The Capacitated Team Orienteering Problem: a hybrid Simulated Annealing and Iterated Local Search Approach

Aldy GUNAWAN[1], Jiahui ZHU[2], Kien Ming NG[2]

aldygunawan@smu.edu.sg, e0032017@u.nus.edu, isenkm@nus.edu.sg
[1]*School of Computing and Information Systems, Singapore Management University, Singapore*
[2]*Industrial Systems Engineering and Management Department, National University of Singapore, Singapore*

1. Introduction

Orienteering Problem (OP) is an NP-hard vehicle routing problem that combines two classical combinatorial optimization problems, the Traveling Salesman Problem (TSP) and the Knapsack Problem (Vansteenwegen et al., 2011). The objective of the problem is to select the most profitable combination of customers from a list of potential customers given that the selected customers do not violate the time constraints (Gunawan et al. 2016). OP was first introduced by Golden et al. (1987) and since then, the problem has received a considerable amount of attention by researchers in the past few decades. The survey papers published by Vansteenwegen et al. (2011) and Gunawan et al. (2016) provided an extensive summary of the range of research works done on OP and its variant from the time of introduction of the OP to as recent as 2016.

Team Orienteering Problem (TOP) is one main variant of the original OP. The objective of TOP is to select the most profitable combination of customers for a fleet of vehicles from a list of potential customers, given that the selected customers do not violate the time constraints. The concept of TOP was first introduced by Butt and Cavalier (1994), who named it the Multiple Tour Maximum Collection Problem, and Chao et al. (1996) came up with the term TOP, which is widely used by researchers around the world currently.

In this paper, we focus on another variant of the OP, namely the Capacitated Team Orienteering Problem. CTOP considers each vehicle to have a limited capacity and each customer is associated with a demand for capacity. The objective of the CTOP is to optimize the profit generated for the fleet of vehicles by choosing customers with the consideration of each customer's demand and profit. The concept was first introduced by Archetti et al. (2009) and it has great practical usage in the logistics industry. Many of the logistics companies are now facing international competition, which forces them to cut costs in order to survive in this competitive market. One of the ways to cut cost is to maximise the number of goods each vehicle can hold. However, the survey papers conducted by Vansteenwegen et al. (2011) and Gunawan et al. (2016) showed that only few research works have been done on this particular topic. Therefore, this paper aims to contribute to the research on CTOP by providing a heuristic approach that can generate a good quality solution efficiently.

Archetti et al. (2009) proposed an exact approach and three heuristic approaches to solve this problem. The exact approach is based on the branch-and-price algorithm. The three heuristic approaches comprise of Variable Neighbourhood Search (VNS) algorithm and Tabu search algorithms. A novel Bi-level Filter-and-Fan method is proposed by Tarantilis

(2013). The proposed method consists of three components: a greedy parallel insertion-based construction heuristic to generate an initial feasible solution; a new Tabu Search based local search to identify a local optimal solution, and a novel filter-and-fan search to explore larger combined neighbourhoods and generate multiple search trajectories in an effort to overcome local optimality. The algorithm was able to match and improve some of the best reported results with competitive computational time. Luo et al. (2013) introduced an approach using an adaptive Ejection Pool (EP) with toggle-rule diversification. The proposed algorithm maintains the current solution in two parts: the first part consists of the selected customers and the second part consists of all the potential customers that are currently not selected. The potential customers are arranged based on their value, with the first one being the most valuable customer. Priority is given to the first potential customer if a replacement is to be made between the selected customer and potential customers. Another heuristic algorithm proposed is the Adaptive Iterative Destruction/Construction Heuristic (AIDCH) (Ben-Said et al., 2016). This algorithm starts with an adaptive construction phase based on the Best Insertion Algorithm, followed by an adaptive diversification phase with local search methodologies. A recent research work on CTOP was published by Gunawan et al. (2019), which proposed a heuristic algorithm based on the Iterated Local Search (ILS). This algorithm comprises of 4 main modules: initial solution, local search, perturbation and acceptance. The algorithm produced promising results as compared to other heuristic algorithms proposed previously.

2. Proposed Algorithm

We propose a heuristic which is inspired by the Simulated Annealing and Iterated Local Search (SAILS) algorithm (Gunawan et al., 2017). The entire algorithm is illustrated in Figure 1. The Simulated Annealing (SA) portion of the proposed algorithm is further modified by adapting the SA process proposed by Lin and Yu (2015) with a few minor adjustments. The first adjustment is done after generating the initial solution, with the addition of local search first before entering the looping process. The addition of an extra local search right after initial solution improves the efficiency of the algorithm by starting off the looping process with a much better initial condition (Gunawan et al., 2019). The second adjustment is done at the generation of the solution based on a previous solution through exploring the neighbourhood. The original SA process introduced by Lin and Yu (2015) has 3 types of iterators for exploring the neighbourhood, namely `Swap`, `Insert` and `Reverse`, each with a probability of 1/3 being chosen. Since Insertion will be performed exhaustively in the local search step, `Insert` is removed from this step for the proposed algorithm. More details of the operators will be explained below.

The Random Walk acceptance criterion (Vansteenwegen, 2014) is adapted for the proposed algorithm. This acceptance criterion provides a good balance between intensification and diversification when searching for solutions. Local search operators adapted from ILS (Gunawan et al., 2017) ensure that only solutions that are better than the current solution are kept, leading to search intensification. Random neighbourhood search with SA process allows the algorithm to explore neighbouring solution and have chance to escape local optima, leading to search diversification.

The algorithm starts off by first generating the initial solution X. The current temperature T is also set to the initial temperature $Tmax$. The algorithm then performs a round of local search on X to improve the initial solution, and the best-found solution $F(Z)$ is updated to $F(X)$. Upon completion of the neighbourhood search, a new solution, Y, is found and the objective value of Y, $F(Y)$, is compared against the objective value of X, $F(X)$. If $F(Y)$ is better or equal to $F(X)$, then X is replaced by Y. However, if $F(Y)$ is worse than $F(X)$, another random number r between 0 and 1 is generated and compared against $e^{\frac{F(Y)-F(X)}{T}}$, where X is replaced by Y if $r < e^{\frac{F(Y)-F(X)}{T}}$. Furthermore, if $F(Y)$ is better than $F(Z)$, Z is also replaced by Y. The neighbourhood search repeats itself until $I = Imax$.

Next, the temperature T is reduced with the formula $T = T \times \alpha$, where α is the cooling ratio. Local search is performed on Z to further improve the solution. Now, the algorithm performs a check to see if $F(Z)$ is improved after the local search. If $F(Z)$ is improved, then $N = 0$, $I = 0$, and the algorithm begins another round of neighbourhood search with $X = Z$. If $F(Z)$ is not improved, the non-improved count, N, increases by one and is compared against $Nmax$. If $N < Nmax$, $I = 0$ and the algorithm begins another round of neighbourhood search with $X = Z$. The algorithm terminates when $N = Nmax$.

```
1: Generate Initial Solution
2: F(Z) ←Apply Local Search
3: Set I = 0, N = 0, T = Tmax, F(Z) = F(X*)
4: while (N < Nmax) do
5:    while (I < Imax) do
6:       F(Y) ← Neighborhood Search
7:       I++
8:       if (F(Y) > F(X*))
9:          F(X*) ← F(Y)
10:      else
11:         Generate r ~ U(0,1)
12:         if (r < exp((F(Y) – F(X*)) / T))
13:            F(X*) = F(Y)
14:         else
15:            return to step 5
16:         if (F(X*) > F(Z))
17:            F(Z) = F(X*)
18:         else
19:            Return to step 5
20:   T = T × α
21:   F(Z*) ← Local Search
22:   if (F(Z*) > F(Z))
23:      N = 0
24:   else
25:      N = N + 1

26:   I = 0; return to step 4
```

Figure 1. Proposed Algorithm

In order to generate the initial solution, we implement the simple insertion heuristic (Luo et al., 2013). This method first ranks all the customers based on their potential value. The ranked customers are then inserted one by one into the available vehicles starting from the highest value customers. The process stops when no more customers can be added into any of the vehicles. Since service time is not included into the calculation of the cost for the given benchmark instances, the value calculation for the given instance would be:

$$value = \frac{profit}{demand} \tag{1}$$

Six different local search operators, as shown in Table 1, were adapted from Gunawan et al. (2017). All the six operators are executed in sequence given in Table 1 for every call of the neighbourhood search. `Swap1` selects the vehicle with the least remaining travel time. All possible combinations of exchanging positions between two different customers are performed. `Swap1` is considered successfully executed only if the exchange increases the travel time of the chosen vehicle. `Swap2` is similar to `Swap1`, with the exception of selecting two vehicles with the least remaining travel times. All possible combinations of exchanging positions of customers between two vehicles are performed. `Swap2` is considered successfully executed only if the exchange increases the total travel time from both vehicles. Both operations terminate when all possible combinations of exchange are performed.

`2-Opt` is executed by first selecting the vehicle with the least remaining travel time. All possible combinations of selecting two different customers are performed, and the

sequences of customers between the two selected customers are reversed. `Move` reallocates customers from one vehicle to another, with the objective of reducing total remaining time for all vehicles. The reallocation of customers should not violate any constraints and the operation terminates when all customers tried reallocating to all locations in all vehicles.

All the above-mentioned operators do not change the objective function value. They modify the current solution in order to increase the total remaining travel time. This may provide more opportunities for the next two operators, namely `Insert` and `Replace`, to improve the objective function value by adding or replacing customers from the group of unassigned customers. `Insert` rearranges all customers that are not assigned to the vehicles, based on their values in ascending order. Each unassigned customer would now be inserted into the vehicles without violating any constraints. If there are multiple insertion locations available for this unassigned customer, the location with the least addition of total traveling time will be chosen. `Replace` replaces customers assigned to vehicles with customers that have not been selected. The vehicle with the most remaining travel time is chosen for this operation. All unassigned customers are rearranged based on their values. The highest-valued unassigned customer is then selected to replace any customer in the vehicle that has a value lower than that of the unassigned customer without violating any constraints. If there are multiple potential customers in the vehicle that can be replaced by the unassigned customer, replace the customer that will result in the least addition of travel time. Each time after a successful replacement, rearrangement of the unassigned customers will be done and the highest-valued unassigned customer is chosen for the next replacement.

Table 1. Local Search operators

Operator	Definition
Swap1	Exchange two customers within one vehicle
Swap2	Exchange two customers between two vehicles
2-Opt	Reverse the sequence of certain customers within a vehicle
Move	Move one customer from one vehicle to another vehicle
Insert	Insert or add customers to a vehicle
Replace	Replace one customer in a vehicle with another customer that has not been selected

3. Computational Results

The proposed algorithm was implemented in C++ programming language and the computational runs were performed on a CPU with MacOS, Intel Core i5 2.7 GHz Dual-Core processor and 8 GB of RAM. Benchmark instances from Tarantilis (2012) were used to test the proposed algorithm.

The comparison of the results from the proposed solution against other well-known algorithms are presented below. The results of the-state-of-the-art algorithms were used to compare with the results generated from the proposed algorithm. Here, Variable Neighbourhood Search (VNS) and Tabu Search (feasible) (TSf) correspond to the algorithms proposed by Archetti et al. (2009); Bi-level Filter-and-Fan Fast (BiF&F-f) corresponds to the algorithm proposed by Tarantilis et al. (2012); Iterated Local Search (ILS) corresponds to the algorithm proposed by Gunawan et al. (2019). It is observed that keeping $Imax$ at 3000, $Nmax$ at 10, $Tmax$ at 1 and α at 0.99 provides the opportunity to find most of the best-known solutions (BK) at relatively short run time. Due to space constraints, only a summary of the result comparisons is presented in this section. Note that "p" refers to the average objective function value and "$t(s)$" refers to the computation time (in seconds). $\%d$ refers to the percentage difference between "p" for proposed algorithm and "p" for best known solution (BK).

Table 2. Computational Results

Instance	BK	VNS			TSf			BiF&F-f			ILS			SA_ILS		
	p	*p*	*t*(s)	%*d*	*p*	*t*(s)	%*d*	*p*	*t*(s)	%*d*	*p*	*t*(s)	%*d*	*p*	*t*(s)	%*d*
Set 1	1814.2	1814.2	**0.0**	0.00	1814.0	43.3	0.02	**1824.0**	0.2	**-0.34**	1814.0	19.0	0.01	**1824.0**	21.4	-0.32
Set 2	295.2	**294.9**	667.1	0.07	295.0	505.9	0.06	295.1	**7.9**	**0.03**	292.5	30.0	0.88	293.1	24.1	0.56
Set 3	728.4	728.4	1028.5	**0.00**	726.2	388.0	0.29	**728.5**	**6.0**	**-0.02**	727.9	128.4	0.08	725.5	55.6	0.39
Average	945.9	945.8	565.2	0.03	945.0	312.4	0.12	**949.0**	**4.7**	**-0.11**	944.8	59.2	0.32	947.0	33.7	0.21

As seen from the table above, the proposed algorithm is comparable against the other algorithms in terms of quality and computation time. Furthermore, it is also able to improve one best-known solution from Set 1 of the benchmark instances. The proposed algorithm's computation time is also shorter as compared to the algorithms VNS, TSf and ILS. However, it is worth mentioning that BiF&F-f algorithm is far more superior than all other algorithms in both the results and computation time.

4. Conclusion

In this paper, Simulated Annealing with Iterated Local Search (ILS) metaheuristic algorithm is proposed to solve the Capacitated Team Orienteering Problem (CTOP). It combines Simulated Annealing process with ILS operators to enhance the algorithm's diversification and intensification. The algorithm is then used to solve benchmark instances and the results are compared against other state-of-the-art algorithms. The proposed algorithm displayed good performance that is comparable with other state-of-the-art algorithms in both the results and computation time. Furthermore, the relatively simple algorithm structure, along with few user-defined parameters, indicates the applicability of the proposed approach towards solving other variants of the Orienteering Problems, as well as the real-life Orienteering Problems. The results are still preliminary with some future research directions, such as developing more local search operators and a more rigorous tuning procedure using statistical tests.

Reference

Archetti, C., Feillet, D., Hertz, A., & Speranza, M. G. (2009). The capacitated team orienteering and profitable tour problems. *Journal of the Operational Research Society, 60*(6), 831-842.

Ben-Said, A., El-Hajj, R., & Moukrim, A. (2016). An adaptive heuristic for the capacitated team orienteering problem. *IFAC-PapersOnLine, 49*(12), 1662-1666.

Butt, S. E., & Cavalier, T. M. (1994). A heuristic for the multiple tour maximum collection problem. *Computers and Operations Research, 21*(1), 101-111.

Chao, I., Golden, B. L., & Wasil, E. A. (1996). The team orienteering problem. *European Journal of Operational Research, 88*(3), 464-474.

Golden, B. L., Levy, L., & Vohra, R. (1987). The orienteering problem. *Naval Research Logistics (NRL), 34*(3), 307-318.

Gunawan, A., Lau, H. C., & Vansteenwegen, P. (2016). Orienteering problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research, 255*(2), 315-332.

Gunawan, A., Lau, H. C., Vansteenwegen, P., & Lu, K. (2017). Well-tuned algorithms for the team orienteering problem with time windows. *Journal of the Operational Research Society, 68*(8), 861-876.

Gunawan, Aldy; Ng, Kien Ming; Yu, Vincent F.; Adiprasetyo, Gordy; and Lau, Hoong Chuin. The capacitated team orienteering problem. (2019). Proceedings of the 9th International Conference on Industrial Engineering and Operations Management Bangkok, Thailand, March 5-7, 2019. 1630-1638.

Lin, S., & Yu, V. F. (2015). A simulated annealing heuristic for the multiconstraint team orienteering problem with multiple time windows. *Applied Soft Computing, 37*, 632-642.

Luo, Z., Cheang, B., Lim, A., & Zhu, W. (2013). An adaptive ejection pool with toggle-rule diversification approach for the capacitated team orienteering problem. *European Journal of Operational Research, 229*(3), 673-682.

Tarantilis, C. D., Stavropoulou, F., & Repoussis, P. P. (2013). The capacitated team orienteering problem: A bi-level filter-and-fan method. *European Journal of Operational Research, 224*(1), 65-78. doi:10.1016/j.ejor.2012.07.032

Vansteenwegen, P., Souffriau, W., & Oudheusden, D. V. (2011). The orienteering problem: A survey. *European Journal of Operational Research, 209*(1), 1-10.

A simulated annealing approach for the tourist trip design problem with the pattern sequence of the points of interest

Vigan Abdurrahmani · Kadri Sylejmani · Lule Ahmedi

1 Introduction

The Tourist Trip Design Problem (TTDP) [2] is about planning the trip itinerary for a tourist, when he/she visits a particular place (e.g. a city) for a certain period of time (e.g. couple of days). It is assumed that the place comprises of a number of Points of Interest (POIs), and the satisfaction factor of the tourist for each of the POIs is a known figure, whereas, each POI is characterized with its own attributes (e.g. opening hours, visit duration, cost of visit, specific types/categories, etc.). In addition, the tourist can enforce constraints, such as amount of money to spent on the trip and maximum number of POIs of certain type/category to visit. In this abstract, we present an extended variant of the TTDP problem, where we allow the tourist to express patterns of the visits to the POIs, in terms of having a certain number of types of POIs visited in a predefined sequence. For example, if the tourist has selected the pattern *monument-castle-museum*, for his/her first day of the tour, then the plan should have at least one *monument*, one *castle* and one *museum* included into the itinerary. Moreover, the specified POIs should be planned in the given sequence, although other ones, of any given type, can be inserted in-between. Our approach for tackling this newly defined variant

V. Abdurrahmani
Faculty of Electrical and Computer Engineering, University of Prishtina, Bregu i Diellit p.n.
Tel.: +383-38-554896
E-mail: vigan.abdurrahmani@uni-pr.edu

✉ K. Sylejmani
Faculty of Electrical and Computer Engineering, University of Prishtina, Bregu i Diellit p.n.
Tel.: +383-38-554896
E-mail: kadri.sylejmani@uni-pr.edu

L. Ahmedi
Faculty of Electrical and Computer Engineering, University of Prishtina, Bregu i Diellit p.n.
Tel.: +383-38-554896
E-mail: lule.ahmedi@uni-pr.edu

of TTDP problem is based on the simulated annealing meta-heuristic and it utilizes the concept of POI pivoting for construction of good starting solutions.

2 Modeling the Tourist Trip Design Problem

In the literature, the TTDP problem is mostly modeled based on the Orienteering Problem (OP) [5] and its derived variants. In the OP, during a single and limited period of time, among a given number of points, a subset of them has to be visited, with the objective of selecting points whose total satisfaction factor is maximized. The Team OP (TOP)[1] enables multiple periods (e.g. days), whereas OP with Time Windows (OPTW) allows modeling service periods (e.g. opening hours) of points. Further, Time Dependent TOP (TDTOP)[3] makes it possible to consider variability within the distances between the points (e.g. walking or traveling by public transport). The Multi Constrained TOPTW (MCTOPTW)[8] problem represents a specific variant, where it is possible to express certain additional knapsack constraints, such as limiting the total cost to spent for visiting points, or enforcing upper limits about the number of points of certain category that can be visited (e.g. at most three points of the category of *architecture*). Furthermore, Multi Constrained Multiple TOPTW (MCMTOPTW)[9] is used to model multiple periods for multiple users (e.g. multiple day trips for multiple tourists), where each user gets its personalized itinerary, which, at certain points can overlap, hence allowing them to be together for some part of the trip. For an extensive study of the existing variants of OP and the respective approaches used for solving them, the reader is referred to Gunawan et al. [6].

In this paper, we define a new model, tagged as MCTOPTW with Patterns (MCTOPTWP), which enables adding an additional hard constraint that enforces the presence of a pattern in the form of a predefined sequence of points within each period of the itinerary. This model will allow the tourist to make certain types/categories of POIs as mandatory to be included into the itinerary [4], and moreover, the order of visits to such types/categories is enforced.

3 Problem Formulation

In more specific terms, the MCTOPTWP problem consists of the following inputs:

Number of tours. The total number of visiting periods (e.g. days).
Budget. The whole budget available for all tours.
Points of interest (POI). These are locations described by geographic coordinates (latitude and longitude), visit duration, satisfaction score, opening and closing time of the location, cost of the visit and the category of the location (might contain multiple categories).
Max allowed visits per category. This represents maximum number of visits per category type of POI for the tour, example *Museum: 3* would allow us to visit at most 3 museums during our tour.

Patterns. Patterns represent sequence of categories of POIs that should be visited, for example a pattern could consist of the following: *Museum*, *Castle*, *Natural Park*.

In our problem there are no soft constraints and all hard constraints must be satisfied in order to have a valid solution. The problem is defined by the hard constraints depicted below:

Visit once. Each POI could be visited at most once during the whole tour no matter in which day.
Time window. A POI cannot be visited before it's opening time or after it's closing time, however a visit could last after it's closing time if the visit starts before closing time.
Day duration. During each day of the tour we cannot exceed the duration defined by opening and closing time of starting POI.
Starting POI. Each daily visit should start from the first POI and end up at same POI.
No simultaneous visits to POIs. During each visit we can visit at most one location at a time.
Budget limit. Total cost of all days should not exceed the input budget.
Max visit per category. During the whole tour we should not visit more POIs of specific category than the defined number in input.
Pattern sequence. It is important that every day should fulfill its category pattern (sequence) and respect its POI order, although other POIs (of any category) could be visited in between. For example if we have the pattern *Museum*, *Castle*, *Natural Park* for a specific day, during that day we could visit POIs of categories *Museum*, *Castle*, *Seaside*, *Natural Park*. Note that *Seaside* category is not part of the pattern, still it can be visited, since the required order (in the pattern) *Museum*, *Castle* and *Natural Park* is maintained.
Travel duration. During the visit we should calculate also the travel duration from one POI to another. The duration is calculated as Euclidean distance between the geographic locations of any two POIs.

The objective function of the problem is maximizing the total satisfaction score of the tour which consists of sum of the satisfaction scores of each visited POI. We can increase this score by visiting as much POIs as we can while respecting hard constraints and reducing the waiting time between consecutive visits.

4 Search Method

The search method is based on the Simulated Annealing (SA) approach as presented in Algorithm 1. In abstract terms, the proposed approach can be described as in the following:

Search Space. The search space consists of the combinations that allocate all POIs into one of the two subsets. The first one makes the assigned

subset, which represents a sequence of POIs (i.e. that make the tourist itinerary), whilst the second subset contains the unassigned POIs (i.e. those that do not get placed into the tourist itinerary). The states that violate the hard constraints are excluded from the search space.

Neighborhood Structure. The neighborhood structure consists of the union of three basic moves:

Insert: Inserts a POI from a unassigned subset into the assigned subset

Remove: Removes a POI from the assigned subset and places it into the unassigned subset, and

Swap: Swaps two POIs between the assigned and unassigned subsets.

In the course of a given iteration, we select a non-pivot POI that is within the itinerary and takes up the larges amount of time, then we replace it with one of the POIs outside of the itinerary. The replacement policy is based on the heuristic that tends to chose POIs of types that are less represented into the itinerary and have high satisfaction factors. After the swap move, we successively apply the insert move aiming to fill empty spaces that might have appeared. Whenever a certain POI is subject to a move, than all the following POIs are shifted forward or backward in time (in the respective itinerary of the affected day) to reflect the changes that occurs due to the process of moving the POI.

Cooling Strategy. The cooling strategy is based on the function defined by Lundy and Mees [7] that is expressed by equation $T_t = T_{t-1}/(1+\beta T_{t-1})$. A low value of β parameter (typically close to zero) makes the temperature change at a slow rate, hence making the algorithm do more exploration into the search space.

Stopping Criterion. The stopping criterion of the algorithm is determined by the minimal temperature parameter, which in this case is set to zero.

Initial Solution. The initial solution is constructed based on two phases, the first adding greediness and the second adding randomness attributes to the initial solution. The first phase uses the so called concept of pivoting, where some selected *pivot* POIs are placed first into the itinerary. For a given day, for each type in the pattern sequence, we make a group of POIs that belong to that specific type, and then we order them (in decreasing mode) based on their satisfaction factor. The POI with the highest satisfaction factor, in a given group, is selected first to act as a pivot POI, hence it will be the first to be considered for insertion into the itinerary of that given day. In case, the first POI does not meet the hard constraints, then the process of selection of a pivot POI is repeated for other candidates in the corresponding group, by using a backtracking strategy. In the second phase, the remaining places are filled randomly with other POIs that are not part of the itinerary yet.

Perturbation. The perturbation mechanism uses all of the three moves in a sequence and it is applied only when a number of iterations without improvement are passed (as defined by *MAX_ITERS* parameter). First, it removes randomly a number (as specified by *MAX_DEL* parameter) of non-pivot POIs from the itinerary. Then, it makes a number (that is selected

at random between *MIN_TRIES* and *MAX_MAX*) of consecutive swaps between pivot POIs within itinerary and pivot POIs outside of the itinerary. Finally, it makes several attempts (as specified by *MAX_INS* parameter) to randomly insert unassigned POIs into the itinerary.

