for how to decompose a graph led to the development of two decomposition strategies: vertex-based and colour-based CMH strategies.

3.1 Vertex-based CMH

The vertex-based CMH (VBC) defines blocks by way of groups of vertices. The induced subgraph defined by a block is coloured to optimality by utilizing its MIP formulation. Once a block is solved, the colour assignments are fixed and the following blocks are solved such that they do not contradict previously fixed colour assignments. Let b_k denote the current block and E_{b_k} denote the induced subgraph corresponding to the vertices belonging to blocks $b_0, \ldots, b_k$. The MIP formulation solved in block b_k can now be formulated as follows.

$$x_{ih} = \begin{cases} 1 & \text{if vertex } i \in b_k \text{ is assigned to colour } h \in H \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

$$y_h = \begin{cases} 1 & \text{if colour } h \in H \text{ is used} \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

$$\text{minimize:} \quad \sum_{h=1}^{n} y_h \tag{11}$$

$$\text{subject to:} \quad \sum_{h=1}^{n} x_{ih} = 1 \quad \forall i \in b_k \tag{12}$$

$$x_{ih} + x_{jh} \leq y_h \quad \forall (i,j) \in E_{b_k}, h = 1, \ldots, n \tag{13}$$

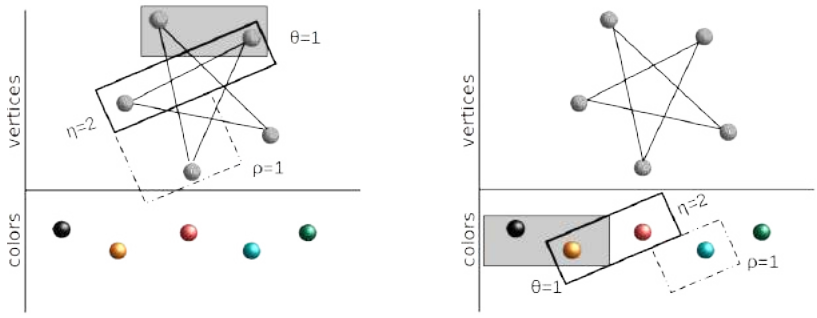
$$y_{h+1} \leq y_h \quad \forall h = 0, \ldots, n-1 \tag{14}$$

$$x_{ih} \in \{0,1\} \quad \forall i \in b_k, h = 1, \ldots, n \tag{15}$$

$$y_h \in \{0,1\} \quad \forall h = 1, \ldots, n \tag{16}$$

3.2 Colour-based CMH

The colour-based CMH (CBC) defines blocks by means of grouping colours. In each block of colours, the CMH solves for the maximal subgraph that can be coloured by the colours in the block and fixes those assignments. The CMH stops when all vertices have been coloured. In contrast with the vertex-based CMH, the block objective function of the CBC maximizes the number of vertices that can be coloured by the colours in a given block. The following is the



(a) Vertex-based CMH (b) Colour-based CMH

Fig. 2: The CBC and VBC strategies. Solid rectangles represent blocks. Gray windows correspond to blocks previously solved, while the dotted rectangles represent the relaxed future part of the current block.

MIP formulation solved in each block b_k:

$$x_{ih} = \begin{cases} 1 & \text{if vertex } i \in V \text{ is assigned to colour } h \in b_k \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

$$y_h = \begin{cases} 1 & \text{if colour } h \in b_k \text{ is used} \\ 0 & \text{otherwise} \end{cases} \tag{18}$$

The model can then be formulated as:

$$\text{maximize: } \sum_{h \in b_k} \sum_{i \in V} x_{ih} \tag{19}$$

$$\text{subject to: } x_{ih} + x_{jh} \leq y_h \quad \forall (i,j) \in E, h \in b_k \tag{20}$$

$$y_{h+1} \leq y_h \quad \forall h \in \{b_{k_0}, \ldots, b_{k_{|b_k|-1}}\} \tag{21}$$

$$x_{ih} \in \{0,1\} \quad \forall i \in V, h \in b_k \tag{22}$$

$$y_h \in \{0,1\} \quad \forall h \in b_k \tag{23}$$

The final block is re-optimized with the original objective of minimizing the number of colours. Figure 2 illustrates CBC and VBC strategies.