Algorithm 1 Simulated Annealing

```
1: procedure SOLVE(input, T_min, T_max, β, MAX_ITERS, MAX_DEL, MIN_TRIES,
       MAX_TRIES, MAX_INS)
2:     S, Q, P ← generateInitialSol(input) // S - solution, Q - unassigned, P - pivots
3:     best ← S, t ← T_max, i ← 0
4:     while t > T_min do
5:         a ← findPoiWithHighestSpace(S)
6:         if a is pivot then
7:             i ← MAX_ITERS
8:         else
9:             Q ← sort(Q)
10:            for b in Q do
11:                R ← clone(S)
12:                type ← selectTypeByPolicy(b)
13:                if swap(a, b, type, R) then
14:                    inserted ← fillEmptySpaces(R, Q)
15:                    if eval(R) > eval(S) or e^((eval(R)−eval(S))/t) > random[0, 1] then
16:                        S ← R
17:                        remove(inserted, Q), remove(b, Q)
18:                        add(a, Q)
19:                    end if
20:                    if eval(S) > eval(best) then
21:                        best ← S, i ← 0
22:                    else
23:                        i ← i + 1
24:                    end if
25:                else
26:                    i ← i + 1
27:                end if
28:            end for
29:        end if
30:        if i > MAX_ITERS then
31:            randomRemove(S, Q, P, MAX_DEL)
32:            swapPivot(S, Q, P, MIN_TRIES, MAX_TRIES)
33:            randomInsert(S, Q, MAX_INS)
34:            i ← 0
35:        end if
36:        t = t/(1 + βt)
37:    end while
38:    return best
39: end procedure
```

Table 1 Comparison of the results of simulated annealing against Iterated Local Search approach

	Iterated Local Search (ILS)		Simulated Annealing (SA)			
Instance	Time (sec)	Fitness (best)	Time (sec)	Fitness (best)	Fitness (avg)	ILS vs. SA (%)
MCTOPTWP-1-pr04	0.49	433	2.21	369	306.9	14.8
MCTOPTWP-1-c105	0.08	314	2.08	330	305.0	-5.1
MCTOPTWP-2-c108	0.25	654	1.08	570	496.0	12.8
MCTOPTWP-2-pr07	0.27	540	1.81	513	435.1	5.0
MCTOPTWP-2-pr08	0.96	764	2.94	655	546.1	14.3
MCTOPTWP-3-pr01	0.19	586	1.25	550	490.0	6.1
MCTOPTWP-3-pr09	2.82	1133	2.45	838	679.5	26.0
MCTOPTWP-3-c107	0.29	861	1.40	800	721.0	7.1
MCTOPTWP-4-r111	0.61	858	0.94	728	633.9	15.2
MCTOPTWP-4-pr04	1.86	1464	4.30	1120	920.5	23.5

5 Results

The above described model is tested by using a test set of ten instances that are derived based on a large instance set in the literature. Further, the results of our approach [1] are compared against the Iterated Local Search approach of [10] for the TOPTW, but which we adopted for the new model presented in this paper (i.e. MCTOPTWP). The adopted ILS solver can be obtained in GitHub [2]. As it can be seen in Table 1, in terms of fitness, in one (out of ten) instances our algorithm performs better than the ILS approach, and, in average, our algorithm falls behind the ILS for about 12% . In addition, in terms of computation time, our approach has an average computation time of 2.04 sec, while the ILS approach takes, in average 0.78 sec.

6 Conclusion and Future Work

The results presented above show that our approach is quite quick as it is able to produce good solutions in a matter of few seconds, which makes it suitable for use in practical applications. As part of future work, new neighborhood operators will be developed and the algorithm will be tested against a greater set of instances that are derived from the existing test in the literature.

Acknowledgements We thank Bsc. Festim Prebreza for implementing, from scratch, the state of the art algorithm ILS[10], who made it possible for us to have reference results for comparison purposes.

References

1. I-Ming Chao, Bruce L Golden, and Edward A Wasil. The team orienteering problem. *European journal of operational research*, 88(3):464–474, 1996.

[1] GitHub `https://github.com/vigan-abd/mctopp`

[2] GitHub `https://github.com/festimprebreza/iteratedLocalSearch`

2. Damianos Gavalas, Charalampos Konstantopoulos, Konstantinos Mastakas, and Grammati Pantziou. A survey on algorithmic approaches for solving tourist trip design problems. *Journal of Heuristics*, 20(3):291–328, 2014.
3. Damianos Gavalas, Charalampos Konstantopoulos, Konstantinos Mastakas, Grammati Pantziou, and Nikolaos Vathis. Heuristics for the time dependent team orienteering problem: Application to tourist route planning. *Computers & Operations Research*, 62:36–50, 2015.
4. Michel Gendreau, Gilbert Laporte, and Frederic Semet. A branch-and-cut algorithm for the undirected selective traveling salesman problem. *Networks: An International Journal*, 32(4):263–273, 1998.
5. Bruce L Golden, Larry Levy, and Rakesh Vohra. The orienteering problem. *Naval Research Logistics (NRL)*, 34(3):307–318, 1987.
6. Aldy Gunawan, Hoong Chuin Lau, and Pieter Vansteenwegen. Orienteering problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research*, 255(2):315–332, 2016.
7. Miranda Lundy and Alistair Mees. Convergence of an annealing algorithm. *Mathematical programming*, 34(1):111–124, 1986.
8. Kadri Sylejmani, Jiirgen Dorn, and Nysret Musliu. A tabu search approach for multi constrained team orienteering problem and its application in touristic trip planning. In *2012 12th International Conference on Hybrid Intelligent Systems (HIS)*, pages 300–305. IEEE, 2012.
9. Kadri Sylejmani, Jürgen Dorn, and Nysret Musliu. Planning the trip itinerary for tourist groups. *Information Technology & Tourism*, 17(3):275–314, 2017.
10. Pieter Vansteenwegen, Wouter Souffriau, Greet Vanden Berghe, and Dirk Van Oudheusden. Iterated local search for the team orienteering problem with time windows. *Computers & Operations Research*, 36(12):3281–3290, 2009.

Local Search Neighborhoods for Industrial Test Laboratory Scheduling with Flexible Grouping

Florian Mischek[1] · **Nysret Musliu**[1] · **Andrea Schaerf**[2]

Keywords Project Scheduling · RCPSP · Industrial Test Laboratory · Local Search

1 Introduction

The Test Laboratory Scheduling Problem (TLSP) arises in a real-world industrial test laboratory, where a large number of activities in multiple projects has to be scheduled, subject to several legal and operational constraints. It is an extension of the well-known (Multi-Mode) Resource-Constrained Project Scheduling Problem ((M)RCPSP) (see e.g. [2,5]) which, in addition to other extensions, includes several unique features.

Most importantly, the activities to be scheduled (*jobs*) are not monolithic, but composed of multiple smaller units called *tasks* and derive all their properties from the tasks they contain. The grouping of tasks into jobs must be determined by the solver as part of the solution process. A similar concept exists in the form of batch scheduling (e.g. [14,12]) or schedule-dependent setup times (e.g. [8,9]). The difference is that in these settings, tasks are scheduled directly and batches arise implicitly from the final schedule.

TLSP also uses heterogeneous resources, with general restrictions on which units of a resource can be used for each task. While usually, variants of RCPSPS assume a homogeneous resource model, similar restrictions can be found in the Multi-Skill RCPSP (MSPSP) [1], where each resource unit possesses a set of skills and requirements are also formulated in terms of skills.

[1] Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling
DBAI, TU Wien
E-mail: {fmischek,musliu}@dbai.tuwien.ac.at
[2] University of Udine
E-mail: schaerf@uniud.it

Finally, TLSP introduces the notion of *linked tasks*, which have to be performed by the same employees. To the best of our knowledge, no other published variant of TLSP contains a similar concept. The only reasonable close approximation can be found in [13], where (some) resource assignments are modeled as different modes and some activities are constrained to be performed in the same mode.

Previous solution approaches have focused on a subproblem of TLSP, where a suitable grouping of tasks into jobs is given as input and cannot be modified by the solver (TLSP-S) [11,4]. However, such a "known-good" grouping cannot always be provided, and simple greedy grouping approaches often result in inferior or even infeasible groupings.

In this extended abstract, we extend the metaheuristic algorithms from [11] by new neighborhoods that deal with the grouping of tasks into jobs. This way, we obtain a solution approach for TLSP that does not require a provided initial grouping. We show that using Simulated Annealing (SA) with these new neighborhoods, we can quickly create high-quality schedules even for large instances. Preliminary results indicate that this approach produces better results than both SA for TLSP-S (even considering that the latter has the advantage of knowing a good grouping from the start). It also outperforms other solution approaches for TLSP, including an exact Constraint Programming (CP) model and, under tight time limits, a Very Large Neighborhood Search (VLNS)[3].

2 Problem definition

TLSP was first defined in a technical report [10], which also contains the full problem description. We provide here a summary of the main properties and constraints.

In TLSP, the solver has to find a schedule which consists of a partitioning of the tasks into jobs, and an assignment of a mode, timeslot and resources for each job.

A job derives its properties from the tasks it contains, which must all come from the same project and family. Within a job, tasks are executed sequentially, but without any defined order. This implies that it must fulfill all requirements of each task for its whole duration, which is the sum of the durations of its tasks plus an additional setup time. For example, the set of available units of each resource is the intersection of the available units of each contained task.

Feasible schedules must satisfy a number of constraints. This includes release dates and deadlines, available resources, linked tasks (and jobs), precedence constraints, fixed assignments, and others.

The quality of a schedule is determined via an objective function that is the weighted sum of several criteria, such as the number of jobs, the total completion time from the start to the end of each project, the number of employees assigned to each project, preferred employee assignments and internal target dates, which are usually slightly before the actual deadline.

3 Local Search framework

In this section, we give an overview of the local search framework and the Simulated Annealing (SA) metaheuristic described in [11], where it was used to solve TLSP-S.

The main concept of the framework is that of *neighborhoods*, which provide various methods to access the *moves* they contain. Solver implementations, such as metaheuristics, can be easily implemented and adapted for different problem variants simply by setting the neighborhoods they operate on.

In [11], we implemented different neighborhoods for TLSP-S. One of the best performing configurations was a combination of two neighborhoods, called *JobOpt* and *EquipmentChange*. JobOpt contains moves that modify the mode, timeslot, workbench and employee assignments of a single job, while EquipmentChange contains moves that replace a single assigned equipment unit by a different one. This special handling for equipment was required due to the sometimes huge number of potential equipment assignments, which made a neighborhood that simultaneously swapped all resources unwieldy in practice.

From among several different well known metaheuristics we implemented for our framework, we achieved the best results for TLSP-S with Simulated Annealing (SA) [7].

4 New neighborhoods

The adaptation required to make the solver for TLSP-S described in the previous section also suitable for TLSP is the introduction of new neighborhoods that contain regrouping moves, together with a careful reconfiguration of the search algorithm. If these neighborhoods are combined with those for TLSP-S [11], the same search algorithm can be used to solve also instances for TLSP.

The main challenge presenting itself is the number of potential partitions for the tasks in a family, which grows exponentially with the size of the family (already 115975 combinations for 10 tasks, which is met or exceeded by 6% of all families in our data sets). Moreover, many of these groupings (in particular if the tasks are short) will result in identical or nearly identical jobs.

We developed three new neighborhoods that modify the task grouping of a project in different ways. In combination with the existing neighborhoods for TLSP-S (JobOpt and EquipmentChange), which deal with mode, timeslot and resource assignments, these form a complete and efficient set of neighborhoods for TLSP.

All three new neighborhoods are implemented in such a way as to guarantee that their moves do not result in any new hard constraint violations, except for constraint H8 (Single Assignment) and H11 (Linked Jobs), where conflicts are allowed. This entails the following general restrictions on moves:

- Whenever two existing jobs are involved, they must belong to the same project and family.
- Fixed tasks cannot be moved to a different job.

- Whenever one or multiple tasks are added to an existing job, its mode and resource assignments must be available for all added tasks.
- In addition to the above point, there must also be a valid timeslot assignment for the job, respecting both time windows (which may changed due to the added tasks) and precedence constraints. In particular, moving tasks must not result in cycles in the precedence graph.

The new neighborhoods are:

Single Task Transfer: A single task from a source job is moved to a target job. Resource assignments are kept as-is, except where requirements change. In this case, resource assignments are adjusted as necessary. Mode and timeslot assignments are not modified (only exception: the target job may by moved to an earlier timeslot if necessary to fulfill due date or precedence constraints).

Split Job: A job is split in two by moving a randomly selected subset of tasks to a new job. It is guaranteed that doing so does not introduce cycles in the dependency graph and each job receives at least one task. For the mode, timeslot and resource assignments of the new job, several strategies are possible (see below). Resource assignments of the source job may be adjusted as described above.

Merge Job: All tasks of a job are moved to another job and the source is deleted. As for the transfer, resource assignments and timeslot of the target may have to be adjusted, but are otherwise kept as-is.

We have implemented different strategies to adjust resource assignments for affected jobs, if the requirements change due to a move (if tasks are added or removed, requirements can only increase or decrease, respectively). The resource units to add/remove can be chosen either randomly from all available units or the best units can be chosen - i.e. those that minimize the conflicts involving the job.

For the newly created job in the Split Job neighborhood, which does not have any initial assignments, different options are available. The timeslot can be either directly after the end of the source, chosen randomly (within time window and respecting precedence constraints) or the position that minimizes conflicts. The resource assignments can be copied from the source (and adjusted using either of the above strategies), assigned randomly or chosen such that they minimize conflicts in the selected position.

5 Preliminary Results

For our experiments, we used the same solver implementation as in [11], with the addition of the three new neighborhoods. We used SMAC3 [6], version 0.11.0, to tune the following configuration parameters (six independent parameters in total): The selection probabilities for the five neighborhoods during each step and the strategies to adjust resource assignments or handle assignments for the newly created job in the Split Job neighborhood.

The benchmark data set is the same as the one used in [11], which can be downloaded at `https://www.dbai.tuwien.ac.at/staff/fmischek/TLSP/`. It contains 30 randomly generated instances, ranging in size from 5 projects and around 30 tasks up to 90 projects and over 1500 tasks, as well as 3 anonymized real-world instances from our industrial partner.

Our results show that with a timeout of 10 minutes, feasible solutions could be found for all instances in nearly all runs, starting out from a simple greedily constructed initial grouping. All neighborhoods contributed to finding feasible solutions. Omitting any single neighborhoods led to a strong decrease in the number of feasible solutions, despite low weights for the Split and Merge neighborhoods in the best configuration found by SMAC.

We also compared our results to those for TLSP-S, using the algorithm and neighborhoods reported in [11]: On average, results for TLSP are better on 19 of the 33 instances, despite the fact that we did not start out from a known good and guaranteed to be feasible grouping. For some instances, we could even generate schedules that are better than the proven optima for TLSP-S under the previously known fixed grouping.

Our results also improve upon those achieved using an exact CP model for TLSP written in MiniZinc. In particular for instances with 20 projects or more, SA always found better solutions, with up to half the penalty of the results using CP.

In comparison with VLNS [3] using the CP model internally, SA finds competitive results overall. As for CP, the relative performance of SA is better on larger instances, where it slightly outperforms VLNS.

Results for longer timeouts of two hours indicate similar findings.

6 Conclusions

We have developed three new local search neighborhoods that contain task (re)grouping moves for TLSP. They can be used in combination with existing neighborhoods for mode, timeslot and resource assignments to solve the full TLSP, without any known initial grouping.

With our local search solver using simulated annealing, we could create high-quality schedules, that show improvements compared to TLSP-S, which starts out from a known good grouping. This approach also outperforms a CP model for TLSP on most instances, and is significantly better on all instances with more than 20 projects. On these large instances, it is also slightly better than a VLNS approach, while remaining competitive on the smaller instances.

For the future, we aim to perform extensive evaluations of our algorithms and their performance, and further improve the task grouping operators. We also plan to investigate other solution approaches for TLSP, such as hyper-heuristics.

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

References

1. Bellenguez, O., Néron, E.: Lower bounds for the multi-skill project scheduling problem with hierarchical levels of skills. In: E. Burke, M. Trick (eds.) Practice and Theory of Automated Timetabling V, pp. 229–243. Springer Berlin Heidelberg, Berlin, Heidelberg (2005). DOI 10.1007/11593577_14
2. Brucker, P., Drexl, A., Möhring, R., Neumann, K., Pesch, E.: Resource-constrained project scheduling: Notation, classification, models, and methods. European Journal of Operational Research **112**(1), 3 – 41 (1999). DOI https://doi.org/10.1016/S0377-2217(98)00204-5. URL http://www.sciencedirect.com/science/article/pii/S0377221798002045
3. Danzinger, P., Geibinger, T., Mischek, F., Musliu, N.: Solving the test laboratory scheduling problem with variable task grouping. In: submitted to Internation Conference on Planning and Scheduling (ICAPS) 2020
4. Geibinger, T., Mischek, F., Musliu, N.: Investigating constraint programming for real world industrial test laboratory scheduling. In: Proceedings of the Sixteenth International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR 2019) (2019)
5. Hartmann, S., Briskorn, D.: A survey of variants and extensions of the resource-constrained project scheduling problem. European Journal of Operational Research **207**(1), 1 – 14 (2010). DOI https://doi.org/10.1016/j.ejor.2009.11.005. URL http://www.sciencedirect.com/science/article/pii/S0377221709008558
6. 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)
7. 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. URL https://science.sciencemag.org/content/220/4598/671
8. Mika, M., Waligóra, G., Węglarz, J.: Modelling setup times in project scheduling. Perspectives in modern project scheduling pp. 131–163 (2006)
9. Mika, M., Waligóra, G., Węglarz, J.: Tabu search for multi-mode resource-constrained project scheduling with schedule-dependent setup times. European Journal of Operational Research **187**(3), 1238 – 1250 (2008). DOI https://doi.org/10.1016/j.ejor.2006.06.069. URL http://www.sciencedirect.com/science/article/pii/S0377221706008344
10. Mischek, F., Musliu, N.: The test laboratory scheduling problem. Technical report, Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling, TU Wien, CD-TR 2018/1 (2018)
11. Mischek, F., Musliu, N.: A local search framework for industrial test laboratory scheduling. Annals of Operations Research **302**, 533–562 (2021). DOI 10.1007/s10479-021-04007-1
12. Potts, C.N., Kovalyov, M.Y.: Scheduling with batching: A review. European Journal of Operational Research **120**(2), 228 – 249 (2000). DOI https://doi.org/10.1016/S0377-2217(99)00153-8. URL http://www.sciencedirect.com/science/article/pii/S0377221799001538
13. Salewski, F., Schirmer, A., Drexl, A.: Project scheduling under resource and mode identity constraints: Model, complexity, methods, and application. European Journal of Operational Research **102**(1), 88 – 110 (1997). DOI https://doi.org/10.1016/S0377-2217(96)00219-6. URL http://www.sciencedirect.com/science/article/pii/S0377221796002196
14. Schwindt, C., Trautmann, N.: Batch scheduling in process industries: an application of resource–constrained project scheduling. OR-Spektrum **22**(4), 501–524 (2000)

A Hybrid Approach for Paint Shop Scheduling in the Automotive Supply Industry

Felix Winter · Nysret Musliu

1 Introduction

Modern day paint shops of the automotive supply industry need to manufacture a large number of products every day. As manual planning approaches can nowadays hardly cope with the growing demands of car manufacturers that are raised by the recent trend towards full automation, there is a strong need for automated scheduling techniques in this area.

One of the most important cost objectives of any automotive paint shop is to minimize the color changes that appear in the production sequence. Since the painting equipment has to be cleaned after each change, any color change will lead to a loss of valuable resources and can cause problematic delays. Minimizing the required color changes is not an easy task on its own, but several constraints that restrict technically feasible color sequences make it even more challenging to automatically create efficient schedules in practice.

In the literature, many publications exist on the topic of minimizing color changes in automotive scheduling problems (e.g. [4,3,2]) and several variants have been shown to be NP-complete [1]. However, most of the previous publications focus on problems that appear in car manufacturing, which although similar, include different constraints and objectives than the ones that are occurring in paint shops of the automotive supply industry.

Recently, we introduced a real-life paint shop scheduling problem from the automotive supply industry together with a set of 24 real-life problem instances in [6]. In [5] and [7] we further investigated exact approaches based on constraint programming for paint shop scheduling and showed that the problem is NP-hard. Experimental results revealed that exact approaches could prove

Felix Winter and Nysret Musliu
Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling
DBAI, TU Wien
E-mail: {winter,musliu}@dbai.tuwien.ac.at

optimal results for some of the smaller instances, but could not solve any of the larger instances in reasonable time. The empirical evaluation further showed that a simulated annealing approach can successfully produce feasible solutions for all real life instances within one hour. However, the number of required color changes in the solutions was still very high for most of the instances, resulting in schedules that have a large potential for further cost improvements in practice.

In this work, we investigate innovative solution methods that can hybridize exact and metaheuristic approaches to quickly find cost efficient schedules for real life paint shop scheduling problems. To realize an enhanced minimization of the required color changes in paint shop schedules, we propose a novel problem formulation that can be used to minimize the color changes of a given feasible solution without violating any of the problem's hard constraints.

The main idea behind the new formulation is to take a candidate solution to the paint shop scheduling problem and formulate a minimization problem based on the given solution that aims to minimize the color changes in the schedule without changing decisions on the scheduled materials and production devices. Therefore, an exact approach solving the proposed minimization problem does not need to consider the complex constraints regarding materials and production devices that appear in the original paint shop scheduling problem which caused previous exact approaches for the original problem to be impractical for large real life instances.

We provide a constraint programming model that we utilize to efficiently find optimized solutions to the new problem formulation and show that we can thereby improve feasible schedules that have been achieved with metaheuristic approaches. Furthermore, we investigate destroy and repair neighborhood operators that allow the incorporation of the proposed exact approach into metaheuristic methods within the framework of large neighborhood search. Finally, we additionally propose a novel construction heuristic that can produce cost efficient initial solutions for local search, and hence can improve the solution process for large scale real life instances.

2 Intermediate Experimental Results

We implemented a first prototype version of the proposed novel construction heuristic and large neighborhood search to produce a first set of intermediate results. All experiments have been conducted on an Intel Xeon E5345 2.33 GHz CPU with 48 GB RAM under a time limit of one hour. Table 1 summarizes the results on 24 publicly available problem instances for paint shop scheduling[1].

The left side of the table shows experimental results on the smaller instances (I1-12), while the right side shows results on the larger instances (I13-24). Columns labeled LS display results achieved with the local search approach from [6], while columns labeled CP show results produced with the

[1] https://www.dbai.tuwien.ac.at/staff/winter/

	LS	LNS	CP		LS	LS+CH	LNS+CH	CP
I1	851	799	775*	**I13**	116235	20060	26168	-
I2	886	853	842*	**I14**	118628	27766	35088	-
I3	995	980	961*	**I15**	180863	54077	51533	-
I4	1256	1061	918*	**I16**	262252	96744	78859	-
I5	598	607	530*	**I17**	421777	-	-	-
I6	892	872	842*	**I18**	581021	-	-	-
I7	1116	1113	1040	**I19**	555829	543264	586260	-
I8	2599	1775	1237*	**I20**	930564	629729	606886	-
I9	2198	1746	992	**I21**	917955	-	-	-
I10	1223	1243	966	**I22**	1128716	-	-	-
I11	5652	5960	-	**I23**	1889804	-	-	-
I12	5973	7348	-	**I24**	2086450	-	-	-

Table 1 Table showing the intermediate experimental results we performed using 24 real life problem instances.

state-of-the-art exact technique from [7]. Column LNS shows results achieved with the prototype for the proposed large neighborhood search, whereas columns LS+CH and LNS+CH show results achieved with the new construction heuristic together with local search and large neighborhood search respectively. Results marked with a * denote proven optimal solutions, where a - indicates that no feasible solution could be reached within the time limit.

We can see in the intermediate results that results produced with large neighborhood search lead to improved results compared to the existing local search method for the majority of the small instances. For the larger instances I13-24 the results show that the proposed construction heuristic leads to a significant improvement in solution quality for six of the instances, but is not able to produce feasible solutions for all instances within the time limit. We further observe that large neighborhood search does not always lead to improved results for the larger instances, but achieves improved results for instances 15 and 16.

The reason why the prototype of the construction heuristic can currently not produce feasible solutions for all instances is that the computation of the initial schedule is not fast enough yet for some of the instances that need to schedule a very large number of jobs. This is because the current implementation finishes execution just after the best position for each single job in the initial schedule has been calculated. We expect that the approach will be able to produce solutions for all instances in our next implementation, when we introduce limits on the time budget for the generation of initial solutions.

3 Future Work

The intermediate results for instances I13-16 and I19-I20 look promising as the first prototype implementation of the proposed techniques already was able to provide six novel improved upper bounds in our experiments. In the next steps, we plan to investigate additional variants of the destroy operator for

LNS. Afterwards, we want to utilize automated parameter configuration tools to further tune the performance of LNS, before we later conduct a systematic experimental study of the proposed techniques, where we expect to reach novel improved upper bounds for several real-life paint shop scheduling instances.

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. Epping, T., Hochstättler, W., Oertel, P.: Complexity results on a paint shop problem. Discrete Applied Mathematics **136**(2), 217 – 226 (2004)
2. Prandtstetter, M., Raidl, G.R.: An integer linear programming approach and a hybrid variable neighborhood search for the car sequencing problem. European Journal of Operational Research **191**(3), 1004 – 1022 (2008)
3. Solnon, C., Cung, V.D., Nguyen, A., Artigues, C.: The car sequencing problem: Overview of state-of-the-art methods and industrial case-study of the roadef'2005 challenge problem. European Journal of Operational Research **191**(3), 912 – 927 (2008)
4. Spieckermann, S., Gutenschwager, K., Voß, S.: A sequential ordering problem in automotive paint shops. International Journal of Production Research **42**(9), 1865–1878 (2004)
5. Winter, F., Musliu, N.: Constraint based modeling for scheduling paint shops in the automotive supply industry. Tech. rep., TU Wien, CD-TR, 2019/1 (2019)
6. Winter, F., Musliu, N., Demirovic, E., Mrkvicka, C.: Solution approaches for an automotive paint shop scheduling problem. In: ICAPS, pp. 573–581. AAAI Press (2019)
7. Winter, F., Musliu, N., Stuckey, P.J.: Explaining propagators for string edit distance constraints. In: AAAI. AAAI Press (2020)

A proposition of a flexible framework for generating nurse rostering instances

Nguyen Dang · Christopher Stone · Ian Miguel

Abstract We propose ideas towards a framework for generating benchmark nurse rostering instances in an automated fashion. The framework makes use of a newly developed automated instance generation approach based on high-level constraint modelling and automated algorithm configuration. It allows a modeller to change the kind of instances produced simply by imposing constraints on the properties expected to be satisfied by the instances generated.

Keywords nurse rostering, constraint modelling · instance generation

1 Introduction

Nurse rostering problem [9,5,20] is one of the most extensively studied problems in operation research [18]. The problem consists of assigning nurses with a certain skill to a working shift of a day. The assignments spread over a planning horizon, normally from a week to a month. The number of nurses required for each triple of days, shifts and skills are given. There are often several hard constraints that need to be satisfied, such as avoiding under-staffing and impractical shift patterns (e.g., a nurse cannot work a Night shift followed by an Early shift). Besides, constraints that can be violated, but should be satisfied as much as possible are expressed as soft constraints. Some examples of soft constraints include the minimum and maximum number of consecutive working days that a nurse can be assigned to. There are several variants of nurse rostering problems, depending on how the hard and soft constraints are stated. We refer to [5,9,20] for an overview of the problem and its variants.

Several algorithmic approaches have been proposed for solving nurse rostering problems during the last three decades. They include mathematical programming [19], constraint programming [17], metaheuristics [8], hyper-heuristics [3] and hybrid approaches [6]. Each approach often has their own

School of Computer Science, University of St Andrews, UK

strength and weakness. For example, constraint programming approaches offer the advantage of being flexible, as new constraints can be easily supported without interfering into solvers' implementation. However, they may suffer from the issue of scalability, and in some cases cannot solve large-size instances effectively as problem-specific solving approaches [23]. Metaheuristic approaches often find good-quality solutions in a reasonable amount of time and have good scalability, but they lack of the ability to prove optimality of solutions found. Even approaches within the same group can have different performance depending on the problem variants and instance distributions [21].

Having benchmark instances with different levels of difficulties is crucial for assessing performance of different solving approaches, and for understanding their strength and weakness [22]. A variety of benchmark instances have been proposed and made publicly available for nurse rostering problems. The KAHO dataset[1] [20] is constructed using real-world data from two Belgian hospitals where for each ward three different scenarios are considered: normal, high amount of unforeseen work and unexpected absence of nurses. The Nottingham dataset[2] [4] consist of artificial and real instances collected worldwide. NSPLib[3] [24] provides an instance generator and large sets of nurse rostering instances. The First [12] and Second International Nurse Rostering Competitions [7] propose artificial benchmark instance sets derived from real-world data. Recently a unified data format, namely `XESTT` [4], for representing several nurse rostering benchmark datasets has been proposed [15].

The necessity of having rich and diverse benchmark instance sets for nurse rostering makes the idea of an automated instance generator plausible. Another motivation comes from the successful applications of automated algorithm configuration in recent years [13]. Imagine that we have a highly configurable nurse rostering solver. When a practitioner wants to adapt a solver to a specific nurse rostering problem in a certain context of a hospital, the first step is to have instances that reflect the pattern of the local context. An automated algorithm configuration tool, such as SMAC [14] or irace [16], can then be used to find the best parameter setting of the solver on the given instance distribution.

In [24], Vanhoucke and Maenhout proposed an instance generator that characterizes an instance through various complexity indicators. They included problem sizes, preference distribution measures, coverage distribution measures, and time related constraints. They implemented a dedicated procedure for generating instances with properties corresponding to specific indicators' values as parameters.

The generator in [24] covers a wide range of nurse rostering instances' aspects. However, due to the complexity and diversity of nurse rostering problems in practice, there can be additional characteristics on the instances that

[1] `https://people.cs.kuleuven.be/~pieter.smet/nurserostering.html`

[2] `http://www.schedulingbenchmarks.org/nurse.html`

[3] `https://www.projectmanagement.ugent.be/research/personnel_scheduling/nsp`

[4] `http://jeffreykingston.id.au/xestt/`

a practitioner might wish to add to reflect the local context. For example, in a normal department of a hospital, the demand for highly skilled nurses can be very high during the middle of the week, while being low during the beginning and the end of the week. On the other hand, in an intensive care department, the demand for highly skilled nurses are generally much higher on most days compared to the demand for less skilled nurses.

In this work, we propose an alternative approach for automated nurse rostering instance generation based on constraint modelling and automated algorithm configuration. The high-level constraint modelling language used by our approach, namely Essence [10], allows for the flexible specification of additional characteristics on instances generated. Moreover, the combination of constraint programming and automated configuration ensures feasibility and diversity of instances generated.

2 Methodology

The flexible framework for generating nurse rostering instances proposed in this paper is an extension of the automated instance generation system for constraint programming proposed in [1,2]. The system makes use of Essence, a modelling language designed for the specification of problems in combinatorial decision and optimisation [10]. This system takes an abstract specification of a problem in Essence and automatically converts it into a parametrised instance generator. Desirable properties of the instances being generated can be incorporated into the generator as constraints. The system comes paired with a constraint programming solver (`minion` [11]) that generates an instance by solving the generator specification, and an automated algorithm configurator (irace [16]) that allows efficient search for generator's parameter settings that cover *feasible instances* of desired properties.

Essence is a high-level constraint modelling language that supports several abstract data types. The language can capture the structure of a problem above the level of abstraction at which modelling decisions are made. In our context, this enables the modeller to easily add or remove specific constraints on properties the instances must satisfy, which consequently change the kind of instances produced by the (automatically created) instance generator.

The nurse rostering instance generation framework will include two components. The first one is a nurse rostering specification in Essence that incorporates all the indicators proposed in [24] as constraints. We will also add a number of additional properties observed on real-world instances. One example is the distribution of demand for highly-skilled nurses during a week. The properties encoded in the specification can be easily deactivated if not needed. As a first step, we will model the specification proposed in the Second International Nurse Rostering Competition [7]. The specification can also be modified to comply with other nurse rostering variants. The second component is the automated instance generation system, which receives as input the spec-

ification from the first component, and automatically generate a diverse set of feasible instances following the distribution indicated in the specification.

This is a work in progress and we plan to make the framework and the instances generated publicly available once they are ready. We believe that this approach will increase the diversity of scenarios covered by nurse rostering's benchmarks and help in the development of new algorithms applicable in real world healthcare.

Acknowledgements

We thank Guilherme Redeker and Juliana Bowles for their help on identifying patterns in real-world nurse rostering instances. This work was supported by the EPSRC grant EP/P015638/1 and used the Cirrus UK National Tier-2 HPC Service at EPCC (`http://www.cirrus.ac.uk`) funded by the University of Edinburgh and EPSRC (EP/P020267/1). Nguyen Dang is a Leverhulme Early Career Fellow.

References

1. Akgün, Ö., Dang, N., Miguel, I., Salamon, A.Z., Spracklen, P., Stone, C.: Discriminating instance generation from abstract specifications: A case study with cp and mip. In: International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research, pp. 41–51. Springer (2020)
2. Akgün, Ö., Dang, N., Miguel, I., Salamon, A.Z., Stone, C.: Instance generation via generator instances. In: International Conference on Principles and Practice of Constraint Programming, pp. 3–19. Springer (2019)
3. Bilgin, B., Demeester, P., Misir, M., Vancroonenburg, W., Vanden Berghe, G., Wauters, T.: A hyper-heuristic combined with a greedy shuffle approach to the nurse rostering competition. In: the 8th International Conference on the Practice and Theory of Automated Timetabling (2010)
4. Burke, E.K., Curtois, T., Post, G., Qu, R., Veltman, B.: A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem. European journal of operational research **188**(2), 330–341 (2008)
5. Burke, E.K., De Causmaecker, P., Berghe, G.V., Van Landeghem, H.: The state of the art of nurse rostering. Journal of scheduling **7**(6), 441–499 (2004)
6. Burke, E.K., Li, J., Qu, R.: A hybrid model of integer programming and variable neighbourhood search for highly-constrained nurse rostering problems. European Journal of Operational Research **203**(2), 484–493 (2010)
7. Ceschia, S., Dang, N., De Causmaecker, P., Haspeslagh, S., Schaerf, A.: The second international nurse rostering competition. Annals of Operations Research **274**(1-2), 171–186 (2019)
8. De Causmaecker, P., Berghe, G.V.: Novel meta-heuristic approaches to nurse rostering problems in belgian hospitals (2004)
9. De Causmaecker, P., Berghe, G.V.: A categorisation of nurse rostering problems. Journal of Scheduling **14**(1), 3–16 (2011)
10. Frisch, A.M., Harvey, W., Jefferson, C., Martínez-Hernández, B., Miguel, I.: E ssence: A constraint language for specifying combinatorial problems. Constraints **13**(3), 268–306 (2008)
11. Gent, I.P., Jefferson, C., Miguel, I.: Minion: A fast scalable constraint solver. In: ECAI, vol. 141, pp. 98–102 (2006)

12. Haspeslagh, S., De Causmaecker, P., Stølevik, M., Schaerf, A.: First international nurse rostering competition 2010 (august 10-13, 2010, belfast, uk). In: PATAT 2010-Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling, Belfast, Northern-Ireland, UK (2010)
13. Hoos, H.H.: Automated algorithm configuration and parameter tuning. In: Autonomous search, pp. 37–71. Springer (2011)
14. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration (extended version)
15. Kingston, J.H., Post, G., Vanden Berghe, G.: A unifed nurse rostering model based on xhstt. In: Proceedings of the 12th International Conference of the Practice and Theory of Automated Timetabling, vol. 12, pp. 81–96. EWG PATAT; Vienna (2018)
16. López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L.P., Birattari, M., Stützle, T.: The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives **3**, 43–58 (2016). DOI 10.1016/j.orp.2016.09.002. URL `http://iridia.ulb.ac.be/irace/`
17. Nonobe, K.: Inrc2010: An approach using a general constraint optimization solver. The First International Nurse Rostering Competition (INRC 2010) (2010)
18. Rais, A., Viana, A.: Operations research in healthcare: a survey. International transactions in operational research **18**(1), 1–31 (2011)
19. Santos, H.G., Toffolo, T.A., Gomes, R.A., Ribas, S.: Integer programming techniques for the nurse rostering problem. Annals of Operations Research **239**(1), 225–251 (2016)
20. Smet, P., De Causmaecker, P., Bilgin, B., Berghe, G.V.: Nurse rostering: a complex example of personnel scheduling with perspectives. In: Automated Scheduling and Planning, pp. 129–153. Springer (2013)
21. Smith-Miles, K., Baatar, D., Wreford, B., Lewis, R.: Towards objective measures of algorithm performance across instance space. Computers & Operations Research **45**, 12–24 (2014)
22. Smith-Miles, K., Lopes, L.: Measuring instance difficulty for combinatorial optimization problems. Computers & Operations Research **39**(5), 875–889 (2012)
23. Teuschl, S.: Casual employee scheduling with constraint programming and metaheuristics. Ph.D. thesis, Wien (2020)
24. Vanhoucke, M., Maenhout, B.: On the characterization and generation of nurse scheduling problem instances. European Journal of Operational Research **196**(2), 457–467 (2009)

Deep Neural Networked Assisted Tree Search for the Personnel Rostering Problem

Ziyi Chen · Patrick De Causmaecker · Yajie Dou

Abstract The personnel rostering problem is the problem of finding an optimal way to assign employees to shifts, subject to a set of hard constraints which all valid solutions must follow, and a set of soft constraints which define the relative quality of valid solutions. The problem has received significant attention in the literature and is addressed by a large number of exact and metaheuristic methods. In order to make the complex and costly design of heuristics for the personnel rostering problem automatic, we propose a new method combining Deep Neural Network and Tree Search. By treating schedules as matrices, the neural network can predict the distance between the current solution and the optimal solution. It can select solution strategies by analyzing existing (near-)optimal solutions to personnel rostering problem instances. Combined with branch and bound, the network can give every node a probability which indicates the distance between it and the optimal one, so that a well-informed choice can be made on which branch to choose next and to prune the search tree.

Keywords Combinatorial Optimization · Deep Learning · Timetabling · Personnel Rostering

This work was supported by the National Natural Science Foundation of China [71901214; 71690233]; the AI in Flanders, KU Leuven Research Fund [RKU-D2932-C24/17/012]; and the Data-driven logistics [FWO-S007318N].

Ziyi Chen
College of Systems Engineering, National University of Defense Technology, Changsha 410073, China
Tel.: +32-485924591
E-mail: chenziyi_nudt@outlook.com
Present address:Department of Computer Science, KU Leuven Kulak, E. Sabbelaan 53, 8500 Kortrijk, Belgium

Patrick De Causmaecker
Department of Computer Science, KU Leuven Kulak, E. Sabbelaan 53, 8500 Kortrijk, Belgium

Yajie Dou
College of Systems Engineering, National University of Defense Technology, Changsha 410073, China

1 Introduction

In various occupations and work scenarios, arranging employees to different shifts is a difficult job. The difficulty is that different employees have different requirements for life and work, which leads to preference of each employee. And there are also requirements of the law that must be followed or diverse properties of different occupations. These regulations are what we call soft constraints and hard constraints. Inflexible or unreasonable work schedules may affect the personnel lives of employees, affect their emotion, make them dissatisfied with their work, and may lead to a high turnover rate, which may have an adverse effect on the employer's operation and the experience impact of customers. If it can ensure that employees are arranged for the right job at the right time. From the perspective of employees, such schedule can improve employee job satisfaction, reduce employee dissatisfaction, fatigue and pressure to improve work efficiency and service quality(Burke, De Causmaecker, Vanden Berghe & Landeghem, 2004). From the perspective of employers and companies, an excellent schedule can increase the retention rate of employees and maintain a reasonable financial budget for employers.(Kazahaya, 2005; M'Hallah & Alkhabbaz, 2013)

Committed to solving such real-world problems, the personnel rostering problems have received great attention in the past decade. The personnel rostering problems aim to generate a scheduling table based on the determined number of employees and the time period. The schedule consists of a series of different types of shifts (for example, morning, evening, and day-off) during the entire time period. The basis for shifts is based on conditions such as the preference of employees for working hours and the requirements of laws or professional regulations, which we call hard constraints and soft constraints. Hard constraints are conditions that must be met for shift scheduling, while soft constraints can be violated to a certain extent, but must pay a price for this. The quality of the schedule can be evaluated by the penalty value for violating soft constraints. Due to the complex and highly-constrained structure, personnel rostering problems are often computationally challenging, and most variants of these problems are classified as NP-hard.

There has been a lot of research on personnel rostering problems, which can be divided into two categories: exact methods and metaheuristic methods (Smet, Brucker, De Causmaecker & Vanden Berghe, 2016). The exact methods mainly include Integer Programming (IP)(Glass & Knight, 2010; Maenhout & Vanhoucke, 2009; M'Hallah & Alkhabbaz, 2013) and Constraint Programming (CP) (Girbea, Suciu & Sisak, 2011; Soto, Crawford, Monfroy, Palma & Paredes, 2013), the exact method can find the optimal solution, but the time cost it pays is also very expensive, and it is usually unacceptable. To solve this problem, the researchers have proposed metaheuristic methods, including Variable Neighborhood Search(Lü & Hao, 2012; Rahimian, Akartunalı & Levine, 2017), Genetic Algorithms(Ayob, Hadwan, Nazr & Ahmad, 2013; Burke, Cowling, De Causmaecker & Vanden Berghe, 2001) and stochastic algorithms (Tassopoulos et al., 2015) and tailored heuristic algorithms (Brucker, Burke, Curtois, Qu & Vanden Berghe, 2010), these methods can generate high-quality feasible solution in a short time, but the shortened time comes at the cost of giving up accuracy.

And even so, these complex methods are not well applied. According to the literature, many organizations are still manually producing schedules. The research

has contributed to producing solutions automatically.(Burke, Kendall & Soubeiga, 2003; De Causmaecker & Vanden Berghe, 2010)

We propose a combined Deep Neural Network and Tree Search(DNNTS)based methdodology. DNNTS is the first method that used Deep Neural Network(DNN) model to guide tree search to solve the personnel rostering problem. It is based on a learned model for branch selection during the tree search. The implementation can be used to solve a number of problems from the literature(Schedulingbenchmarks.org). These problems have the following characteristics and observations:

1. The (intermediate) solutions of a problem can be represented as m×n matrices that can be transferred as input for the neural network.
2. The best solution can be found by modifying the initial solution in a number of simple steps which defining the possible child nodes.
3. The problems have soft constraints which can be expressed in the penalty function and set the lower bound for the tree search.

The main contributions of this work can be summarized as follows:

1. An automatic method DNNTS is applied to personnel rostering problem.
2. For some specific problems, this method is shown to find a good solution equal to the best known lower bound.
3. The Deep Neural Network is used to make branch selection in the process of tree search, which speeds up the search process.
4. Experimental evaluation of different search strategies.

This paper is organized as follows. In Chapter 2, we discuss work on personnel rostering and on combining machine learning and optimization techniques. In Chapter 3, we describe the problem and formal problem definition. In Chapter 4, we introduce the tree search method, the DNN model and the process of combining these to solve the personnel rostering problem. In Chapter 5 we test our method, show the results and make relevant comparisons. In Chapter 6 we summarize and introduce the future work.

2 Literature review

In this chapter, we first review existing methods for personnel rostering. Similar methods combining deep learning and optimization are discussed. There are many optimization methods that integrate deep learning methods in other fields. We summarize the methods that inspired us, and discuss the connection between these and our methods.

2.1 Rostering problem

As mentioned before, the methods to solve personnel rostering problems are mainly divided into two categories, exact methods and heuristic methods. Exact method are concerned with finding proven optimal solutions. For the direction of IP, Maenhout & Vanhoucke (2009) present an exact branch-and-price algorithm for solving the nurse scheduling problem incorporating multiple objectives and discuss different branching and pruning strategies. Some authors(Girbea et al., 2011; Soto

et al., 2013) concentrate on CP, and introduce a model including soft constraints. Mixed integer programming (MIP) is also a method that has attracted much attention. M'Hallah & Alkhabbaz (2013) describe the nurses' timetabling problem of a Kuwaiti health care unit and model it as a MIP. Glass & Knight (2010) start from benchmark problems and extend their MIP approach to nurse rostering to take better account of the practical considerations. Column generation has been effective to determine preference scheduling (Bard & Purnomo, 2005). applied Mixed Integer Quadratic Programming.

Heuristic methods are mainly concerned with finding a solution quickly, and even if the number of employees waiting to be scheduled is large and the constraints are complicated, an acceptable and feasible solution can be obtained in an appropriate time. Genetic and Memetic algorithms form an important class of metaheuristics that have been extensively applied in personnel rostering problem.(Aickelin & Dowsland, 2000, 2004; Easton & Mansour, 1999; Kawanaka, Yamamoto, Yoshikawa, Shinogi & Tsuruoka, 2001)(Burke et al., 2001) In addition, there are a lot of attempts on other types of methods, such as Tabu Search Algorithms, some researchers (Burke, De Causmaecker & Vanden Berghe, 1999) proposed a hybrid Tabu Search Algorithm to solve the personnel rostering problem in Belgian hospitals. Also simulated annealing algorithms. As a representative, an iteratively local searching method based on simulated annealing(Cheng, Ozaku, Kuwahara, Kogure & Ota, 2008), and a shift mode method using simulated annealing were proposed.(Hadwan & Ayob, 2010) Aickelin & Dowsland (2004) proposed a mode conversion technique arising at a major UK hospital. And Todorović, Petrović & Teodorović (2013) proposed the Bee Colony Optimization method to solve this problem.

Burke et al. (2004) analyze the state of the art of research on nurse rostering problems and categorized papers according to solution methods, constraints, performance measures, and information on the planning period, the data that is used, the number of skills, and their substitutability, etc. De Causmaecker & Vanden Berghe (2010) build on the work of the last decades to produce a classification system for nurse rostering problems. Vanden Berghe, Beliën, Bruecker, Demeulemeester & Boeck (2013) present a review of literature on personnel scheduling and evaluate the literature from different perspectives of personnel characteristics, decision delineation and shifts definitions, constraints, performance measures, flexibility, application area and applicability of research.

2.2 Deep learning and Optimization

For the combination of deep learning and optimization, some researchers have made some attempts. For example, Hottung, Tanaka & Tierney (2020) use two DNN models to guide the tree search to solve Containers Pre-Marshalling Problem. As is mentioned in the paper written by De Causmaecker (2017), data science meets optimization when using data science in algorithm construction and applying deep learning while engineering an algorithm.

Combining operations research and artificial intelligence allows using powerful solvers such as IBM CP Optimiser(Optimizer, 2015) and Gurobi (Gurobi Optimization, 2015), to be used in hybrid environments. e.g. a method combing CP and heuristics, which is an iterated local search framework that uses CP for initial

solution construction and diversification, and a variable neighborhood descent for iterative improvement. (Stølevik, Nordlander, Riise & Frøyseth, 2011). Another method divides the original problem into sub-problems, solves sub-problems by IP, and combines IP and local search to get results (Valouxis, Gogos, Goulas, Alefragis & Housos, 2012). And a less studied combination of IP and CP (Rahimian, Akartunali & Levine, 2015) allow to take advantage of the complementarity in the different methodologies. Rahimian et al. (2015) proposed a new hybrid algorithm of IP and CP to solve the personnel rostering problem. It uses IP to find the best solution, and CP to find feasible solutions effectively. This hybrid algorithms uses information from specific problems to reduce the search space, fine-tunes search parameters and improves the efficiency of the entire search process in a novel way.

Lodi & Zarpellon (2017) and Dilkina, Khalil & Nemhauser (2017) outline methods for applying learning to variable and node selection problems in MIP. Khalil, Dilkina, Nemhauser, Ahmed & Shao (2017) use logistic regression to predict when to apply the original heuristic when solving MIPs. The author uses similar features as mentioned by Khalil, Le Bodic, Song, Nemhauser & Dilkina (2016) and can improve the performance of the MIP solver. There are other papers that use machine learning techniques to solve MIP (Kruber, Lübbecke & Parmentier, 2017; Bonfietti, Lombardi & Milano, 2015).

Vinyals, Fortunato & Jaitly (2015) propose a method called a pointer network and train it to generate solutions to traveling salesman problems through supervised learning. Bello, Pham, Le, Norouzi & Bengio (2016) use reinforcement learning to train a pointer network for the traveling salesman problem. Kool & Welling (2018) propose a similar method, which can also be used to solve other routing problems, such as vehicle routing problems. All these methods focus on the training and architecture of the DNNs network, rather than merging the DNNs network into a complex search process.

3 Problem description

There are many personnel rostering problems in different work contexts, such as nurse scheduling, hotel reception scheduling or other situations. In the personnel rostering problems, the people waiting to be assigned are called employees, the different time in everyday waiting to be occupied are called shifts. Employees are assigned to shifts in a certain period of time according to certain constraints. Constraints define the limitations of assignments for each employee, there are hard constraints which the solution must obey and soft constraints which will give some penalty when the solution does not meet the requirements. These constraints can be used to model restrictions such as ‘employees should work more than 3 days and less than 6 days a week’ or ‘employee A does not want to work on Wednesday’(De Causmaecker & Vanden Berghe, 2010; Paul & Knust, 2015). The obtained feasible assignments that meet all hard constraints are called solutions. Each solution has a value of penalty, which is determined by the level of compliance with soft constraints. The personnel rostering problem aims to find an allocation scheme with the lowest penalty value that meets all hard constraints. It is an NP-hard problem(Osogami & Imai, 2000).