The proposed CMH strategy falls in the category of successive augmentation techniques discussed in Johnson, Aragon, McGeoch, and Schevon (1991). Such techniques begin with a feasible partial colouring of the graph and then progressively extend it, examples of which include greedy colouring heuristics such as DSATUR (Brélaz, 1979) and recursive largest first or RLF (Leighton, 1979). DSATUR colours vertices one by one whereas RLF iteratively construct colour classes - groups of vertices that can be coloured by the same colour. The general CMH strategy employed in this paper can be interpreted as a generalization of these classical colouring heuristics. Rather than colouring single vertices or isolating a single colour class, vertex-based CMH optimally colours

subgraphs whereas the colour-based CMH isolates a block of colour classes by solving for it mathematically. It is worth noting that in case of the colour-based CMH, subproblems are attempting to generate maximal independent sets mathematically and therefore might require very long computation times for certain graph classes, as opposed to RLF where it is done heuristically.

Johnson et al. (1991) and Matula, Marble, and Isaacson (1972) studied the influence of the order in which vertices are coloured on final solution quality. Some heuristics utilize a fixed static ordering of vertices, whereas some change this ordering dynamically as the algorithm proceeds. Among static colourings, it is proven that the smallest last (SL) ordering yields the best performance when implemented in a greedy colouring heuristic that colours one vertex at a time. The random vertex-based CMH (RVC) is produced by adapting a random order for colouring vertices in the vertex-based CMH. On adapting the adapting the SL ordering, the SL-based CMH (SLC) is produced. In addition, another approach called the DSATUR-SL based CMH (DSC) is designed to utilize a dynamic ordering as in the DSATUR heuristic. In this CMH approach, the first block is composed of η vertices from the SL ordering. Once this block is solved, elements of the next block are selected such that it composed of uncoloured vertices with the first η largest saturation degrees in the partially coloured graph. Ties are broken utilizing the SL order.

4 Computational Study

Experiments are conducted on four threads of an Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz computer running Ubuntu 16.04.2 LTS. The CMH algorithm was coded in Java and used Gurobi 8.1 to solve blocks. The DIMACS10 vertex-colouring benchmark instances were utilized in the computational study and are available at `http://www.cc.gatech.edu/dimacs10/`. The benchmark time limit is considered to be 3600s.

From Table 1 it is clear that VCP-ASS, being the IP formulation, is capable of solving only 48 instances of the 131 benchmark instances. When the number of vertices is above 500, the method fails or exhibits poor performance, something which, again, justifies the need for powerful heuristics. The performance of CBC and RVC for $\eta = 1$ are also summarized for comparison purposes. Since RVC utilizes a random order of vertices, the results are averaged over 10 runs. Here, one colour or one vertex is considered per block. If the runtime exceeds the benchmark time limit, the program outputs the number of vertices (n) as the result. %Gap is calculated with respect to the best known solution available in the literature for a particular instance. In order to develop an efficient CMH for the VCP, the impact of CMH design parameters η and θ on all its variants have been tested.

Table 1: Performance comparison of baseline algorithms VCP-ASS, CBC and RVC

	VCP-ASS	RVC(1,0,0)	CBC(1,0,0)
Number of feasible solutions	74	104	116
Number of best known solutions	48	35	45
Average calculation time(s)	2718.36	917.08	569.17
%Gap - average	3100.82	1103.59	521.17

4.1 Colour-based CMH

From Table 1 it is evident that for $\eta = 1$, CBC has a better performance compared to that of the RVC. In general, larger block size is expected to result in higher solution quality, but also lead to longer runtimes. However, decreasing the block size too much may lead to too many sub problems and longer overall set up times. See Table 2 for a summary of the results obtained by CBC. From the performance of CBC, it is clear that larger block sizes lead to longer runtimes but this does not correspond to improved solution quality trends. This can be attributed to the fact that the algorithm might terminate due to the runtime limit being exceeded for larger block sizes, which contributes highly to the average %gap. This becomes more evident with the %gap calculated exclusively for the feasible solutions.