	Day 1	Day 2	···	Day t
Employee 1	shift 1	rest	···	shift s
Employee 2	shift 2	shift 1	···	rest
···	···	···	···	···
Employee e	shift s	shift s	···	shift 1

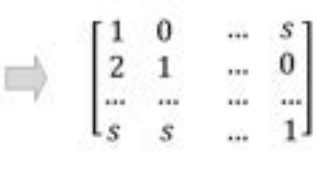


Fig. 1 Solution and corresponding mathematical matrix expression

3.1 Formal problem definition

Personnel rostering problems are characterized by a set of employees, a set of a scheduling period of days, and a set of shifts. A shift is a fixed time interval which denotes a working period. Each shift is characterized by a unique type which classifies the shifts in various ways, e.g., by time interval (morning, late), by required qualifications (senior, junior), or by a combination of these (morning-senior, late-junior). A shift is considered to occur on the day where its time interval starts. The number of employees required for each shift can vary from day to day, and is typically more than one employee. An assignment is the allocation of an employee to a shift on a day. In this paper, the solution is regarded as an e×t matrix which contains in each cell either an assignment or a day-off. Days T is a period during which the assignment begins and ends. Fig.1 shows how to transfer a solution to a matrix.

Constraints can be expressed as an exact, ranged, minimum or maximum requirement. In the case of exact demand, the specified value is exactly the number of employees to be assigned. A ranged definition requires that the number of assigned employees should be within a specified time interval. When such an interval has no upper(lower) limit, the requirement is defined as a minimum(maximum).(Smet et al., 2016)

The parameters and decision variables (Curtois & Qu, 2014) are described as following:

Parameters:

E set of employees.
h number of days in the planning horizon.
T set of days in the planning horizon= $\{1...h\}$.
W set of weekends in the planning horizon= $\{1...h/7\}$.
S set of shift types.
R_t set of shift types that cannot be assigned immediately after shift type t.
N_i set of days that employee i cannot be assigned a shift on.
l_t length of shift type t in minutes.
m_{es}^{max} maximum number of shifts of type s that can be assigned to employee e.
b_e^{min} minimum number of minutes that employee e must be assigned.
b_e^{max} maximum number of minutes that employee e can be assigned.
c_e^{min} minimum number of consecutive shifts that employee e must work.
c_e^{max} maximum number of consecutive shifts that employee e can work.
o_e^{min} minimum number of consecutive days off that employee e can be assigned.
a_e^{max} maximum number of weekends that employee e can work.
q_{ets} penalty if shift type s is not assigned to employee e on day t.

p_{ets} penalty if shift type s is assigned to employee e on day t.
u_{dt} preferred total number of employees assigned shift type s on day t.
v_{ts}^{min} weight if below the preferred cover for shift type s on day t.
v_{ts}^{max} weight if exceeding the preferred cover for shift type s on day t.

Decision variables:

x_{ets} 1 if employee e is assigned shift type s on day t, 0 otherwise
k_{ew} 1 if employee e works on weekend w, 0 otherwise.
y_{ts} total below the preferred cover for shift type s on day t.
z_{ts} total above the preferred cover for shift type s on day t.

Then the problem can be formulated below:

$$\min \sum_{e\in E}\sum_{t\in T}\sum_{s\in S} q_{ets}(1-x_{ets}) + \sum_{e\in E}\sum_{t\in T}\sum_{s\in S} p_{ets}x_{ets} + \sum_{t\in T}\sum_{s\in S} y_{ts}v_{ts}^{min} + \sum_{t\in T}\sum_{s\in S} z_{ts}v_{ts}^{max} \tag{1}$$

$$\sum_{s\in S} x_{ets} \leq 1, \quad \forall e \in E, t \in T \tag{2}$$

$$x_{ets} + x_{e(t+1)s'} \leq 1, \quad \forall e \in E, t \in \{1, ..., h-1\}, s, s' \in S \tag{3}$$

$$\sum_{t\in T} x_{ets} \leq m_{es}^{max}, \quad \forall e \in E, s \in S \tag{4}$$

$$b_e^{min} \leq \sum_{t\in T}\sum_{s\in S} l_s x_{ets} \leq b_e^{max}, \quad \forall e \in E \tag{5}$$

$$\sum_{j=d}^{d+c_e^{max}} \sum_{s\in S} x_{ejs} \leq c_e^{max}, \quad \forall e \in E, d \in \{1...h - c_e^{max}\} \tag{6}$$

$$\sum_{s\in S} x_{ets} + (k - \sum_{j=t+1}^{t+k}\sum_{s\in S} x_{ejs} + \sum_{s\in S} x_{i(t+k+1)s} > 0, \quad \forall e \in E, k \in \left\{1...c_e^{min} - 1\right\}, t \in \{1...h-(k+1)\} \tag{7}$$

$$(1 - \sum_{s\in S} x_{ets}) + \sum_{j=t+1}^{t+k}\sum_{s\in S} x_{ejs} + (1 - \sum_{s\in S} x_{i(t+k+1)s}) > 0 \quad \forall e \in E, k \in \left\{1...o_e^{min} - 1\right\}, d \in \{1...h-(k+1)\} \tag{8}$$

$$\sum_{w \in W} k_{ew} \leq a_e^{max} \quad \forall e \in E \tag{9}$$

$$x_{ets} = 0, \quad \exists e \in E, t \in T, s \in S \tag{10}$$

Formula (1) is the objective function, which is to minimize the penalty value. Constraint(2) ensures that an employee cannot be assigned more than one shift on a single day. Constraint(3) describes some types of shifts cannot follow others. Constraint(4) is used to limit the maximum number of shifts of each type that can be assigned to employees. For example, some employees will have contracts which do not allow them to work night shifts or only a maximum number of night shifts. Constraint(5) ensures that the minimum and the maximum work time. The total minutes worked by each employee must be between a minimum and maximum. These limits can vary depending on whether the employee is full-time or part-time. Constraints(6) and (7) describe the consecutive constraints. Constraint(8) models the minimum consecutive days off in a similar way to the minimum consecutive shifts constraint. Constraint(9) sets the maximum number of weekends. A weekend is considered as being worked if the employee has a shift on the Saturday or the Sunday. Constraint(10) describes some days that employees cannot work, for example, they are on vacation.(Curtois & Qu, 2014)

4 Method

The DNNTS method integrates DNN into a heuristic tree search to decide which branch to choose next. The DNN is trained offline by supervised learning on existing (near-) optimal solutions for the defined personnel rostering problem and are then used to make branch decisions during the search.

The core idea of the method is to treat each feasible solution as an e×t matrix as Fig 1, and change the matrix through several predetermined change strategies, thereby gradually approaching the optimal solution. In the process of solving the problem, we use tree search to explore all the possibilities, and use DNN to decide the order to explore. As shown in Fig 2, whenever a new unexplored node A is found by the tree search, according to predetermined change strategies, it has three corresponding child nodes B, C and D, the DNN will predict which node has the highest probability to arrive at the best solution, thereby determining the next search node C.

In this section, we first describe how to form and train a DNN model to do prediction. We then explain in detail how personnel rostering problems can be solved using tree search. Finally, we show how to use DNN in tree search to do branching decisions.

4.1 Tree search

Methods based on tree search can be used to solve optimization problems. Beginning with the root node, the search tree is explored by systematically exploring

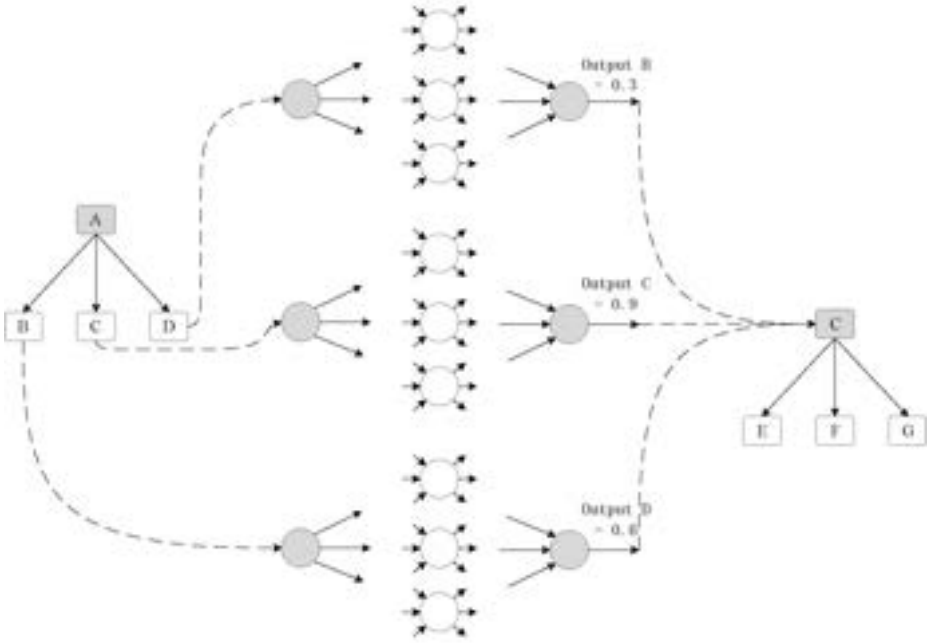


Fig. 2 Overview of the combination method

the child nodes of the root node and subsequent nodes. The solution of a given optimization problem can be understood as a leaf node in the tree.

In the process of tree search, each node in the tree represents a feasible arrangement for employees according to the hard constraints. The initial solution is represented by the root node. The child nodes of a node represent best solutions that can be reached by only one change from a set of predetermined change strategies.

4.1.1 Change strategies

There are many ways to simply modify the matrix representing a solution, such as exchanging two rows in the matrix randomly or changing some numbers in the matrix. To make the search process converge faster, we did related experiments in section 5.2 and identified three change strategies.

- Strategy 1: randomly change 2 employees' shifts in one day.
- Strategy 2: randomly change 2 day's shifts for one same employee.
- Strategy 3: randomly chose one shift where one of the employees doesn't need to work, and change his status to work on that shift.

4.2 DNN

The DNN is a method inspired by biological neural networks. A DNN consists of multiple layers of neurons. Each neuron receives one or more weighted inputs from

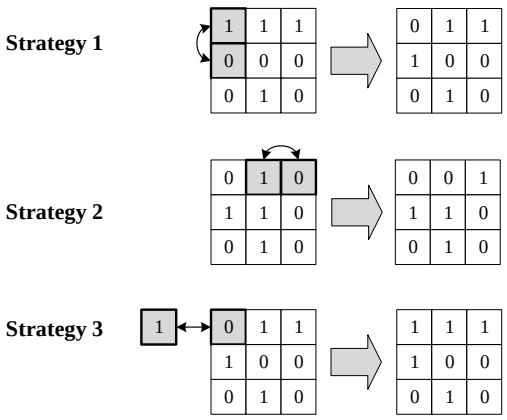


Fig. 3 Three change strategies

neurons of the previous layer, summarizes those inputs, and applies an activation function to the inputs. The value from the activation function is then sent out to the neurons of the next layer. The DNN "learns" by optimizing the weights on the arcs of the network.(Hottung et al., 2020)

DNN can be used for both classification (the space Y consists of a set of discrete values) as well as regression (Y can take any value in R), and we use regression DNN in this work to predict the probability that a solution might be the optimal one or might become the optimal one. To substantiate the word "might", we will use the distance to express the probability.

Definition 1 (Distance) The distance is the number of simple changes needed to be used in the current solution to become the best one.

Definition 2 (Probability) The probability is used to measure the possibility that the current solution might be the optimal solution. For $distance \in N, k > 0$.

$$probability = 1 - k \times distance \tag{11}$$

Simple change has been explained in section 4.1.1. Correspondingly, the shorter the distance between the current solution and the optimal solution, the higher is the probability, and the more likely the current solution might be or become the optimal solution. Fig.4 shows how to understand the distance and probability.

4.2.1 DNN forming

The DNN we use in this paper is a multi-layer DNN as Fig.5 shows. The first layer is the input layer, which has $e \times t$ nodes and is used to one-dimensional the input multidimensional matrix. The middle layers are hidden layers, the number of their layers and nodes is determined through parameter adjustment. Each node in hidden layers uses the activation function, defined as $ReLU(x) = \max\{0, x\}$. The output layer contains only one node. Its activation function is a sigmoid function, defined as $S(x) = 1/(1 + e^{-x})$, in order to output probability values between 0-1. As for the optimizer, we use the Adam optimizer (Kingma & Ba, 2014), which is based on a gradient descent.

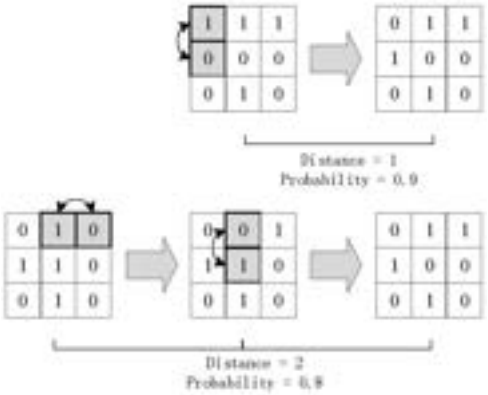


Fig. 4 The link between distance and probability

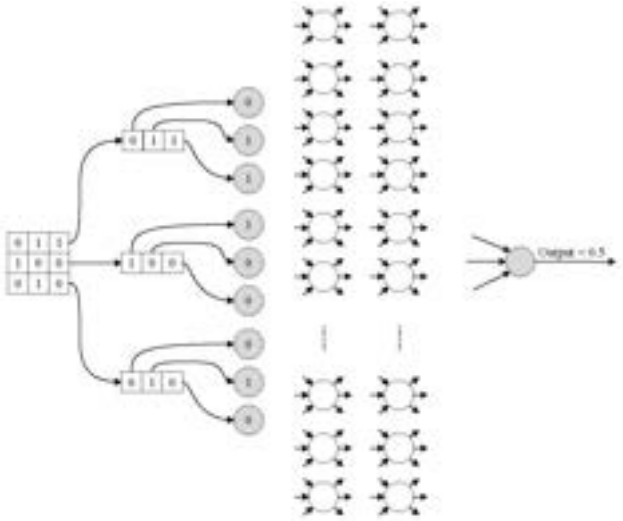


Fig. 5 The matrix type input of our DNN model

4.2.2 DNN training

The work of training the DNN model is described below. As is well known, the training set is a dataset of examples used for learning, that is to fit the parameters of the model, such as weight. A set of representative instances is divided into a training set and a validation set. Also, these two kinds of sets consist of a set of data(input) and labels(output). During the process of training, each instance in the training set is input into the DNN, and propagated by the network, in order to generate relative output. Then these values are compared to the labels in the training data using a loss function, which is used to compute the accuracy of the prediction. In the next step, according to the influence of network weights on the loss function, the DNN model needs to adjust the weights of the network in order to reduce the value of the loss function in the next iteration (gradient

descent). After dealing with all the instances in the training set, the first epoch of the training is completed. We can repeat the training for many epochs until the error doesn't improve at all.(Goodfellow, Bengio & Courville, 2016)

As a research of solving classical problems with new methods, we have many instances that can be used as the data(input) of the training set, but there is nothing to use as the label(output) for the training set. It's impractical to give every instance a label manually. We will use another method to obtain a relatively convincible training set for training the DNN model. The method will be explained in chapter 4.3, after the whole process is described in detail.

4.3 DNN assisted tree search

4.3.1 Search Strategies

There are many kinds of search strategies in tree search, such as depth-first search (DFS) of nodes that traverse the tree along the depth of the tree, breadth-first search (BFS) of nodes that traverse the tree along the width of the tree. In this paper, we refer to the idea of Depth-First-Search and improve it from different ways. New search strategies can explore the tree according to the probability value given by the DNN model and also take the penalty into consideration. We will discuss these strategies in detail below.

Depth-First-Search Depth-First-Search (DFS) is an algorithm for traversing or searching trees or graphs. This algorithm searches branches of the search tree as deep as possible. When the edge of node v has been searched, the search will go back to the starting node of the edge where node v was found. This process continues until all nodes have been found reachable from the source node. If there are still undiscovered nodes, select one of them as the source node and repeat the above process. The entire process is repeated until all nodes are visited. This algorithm does not adjust the execution strategy based on the information such as the structure of the graph.

For example, the tree in Fig.7. According to depth-first principle, A is the initial node which is explored first. Then start from the left node of all unexplored children of A, that means B is the next node after A. The same way, after exploring B, then E rather than C, until the bottom of the branch H is explored. Next returning to an unvisited node next to B, the depth-first traversal is repeated until all nodes in the tree have been visited. The search order in Fig.6 is $A \rightarrow B \rightarrow E \rightarrow H \rightarrow F \rightarrow C \rightarrow G \rightarrow D$.

Algorithm1 shows the depth first search strategy. The algorithm starts with the initial solution s_0, stored in a node n, that has several properties. These are whether the current node is visited, $visited(n)$, the current penalty of associated solution, $penalty(n)$, the children nodes of the current node, $child(n)$. The penalty of the best solution p and the best solution $best_n$ are set as the global value.

Probability-Fist-Strategy This strategy also follows the principle of searching the tree as deep as possible. But we add each node in the tree a value of probability by the DNN model, and we will use it to replace the left-first order (Probability-Fist-Strategy (PFS)). That means that when exploring the child nodes of v, the

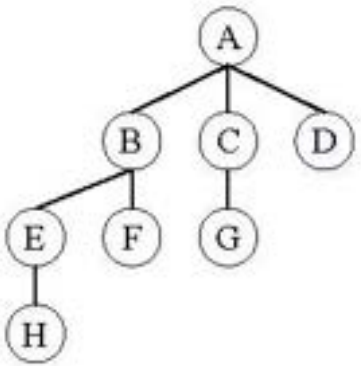


Fig. 6 DFS path

Algorithm 1 Depth first search strategy

Input: A node n of search tree.
Global: Penalty of the best solution p, best solution $best_n$
Output: Node representing the best solution.
function DNNTS-DFS(n)
if $isvisited(n)$ = true **then return** $best_n$
    $isvisited(n)$ = true
if $penalty(n) < p$ **then**
    $p \leftarrow penalty(n)$
    $best_n \leftarrow n$
for n' **in** $child(n)$
    DNNTS-DFS(n')
return $best_n$

child node with higher probability value will be explored first, rather than the left child node. But this principle only applies to the child nodes of the same layer and the same parent node. For nodes that are not in the same layer or nodes that are not a parent node, this principle does not apply.

For example, the tree in Fig.8. node A which is explored first, but the probability of its left child node is 0.7, less than the right one. So, the next step starts from D. The same way, after exploring D, then exploring B and its subtree. The resulting search order in Fig.7 is $A \rightarrow D \rightarrow B \rightarrow E \rightarrow H \rightarrow F \rightarrow C \rightarrow G$. Algorithm2 shows the probability first strategy. The setting is the same as the Algorithm1, but there is an additional DNN(n), which represent the output of the DNN model when the solution associated with node n is the input. i.e. The probability of solution associated with node n.

Probability-Penalty-Strategy In the personnel rostering problem, the penalty determines whether the solution is optimal. When the penalty is taken into consideration is addressed as the Probability-Penalty-Strategy (PPS). We need to normalize the probability and penalty values to the same unit of measurement, and give them different weights to get a value to replace the probability value in the previous strategy.

Algorithm3 shows the probability penalty strategy. The setting is the same as the Algorithm2, but there are two additional global values w_1 and w_2, which represent the weight for probability and weight for penalty respectively.

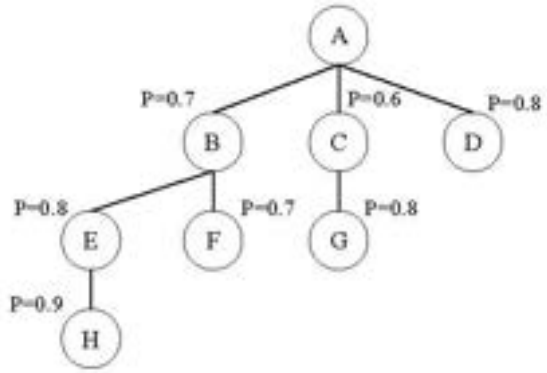


Fig. 7 PFS path

Algorithm 2 Probability first strategy

Input: A node n of search tree.
Global: Penalty of the best solution p, best solution $best_n$
Output: Node representing the best solution.
  function DNNTS-PFS(n)
  if $isvisited(n)$ = true **then return** $best_n$
    $isvisited(n)$ = true
  if $penalty(n) < p$ **then**
    $p \leftarrow penalty(n)$
    $best_n \leftarrow n$
  Sort $child(n)$ by DNN(n') for each $n' \in child(n)$
  for n' **in** $child(n)$
    DNNTS-PFS(n')
  return $best_n$

Algorithm 3 Probability Penalty strategy

Input: A node n of search tree.
Global: Penalty of the best solution p, best solution $best_n$,weight for probability w_1, weight for penalty w_2
Output: Node representing the best solution.
  function DNNTS-PPS(n)
  if $isvisited(n)$ = true **then return** $best_n$
    $isvisited(n)$ = true
  if $penalty(n) < p$ **then**
    $p \leftarrow penalty(n)$
    $best_n \leftarrow n$
  Sort $child(n)$ by $w_1 \times$ DNN(n')+$w_2 \times penalty(n')$ for each $n' \in child(n)$
  for n' **in** $child(n)$
    DNNTS-PPS(n')
  return $best_n$

4.3.2 DNN assisted tree search model

After forming and training the DNN model, it is used in the tree search as follows. When the node nk is to be explored, the associated $SolutionK$ is changed by 3 strategies as mentioned before, iterating through all the possibilities in each strategy and get 3 sets. The set obtained through strategy 1, 2, 3 are $\{SolutionK1_0$,

$SolutionK1_1, SolutionK1_2, \dots\}, \{SolutionK2_0, Solu- \ tionK2_1, \ SolutionK2_2, \dots\}, \{SolutionK3_0, Solution-$
$K3_1, SolutionK3_2, \dots\}$. Next all possibilities are given to the DNN model. These matrices of Solution are then propagated through the DNN model. We use a *sigmoid* activation function in the output layer to get the result between 0 and 1, this allows us to use the output as probability. Left the highest probability as the child node for each strategy, so among 3 strategies, we get 3 child nodes for each father node. The left probability is then used to decide which child node should be explored. The decision-making process is determined by the search strategies proposed in 4.3.1, for example, exploring the node with highest probability first by PFS. Fig.8 shows a whole process.

4.3.3 Initial dataset

We mentioned that the work applied to personnel rostering problem, although there are enough solutions (input), but no labels(output). So, we will use an automated method to generate a dataset for training the DNN model, including solution and labels.

First of all, for a specific personnel rostering problem that people already know the best known solution so far, we set the probability of the best known solution so far to 1, then randomly generate other solutions based on the best known solution. Actually, it means randomly generate $e \times t$ matrices according to the best known solution. And assign different labels according to the following conditions.

1. The most important thing is to check whether the randomly generated matrix meets the hard constraint. In case the hard constraint is not satisfied, the data will not be stored in the generated data set. Only matrices meeting the hard constraint can enter the next step to judge and assign different labels.
2. In order to give each matrix a label, we need to check the number of change strategies required(mentioned in chapter 4.3.1) to let the current matrix change to become the optimal solution for the current specific personnel rostering problem, then use a piecewise constant function to correspond the number of changes to the label. The less change strategies are needed, it means the more probable it is for the current matrix to change into the optimal solution. Equation12 is the piecewise constant function we use in this work.

$$f(x) = \begin{cases} 0.9 & 0 < x \leq 3 \\ 0.7 & 3 < x \leq 6 \\ 0.5 & 6 < x \leq 9 \\ 0.3 & 9 < x \leq 12 \\ 0.1 & x > 12 \end{cases} \tag{12}$$

After generating all the data, we use these data to initially train our DNN model. After training, we can then use the trained DNN model to solve the specific personnel rostering problem which is already known best known solution. In a complete solution process, we can get a complete path $\{s_0, s_1, s_2, \dots, s_{op}\}$ from how the initial solution s_0 changes into the final best known solution sop. According to the length of the entire path and the position of the s_k in the path, the

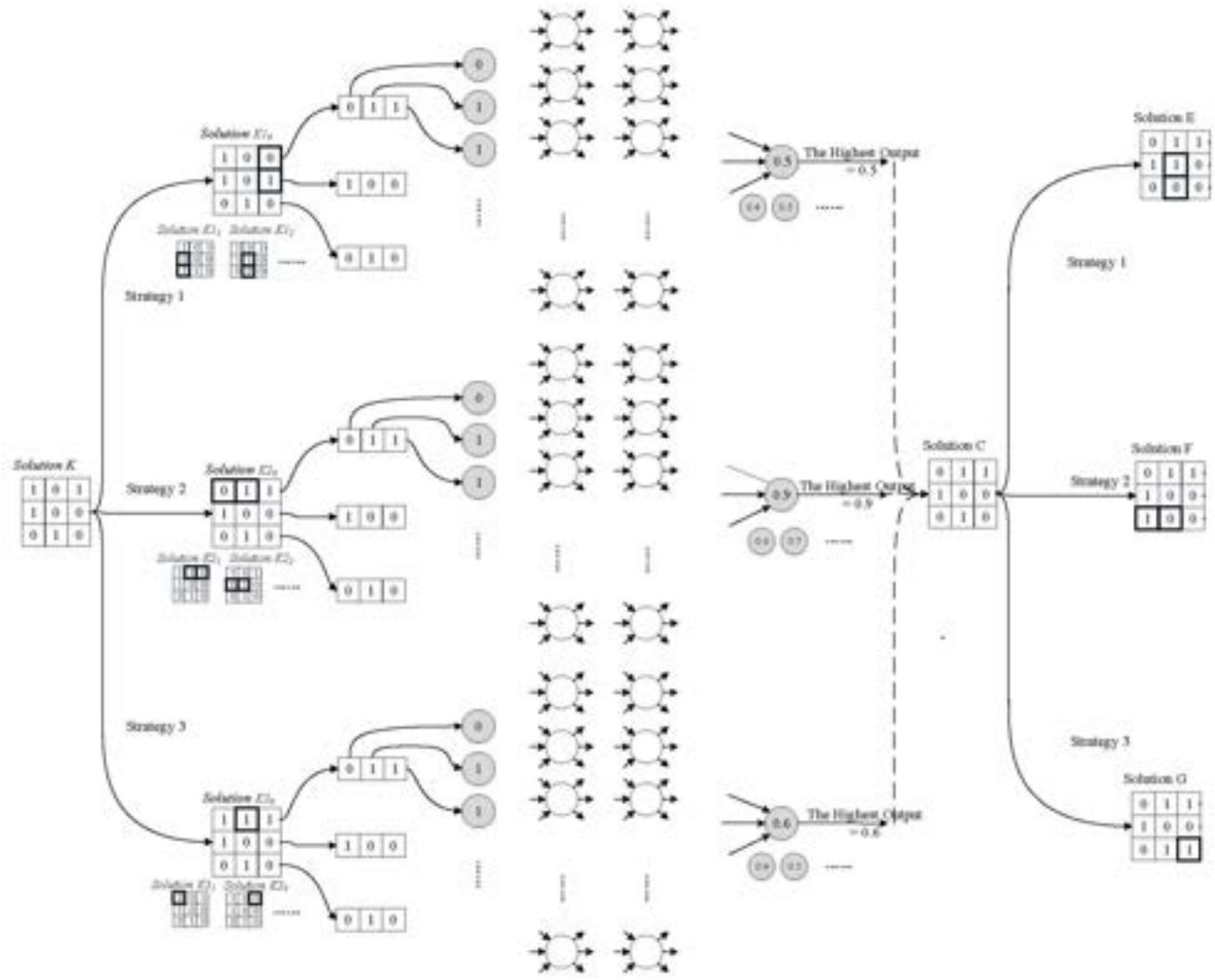


Fig. 8 The process of Neural Networked Assisted Tree Search for the Personnel Rostering Problem

distance from the best known solution can be found, from which the higher confidence of probability value can be given. At the same time, whenever enough data for training the DNN model is obtained, the model can be retrained. The specific problem model and initial solution s_0 are continuously changed to get enough data to finally complete the model training.

The use of these randomly generated data to train the DNN model at the beginning will not have a great impact on the efficiency of the final model, because these data are just to make the model have the ability to solve the problem, maybe the efficiency of the DNN model trained from the randomly generated data set is not very high, but at least it can ensure that the model can successfully solve a specific problem and give a complete path from the initial solution s_0 to the best

known solution s_{op}. It is our purpose to obtain a large number of such paths, so that we can use the data and labels in these paths to retrain our DNN model with more accurate data, and through continuous iteration, our model reaches a good level.

5 Result

We now use the instance to evaluate the performance of our DNNTS method in 3 aspects. In order to measure whether our method is competitive with current methods, we experiment on a variety of personnel rostering problem instances by different methods. To find out the most efficient change strategies, we use different combinations of change strategies for comparison. We also compare DFS, PFS and PPS by the same instance for several times to find the suitable search strategy.

5.1 Experimental setup

In order to get a good performing DNN, we need a large number of instances. We use the method from section 4.3.3 to generate 2 different types of personnel rostering problem instance sets: I1 and I2. In I1 and I2, both of them start from Monday, the horizon length of days is 14, $T = \{1, 2, \dots 14\}$. There is only 1 type of shift, the set of shifts $S = \{1\}$. 8 employees are assigned to different shifts, $E = \{1, 2, \dots, 8\}$. More detailed parameters setting are referred from Nurse Rostering Benchmark Instances(Schedulingbenchmarks.org).

We train our DNN on the instance sets mentioned above. I1 is an instance set including more than 4000 instances, which is used to train the DNN model initially. To ensure the better performance of the DNN model, we generate I2 of 2000 accurate instances, which is used to retrain the DNN model. After completing the training, we change one of the above specific problem parameters or add constraints to make it a different problem scenario, and then try to use our method to find the best solution to the new problem within a certain time limit. If the best solution cannot be found within the limited time, the current best solution is used.

We implement our algorithm in Python 3.7 using keras 2.3.0 under tensorflow 2.0.0 as the backend for the implementation of the DNN. All networks are trained using the Adam optimizer, which is based on a gradient descent(Kingma & Ba, 2014). All experiments are conducted using Tier-2 Cluster of the Vlaams Supercomputer Center.

5.2 Experiment 1: Comparison of change strategies

The correct change strategy in the search process of the selection tree is critical to the good performance of the algorithm. Therefore, we propose three different possible combinations of change strategies for DNNTS. We train the DNN model on the I2 data set, and then we apply the DNN model to the tree search with different change strategies combinations to evaluate their impact on the performance of the algorithm.

The combinations of change strategies we used for comparison is as follows:

Combination 1:

– Strategy1: randomly change 2 employees' shifts in one day.
– Strategy2: randomly change 2 days shifts for one same employee.
– Strategy3: randomly chose one shift which one of the employees who doesn't need to work, and change his status to work on that shift.

Combination 2:

– Strategy1: randomly change 2 employees' shifts in one day.
– Strategy2: randomly change 2 employees' shifts for 2 neighboring days.
– Strategy3: randomly change 2 days shifts for one same employee.

Combination 3:

– Strategy1: randomly change 2 employees' shifts in one day.
– Strategy2: randomly change 2 employees' shifts for 2 neighboring days.
– Strategy3: randomly change 2 days shifts for one same employee.

The optimization criterion is minimal penalty, the lower the better. A performance measure is the average time to solve. Another performance measure are the Figures showing declining process of the current optimal penalty. Since the final solution found by our method does not prove to be optimal, we will use final solution to represent the best solution we find in a complete algorithm process.

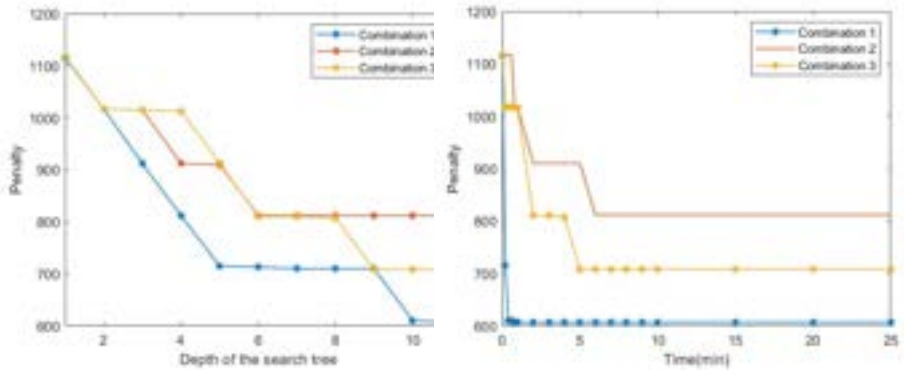
As can be seen from Table 1, whether the algorithm meets the stop criterion after running for 60s, 600s, or the end of the algorithm, the performance of the change strategies combination 1 is much better than the other two combinations in all aspects. As shown in the Table 1, combination 1 can find the current known best solution with the penalty value of 607 in less than 1 minute. In contrast, although strategy 3 can find a relatively good solution, the total time of the entire algorithm process is too long, nearly half an hour. Although combination 2 is slightly better than combination 3 in the total time, its final solution penalty value is worse than combination 3. As can be seen from Fig.9, whether it is based on the depth of the tree search or time, the convergence speed of combination 1 is much faster than the other two combinations, and the search process will not cost too much time on these feasible solutions with similar penalty value, it almost directly comes to the final solution with the shortest path. However, the other two combinations consume too much time in the local search, which causes many platforms on the curve in Fig.9, so the convergence rate becomes slow. This is closely related to the combination of changing strategies. The strategy "Randomly change 2 employees' shifts in one day." can ensure that the elements of the solution matrix are exchanged up and down, and the strategy "Randomly change 2 employees' shifts for one same employee." can ensure that the elements of the solution matrix are exchanged left and right. The strategy "Randomly chose one shift which one of the employees who doesn't need to work, and change his status to work on that shift." can control the number of different elements in the matrix. These three strategies can guarantee that the matrix changes covers every possibility.

5.3 Comparison of search strategies

We need to compare three search strategies mentioned above in order to find the best one, namely DFS, PFS and PPS. To ensure a fair comparison among

Table 1 Comparison of different obfuscations in terms of their transformation capabilities

Combination	Convergence Speed		Final result	
	60s	600s	Final penalty	Time(s)
1	607	607	607	43.41
2	1015	812	812	680.78
3	1015	708	708	1363.25



(a) Penalty with the depth of the tree (b) Penalty with the time

Fig. 9 The image of the convergence speed

strategies, we use the DNN model trained on the I2 dataset for each strategy to evaluate the performance. Table 2 also provides the current best solution found in a specific period of time(60s and 600s), the final solution, and the time of the three search strategies.

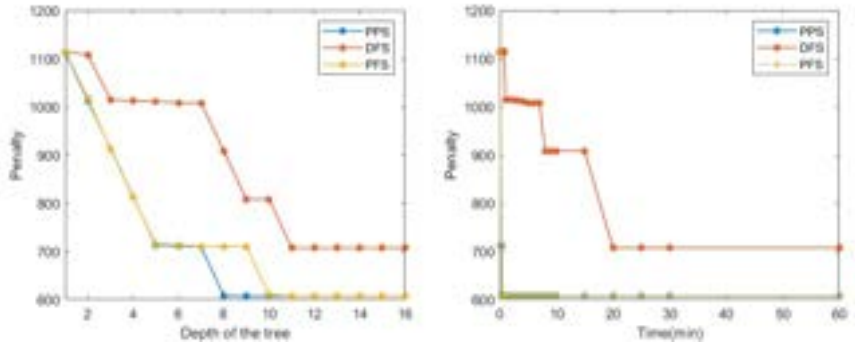
As can be seen from Table 2, PFS and PPS can find the final solution with the same penalty value, but PFS takes a little bit longer - just 13s. However, whether the current best solution in 60s , 600s or the final solution in the whole process, the performance of DFS is far inferior to PFS and PPS. It can be seen from Table 2 that the introduction of the DNN model to the branch selection in the process of ordinary tree search has a certain effect. It can be further seen from Fig.10 that, regardless of the depth or time of the search tree, DFS will encounter the problem that the penalty value convergence is too slow during the search process, which is actually caused by not selecting the correct branch, thus wasting more time. From the final result, the final solution of DFS stagnates at a penalty value of 707, while the convergence rate and the final solution of PPS and PFS are far superior to DFS. The final result further proves that the method of combing the DNN model with the tree search really works. Compared to PFS, the penalty value of PPS will converge slightly faster than PFS. This is why we proposed PPS. By giving Penalty a certain weight when selecting branches, it helps the search tree reach to the root as fast as possible in the beginning of the search, and the fast approach to the final solution also plays the guiding role of the DNN model.

Compared with ordinary search methods, because of the guidance of the DNN model, the number of nodes searched by DNNTS is many orders of magnitude

Table 2 Comparison of three search strategies

Search Strategy	Convergence Speed		Final result	
	60s	600s	Final penalty	Time(s)
DFS	1015	908	707	3453.47
PFS	607	607	607	43.41
PPS	607	607	607	30.54

less. Usually the best results can be achieved in twenty selections, which clearly shows that the DNN model is very effective.



(a) Penalty with the depth of the tree (b) Penalty with the time

Fig. 10 The image of the convergence speed

5.4 Experiment 3: Evaluation of methods

We compare the DNNTS and Roster viewer (Curtois & Qu, 2014), which has two built-in solutions-VDS (3.11) and Branch and Price (B&P). We compare these three methods on eight Groups with different problem types and report the performance of each method. Group 1 and Group 2 are designed by ourselves, and Group 3-11 are the instances from Nurse Rostering Benchmark Instances(Schedulingbenchmarks.org). Method VDS requires a limit on the maximum time. So we first run DNNTS, and set the maximum time of VDS equal to the time costed by DNNTS. So that we can better observe the performance of the different methods during the same time. The parameters of different group are shown in Table 3.

The results are shown in Table 4. For small instances (Group 1 to Group 9), B&P can find the best solution in a really short time, but for big instances(Group 10 and Group 11), it meets some problem. This is the exponential explosion problem often encountered in B&P method. Since there is a DNN model to do branch decision, our DNNTS method consumes acceptable time in all groups. In terms of the quality of the final solution, The performance of VDS isn't good, probably because it is an traversal algorithm, so maybe it needs more time to get better

Table 3 Parameters of each Group

Group	Weeks	Employees	Shift types	Hard constraints	Best known Solution
1	2	8	1	2	/
2	2	8	1	3	/
3	2	8	1	10	607
4	2	14	2	10	828
5	2	20	3	10	1001
6	4	10	2	10	1716
7	4	16	2	10	1143
8	4	18	3	10	1950
9	4	20	3	10	1056
10	4	30	4	10	1300
11	6	45	6	10	3833

results. Some penalty value of the final solution found by DNNTS are the same as the best known solutions, others are near to the best known solutions, which proves our method can be applied to the public instance set and propagated. The result shows that DNNTS can compete with the latest methods in terms of solution time and solution quality. And the result also illustrates the positive significance of introducing the DNN model into the tree search, and also emphasizes the importance of adequate training of the DNN model.

Table 4 Comparison to other methods on the test set

Group	Best Known Penalty	Final Penalty			Time(s)		
		DNNTS	B&P	VDS	DNNTS	B&P	VDS
1	/	0	0	0	1.98	0	2
2	/	1	1	1	1.60	0	2
3	607	607	607	607	43.42	0.27	44
4	828	828	828	937	205.47	0.13	206
5	1001	1001	1001	1103	170.77	0.45	171
6	1716	1718	1716	1721	283.9	1.5	284
7	1143	1143	1160	1636	78.28	25.61	79
8	1950	1952	1952	2340	1800.78	10.45	1801
9	1056	1057	1058	1278	1589.32	93.73	1590
10	1300	1317	1308	2643	560.93	11831.06	561
11	3827	4378	/	5514	2660.91	Out of memory	2661

6 Conclusion and future work

We propose a method aim to the personnel rostering problem combining DNN and tree search, which uses deep learning to assist branch selection. We prove that compared with the examples in the literature, DNNTS finds good solutions equal or near to the best known solutions, which are able to compare with the rostering problem solver. DNNTS can solve problems with very little user input. It mainly relies on the best known solution provided to learn how to build a solution by itself. There are many ways for DNNTS to work in the future. If we can model

other optimization problem as a standard model with the following characteristics, applying DNNTS to other similar optimization problems is a possible choice.

- The solutions of the problems can be regarded as m×n matrices, so that they can be transferred as the input of the neural network.
- The best solution can be found by some changes from the initial solution, so that some child nodes can be derived at each parent node.
- The problems have some soft constraints, so that we can use the penalty to set the lower bound of the tree search.

Other areas of future work include the use of reinforcement learning and others to further improve efficiency. We suggest that by using a faster programming language or using a GPU instead of a neural network's CPU, the runtime of the results we obtained may be improved. In addition, many changes can be made to DNNTS, such as reconfiguring the DNN network structure or adjusting branch pruning functions. These changes can improve performance in terms of runtime and solution quality.

References

Aickelin, U., & Dowsland, K. A. (2000). Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem. *Journal of scheduling*, *3*, 139–153.

Aickelin, U., & Dowsland, K. A. (2004). An indirect genetic algorithm for a nurse-scheduling problem. *Computers & Operations Research*, *31*, 761–778. doi:10.1016/s0305-0548(03)00034-0.

Ayob, M., Hadwan, M., Nazr, M. Z. A., & Ahmad, Z. (2013). Enhanced harmony search algorithm for nurse rostering problems. *Journal of Applied Sciences*, *13*, 846–853. doi:10.3923/jas.2013.846.853.

Bard, J. F., & Purnomo, H. W. (2005). Preference scheduling for nurses using column generation. *European Journal of Operational Research*, *164*, 510–534. doi:10.1016/j.ejor.2003.06.046.

Bello, I., Pham, H., Le, Q. V., Norouzi, M., & Bengio, S. (2016). Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, .

Bonfietti, A., Lombardi, M., & Milano, M. (2015). Embedding decision trees and random forests in constraint programming. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems* (pp. 74–90). Springer.

Brucker, P., Burke, E. K., Curtois, T., Qu, R., & Vanden Berghe, G. (2010). A shift sequence based approach for nurse scheduling and a new benchmark dataset. *Journal of Heuristics*, *16*, 559–573.

Burke, E., Cowling, P., De Causmaecker, P., & Vanden Berghe, G. (2001). A memetic approach to the nurse rostering problem. *Applied Intelligence*, *15*, 199–214. doi:10.1023/a:1011291030731.

Burke, E., De Causmaecker, P., & Vanden Berghe, G. (1999). A hybrid tabu search algorithm for the nurse rostering problem. In B. McKay, X. Yao, C. S. Newton, J.-H. Kim, & T. Furuhashi (Eds.), *Simulated Evolution and Learning* (pp. 187–194). Berlin, Heidelberg: Springer Berlin Heidelberg.

Burke, E., Kendall, G., & Soubeiga, E. (2003). A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics*, *9*, 451–470. doi:10.1023/b:heur.0000012446.94732.b6.

Burke, E. K., De Causmaecker, P., Vanden Berghe, G., & Landeghem, H. V. (2004). The state of the art of nurse rostering. *Journal of Scheduling*, *7*, 441–499. doi:10.1023/b:josh.0000046076.75950.0b.

Cheng, M., Ozaku, H. I., Kuwahara, N., Kogure, K., & Ota, J. (2008). Simulated annealing algorithm for scheduling problem in daily nursing cares. In *2008 IEEE International Conference on Systems, Man and Cybernetics* (pp. 1681–1687).

Curtois, T., & Qu, R. (2014). Computational results on new staff scheduling benchmark instances. *tech. report*, .

De Causmaecker, P. (2017). Data science meets optimization. In A. Sforza, & C. Sterle (Eds.), *Optimization and Decision Science: Methodologies and Applications* (pp. 13–20). Cham: Springer International Publishing.

De Causmaecker, P., & Vanden Berghe, G. (2010). A categorisation of nurse rostering problems. *Journal of Scheduling*, *14*, 3–16. doi:10.1007/s10951-010-0211-z.

Dilkina, B., Khalil, E. B., & Nemhauser, G. L. (2017). Comments on: On learning and branching: a survey. *Top*, *25*, 242–246.

Easton, F. F., & Mansour, N. (1999). A distributed genetic algorithm for deterministic and stochastic labor scheduling problems. *European Journal of Operational Research*, *118*, 505–523. doi:10.1016/s0377-2217(98)00327-0.

Girbea, A., Suciu, C., & Sisak, F. (2011). Constraint based approach for optimized planning-scheduling problems. *Bulletin of the Transilvania University of Brasov. Engineering Sciences. Series I*, *4*, 123.

Glass, C. A., & Knight, R. A. (2010). The nurse rostering problem: A critical appraisal of the problem structure. *European Journal of Operational Research*, *202*, 379–389. doi:10.1016/j.ejor.2009.05.046.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.

Gurobi Optimization, I. (2015). Gurobi ip solver, .

Hadwan, M., & Ayob, M. (2010). A constructive shift patterns approach with simulated annealing for nurse rostering problem. In *2010 International Symposium on Information Technology* (pp. 1–6). volume 1.

Hottung, A., Tanaka, S., & Tierney, K. (2020). Deep learning assisted heuristic tree search for the container pre-marshalling problem. *Computers & Operations Research*, *113*, 104781. doi:10.1016/j.cor.2019.104781.

Kawanaka, H., Yamamoto, K., Yoshikawa, T., Shinogi, T., & Tsuruoka, S. (2001). Genetic algorithm with the constraints for nurse scheduling problem. In *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546)* (pp. 1123–1130 vol. 2). volume 2.

Kazahaya, G. (2005). Harnessing technology to redesign labor cost management reports: labor costs typically represent over 50 percent of a hospital's total operating expenses. can the data management process be harnessed to create meaningful labor cost management tools? *Healthcare Financial Management*, *59*, 94–101.

Khalil, E. B., Dilkina, B., Nemhauser, G. L., Ahmed, S., & Shao, Y. (2017). Learning to run heuristics in tree search. In *IJCAI* (pp. 659–666).

Khalil, E. B., Le Bodic, P., Song, L., Nemhauser, G., & Dilkina, B. (2016). Learning to branch in mixed integer programming. In *Thirtieth AAAI Conference on*

Artificial Intelligence.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, .

Kool, W., & Welling, M. (2018). Attention solves your tsp. *arXiv preprint arXiv:1803.08475*, .

Kruber, M., Lübbecke, M. E., & Parmentier, A. (2017). Learning when to use a decomposition. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems* (pp. 202–210). Springer.

Lodi, A., & Zarpellon, G. (2017). On learning and branching: a survey. *TOP*, *25*, 207–236. doi:10.1007/s11750-017-0451-6.

Lü, Z., & Hao, J.-K. (2012). Adaptive neighborhood search for nurse rostering. *European Journal of Operational Research*, *218*, 865–876. doi:10.1016/j.ejor.2011.12.016.

Maenhout, B., & Vanhoucke, M. (2009). Branching strategies in a branch-and-price approach for a multiple objective nurse scheduling problem. *Journal of Scheduling*, *13*, 77–93. doi:10.1007/s10951-009-0108-x.

M'Hallah, R., & Alkhabbaz, A. (2013). Scheduling of nurses: A case study of a kuwaiti health care unit. *Operations Research for Health Care*, *2*, 1–19. doi:10.1016/j.orhc.2013.03.003.

Optimizer, I. I. C. C. (2015). V12. 3.

Osogami, T., & Imai, H. (2000). Classification of various neighborhood operations for the nurse scheduling problem. In *International Symposium on Algorithms and Computation* (pp. 72–83). Springer.

Paul, M., & Knust, S. (2015). A classification scheme for integrated staff rostering and scheduling problems. *RAIRO-Operations Research*, *49*, 393–412.

Rahimian, E., Akartunali, K., & Levine, J. (2015). A hybrid constraint integer programming approach to solve nurse scheduling problems, .

Rahimian, E., Akartunalı, K., & Levine, J. (2017). A hybrid integer programming and variable neighbourhood search algorithm to solve nurse rostering problems. *European Journal of Operational Research*, *258*, 411–423. doi:10.1016/j.ejor.2016.09.030.

Schedulingbenchmarks.org (). Shift scheduling benchmark instances. http://www.schedulingbenchmarks.org/index.html. Accessed 28 Feb. 2020.

Smet, P., Brucker, P., De Causmaecker, P., & Vanden Berghe, G. (2016). Polynomially solvable personnel rostering problems. *European Journal of Operational Research*, *249*, 67–75. doi:10.1016/j.ejor.2015.08.025.

Soto, R., Crawford, B., Monfroy, E., Palma, W., & Paredes, F. (2013). Nurse and paramedic rostering with constraint programming: A case study. *Romanian Journal of Information Science and Technology*, *16*, 52–64.

Stølevik, M., Nordlander, T. E., Riise, A., & Frøyseth, H. (2011). A hybrid approach for solving real-world nurse rostering problems. In J. Lee (Ed.), *Principles and Practice of Constraint Programming – CP 2011* (pp. 85–99). Berlin, Heidelberg: Springer Berlin Heidelberg.

Tassopoulos, I. X., Solos, I. P., & Beligiannis, G. N. (2015). A two-phase adaptive variable neighborhood approach for nurse rostering. *Computers & Operations Research*, *60*, 150–169.

Todorović, N., Petrović, S., & Teodorović, D. (2013). Bee colony optimization for nurse rostering. *IEEE Transactions on Systems Man & Cybernetics: Systems*,

43, 467–473.

Valouxis, C., Gogos, C., Goulas, G., Alefragis, P., & Housos, E. (2012). A systematic two phase approach for the nurse rostering problem. *European Journal of Operational Research*, *219*, 425–433. doi:10.1016/j.ejor.2011.12.042.

Vanden Berghe, G., Beliën, J., Bruecker, P. D., Demeulemeester, E., & Boeck, L. D. (2013). Personnel scheduling: A literature review. *European Journal of Operational Research*, *226*, 367–385. doi:10.1016/j.ejor.2012.11.029.

Vinyals, O., Fortunato, M., & Jaitly, N. (2015). Pointer networks. In *Advances in neural information processing systems* (pp. 2692–2700).