For CBC, the block objective function tries to solve for the largest subgraph that can be coloured by the elements of the block. The underlying problem therefore seeks to isolate independent sets in the graph, which can be an NP hard problem. Since increasing the block size beyond $\eta = 8$ may lead to very long algorithm runtimes, increasing block size further will probably not improve CMH performance. This motivates testing the impact of the overlap design parameter on the CBC. Surprisingly, overlap feature has a negative effect on CBC's performance. Both average algorithm runtime and %gap increase when the overlap feature is introduced. However, when only feasible solutions are considered there is an improvement concerning the average gap, indicating that the overlap feature need not necessarily have a negative impact on CBC's solution quality and this instead may be attributed to the premature termination of the algorithm due to the runtime limit being exceeded.

Table 2: Summary of performance details of colour-based CMH strategies

Configuration	Average runtime	Average %gap	#feasible solutions	#optimal solutions	Average %gap(only feasible solutions)
"CBC(1,0,0)"	569.17	521.17	116	45	52.41
"CBC(4,0,0)"	1,694.07	2,806.29	72	41	63.77
"CBC(8,0,0)"	1,839.26	1,247.49	86	51	57.85
"CBC(4,50,0)"	2,129.31	3,490.89	52	14	48.79
"CBC(8,50,0)"	1,842.23	2,102.66	65	20	39.46

4.2 Vertex-based CMH

The performance of various vertex-based approaches is summarized in Table 3. In case of RVC, performance trends are rather irregular, with RVC(20,0,0) exhibiting the best performance when $\theta = 0$. In contrast to that of CBC, it is evident that overlap feature can be employed in the RVC to produce more feasible solutions. The overall average gap still remains high whereas the average gap calculated only over the feasible solutions decreases. This could be due to the large RVC runtimes as a result of implementing overlap. In case of exceeding runtime limit, the CMH terminates and returns the number of vertices (n) as the result. Therefore, a subset S of 95 instances on which all vertex-based CMH strategies produce feasible solutions has been constructed in order to make a fair comparison between their performances.

Table 3: Summary of performance details of various vertex-based CMH strategies

Configuration	Average runtime	Average %gap	#feasible solutions	#optimal solutions	Average %gap(only feasible solutions)	Average %gap(over S)
"RVC(1,0,0)"	917.08	1,103.59	104	35	34.45	35.41
"RVC(10,0,0)"	797.95	1,029.42	109	31	34.68	35.65
"RVC(20,0,0)"	767.18	1,142.73	107	38	34.02	34.35
"RVC(10,50,0)"	954.28	1,105.3	104	38	36.6	36.95
"RVC(20,50,0)"	913.79	1,103.09	104	37	33.82	34.8
"SLC(1,0,0)"	973.73	595.88	108	49	27.18	26.47
"SLC(10,0,0)"	703.43	879.43	112	60	23.04	23.29
"SLC(20,0,0)"	669.67	878.55	112	55	22.02	22.26
"SLC(10,50,0)"	899.41	1,250.6	106	58	21.41	22
"SLC(20,50,0)"	838.19	1,054.18	108	61	19.66	20.8
"DSC(1,0,0)"	1,204.19	2,148.55	95	48	26.14	26.14
"DSC(10,0,0)"	831.33	1,208.23	109	55	22.24	22.8
"DSC(20,0,0)"	762.19	1,207.75	109	60	21.67	21.98
"DSC(10,50,0)"	992.31	1,539.85	102	56	16.33	16.68
"DSC(20,50,0)"	928.37	1,489.96	105	56	17.07	17.64

Sequential colouring (SC) refers to greedy strategies which colours vertices of a graph one by one. It is proven that there exists an order which, when utilized by the SC, results in an optimal colouring. Matula et al. (1972) presents an in depth study of the influence of the order in which vertices are coloured in a SC heuristic. This work introduced the SL ordering and proved that when utilized by the SC, this order guarantees the max-subgraph-min-degree bound given by

$$\chi(G) \leq 1 + \max_{H:\text{subgraph of G}} \min_{v \in H}\{\deg_H(v)\}$$

Note that the configuration $(1, 0, 0)$ of vertex-based CMH corresponds to a SC heuristic and guarantees this bound when one uses SL ordering. The im-

provement of RVC solution quality with larger block sizes and when applying the overlap feature motivates similar experiments using SL-ordering in the vertex-based CMH, resulting in the SL-CMH or SLC.

Figure 3 compares the vertex-based CMH performance with respect to design parameters η and θ. These experiments are based on the performance on the insances from set S. It is clear that utilizing SL ordering in the vertex-based CMH is an improvement over the RVC. The figure 3 also shows how increasing the block size improves solution quality. However, it must be noted that very large blocks lead to very long runtimes and hence the number of optimal solutions decreases for $\eta = 20$. Thus it is not feasible to increase block size further to improve SLC's performance. Overlap experiments show that this feature improve the overall performance of SLC. While it is clear that the algorithm generates high quality solutions, overlap also increases the algorithm runtime, thereby leading to the termination of the program before it generates a solution. This results in fewer feasible and optimal solutions and larger average %gaps compared to the results of SLC implemented without overlap.

From these experiments it is clear that while overlap can improve the CMH solution quality, its influence on algorithm runtime makes the overall CMH strategy rather inefficient. However, it is worth noting that, for overlap to be able to handle constraints linking blocks more effectively, it requires vertices which share Constraints 5 to be present in consecutive blocks. This idea motivates utilizing the DSATUR-based dynamic ordering in the CMH strategy, resulting in the DSC. While using DSATUR may not lead to solution quality improvements, it may exhibit better performance when employed with non-zero overlap.

Similar to all other CMH strategies discussed, the solution quality of DSC also improves upon increasing block size, as is evident from Figure 3 and Table 3. Note that concerning experiments without overlap, the SLC exhibits the best overall performance. Table 3 also presents the results of DSC when implemented with overlap $\theta = 50$. Out of all the CMH strategies presented in the paper, DSC when implemented with overlap leads to the lowest %gap computed over the feasible solutions, indicating its ability to generate high quality solutions for the VCP. However, algorithm runtime also increases while adding overlap feature and affects algorithms overall efficiency. SLC exhibits the lowest impact on algorithm runtime when implementing overlap.

To further improve the method, an upper bound for the chromatic number is utilized to reduce the size of the formulation. This is done by executing a simple greedy coloring heuristic that colors vertices one after the other based on the SL ordering. Moreover, using the number of vertices(n) as the algorithm output on reaching timelimit contribute a disproportionately high value towards the average gap, making it an inefficient measure to study CMH performance with respect to other best performing algorithms. The SLC and DSC algorithms after implementing this modification is renamed as SLC' and DSC' respectively and their performance on the difficult VCP instances is presented

in Table 4 along with that of the other best performing heuristics for comparison.