Plant shut-down maintenance workforce allocation and job scheduling

Hesham K. Alfares[0000-0003-4040-2787]

King Fahd University of Petroleum & Minerals, Dhahran 31261, Saudi Arabia
alfares@kfupm.edu.sa

Abstract. An important and challenging real-life problem is considered, involving workforce allocation and job scheduling for shut-down maintenance in a large oil refinery. A limited number of maintenance employees must be divided into several teams that work in parallel on different maintenance tasks. Since shut-down is costly, the aim is to minimize the total shut-down period, i.e., the time to complete all the maintenance tasks. Different team sizes are possible, and the size of the given team determines the speed of finishing the assigned maintenance tasks. Constraints include the total workforce size, the allowed team sizes, job availability (arrival) times, and precedence relations between different jobs. This problem can be considered as a resource-constrained parallel-machine scheduling problem, in which the objective is to minimize the makespan, and both the number and the speeds of the individual machines are decision variables. Optimization models and effective heuristic algorithms are developed for this NP-hard scheduling problem.

Keywords: Maintenance scheduling, parallel machine scheduling, makespan, resource-constrained scheduling

1 Introduction

The real-life problem considered in this paper is the optimum scheduling of shut-down maintenance activities in a large oil refinery. Shut-down maintenance is performed by a limited number of skilled workers supplied by a contractor at a high daily rate. These workers have to be divided into several work teams that work in parallel on different maintenance tasks. The number of workers in each team, which is subject to certain restrictions, determines the speed of completing each maintenance task assigned to the team. Models are formulated to determine the number of work teams, the size of each team, the jobs assigned to each team, and the sequence of processing these jobs. The objective is to minimize the time needed to complete all jobs, i.e., the shut-down duration, in order to reduce the cost of lost revenue and the cost of contract workers.

The problem analyzed in this paper can be considered as a resource-constrained parallel-machine scheduling problem, in which the limited resource is the maintenance workforce, and the parallel machines are the different maintenance teams. In machine scheduling terms, the objective is to minimize the makespan, and both the number and

the speeds of the individual machines are decision variables. The proposed machine scheduling problem involves resource constraints, precedence constraints, a variable number of machines, and variable machine speeds. This is a unique problem that has not been considered previously in scientific literature. It is also a challenging NP-hard scheduling problem, which is very hard to solve to optimality. An integer programming model is formulated to solve smaller instances of this problem. In order to solve larger real-life instances, an effective heuristic solution algorithm is developed and evaluated using a large number of test problems.

Remaining sections of this paper are organized as follows. Relevant recent literature is reviewed in section 2. The integer programming model is formulated in section 3. The heuristic solution is described in section 4. Finally, conclusions and suggestions are provided in section 5.

2 Relevant literature

Three aspects of the problem have been separately considered in prior literature: plant maintenance workforce scheduling, plant shut-down maintenance scheduling, and resource-constrained parallel machine scheduling. Relevant recent literature on these aspects is reviewed below.

Several integer programming models have been proposed for plant maintenance workforce scheduling. Alfares and Emovon (2007) use integer programming to compare alternative maintenance work schedules at a power station. The objective is to satisfy workload requirements with minimum cost and highest efficiency. Safaei et al. (2011) present a bi-objective mixed-integer programming model for workforce-constrained maintenance scheduling in a steel plant. The first objective is to minimize the total workforce requirements, and the second is to maximize the equipment availability. Koochaki et al. (2013) analyze the impact on maintenance workforce scheduling of two plant maintenance policies, namely condition-based maintenance and age-based maintenance. Leite and Vellasco (2020) adapt the particle swarm optimization (PSO) algorithm to schedule offshore maintenance activities and staff in order to maximize profitability.

Plant shut-down maintenance scheduling has been well studied in the literature. Castro et al. (2014) analyze long-term shut-down maintenance scheduling in a power plant with time restrictions on labor availability, demand variability, and price variability. A generalized disjunctive programming model is developed to maximize the revenue by maximizing labor utilization and minimizing shut-downs during high-price seasons. Adhikary et al. (2016) construct a nonlinear optimization model for preventive maintenance in continuous operating systems with two objectives: maximizing system availability and minimizing maintenance cost. To solve the model, a dual-objective genetic algorithm is developed and applied to optimize preventive maintenance in a real power plant.

The resource-constrained parallel machine scheduling problem, which is analogous to the maintenance scheduling problem analyzed in this paper, has received considerable attention in the literature. Edis et al. (2013) review and classify the literature on

parallel machine scheduling where additional resources are required, such as operators, tools, fixtures, and robots. Fanjul-Peyro et al. (2017) propose two integer programming models to minimize makespan for parallel machine scheduling with limited resources. Assuming the amounts of required resources depend on the job-machine assignment, three heuristics are developed to solve the two optimization models. Zheng and Wang (2018) analyze a resource-constrained parallel machine scheduling problem, aiming to minimize both the makespan and the total carbon emission. A multi-objective fruit fly optimization algorithm is used to determine the optimum job schedule and the speed of each machine.

Compared to previous work, this paper presents a new scheduling problem. If considered as a machine scheduling problem, it would have a variable number of parallel machines, variable machine speeds, resource-dependent job processing times, resource limitations, and precedence relations. As far as the author knows, this is the first paper in which the number of machines is itself a decision variable.

3 Model description

In the problem under study, a limited number of maintenance employees are available to perform a given set of maintenance jobs. These employees must be divided into several groups (work teams) that work in parallel on different subsets of the maintenance jobs, in order to finish all jobs in the minimum time duration. Therefore, the proposed maintenance scheduling problem involves employee team formation, job-team assignment, and job sequencing for each team. The aim is to minimize the makespan (time interval) needed to complete the shut-down maintenance. The binary integer programming model presented below is formulated to represent and optimally solve this problem.

3.1 Assumptions

The assumptions used to define the problem and construct the optimization model are listed below.

1. The maintenance workforce is homogeneous and composed of multi-skilled employees who are equally qualified to work on any maintenance job.
2. The size (number of employees) of any employee group (team) is limited to a given set of feasible values. Usually, two team sizes are specified: a standard (smaller) team size, and a rush (larger) team size.
3. The processing time of each job depends on the size of the group assigned to do the job. Usually, two processing time durations are possible for each job: a normal (longer) time by the smaller team size, and a crash (shorter) time by the larger team size.
4. Precedence relations exist between certain pairs of jobs, where a job cannot be started before the completion of its predecessor job(s).
5. A job cannot be started before its arrival time. Some jobs have an arrival delay period after which they become available for maintenance.

3.2 Decision variables

C = makespan, i.e., completion time of the last maintenance job
N = number of groups (work teams)

$$Q_g = \begin{cases} 1 \text{ if group } g \text{ is active} \\ 0 \text{ otherwise} \end{cases}$$
$g = 1,\ldots, G$

$$Y_{gk} = \begin{cases} 1 \text{ if the size of group } g \text{ is } s_k \text{ employees} \\ 0 \text{ otherwise} \end{cases}$$
$g = 1,\ldots, G,\ k = 1,\ldots, K$

$$X_{jgkt} = \begin{cases} 1 \text{ if job } j \text{ is assigned to group } g, \text{ of size } s_k, \text{ and started on day } t \\ 0 \text{ otherwise} \end{cases}$$
$j = 1,\ldots, J,\ g = 1,\ldots, G,\ k = 1,\ldots, K,\ t = 1,\ldots, T$

3.3 Structure of the integer programming model

A pure-integer linear programming (ILP) model is developed to represent and optimally solve the problem. is linear, All the decision variables are integer, and all of them except C and N are binary. The objective of the model is to minimize the makespan C , i.e., the total shut-down duration, subject to the following constraints:

1. Unique job scheduling constraints: ensure that each job j is assigned to one group g of one size s_k and has one start time t. Job-to-group and employee-to-group assignments are fixed during the shut-down maintenance period.
2. Unique job assignment constraints: assure that, for each work group, no more than one job is assigned in each time period.
3. Precedence constraints: guarantee that each job can start only after the completion time of all its predecessors.
4. Makespan constraints: set the makespan to be greater than or equal to the finish time of all jobs.
5. Logical constraints: relate the decision variables X_{jgkt} and Y_{gt}, by assigning jobs only to the active groups with the correct size.
6. Workforce constraint: ensure that the sum of group sizes does not exceed the available workforce size W.
7. Unique team size constraints: to guarantee that only one size is selected for each active group.
8. Number of teams' constraint: to equate the number of groups to the sum of active groups.

4 Heuristic solution method

Obtaining the optimum solution of the ILP model defined above is very difficult, especially for large problem sizes. The number of decision variables, especially X_{jgkt}, grows very rapidly with increasing problem size. To effectively solve larger sizes of this workforce allocation and job scheduling problem, the unique problem structure was utilized

to develop an efficient two-stage heuristic. First, ignoring job sequencing and precedence constraints, a simplified ILP model is used to assign jobs to teams. Next, jobs are sequenced for their respective assigned teams to incorporate arrival and precedence constraints.

In order to assess the real-world effectiveness of proposed heuristic, its performance has been verified assuming different problem characteristics. Therefore, to compare the heuristic solution with the optimum ILP solution, computational experiments were carried out using a wide variety of randomly generated test problems. The heuristic method has been shown to produce optimal solutions for all test problems significantly faster than the ILP model.

5 Conclusions

This paper presented a model for scheduling employees and tasks in plant shut-down maintenance. Given a limited number of maintenance employees and a set of maintenance tasks, the employees are divided into several teams of specific sizes such that each team is responsible for a given subset of the tasks. This allows several maintenance activities to be processed simultaneously (in parallel) by the different workforce teams. Naturally, the speed of processing each task depends on the size of the maintenance team assigned to perform it. Considering the work teams as machines, this problem can be presented as a resource-constrained parallel-machine scheduling model. As a parallel machine scheduling problem, this model is unique because both the number and the capacities of the machines are decision variables. The model has been successfully applied in a real-life shut-down maintenance scheduling problem in a large oil refinery. A heuristic algorithm is developed and shown to efficiently solve this scheduling problem.

6 References

1. Adhikary, D. D., Bose, G. K., Jana, D. K., Bose, D., & Mitra, S.: Availability and cost-centered preventive maintenance scheduling of continuous operating series systems using multi-objective genetic algorithm: A case study. Quality Engineering 28(3), 352–357 (2016).
2. Alfares, H. K., Lilly, M. T., & Emovon, I.: Maintenance staff scheduling at Afam power station. Industrial Engineering & Management Systems 6(1), 22–27 (2007).
3. Castro, P. M., Grossmann, I. E., Veldhuizen, P., & Esplin, D.: Optimal maintenance scheduling of a gas engine power plant using generalized disjunctive programming. AIChE Journal, 60(6), 2083–2097 (2014).
4. Edis, E. B., Oguz, C., & Ozkarahan, I.: Parallel machine scheduling with additional resources: Notation, classification, models and solution methods. European Journal of Operational Research, 230(3), 449–463 (2013).
5. Fanjul-Peyro, L., Perea, F., & Ruiz, R.: Models and matheuristics for the unrelated parallel machine scheduling problem with additional resources. European Journal of Operational Research, 260(2), 482–493 (2017).

6. Koochaki, J., Bokhorst, J. A., Wortmann, H., & Klingenberg, W.: The influence of condition-based maintenance on workforce planning and maintenance scheduling. International Journal of Production Research, 51(8), 2339–2351 (2013).
7. Leite, G. A., & Vellasco, M. M. B. R.: Development of offshore maintenance service scheduling system with workers allocation. In: 2020 IEEE Congress on Evolutionary Computation (CEC), pp. 1–6. IEEE, Glasgow, UK (2020).
8. Safaei, N., Banjevic, D., & Jardine, A. K.: Bi-objective workforce-constrained maintenance scheduling: a case study. Journal of the Operational Research Society, 62(6), 1005–1018 (2011).
9. Zheng, X. L., & Wang, L.: A collaborative multiobjective fruit fly optimization algorithm for the resource constrained unrelated parallel machine green scheduling problem. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 48(5), 790–800 (2018).

International Timetabling Competition 2019: A Mixed Integer Programming Approach for Solving University Timetabling Problems

Efstratios Rappos · Eric Thiémard · Stephan Robert · Jean-François Hêche

Abstract This summary article presents the mathematical programming approach used to solve and optimize the problem instances of the International Timetabling Competition 2019. The optimization problem was modeled as a mixed integer program which was solved using traditional branch-and-cut methods. Several innovative elements enabled to achieve good performance, such as the precalculation of several characteristics of the instances, the aggregation of constraints and the efficient use of auxiliary variables in the formulation. The computational implementation consisted of a first stage algorithm to obtain a feasible solution and an iterative local search metaheuristic to improve the quality of the resulting timetable. The solutions produced using this algorithm resulted in a ranking of second place in the competition.

Keywords ITC 2019 · timetabling problems · integer programming · combinatorial optimization

1 Introduction

This extended abstract describes the method used to solve the timetabling problems of the 2019 International Timetabling Competition [1]. Overall, we were able to solve 29 out of the 30 problem instances; the instance "agh-fal17" was not solved in time for the end of the competition. The mathematical modeling approach formulated the problem as a linear mixed integer program (MIP) and used techniques from large neighborhood search [2],[3],[4] and metaheuristics [5] to improve the solution quality.

2 Model formulation

The MIP formulation uses four sets of binary 0-1 variables, x, y, z and Z, representing the class times, class rooms, student-class allocation and student-course configuration alloca-

Efstratios Rappos · Eric Thiémard · Stephan Robert · Jean-François Hêche
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud (HEIG-VD)
University of Applied Sciences of Western Switzerland (HES-SO)
Yverdon-les-Bains, Switzerland
E-mail: efstratios.rappos@heig-vd.ch

tion respectively. The indices of these variables are described in Table 1 and their values uniquely represent a solution to a timetabling problem.

Variable	*Value*
$x_{c,t}$	1 if the class c takes place at time t, 0 otherwise
$y_{c,r}$	1 if the class c takes place in room r, 0 otherwise
$z_{s,c}$	1 if the student s is assigned to the class c, 0 otherwise
$Z_{s,f}$	1 if the student s follows the course configuration f, 0 otherwise

Table 1 Variables used in the formulation

The binary variables are linked by a set of linear constraints representing the solution requirements of the timetabling problems of the 2019 International Timetabling Competition. The hard constraints must be satisfied by any feasible solution and are summarized in Table 2.

Constraint	*Meaning*
C-1	Every class must be assigned a time
C-2	Every class must be assigned a room, where applicable
C-3	Every student must attend exactly one class from each subpart of the selected course configuration for each course that he must attend
C-4	For two classes with a parent-child relationship, if a class is assigned to a student then the parent class must also be assigned
C-5	Every student must be assigned a course configuration for every course that he follows
C-6	The capacity of each class in terms of the number of students must be satisfied
C-7	A room cannot be used when it is unavailable
C-8	Two classes cannot take place at the same time in the same room
C-9	Any hard distribution constraints must be satisfied

Table 2 Constraints used in the formulation

The first seven constraints are straightforward to formulate as linear inequalities using the above decision variables. For example, C-1 is represented by the formula $\sum_t x_{c,t} = 1$ for each class c; C-3 is expressed as $\sum_c z_{s,c} = Z_{s,f}$ for each student s, where the sum is over all classes c of one subpart of the course configuration f, and C-5 is $\sum_f Z_{s,f} = 1$ for every student s. The last two constraints C-8 and C-9 are modeled as inequalities of the form:

$$x_{c_1,t_1} + y_{c_1,r_1} + x_{c_2,t_2} + y_{c_2,r_2} \leq 3 \tag{1}$$

for every combination of class times and rooms which leads to a violated constraint. The inequality (1) prevents all four terms from taking the value 1 and therefore disallows this specific combination. There are however four types of hard constraints (namely, the special MaxDays, MaxDayload, MaxBreaks and MaxBlock constraints) that cannot be represented by inequalities of the form (1). This is because these constraints cannot be expressed in the form "for each pair of classes"; for these constraints an additional step is taken. Each time a potential new solution is found we need to check that these special constraints are satisfied, and if they are not, reject the solution by adding the appropriate inequalities, similar to (1) but with more terms, that forbid this combination of variables.

The objective function consists of four linear terms which correspond to the four solution quality criteria of the competition [1], namely the class time and room assignment costs, soft distribution constraint costs and student conflicts. The first two terms are simply

the weighted sum of the x and y variables. For the costs associated with the soft distribution constraints, an auxiliary variable is introduced in (1) which specifies if the constraint is satisfied or not, and the sum of these variables is used as the third term of the objective function. A similar method is used for the student conflicts. An auxiliary variable is introduced for every student and pair of classes he may follow, specifying whether a conflict exists or not. However, as the number of potential student conflicts can be very large, we aggregate the penalties associated with every pair of classes into one equation (for all students) by introducing an indicator variable which specifies whether a student follows both classes or not. Note that we are able to take into account both types of student conflict: where two classes overlap and when the travel time between the classes is insufficient.

Several innovative techniques are used to deal with the very large number of constraints of the above formulation, which makes it possible to solve the competition instances in a reasonable amount of time. The efficiency optimizations include:

- Extensive use of precalculated problem characteristics to save time between runs, for example the minimum and maximum gap and travel distances between classes
- Four types of logical checks to eliminate variables whose value can be deduced
- Removal of constraints that are always satisfied
- Reduction of the problem size by constraint aggregation

As an example of a logical check, we examine the existence of two classes which must take place in the same room. If one of these classes has its time assignment fixed, we can then exclude those time assignments of the second class which produce a conflict.

The computational implementation was done in Java using the commercial software CPLEX and Gurobi as the mixed integer programming solvers. The solution strategy is outlined in Agorithm 1 and consisted of two stages: the first stage focused at obtaining a feasible solution via an incremental addition of the hard constraints, whereas the second stage is a metaheuristic which combines iterative local search with mixed integer programming and aims to improve the solution quality.

The first stage performs a progressive addition of constraints into the model using slack variables to account for any violated constraints. Once a feasible solution is obtained the second stage iteratively improves the solution quality. In the end, the algorithm will produce better and better solutions until the optimization is stopped for practical reasons.

In both stages a number of decision variables is fixed to their last-solution values to reduce the size of the MIP. Several strategies of fixing the variables were developed and implemented sequentially, such as fixing only the x or y variables, fixing all variables within a class or fixing all classes within a soft distribution constraint. The strategy for fixing the variables is very important; the aim is to find a balance between fixing too many variables, which has a short running time but would produce a small improvement in the quality, and fixing too few variables which allows a wider exploration of the solution space at the expense of increased computational time.

The overall time taken for the first stage of the algorithm ranged from around 5 minutes to 21 hours (5 to 1850 optimization runs), except for the instance “pu-proj-fal19” which required around 260 hours to produce the first feasible solution. The amount of time needed for the first stage depended not only on the size of the problem but also the size of the solution space: small problems with few feasible timetable configurations can be hard to solve. The second stage took between 1 and 240 hours (50 to 5000 runs). Since the second stage is a local improvement metaheuristic, the decision to stop the optimization was partially based on practical considerations and, outside the competition, one could in theory continue these runs to further improve the solution quality.

Algorithm 1 A two-stage algorithm to solve and optimize the timetabling instances

```
{The first stage to obtain a feasible solution}
Solve a minimal MIP containing constraints C-1, C-2, C-5
while Some constraint is violated do
    Read the solution values of the last MIP solved
    Fix randomly some decision variables to their solution values (0 or 1)
    Create a MIP with all constraints C-1 to C-9 satisfied by the current solution
    Add to the MIP all the constraints C-1 to C-9 which are violated, using slack variables
    Optimize the MIP: minimize the sum of the slack variables
end while
return feasible solution
{The second stage to improve the solution quality}
Require: initial feasible solution
while time limit not reached do
    Read the solution values of the last MIP solved
    Fix some decision variables to their solution values (0 or 1)
    Create a MIP containing all the hard constraints
    Add variables and constraints related to student conflicts
    Optimize the MIP: minimize the four cost terms
end while
return solution
```

3 Conclusions and future work

This extended abstract presented a mixed integer programming approach for solving the timetabling problems of the International Timetabling Competition 2019, which produced a ranking of second place in this competition. Although the problem size for a typical timetable was very large to be solved exactly using traditional mixed integer programming tools, several improvements significantly reduced the size to manageable levels. Once a feasible solution was obtained, the use of mixed integer programming for the local search optimization stage proved to be very powerful in improving the quality of the solutions very quickly. A detailed article containing the detailed mathematical formulation, computational implementation, comprehensive results and in-depth analysis is in production and will appear in due course.

References

1. Müller, T., Rudová, H., Müllerová, Z.: University course timetabling and International Timetabling Competition 2019. In: PATAT 2018—Proceedings of the 12th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2018), 5–31 (2018)
2. Burke, E., Eckersley, A., McCollum, B., Petrovic, S., Qu, R.: Hybrid variable neighbourhood approaches to university exam timetabling. European Journal of Operational Research **206**(1), 46–53 (2010)
3. Ergun, Ö., Orlin, J.B., Steele-Feldman, A.: Creating very large-scale neighborhoods out of smaller ones by compounding moves. Journal of Heuristics **12**, 115–140 (2006)
4. Pisinger, D., Ropke, S.: Large neighborhood search. In: M. Gendreau, J.Y. Potvin (eds.) Handbook of Metaheuristics, pp. 399–419. Springer US, Boston, MA (2010)
5. Lindahl, M., Sørensen, M., Stidsen, T.R.: A fix-and-optimize matheuristic for university timetabling. Journal of Heuristics **24**(4), 645–665 (2018)

Simulated Annealing with Penalization for University Course Timetabling

Edon Gashi · Kadri Sylejmani · Adrian Ymeri

1 Introduction

In this paper we present a simulated annealing algorithm for solving the problem of University Course Timetabling as formulated in the International Timetabling Competition (ITC) 2019 [6].

The presented algorithm is based on a modified version of simulated annealing. It utilizes a suitable cooling function [5] and an adaptive evaluation function [7], both of which have proven useful for normalizing the varying characteristics of problem instances.

The algorithm searches both feasible and infeasible regions of the solution space, where the latter is dealt with by using a combination of incremental penalization and narrowed search on specific hard constraints.

The solver based on this algorithm has placed among the five finalists of the International Timetabling Competition 2019.

2 Solution approach

Solution representation A given solution is always modeled to be complete. This means that variables are always assigned to some value, even if their configuration evaluates to an invalid solution. A solution is the list of all variables V_i (where $i = 1..n$) and their values. A sample representation of

E. Gashi
University of Prishtina
E-mail: edon.gashi@uni-pr.edu

✉ K. Sylejmani
University of Prishtina
E-mail: kadri.sylejmani@uni-pr.edu

A. Ymeri
University of Prishtina
E-mail: adrian.ymeri@studentet.uni-pr.edu

a given solution is $S = \{V_1, V_2, V_3, ..., V_i, ..., V_n\}$, where the assignment of variable V_i in a solution S is modeled to have three components CT - Class Time, CR-Class Room or SE- Student Enrollment (i.e. $V = \{CT, CR, SE\}$), where CT = {Class ID, Time slot}, CR = {*Class ID*, *Room Index*}, and SE = {(*Student ID*, *Course Index*), *Class Chain Index*}. Variable n represents the total number of all course configurations and the *Class Chain Index* represents a particular assignment combination of a given course.

We maintain three separate penalties to determine the state of the solution: hard penalty, class overflows, and soft penalty. Soft penalty is calculated using the identical rules described by [6], and hard penalty is calculated as follows:

- A conflict between a pair of classes gives 1 hard penalty point.
- A time assignment conflicting with a room's unavailable schedule gives 1 hard penalty point.
- An unsatisfied required constraint gives hard penalty points equal to the soft penalty the constraint would give if it were not required.

Class overflows penalty is the sum of all over-enrollments on classes. We maintain this as a separate penalty because it is not as constrained as hard penalty and is easier to satisfy.

Neighborhood function A mutation is an operation that changes a single variable. We hold two lists of possible mutations: feasible mutations and infeasible mutations. Infeasible mutations are applied on solutions with non-zero hard penalty, and they do not contain operations on students.

In concrete terms, a mutation is a single operation that either changes: (1) the schedule of a class, (2) the room of a class, or (3) the class configuration of a student attending a particular course.

The neighborhood operator has a 50% chance of performing a single mutation (selected randomly from the possible mutations described above), and a 50% chance of performing up to a maximum of three mutations, where the number of mutations is selected at random with uniform probability.

Initial solution The initial solution S is deterministically assigned by giving each variable an iterating natural number in the domain from 1 to 3, which map the set of possible variable values $V = \{CT, CR, SE\}$, respectively. If the attempted number exceeds the domain size, then the last number in the domain is taken.