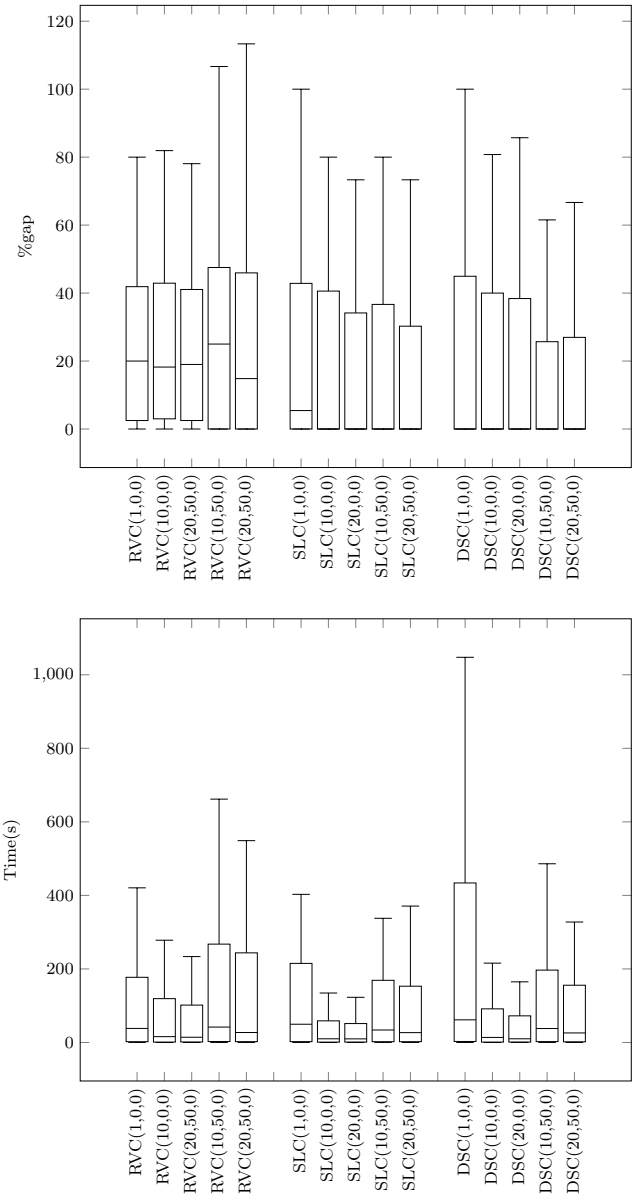


Fig. 3: Performance comparison of various vertex-based CMH approaches

Table 4: Performance of SLC and DSC on some of the difficult VCP instances. Performance details of some best performing algorithms are also provided for comparison such as Impasse: (Morgenstern, 1996), VSS-Col: (Hertz et al., 2008), MIPS_CLR: (Funabiki and Higashino, 2000) and MMT: (Malaguti et al., 2008). k denotes the number of colors used and T(s) denotes the algorithm runtime in seconds.

				Impasse		**VSS-Col**		**MIPS_CLR**				**MMT**		**SLC'(10,50,0)**		**SLC'(20,50,0)**		**DSC'(10,50,0)**		**DSC'(20,50,0)**	
Instance	n	m	χ_G	best	k	T(s)	k	T(s)	best	avg.	T(s)	best	avg.	T(s)	k	T(s)	k	T(s)	k	T(s)	k	T(s)
DSJC125.1	125	736		5					5	5	0	5	5	21	7	1.15	7	0.42	6	0.95	6	0.58
DSJC125.5	125	3,891		17	17	1			17	17	1	17	17	122	24	8.97	24	6.11	21	9.8	23	5.99
DSJC125.9	125	6,961		44					44	44	0	44	44	121	55	30.02	51	19.17	51	31.29	50	19.49
DSJC250.1	250	3,218		8					8	8	5	8	8	21	12	7.46	11	3.84	11	11	11	5.75
DSJC250.5	250	15,668		28	28	22			28	28.4	14	28	28	117	41	110.45	42	66.57	37	124.01	37	71.62
DSJC250.9	250	27,897		72					72	72.4	31	72	72	89	95	381.77	92	217.38	92	414.78	92	234.5
DSJC500.1	500	12,458		12			12	97	12	12.4	84	12	12.25	210	18	84.57	18	45.93	16	192.83	17	97.7
DSJC500.5	500	62,624		48	49	660	48	1,331	49	49.4	349	48	48.25	388	72	1,368.51	69	783.7	65	1,721.85	66	958.1
DSJC500.9	500	$1.12 \cdot 10^6$		126			126	1,686	127	127.8	480	127	127.75	433	176	3,600	176	3,600	176	3,600	176	3,600
DSJC1000.1	1,000	49,629		20			20	2,396	21	21	90	20	20.25	260	30	1,032.55	30	3,600	26	2,682.38	26	2,013.54
DSJC1000.5	1,000	$2.5 \cdot 10^5$		83	89	1,148	88	2,028	88	89	4,658	84	84.25	8,407	124	3,600	124	3,600	124	3,600	124	3,600
DSJC1000.9	1,000	$4.49 \cdot 10^5$		224			224	3,326	228	229.6	1,565	225	226	3,234	318	3,600	318	3,600	318	3,600	318	3,600
DSJR500.1	500	3,555	12	12	12	0			12	12	0	12	12	25	12	16.25	12	8.34	12	66.43	12	33.99
DSJR500.1c	500	$1.21 \cdot 10^5$		85	85	5	85	736	85	85	6	85	85	88	107	3,600	103	2,366.91	107	3,600	95	2,703.06
DSJR500.5	500	58,862	122	122	123	14	126	1,409	122	123.4	276	122	122	163	125	2,729.63	128	1,440.24	127	3,545.39	126	1,855.03
le450_15a	450	8,168	15	15	15	0			15	15	1	15	15	0	18	47.86	18	24.18	18	119.75	18	76.34
le450_15b	450	8,169	15	15	15	0			15	15	1	15	15	0	17	3,600	17	3,600	17	120.08	16	58.96
le450_15c	450	16,680	15	15	15	5	15	6	15	15.2	11	15	15	3	26	74.15	25	40.58	24	251.45	25	130.83
le450_15d	450	16,750	15	15	15	3	15	44	15	15	5	15	15	4	27	78.45	27	3,600	25	254.95	26	131.65
le450_25c	450	17,343	25	25			26	1	26	26	7	25	25	1,321	30	94.31	30	50.18	29	319.64	30	162.16
le450_25d	450	17,425	25	25			26	1	26	26.4	1	25	25	436	30	3,600	30	48.09	28	310.25	29	154.7
r250.1	250	867	8	8	8	0			8	8	0	8	8	26	8	0.91	8	0.44	8	3.17	8	1.7
r250.1c	250	30,227	64	64	64	0			64	64	2	64	64	21	67	191.97	67	103.77	67	401.03	65	212.73
r250.5	250	14,849	65	65	65	7			65	65.8	16	65	65	64	67	91.83	68	3,600	68	204.42	67	106.4
r1000.1	1,000	14,378	20	20	20	1			20	20	0	20	20	37	20	113.8	20	58.51	20	2,021.99	20	1,004.68
r1000.1c	1,000	$4.85 \cdot 10^5$		98	98	46			98	98.8	557	98	98	518	120	3,600	120	3,600	120	3,600	120	3,600
r1000.5	1,000	$2.38 \cdot 10^5$	234	234	241	77			237	238.6	1,345	234	234	753	251	3,600	251	3,600	251	3,600	251	3,600
latin_square_10	900	$3.07 \cdot 10^5$		98	98	415			99	100.2	938	101	102	5,156	213	3,600	213	3,600	213	3,600	213	3,600
flat300_20_0	300	21,375	20	20	20	0			20	20	2	20	20	21	46	96.89	44	56.49	43	205.43	42	118
flat300_26_0	300	21,633	26	26	26	1			26	26	1	26	26	36	46	96.67	44	57.59	42	211.27	41	121.14
flat300_28_0	300	21,695	28	28	31	156	29	867	31	31	133	31	31	212	45	97.08	46	57.64	43	212.38	44	122.38
flat1000_50_0	1,000	$2.45 \cdot 10^5$	50	50	50	0	50	318	50	50	14	50	50	1,417	125	3,600	125	3,600	125	3,600	125	3,600
flat1000_60_0	1,000	$2.46 \cdot 10^5$	60	60	60	0	60	694	60	60	59	60	60	3,645	123	3,600	123	3,600	123	3,600	123	3,600
flat1000_76_0	1,000	$2.47 \cdot 10^5$		82	89	897	87	1,689	87	87.8	2,499	83	83.5	7,325	125	3,600	125	3,600	125	3,600	125	3,600
Average gap											1.78			0.71		44.13		42.66		38.19		38.48
Average runtime											386.79			1,020.41		1,469.27		1,468.22		1,454.01		1,258.85
Maximum runtime											4,658			8,407		3,600		3,600		3,600		3,600

5 Results and Discussion

The present paper presents preliminary research conducted in order to design an efficient constructive matheuristic (CMH) strategy for the vertex colouring problem. The primary insight gained from this work is that it is possible to extend subproblems of successive augmentation techniques and employ simple integer programming approaches to arrive at high quality solutions. This, in turn, becomes a way of designing a CMH. Experiments show that algorithm design parameters such as overlap can be effectively utilized to improve the solution quality. The major challenge, however, is the very high impact of such features on algorithmic runtime. Previous research has shown that identifying smarter decomposition strategies and introducing components that better navigate the CMH may overcome this drawback. Therefore, future research will explore decomposition strategies for CMH and study their suitability when implemented alongside CMH design parameters.