Cooling function The cooling schedule in Equation (1) is based on [5], where the value of β has been assigned empirically. Because of the varying amount of time it takes to solve different instances, a fixed restart temperature is used once the hard penalty hits 0. In Equation (1), variable t represents the temperature in the current iteration, whereas variable t' represents the temperature for the next iteration.

$$t' = \frac{t}{1 + \beta t} \tag{1}$$

Evaluation function There are a few types of evaluation functions, depending on the phase and context. In equations (2) and (3), p_h stands for hard penalty, p_s for soft penalty, and p_c for class overflows penalty, whereas c_1 and c_2 are empirically defined constants. The function $round_2$ means rounding to 2 digits after the decimal separator. The worst soft penalty of a problem is calculated statically by taking the worst case penalty of all constraints and assignments.

$$\text{searchPenalty}(s) = \begin{cases} c_1 p_h + \text{round}_2(c_2 p_c + \text{normalize}(p_s)), & p_h > 0 \\ c_2 p_c + \text{normalize}(p_s), & p_h = 0 \end{cases} \tag{2}$$

$$\text{normalize}(p_s) = \frac{p_s}{\text{worst soft penalty of problem}} \tag{3}$$

The simulated annealing acceptance condition compares the difference between values of f_{stun} [7], which is shown in Equation (4). The constant γ is defined empirically, whereas f_0 denotes the quality of the best solution. Thus, the energy difference ΔE in simulated annealing is defined in Equation (5), where s and s' represent the current and next solution, respectively.

$$f_{\text{stun}}(x) = 1 - \exp\left[-\gamma(f(x) - f_0)\right] \tag{4}$$

$$\Delta E(s', s) = f_{\text{stun}}\left(\text{searchPenalty}(s')\right) - f_{\text{stun}}\left(\text{searchPenalty}(s)\right) \tag{5}$$

Particular class-time and class-room combinations can become penalized over time. The *modifiedPenalty* function defined in Equation (6) is expressed as the search penalty combined with the total sum of all penalties of present features in a solution s.

$$\text{modifiedPenalty}(s, \mathit{penalties}) = \text{searchPenalty}(s) + \sum_{x}^{\mathit{features}_s} \mathit{penalties}_x \tag{6}$$

Penalization is performed after local timeouts for each feature by incrementing a constant multiplied by the hard penalty contribution of that particular feature.

Sometimes hard constraints persist for many timeouts. In such cases, we pivot the solution by performing hill-climbing with a different penalization function, as shown in Equation (7). A limited subset of persistent constraints (*focus*) is taken and random walks are performed until the solution has reached a sufficiently different shape.

$$\text{focusedPenalty}(s, \mathit{focus}) = f_{\text{stun}}\left(\text{searchPenalty}(s)\right) + \sum_{x}^{\mathit{focus}} \text{penalty}(x) \tag{7}$$

Algorithms 1 and 2 provide a simplified overview of the solver.

Algorithm 1 Simulated annealing with penalization

```
 1: procedure SOLVE(initial solution)
 2:     t ← initial temperature
 3:     penalties ← initial penalties
 4:     best ← initial solution
 5:     local best ← ∞
 6:     local timeout ← 0
 7:     current ← best
 8:     while stopping criteria not met do
 9:         t ← cool(t)
10:         candidate ← mutate(current)
11:         if candidate better than best then
12:             best ← candidate
13:         end if
14:         if searchPenalty(candidate) < local best then
15:             local best ← searchPenalty(candidate)
16:             local timeout ← 0
17:         else
18:             local timeout ← local timeout + 1
19:         end if
20:         if modifiedPenalty(candidate) < modifiedPenalty(current) then
21:             current ← candidate
22:         else if accept(current, candidate, t) then
23:             current ← candidate
24:         end if
25:         if local timeout > limit then
26:             local best ← ∞
27:             local timeout ← 0
28:             t ← restart temperature
29:             persistent constraints ← {constraints with age > age limit}
30:             if infeasible(current) ∧ persistent constraints ≠ ∅ then
31:                 focused constraints ← oldest 3 persistent constraints
32:                 current ← ConstraintSearch(current, focused constraints)
33:             else
34:                 penalties ← scale(penalties)
35:             end if
36:         end if
37:     end while
38:     return best
39: end procedure
```

Algorithm 2 Focused search on particular constraints

```
 1: procedure CONSTRAINTSEARCH(solution, focused constraints)
 2:     timeout ← 0
 3:     while timeout < timeout limit do
 4:         candidate ← RandomWalk(solution, distance)
 5:         if focusedPenalty(candidate) < focusedPenalty(solution) then
 6:             solution ← candidate
 7:             timeout ← 0
 8:         else
 9:             timeout ← timeout + 1
10:         end if
11:     end while
12:     return solution
13: end procedure
```

3 Results and conclusion

The presented algorithm managed to solve all instances of the ITC2019 in time limit of maximum 24 hours. It has won first place in both 1st and 2nd milestone of the competition [6], and has placed third in the final round. In addition, our solver has won the prize as the best approach in the open source category.

In Table 1, we present the comparison results of our approach against the results of four of the other finalists, namely Holem et al.[3], Rappos et al. [1], Er-rahaimini [2] and Lemos et al. [4]. In addition, we have also included the results presented by Müller (one of the organizers of the competition), who, after the end of the competition, has published the results obtained by the solver that is used by UniTime timetabling system. The presented results of our approach (tagged as Edon et al.) are the best results that have been achieved when running the solver for 24 hours for each instance. The results in Table 1 show that our approach is outperformed, in all of the instances, by the the approaches of Holem et al.[3] and Müller, whereas Rappos et al.[1] performs better in 23 (out of 30) instances. Our approach performs better than the approach of Rappos et al.[1] in seven instances, better than the approach Er-rahaimini [2] in 19 instances, and better than the approach of Lemos et al. [4] in 21 instances. In addition, the solutions of our solver have a gap of less than 15% from the best known solutions in 5 (out of 30) instances. Furthermore, the average gap from the best known solution for early, middle and late chunks of instances are about 80%, 90% and 230%, respectively.

Overall, this model shows that it can solve complex and large instances with distinct features in terms of number of courses, number of students, number of rooms, as well as other distribution and special constraints.

References

1. Rappos Efstratios, iemard Eric, Stephan Robert, and Jean-Francois Heche. International timetabling competition 2019: A mixed integer programming approach for solving university timetabling problems. 2021.
2. Karim Er-rhaimini. Forest growth optimization for solving timetabling problems. 2021.
3. Dennis Søren Holm, Rasmus Ørnstrup Mikkelsen, Matias Sørensen, and Thomas Jacob Riis Stidsen. A mip formulation of the international timetabling competition 2019 problem. 2020.
4. Alexandre Lemos, Pedro T. Monteiro, and Ines Lynce. Itc-2019: A maxsat approach to solve university timetabling problems. 2021.
5. Miranda Lundy and Alistair Mees. Convergence of an annealing algorithm. *Mathematical programming*, 34(1):111–124, 1986.
6. Tomáš Müller, Hana Rudová, Zuzana Müllerová, et al. University course timetabling and international timetabling competition 2019. In *Proceedings of 12th International Conference on the Practice and Theory of Automated Timetabling (PATAT)*, pages 5–31, 2018.
7. Wolfgang Wenzel and Kay Hamacher. Stochastic tunneling approach for global minimization of complex potential energy landscapes. *Physical Review Letters*, 82(15):3003, 1999.

Table 1 Presentation of the gap (in percentage) from the best known results

Instance name	Best	Holm et al.(%)	Müller (%)	Rappos et al.(%)	Gashi et al.(%)	Er-rhaimini(%)	Lemos et al.(%)
agh-fis-spr17	3039	0	12.2	49.9	123.7	87.8	N/A
agh-ggis-spr17	34285	0	6.2	6.7	127.3	65.5	N/A
bet-fal17	289965	0	0.2	1.8	3.1	N/A	2.1
iku-fal17	18968	0	24.5	41.5	166.8	134.5	57.7
mary-spr17	14910	0	1.4	0.7	6.5	11.9	N/A
muni-fi-spr16	3756	0	7.5	2.3	33.2	38.6	N/A
muni-fsps-spr17	868	0	1.2	1.7	123.2	376.3	N/A
muni-pdf-spr16c	33724	0	18.3	11.1	72.5	130	59.5
pu-llr-spr17	10038	0	7.5	33.3	68.1	91.5	N/A
tg-fal17	4215	0	0	0	90.8	74.5	60.7
agh-ggos-spr17	2864	0	19.3	120.6	225.6	169.7	2684.3
agh-h-spr17	22175	0.05	0	17.9	13.1	16.1	N/A
lums-spr18	95	0	3.1	20	12.6	87.3	475.7
muni-fi-spr17	3825	0	3.6	12.1	22.6	42	372.6
muni-fsps-spr17c	2596	0	16	27.2	255	806	23714.2
muni-pdf-spr16	17208	0	16.8	41.3	132.8	125.6	1707.2
nbi-spr18	18014	0	3.7	5.7	47.2	68.2	177.1
pu-d5-spr17	15204	4.6	0	23.7	27.8	33.1	3.4
pu-proj-fal19	117425	25.7	0	377.9	102.6	49.9	9.32
yach-fal17	1074	15.3	0	71.6	60.8	196.1	N/A
agh-fal17	118038	24.5	0	N/A	20.8	55.9	29.8
bet-spr18	348524	0	0.2	3.3	1.5	3.4	7
iku-spr18	25868	0	39.1	41.9	76	232.3	174.2
lums-fal17	349	0	5.7	10.6	132.9	39.2	59.8
mary-fal18	4422	0	8.8	27.5	897.2	62.8	57
muni-fi-fal17	2999	0	8.2	26.5	38.7	57.1	60.7
muni-fspsx-fal17	10123	68.3	0	226	900.8	314.2	933.5
muni-pdfx-fal17	98373	13.7	0	53.9	53.9	61.8	95.1
pu-d9-fal19	39942	0	11.9	235.5	19	107.2	76.3
tg-spr18	12704	0	14.5	1.2	151.1	25.8	55

Metaheuristic for the Personalized Course Sequence Recommendation Problem

Aldy Gunawan · Audrey Tedja Widjaja · Roy Ka-Wei Lee · Ee-Peng Lim

1 Introduction

Scheduling problems have been studied in various domains, such as machine scheduling, staff scheduling, transportation and sport scheduling. In this paper, we focus on a novel scheduling problem in the university context, namely personalized course sequence recommendation. Many universities today offer courses in such a way that gives students more flexibility in selecting courses. The universities offer not only compulsory courses but also elective courses that students can choose based on certain criteria, such as majors, specific programmes, specialized courses and elective courses. Students are expected to take a required number of courses over a sequence of terms. Among the factors considered by students when selecting courses, course instructors, academic preference, relevance to career plan, and GPA (Grade Point Average) are often the important ones. GPA is commonly used in a university in Singapore. This scoring system refers to a student's academic performance and reflects it as a number - the higher the better.

Recommender systems become an important research area since it helps users to find the right content, products, or services [4]. Much of the research pertaining to recommendation systems has been conducted in the domain of e-commerce. [5] describe a recommender system as follows:

In a typical recommender system, people provide recommendations as inputs, which the system then aggregates and directs to appropriate recipients. In some cases the primary transformation is in the aggregation; in others the system's value lies in its ability to make good matches between the recommenders and those seeking recommendations.

A. Gunawan, A.T. Widjaja, E.-P. Lim
Singapore Management University
E-mail: aldygunawan, audreyw, eplim@smu.edu.sg

R. K.-W. Lee
Singapore University of Technology and Design
E-mail: roy_lee@sutd.edu.sg

The application of the recommendation system in the education sector has recently gained popularity. [6] summarized the main challenges of developing recommendation systems. Unlike most existing recommender systems, such as movies or products to buy, course sequence recommender system generates sequences of courses rather than a single item at a time. The complexity increases when the number of courses offered is large. Other factors such as considering the student's performance and the university (or school) requirements simultaneously further add challenges to the task. [2] developed an application, namely SUcheduler, for helping students to plan their personalized course schedules by considering their preferences over sections, instructors and other factors. The application utilizes a declarative problem solving method based on Answer Set Programming, for generating course schedule plans. However, the model assumes that students have selected their courses before hand. Basically, it generates possible candidate solutions, using choice rules and eliminates the candidates that violate predefined constraints.

In this paper, we develop a personalized course sequence recommendation system with the main objective of generating a sequence of courses for all subsequent terms, i.e. until the final graduation term. [6] highlighted that prolonged graduation time may arise when courses are only taken myopically, without a clear plan. Therefore, it is important to tailor course sequences to students since students may not have the same learning path. We first analyze the past data that covers course titles and grades from previous terms of undergraduate students from a university in Singapore. From the data, we have learnt that students may achieve better GPA if they choose suitable sets of courses and order them in certain sequences. Performance of a student evolves in the process of learning [6]. Based on past courses taken by a student and the course grades, we design an objective function that returns a course sequence which maximizes the student's GPA. We propose an algorithm based on simulated annealing to determine the optimal course sequence. We conduct experiments based on a real-world student record dataset to verify the efficacy of the proposed algorithm.

2 Personalized Course Sequence Recommendation Problem

For a particular degree with a given set of courses $C = \{1, 2, ...|C|\}$, C_c and C_e are defined as sets of compulsory and non-compulsory (elective) courses respectively $(C_c, C_e \subset C)$. Each student must complete C_c and take a subset of C_e in order to fulfill the graduation requirements. Assume that a student has completed the first T terms. The set of courses taken in term t is denoted as $C'_t (t \in T)$, and the actual grade received for the course i in term t is denoted as $G_{it} (t \in T, i \in C'_t)$. In our problem, there are five possible grades for courses $G = \{A, B, C, D, F\}$. They are converted to scores of 4, 3, ..., 0 respectively. Every student can take a maximum of C_{max} courses in each term, and is expected to complete C_{total} courses for graduation.

The main objective of the course sequence recommendation is to recommend a sequence of courses to a student, based on his first T terms information, such that all of the graduation requirements and prerequisites of courses are satisfied, and his overall GPA is maximized. In this work, we have chosen to maximize the overall GPA which has been highly valued among students within the highly competitive ed-

ucation systems [1]. Therefore, selecting what courses to be taken in their following terms should be done very carefully. For example, students who perform well in programming courses tend to perform well in advanced programming and data analytics courses. For such students, recommending both advanced coding and data analytics courses to be taken within the same term would not be a problem. However, for students who had performed badly in programming courses, it will be more challenging for them to take both advanced programming and data analytics courses within the same term.

Our proposed personalized course sequence recommendation approach is divided into two main stages:

Stage 1: Grade Estimation. The grade in course j after completing course i with grade g, denoted as E_{ij}^{g}, where $i, j \in C, g \in G$ is estimated. We define the probability of completing course j with grade g' after completing course i with grade g, denoted as $P_{ij}^{gg'}$, in equation (1). $S_j^{g'}$ is the set of students who have taken course j and obtained grade g', S_i^g is the set of students who have taken course i and obtained grade g, and S_j is the set of students who have taken course j.

$$P_{ij}^{gg'} = \frac{|S_i^g \cap S_j^{g'}|}{|S_i^g \cap S_j|} \tag{1}$$

The expected grade of course j after completing course i with grade g, E_{ij}^{g}, is defined by Equation (2). $GP^{g'}$ represents the grade point of grade g', where $GP^{g'} = \{4,3,2,1,0\}$.

$$E_{ij}^{g} = \sum_{g' \in G} P_{ij}^{gg'} \times GP^{g'} \tag{2}$$

Stage 2: Course Sequence Construction. The course sequence is constructed in order to maximize the overall estimated grade. For a particular student, an initial solution is constructed by recommending the next course j, starting from $|T|+1$, that satisfies the prerequisite requirements and Equation (3). The prerequisite requirement is defined as a certain list of courses that must be completed until term $|T|$ in order for a particular student is able to take course j in term $|T|+1$.

$$\underset{j \in C}{\operatorname{argmax}} \left\{ G_{j,|T|+1} = \frac{\sum_{t \in T} \sum_{i \in C_t'} E_{ij}^{G_{it}}}{\sum_{t \in T} |C_t'|} \right\} \tag{3}$$

We first recommend as many compulsory courses C_c as possible, before recommending C_e. Once we recommend C_{max} courses, we increase $|T|$ by one, update C_t' and G_{it}, and repeat the procedure until C_{total} is reached. The overall expected GPA, $E(GPA)$, of a student is defined in Equation (4). $E(GPA)$ is also the objective function to be maximized.

$$E(GPA) = \frac{\sum_{t \in T} \sum_{i \in C_t'} G_{it}}{\sum_{t \in T} |C_t'|} \tag{4}$$

A course sequence for a particular student, say student A, can be represented as a two-dimensional matrix $|T| \times C_{max}$, as illustrated in Figure 1. For example, in term

3 (see row 3), we recommend student A to take courses 166, 204, 37, 72, and 4. However, for cases where $|T| \times C_{max} \neq C_{total}$, -2 is added as a dummy value.

	Course_code				
Term	51	43	27	101	54
	114	41	108	212	15
	166	204	37	72	4
	238	262	240	288	33
	194	230	137	38	254
	155	224	214	394	157
	106	14	301	339	557
	281	536	245	-2	-2

Fig. 1: Example of solution representation with $|T| = 8, C_{max} = 5, C_{total} = 38$

To improve the initial solution, we propose an adaptive simulated annealing algorithm [3]. The algorithm includes parameters that control temperature schedule and the operator selections are automatically adjusted as the algorithm evaluates the later iterations. This makes the algorithm more efficient and less sensitive to user-defined parameters than pure simulated annealing. We adjust the probability of choosing the local search operators, such that operators with good performance in the past iterations will get a higher chance to be selected in the subsequent iterations. We implement six local search operators:

- **Swap**: choose two courses randomly from different terms and exchange their positions.
- **Move**: choose two courses randomly from different terms, move the position of the second course before the first course, and push back the courses in-between both courses.
- **2-opt**: choose two courses randomly and reverse the sequence between both courses.
- N**-replacement**: remove N courses (i.e. elective courses) from the solution, then, add other N unselected courses to the solution. We implemented $N = 1, 2, 3$.

3 Experimental Results and Discussion

We use a real dataset from a particular school of a university in Singapore. To perform grade estimation, we consider students who enrolled in years 2010 until 2018 with a total of 3905 students and 644 courses. We classify grades into 5 categories, $|G| = 5$. This grade estimation approach is tested on 572 students, we found that a mean absolute error of 0.32 with 172 (366) students' results are overestimated (underestimated). This ensures the fairness of the measurement conducted in the following experiments as the grade estimation is not always overestimated.

The proposed algorithm is tested on a subset of students: students enrolled in years 2011 until 2014, from two different program tracks, Track 1 and Track 2. Since the final grade upon graduation are known, we are able to evaluate the performance of our proposed algorithm by: (i) the number of students enjoying grade improvement from the recommended course sequence compared to the actual course sequences. This is derived by the number of students with $E(GPA_{recommendation}) > GPA_{student}$ and (ii) the grade improvement obtained if students follow our recommended course sequences compared to their actual performance (measured by Equation (5)).

$$\%imp = \frac{E(GPA_{recommendation}) - GPA_{student}}{GPA_{student}} \times 100\% \tag{5}$$

From a total of 385 students, our proposed algorithm is able to improve the expected overall grade for 218 students, with an average improvement of 3.15%. Most of the improvement comes from students with low $GPA_{student}$ grades. It is harder to improve overall grades when the students have already obtained high actual grades (i.e. above 3.0).

To overcome this matter, we try other two scenarios: (i) to increase $|T|$ and (ii) to perform grade moderation. The results are summarized in Table 1.

Scenario (i) - to increase $|T|$. This approach is implemented such that we are able to "know more" about the student's past performance. Here, we use $|T| = 2$, meaning that we use student's first and second terms information to recommend the course sequence for the following terms. By using this approach, we are able to improve the expected overall grade for 228 students, with an average improvement of 3.24%.

Scenario (ii) - grade moderation. This approach is implemented to "adjust" our grade estimation, depending on student's performance in their first term ($|T| = 1$). The moderation is done by deriving the performance index (PI) for each student, by Equation (6), where $\overline{x_s}$ is the average student's grade in his first term and $\overline{x_d}$ is the average grade obtained by all students who have taken the same set of courses.

$$PI = \frac{\overline{x_s}}{\overline{x_d}} \tag{6}$$

$PI = 1$ indicates that the student is normal (performs as well as the average student), $PI > 1$ indicates the student has an academic ability above other students, and therefore we try to "upgrade" our grade estimation to match his ability, while $PI < 1$ indicates the student does not perform well compared with other students, and therefore we try to "downgrade" our predicted grade to match his ability. When applying this approach, Equation (3) is replaced by Equation (7). In our experiments, this new grade estimation scheme is shown to improve the expected overall grade for 236 students; but, the average improvement falls to 2.71%.

$$\underset{j \in C}{\operatorname{argmax}} \left\{ G_{j,|T|+1} = \frac{\sum_{t \in T} \sum_{i \in C'_t} E_{ij}^{G_{it}}}{\sum_{t \in T} |C'_t|} \times PI \right\} \tag{7}$$

Table 1: Results of different scenarios

	# improvement			Average of $\%imp$		
	Track 1	**Track 2**	**Both**	**Track 1**	**Track 2**	**Both**
Initial approach	76	142	218	1.62	4.14	3.15
Scenario (i)	77	151	228	1.90	4.10	3.24
Scenario (ii)	83	153	236	0.97	3.83	2.71

4 Conclusion

In this paper, we introduce a personalized course sequence recommendation system to suggest courses for students to achieve good academic performance. In our proposed model, the main objective is to maximize the expected GPA with respect to several constraints, such as the maximum number of courses taken in each term and prerequisite constraints. We propose an adaptive simulated annealing algorithm in order to solve the problem. The operator selections are dynamically adjusted. Our preliminary results show that the proposed algorithm is able to improve the expected GPA by recommending course sequences for 218 out of 385 students, with an average improvement of 3.15%. The current model only concerns about maximizing GPA while other factors that may affect the performance of students have not been considered yet, such as instructors, colleges of students, and so on. Furthermore, the popularity of a particular course has not been addressed in this work, e.g. courses with higher number of students enrolled are more likely to be recommended. Machine learning for education has recently gained attention in this recommendation system. For future work, we will focus on using machine learning techniques to further improve the grade prediction accuracy and develop a User Interface to allow students to use.

Acknowledgements This research is supported by the National Research Foundation, Singapore under its Strategic Capabilities Research Centres Funding Initiative. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

References

1. Feldman, D.B., Kubota, M.: Hope, self-efficacy, optimism, and academic achievement: Distinguishing constructs and levels of specificity in predicting college grade-point average. Learning and Individual Differences **37**, 210–216 (2015)
2. Kahraman, M.K., Erdem, E.: Personalized course schedule planning using answer set programming. In: J.J. Alferes, M. Johansson (eds.) Practical Aspects of Declarative Languages, pp. 37–45. Springer International Publishing, Cham (2019)
3. Aguiar e Oliveira Junior, H., Ingber, L., Petraglia, A., Rembold Petraglia, M., Augusta Soares Machado, M.: Adaptive Simulated Annealing, pp. 33–62. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
4. Park, D.H., Kim, H.K., Choi, I.Y., Kim, J.K.: A literature review and classification of recommender systems research. Expert Systems with Applications **39**(11), 10,059–10,072 (2012)
5. Resnick, P., Varian, H.R.: Recommender systems. Communications of the ACM **40**(3), 56–58 (1997)
6. Xu, J., Xing, T., van der Schaar, M.: Personalized course sequence recommendations. IEEE Transactions on Signal Processing **64**, 5340–5352 (2016)

Conflicts in Examination Timetabling under Uncertainty

Bernd Bassimir · **Rolf Wanka**

Abstract In the literature the examination timetabling problem (ETTP) is mostly described as a post enrollment problem (PE-ETTP). As such it is known at optimization time how many students will take an exam and consequently how big a room is needed for the exam and which exams should not be held at the same time because of overlapping student lists. In contrast some universities start their scheduling process before students register. As such the model is subject to uncertainty in respect to the number of students per exam and the conflicts between exams. In this work we focus on the uncertainty of conflicts between exams and introduce two soft criteria for handling this uncertainty. We show results for two real world instances taken from the School of Engineering at the Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU).

Keywords Examination Timetabling, Curriculum-Based Timetabling, Robust Optimization

1 Introduction

In every academic term, universities are faced with a number of different aspects of academic timetabling. Academic timetabling can be divided into two distinct, but similar problems, namely the *Course Timetabling Problem* (CTTP) prior to each term and the *Examination Timetabling Problem* (ETTP) at the end of the term.

In the literature the ETTP is often treated as a timetabling problem, where all data is provided as input to a scheduling algorithm and no uncertainty is present. This approach is sometimes called *post-enrollment* ETTP (PE-ETTP) or is at least treated as PE-ETTP, e. g., see [5,6,8]. First the students register for their different exams and after registration is finished the optimization takes place. In this approach the complete input is known, as we have the exact number of students registered for an exam and exams are in conflict, i. e., should not be scheduled at the same

Department of Computer Science, University of Erlangen-Nueremberg, Germany
E-mail: bernd.bassimir@fau.de · rolf.wanka@cs.fau.de

time, if there are students taking both exams. In practice however some universities generate their schedules for the exams before this registrations takes place to provide a time and date for the exams when students register, e. g., see [2,3,4]. Most often the values for the number of students per exam are taken from the last year as well as the conflicts that were present in the previous year.

In this work we focus on the implications of the uncertainty present in the conflict data. We discuss pitfalls when using an estimation approach for handling this uncertainty and propose new soft criteria that avoid these issues and show promising results on two real world instances of the School of Engineering at the FAU.