Acknowledgements This research received funding from the Flemish Government under the *Onderzoeksprogramma Artificile Intelligentie (AI) Vlaanderen* programme. Editorial consultation provided by Luke Conolly (KU Leuven).

References

Karen I Aardal, Stan PM Van Hoesel, Arie MCA Koster, Carlo Mannino, and Antonio Sassano. Models and solution techniques for frequency assignment problems. *Annals of Operations Research*, 153(1):79–129, 2007.

Hamed Babaei, Jaber Karimpour, and Amin Hadidi. A survey of approaches for university course timetabling problem. *Computers & Industrial Engineering*, 86:43–59, 2015.

Daniel Brélaz. New methods to color vertices of a graph. *Commun. ACM*, 22: 251–256, 1979.

Reshma Chirayil Chandrasekharan, Túlio AM Toffolo, and Tony Wauters. Analysis of a constructive matheuristic for the traveling umpire problem. *Journal of Quantitative Analysis in Sports*, 15(1):41–57, 2019.

Reshma Chirayil Chandrasekharan, Pieter Smet, and Tony Wauters. An automatic constructive matheuristic for the shift minimization personnel task scheduling problem. *Journal of Heuristics*, pages 1–23, 2020.

Fred C Chow and John L Hennessy. The priority-based coloring approach to register allocation. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 12(4):501–536, 1990.

Nobuo Funabiki and Teruo Higashino. A minimal-state processing search algorithm for graph coloring problems. *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, 83(7): 1420–1430, 2000.

Philippe Galinier and Jin-Kao Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of combinatorial optimization*, 3(4):379–397, 1999.

Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

Alain Hertz, Matthieu Plumettaz, and Nicolas Zufferey. Variable space search for graph coloring. *Discrete Applied Mathematics*, 156(13):2551–2560, 2008.

David S Johnson, Cecilia R Aragon, Lyle A McGeoch, and Catherine Schevon. Optimization by simulated annealing: an experimental evaluation; part ii, graph coloring and number partitioning. *Operations research*, 39(3):378–406, 1991.

Frank Thomson Leighton. A graph coloring algorithm for large scheduling problems. *Journal of research of the national bureau of standards*, 84(6): 489–506, 1979.

Enrico Malaguti and Paolo Toth. A survey on vertex coloring problems. *International transactions in operational research*, 17(1):1–34, 2010.

Enrico Malaguti, Michele Monaci, and Paolo Toth. A metaheuristic approach for the vertex coloring problem. *INFORMS Journal on Computing*, 20(2): 302–316, 2008.

Enrico Malaguti, Michele Monaci, and Paolo Toth. An exact approach for the vertex coloring problem. *Discrete Optimization*, 8(2):174–190, 2011.

V Maniezzo, T Stützle, and S Voß. Matheuristics, volume 10 of annals of information systems, 2010.

David W. Matula, George Marble, and Joel D. Isaacson. Graph coloring algorithms. pages 109 – 122, 1972. doi: https://doi.org/10.1016/B978-1-4832-3187-7.50015-5.

Isabel Méndez-Díaz and Paula Zabala. A branch-and-cut algorithm for graph coloring. *Discrete Applied Mathematics*, 154(5):826–847, 2006.

Isabel Méndez-Díaz and Paula Zabala. A cutting plane algorithm for graph coloring. *Discrete Applied Mathematics*, 156(2):159–179, 2008.

Craig Morgenstern. Distributed coloration neighborhood search. *Discrete Mathematics and Theoretical Computer Science*, 26:335–358, 1996.

Pieter Smet, Tony Wauters, Mihail Mihaylov, and Greet Vanden Berghe. The shift minimisation personnel task scheduling problem: A new hybrid approach and computational insights. *Omega*, 46:64–73, 2014.