2 Model

Most of the following model is similar to the model presented in [7] except that we allow multiple rooms per exam, which already makes the subproblem of assigning rooms to exams in a fixed timeslot NP-complete, which can be shown by a reduction from 3-partition.

Definition 1 *An instance of the Examination Timetabling Problem is represented as follows.*

- $\mathscr{E}$*: A set of exams*
- $\mathscr{R}$*: A set of rooms*
- $\kappa : \mathscr{R} \to \mathbb{N}$*: the capacity available for each room and* $\kappa : \mathscr{P}(\mathscr{R}) \to \mathbb{N}$ *as the canonical extension such that for* $X \subseteq \mathscr{R} : \kappa(X) := \sum_{r \in X} \kappa(r)$
- $\nu \in \mathbb{N}^{\mathscr{E}}$*: A vector specifying for each exam how many students are attending*
- *Conflict matrix* $C \in \mathbb{N}^{\mathscr{E} \times \mathscr{E}}$*: specifying the number of overlapping students of two exams*
- *A set of hard constraints and a set of soft constraints with associated weights*
- *A number of available timeslots*

Note that the vector ν *and the conflict matrix* C *are subject to uncertainty in our case.*

To be feasible, a timetable must meet the following hard constraints:

(H1) each exam is assigned to exactly one timeslot
(H2) each exam is assigned to one or more rooms
(H3) two exams that are in conflict are not scheduled at the same time
(H4) no room is used at the same time by two different exams
(H5) the sum of capacities of the assigned rooms is larger than the number of students taking the exam

In our reduced model we use the soft constraints *two-in-a-row*, *two-in-a-day* and *period-spread* as defined in [7] and the robustness soft constraint introduced in [2]

(S1) *two-in-a-row*: If two exams are in conflict according to C they should not be assigned to two adjacent timeslots on the same day.
(S2) *two-in-a-day*: If two exams are in conflict according to C they should not be assigned to two timeslots on the same day. Note that we exclude the directly adjacent timeslot to avoid double counting.

(S3) *period-spread*: If two exams are in conflict according to C they should not be assigned to timeslots less than λ apart.

(S4) $\min \left\{ \frac{\kappa(RP(e))}{\nu(e)} \;\middle|\; e \in \mathscr{E}, RP(e) \text{ is room pattern of } e \text{ in current timetable} \right\}$

Each violation of a soft criterion induces a penalty corresponding to the number of students that are in conflict given by conflict matrix C. For a feasible timetable, i. e., a timetable that satisfies hard constraints (H1) - (H5), we can formulate the objective value to be minimized as the weighted sum of the penalties induced by the soft constraints (S1) - (S3) and the weighted negative value of soft constraint (S4), as (S1) - (S3) have to be minimized and (S4) maximized.

3 New contribution

Hard constraint (H3) ensures for a feasible timetable, that for each student the chosen exams do not overlap. However in a pre-enrollment setting these choices are not known at scheduling time. It therefore becomes necessary to account for this uncertainty in the scheduling process. We call a conflict *active* iff after registration there is a student that takes both exams, otherwise the conflict is called *inactive*.

For a given major available lectures and therefore exams can usually be classified into two distinct categories. The first group consists of exams a student has to take, often called *mandatory*. Furthermore there might exist a portfolio of choices from which a student has to select a certain amount of lectures and exams respectively, called *elective*.

The strict robustness approach for this setting is to have no distinction between these two categories and enforce conflict freeness, i. e., (H3), for all possible combinations of exams of a major. However this approach leads to large numbers of conflicts per major and therefore no feasible timetable might exist for the model. This is the case for the School of Engineering at FAU.

It becomes necessary to limit the conflicts that are taken into account in the scheduling process. The resulting question is which conflicts to consider and which to ignore. As for *mandatory* exams all students of the major have to take these exams in a specified term and therefore all conflicts involving these exams have to be considered such that all students can take the exam in this term. Consequently there is no uncertainty if such a conflict is active or inactive.

For conflicts between *elective* exams there is uncertainty whether the conflict is active or inactive. The first possible solution for this uncertainty is to use data from previous terms to estimate if a conflict is active and to take all conflicts that are estimated to be active into account in the scheduling process. This was the approach we used in previous works, when talking about the uncertainty in the number of students per exam.

However there are a few inherent problems with this approach. The first problem when using data for estimations, especially when taken from more than one previous term is that almost all conflicts might be active and the model therefore becomes infeasible again. The other problem, which is far more problematic is that the resulting timetables might get biased against certain possible choices. If a conflict is considered

inactive the exams might be scheduled in the same timeslot, therefore no student can attend both exams. In the following year the estimation will again treat this conflict as inactive as no student has taken both exams in the previous term. This problem might not even be visible to the responsible scheduler, however still reducing the acceptance of the calculated timetables by the student body.

Instead of estimating which *elective* conflicts are active and enforcing them via hard constraint (H3) we consider all elective conflicts to be inactive in regard to feasibility. For these conflicts we introduce a new soft criterion to minimize the number of students that cannot take all exams they want.

Given a feasible timetable TT as a set of timeslots we want to minimize the following soft constraint.

$$\sum_{T \in TT} \sum_{\substack{e_1, e_2 \in T \\ e_1 \neq e_2}} \bar{\mathrm{E}}[\#\text{students attending } e_1 \text{ and } e_2] \tag{S5}$$

$\bar{\mathrm{E}}$ is an estimator for how many students induce a given conflict between two exams. Therefore (S5) measures the estimated number of students with a conflict in timetable TT. Using this soft criterion instead of enforcing all estimated conflicts between elective exams through hard constraint (H3) there always exits a feasible timetable.

However this soft criterion will not prevent the issue of bias in regard to the estimations. To also address this issue we formulate a second soft criterion.

$$\sum_{T \in TT} \sum_{\substack{e_1, e_2 \in T \\ e_1 \neq e_2}} \max\{\bar{\mathrm{E}}[\#\text{students attending } e_1 \text{ and } e_2], \mathbb{1}_{\{e_1, e_2\} \text{ is elective conflict}}\} \tag{S6}$$

Instead of using only the sum of estimated students for elective exam conflicts we consider all elective exam conflicts with a value of at least 1. Therefore even if in the last terms no student did choose both exams our optimization will try to schedule the exams conflict free and thus enabling students to choose both.

4 Preliminary Results

To test the performance of the introduced soft criteria and to evaluate the impact on the overall objective value, we used two real world data instances taken from the School of Engineering at FAU. In our experiments we used a simulated annealing algorithm, with a kempe-exchange neighborhood. A more detailed description can be found in [2].

Table 1 and 2 show the arithmetic mean rounded to the nearest integer of 24 runs using the simulated annealing algorithm for the summer term 2018 and the winter term 2017 at the School of Engineering at FAU. The solutions are based on our introduced model using estimations for the number of students attending an exam as described in [1] and the arithmetic mean over the previous years for the elective conflicts and the minimum of the attending students for the mandatory conflicts. Each solution is then evaluated with the actual values of the corresponding term. The values for the soft criteria (S1) - (S3) are shown, with the final value being the number of students that have elective exam conflicts.

Version	(S1)	(S2)	(S3)	#Conflicting
All Conflicts $\neq 0$	70	171	1720	73
All Conflicts $\neq 0$ + (S6)	132	146	1919	20
Only Mandatory + (S5)	78	163	1792	74
Only Mandatory + (S6)	142	141	1937	20

Table 1: Results on the Summer Term 2018 instance of the School of Engineering at FAU.

Version	(S1)	(S2)	(S3)	#Conflicting
All Conflicts $\neq 0$	90	170	1832	109
All Conflicts $\neq 0$ + (S6)	168	169	2112	33
Only Mandatory + (S5)	106	182	1854	101
Only Mandatory + (S6)	173	178	2146	38

Table 2: Results on the Winter Term 2017 instance of the School of Engineering at FAU.

Preliminary results show only a slight decrease in the objective values when using the soft criterion (S5) compared to the solution when considering the estimated elective conflicts as active and enforcing them with hard constraint (H3). For the soft criterion (S6) preliminary results show that we can reduce the number of students that could not choose their intended exams by a factor of around 3. For this soft criterion the objective value shows a larger increase. Partly this increase is a result of the more robust solution, as the soft constraints measure the number of students with conflicting exams in different timeslots, therefore if two exams are in conflict in the same timeslot they do not distribute to the objective value. As such we can argue that the price of robustness is tolarable.

5 Conclusion

In this work we address the issue of uncertainty in the ETTP model in regard to the conflicts, when using a pre-enrollment approach. We discuss pitfalls when using estimations for the conflicts between exams and introduce two soft criteria (S5) and (S6) to address this uncertainty and the resulting issues. We present a case study for two real world instances at FAU and give preliminary results that show only a moderately large increase in the objective value, while providing a large decrease in the number of conflicts.

References

1. Bernd Bassimir and Rolf Wanka. Probabilistic Curriculum-based Examination Timetabling. In *Proc 12th International Conference on the Practice and Theory of Automated Timetabling (PATAT)*, pages 273–285, 2018.

2. Bernd Bassimir and Rolf Wanka. Robustness Approaches for the Examination Timetabling Problem under Data Uncertainty. In *Proc. 9th Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA)*, pages 381–395, 2019.
3. Alejandro Cataldo, Juan-Carlos Ferrer, Jaime Miranda, Pablo A. Rey, and Antoine Sauré. An integer programming approach to curriculum-based examination timetabling. *Annals of Operations Research*, 258(2):369–393, Nov 2017.
4. Peter Demeester, Burak Bilgin, Patrick De Causmaecker, and Greet Vanden Berghe. A hyperheuristic approach to examination timetabling problems: benchmarks and a new problem from practice. *Journal of Scheduling*, 15(1):83–103, Feb 2012.
5. Michael Eley. Ant algorithms for the exam timetabling problem. In *Proceedings of the 6th International Conference on Practice and Theory of Automated Timetabling VI*, PATAT'06, pages 364–382, Berlin, Heidelberg, 2007. Springer-Verlag.
6. Christos Gogos, Panayiotis Alefragis, and Efthymios Housos. An improved multi-staged algorithmic process for the solution of the examination timetabling problem. *Annals of Operations Research*, 194(1):203–221, Apr 2012.
7. Barry McCollum, Paul McMullan, Andrew J. Parkes, Edmund K. Burke, and Rong Qu. A new model for automated examination timetabling. *Annals of Operations Research*, 194(1):291–315, Apr 2012.
8. Tomáš Müller. Itc2007 solver description: A hybrid approach. 172:429–446, 01 2008.

A mixed-integer linear programming algorithm for final exam scheduling

Szilvia Erdős · **Bence Kővári**

Abstract The automatic generation of schedules has been in the focus of researches for decades. Since small changes in the input have an exponential impact on the tested state space, methods based on heuristics, linear programming, and artificial intelligence are the most successful. Final exam scheduling is a special subtask of the generation of schedules, where special requirements restrict the state space. The problem is examined with an integer linear programming approach. A scoring system is elaborated, wherewith the goodness of the generated schedules is measurable and comparable. The algorithm is tested on an actual test set, which contained the registration of 100 students on the finals of bachelor's degrees. The results show that there is an optimal solution for this complexity. With some improvements on the algorithm, there can be solutions, which are better and fairer than the manually compiled schedules.

Keywords Final exam scheduling · Mixed-integer linear programming · Scheduling algorithm · State examination · Examination timetabling · Operations research · Optimization

1 Introduction

The final examination takes place at the end of the course in most universities. One of the most common forms of this exam is the oral examination, where at one time in one room only one student takes the exam in front of a board of examiners. All these instructors have a special role, and some roles have so high requirements that only a few people can fulfil it. The scheduling of the final exams may be done manually, but the non-automated process can cause human errors, and it is hard to see it all, if all the requirements were fulfilled, and if the scheduling is suitable for everyone. The complexity of this problem is based on two levels.

Szilvia Erdős
Budapest University of Technology and Economics
Department of Automation and Applied Informatics
E-mail: Erdos.Szilvia@aut.bme.hu

Bence Kővári
Budapest University of Technology and Economics
Department of Automation and Applied Informatics
E-mail: Kovari@aut.bme.hu

The scheduling problem is NP-complete [8]. Besides, many final-exam specific conditions must be fulfilled, and some of them contradict each other readily. For example, we want to distribute the workload among the examiners equally, but some instructors may have many more students than others. It is also possible that the instructors (who have to be there on a student's exam) are not available at the same time.

The organization of this extended abstract is as follows. Besides Introduction, the extended abstract features six sections. In Section 2, we overview existing research on the need for oral exams and examination timetabling. Section 3 describes the problem statement and shows the challenges of final exam scheduling. Section 4 presents the structure of the proposed approach and builds up the integer linear programming model for final exam scheduling. In Section 5, the results are evaluated on a real-world data set. Section 6 presents the results and success of the algorithm compared with our previous algorithms and algorithms from the literature made for similar problems. Finally, Section 7 includes concluding remarks.

2 Background

Scheduling is a widely researched topic in literature as several fields in our lives need to be scheduled: preparing a timetable, scheduling our agenda, or scheduling working shifts also belongs to this field.

A large area of research is examination timetabling, which includes the scheduling of exams in universities. This field is the closest to the final exam scheduling problem presented in detail in Section 3.

The goal is to allocate a session and a room to every exam to satisfy a given set of constraints in the general problem. The result is a feasible exam timetable. However, each institution will have some unique combination of constraints, as policies differ from institution to institution. Due to this diversity, the constraints of algorithms in the literature are also various.

For example, in the studies of Wijgers and Hoogeveen (2007) [12], the examination days were fixed, and every student could have only one exam per day. The availabilities of instructors were considered but having an equal workload was not in focus. Al-Yakoob, Sherali, and Al-Jazzaf (2010) [1], in their paper, considered availabilities too. Moreover, they also took into account the distance between buildings. However, they did not handle the exceptional cases like specific instructors could be assigned to given exams and instructors could not have unique roles.

Kochaniková and Rudová, in their article from 2013 [9], gave a solution for oral final exam scheduling, where a student and an instructor are assigned to every exam. They dealt with the availabilities and parallel sessions, but the workloads were not in scope. In addition to the shortcomings of the previously mentioned article, the paper of Ivancevic, Knezevic, and Lukovic (2014) [7] did not take into account the availabilities. However, the instructors were scheduled for whole blocks, and parallel exams were also allowed. Bergmann, Fischer, and Zurheide (2014) [3] also considered the parallelization and workloads too, but instructors could not have particular roles. Aslan, Şîmşek, and Karkacier in 2017 [2] presented an algorithm with equal workloads and collision prevention, but the availabilities of the people were not in scope.

As can be seen from the previous examples, although the same problem is being addressed, the requirements taken into account can vary widely depending on the researchers' priorities.

3 Problem statement

This section shows how the final exam scheduling builds up. A small example is shown in Figure 1, where the schedule of a day is shown.

Day1	Student	Supervisor	Chair	Secretary	Member	Examiner
1	Levente	Hideg Attila	Vajk István	Hideg Attila	Dudás Ákos	Dudás Ákos
2	Tímea	Dudás Ákos	Vajk István	Hideg Attila	Dudás Ákos	Benedek Zoltán
3	Kata	Ekler Péter	Vajk István	Hideg Attila	Ekler Péter	Benedek Zoltán
4	Bence	Kővári Bence András	Vajk István	Hideg Attila	Kővári Bence András	Benedek Zoltán
5	Bence Zsigmond	Ekler Péter	Vajk István	Hideg Attila	Kővári Bence András	Kővári Bence András
6	Péter Szabolcs	Forstner Bertalan	Forstner Bertalan	Pomázi Krisztián	Kővári Bence András	Kővári Bence András
7	Márton	Tóth Tibor	Forstner Bertalan	Pomázi Krisztián	Kővári Bence András	Kővári Bence András
8	Attila	Szabó Gábor	Forstner Bertalan	Pomázi Krisztián	Asztalos Márk	Goldschmidt Balázs
9	Dániel Gábor	Mezei Gergely	Forstner Bertalan	Pomázi Krisztián	Kővári Bence András	Kővári Bence András
10	Gergő	Hamar János Krisztián	Forstner Bertalan	Pomázi Krisztián	Csorba Kristóf	Csorba Kristóf

Fig. 1 Example scheduling for one day

An examination period consists of timeslots (T is the set of all timeslots, $t \in T$ is one of the timeslots). The number of timeslots is equal to the number of students who must take the exam in that semester. The set of students is S. In each timeslot, there is one final exam named e. All exams together compound the whole scheduling named E.

A block is a homogeneous group of students per morning or afternoon, namely half a day, where students are from the same faculty. The set of all blocks is B, and one block is $b \in B$.

The participants and the relationships between them are shown in the Figure 2.

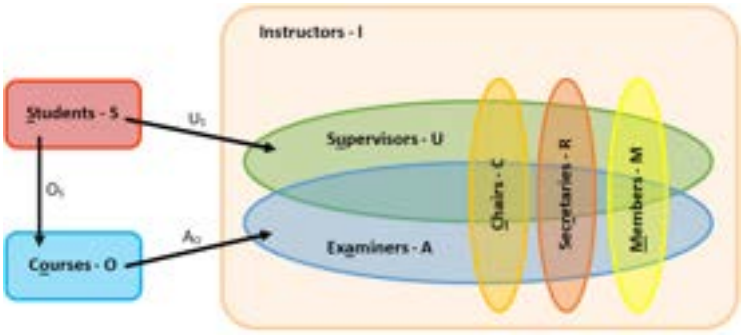


Fig. 2 Participants and their relationships

The instructors are a separate group named I. Several specific subsets can be identified within I:

- The set of supervisors is named $U \subseteq I$. Every $s \in S$ has a specialized $u \in U$, thus $\{s \in S\} \rightarrow \{u \in U\}$. A supervisor may be assigned to several students: $\{u \in U\} \rightarrow \{s_1, s_2, \ldots, s_n \mid \forall\, s_i \in S\}$. Afterwards u_s means the supervisor of student s.
- The group of chairs is named $C \subseteq I$. The regulations of the given final exams specify who can fill this role.
- The group of secretaries is named $R \subseteq I$. They are the rapporteurs of the exams.

– The group of inner members is named $M \subseteq I$. They are the inner instructors of the department where the exam takes place.

There is also a resource group, which consists of courses (noted with O). Every student has to choose a course, thus s student has an exam in course o_s. For each course, it is predefined who can sit the exam for that course. It is defined for every $o \in O$ who can examine from that course. These instructors are examiners named $A_o \subseteq I$, so for every $\{o \in O\} \rightarrow \{a_1, a_2, \ldots, a_n \mid \forall\, a_i \in A_o\}$. Afterwards, $\bigcup_{o \in O} A_o = A$

An exam stands up, as shown in equation 1.

$$e \in \{t, s, u, c, r, m, a \mid t \in T,\ s \in S, u \in U, c \in C, r \in R, m \in M, a \in A_o, o \in O_s\} \tag{1}$$

4 The proposed mixed-integer linear programming model

This section discusses how the mixed-integer linear programming model builds up for this unique timetabling problem – for the final exam scheduling.

Integer linear programming is a well-known method for optimization problems[4]. The standard form of it can be seen henceforth: there are some integer decision variables, which values are searched, while the linear objective function is optimized, and linear equality and inequality constraints are subjected.

4.1 Decision variables

By choosing the variables, they must be as suitable as they can to the actual problem. Besides, there should be no unnecessary variables to avoid redundant calculations.

There are two primary variables, and both are binary. One is for instructors, and the other represents the students. Both variables have two dimensions: people and timeslots. If the value of $i \in I$ instructor in timeslot $t \in T$ is 0, then instructor i is not scheduled in t timeslot, and if this value is 1, then i is scheduled in t. The same holds for students. The decision variables of instructors are $x_{i,\,t} \in \{0,1\} \mid \forall i \in I,\ \forall t \in T$, and the variables of students are $x_{s,\,t} \in \{0,1\} \mid \forall s \in S,\ \forall t \in T$.

The instructors, who have special roles, are noted as follows. The instructor who can be a chair on an exam is shown as $x_{c,\,t}$, which is identical to the variable $x_{i,\,t}$, where $i \in C$ holds.

Some additional decision variables are necessary for the fulfilment of certain constraints. These were calculated based on the primary decision variables.

A hard requirement of the final exam scheduling is that the chairs and the secretaries must be scheduled in whole blocks. Additional binary variables were introduced that present the scheduling blocks of chairs and secretaries. These are shown in equation 2.

$$\begin{aligned} x_{c,\,b} &\in \{0,1\} \ \ \forall c \in C,\ \forall b \in B \\ x_{r,\,b} &\in \{0,1\} \ \ \forall r \in R,\ \forall b \in B \end{aligned} \tag{2}$$

Furthermore, some variables were constructed for optimizing the workload of instructors. The main point of this is to determine the difference between the optimal and the actual workloads. However, in linear programming, there is no simple way to express the absolute value of two decision variables. A method is to introduce two additional variables for one person. The workload of chairs, secretaries, and members is optimized by applying the variables like in formula set 3.

$$\begin{array}{llll} x_c^\alpha \in \mathbb{Z}^+ & \forall c \in C & x_r^\alpha \in \mathbb{Z}^+ & \forall r \in R \quad x_m^\alpha \in \mathbb{Z}^+ \quad \forall m \in M \\ x_c^\beta \in \mathbb{Z}^+ & \forall c \in C & x_r^\beta \in \mathbb{Z}^+ & \forall r \in R \quad x_m^\beta \in \mathbb{Z}^+ \quad \forall m \in M \end{array} \tag{3}$$

4.2 Objective function

Final exam scheduling has to fulfil some soft requirements (like instructors' optimal workload or availabilities). These requirements are primarily defined in the objective function, which looks like the expression 4.

$$min \sum_{i}^{i \in I} \sum_{t}^{t \in T} (x_{i,t} * Cost_{i,t}) + \sum_{c}^{c \in C} \left(x_c^\alpha + x_c^\beta \right) + \sum_{r}^{r \in R} \left(x_r^\alpha + x_r^\beta \right) + \sum_{m}^{m \in M} \left(x_m^\alpha + x_m^\beta \right) \tag{4}$$

The first part is for the availabilities of instructors where $Cost_{i,t}$ is a positive integer constant, which belongs to the penalty point of instructor $i \in I$ if he or she is not available in timeslot $t \in T$.

The rest of the objective function belongs to the requirements of having equal workloads for chairs, secretaries, and members. Here the derived decision variables of each instructor in these roles are summarized, where for example, by the chairs x_c^α means the difference from the optimal workload in the positive direction (how much more than optional scheduled exams the chair c has), and x_c^β means the difference in the negative direction. Of course, at least one of these two variables have to be null.

4.3 Problem constraints

The scheduling has to fulfil many different requirements, and most of them are defined in various ways as linear equality and inequality constraints. There are three types of problem constraints which are defined below.

4.3.1 Constraints for derived decision variables

The first type of constraint refers to the derived decision variables. This formula set contains the binary variables that present the scheduling blocks of chairs (shown in equation 5) and secretaries (shown in equation 6). These variables have the value of 1 only if the decision variables in every timeslot in that given block are 1 – this means that the instructor is scheduled in the whole block. The condition was formulated using "logical and" operations, taking advantage of the solver's capabilities. (Gurobi was used in the modelling, but many other solvers can map these operations to MILP conditions.)

$$x_{c,b} = \bigwedge_{t \in b} x_{c,t} \qquad \forall c \in C \, , \, \forall b \in B \tag{5}$$

$$x_{r,b} = \bigwedge_{t \in b} x_{r,t} \qquad \forall r \in R \, , \, \forall b \in B \tag{6}$$

The other derived variables are for optimizing the workload of instructors. In the case of chairs, formula 7 should be minimized , where D_c is the optimal value of the workload of a chair.

$$\left| \sum_{b}^{b \in B} x_{c,b} - D_c \right| \tag{7}$$

However, as stated before, the absolute value could not be placed in the objective function, so instead of it, there should be two derived variables x_c^{α} and x_c^{β}, subject to equation 8.

$$x_c^{\alpha} - x_c^{\beta} = \sum_{b}^{b \in B} x_{c,b} - D_c \qquad \forall c \in C \tag{8}$$

Furthermore, in the objective function there is $x_p^{\alpha} + x_p^{\beta}$ instead of the absolute value 7.

The constrains for secretaries (9) and members (10) are similar. D_r is the optimal workload of secretaries and D_m refers to the optimal workload of members.

$$x_r^{\alpha} - x_r^{\beta} = \sum_{b}^{b \in B} x_{r,b} - D_r \qquad \forall r \in R \tag{9}$$

$$x_m^{\alpha} - x_m^{\beta} = \sum_{t}^{t \in T} x_{m,t} - D_m \qquad \forall m \in M \tag{10}$$

4.3.2 Constraints refer to the basis of scheduling

Many constraints are necessary for getting an adequate final exam scheduling. The list of them can be found in Table 1.

Table 1: Constraints for the fundamental contribution of scheduling

$\sum_i x_{i,t} \leq 5 \quad i \in I, \forall t \in T \qquad (11)$	There should be a maximum of 5 instructors in a timeslot.
$\sum_c x_{c,t} = 1 \quad c \in C, \forall t \in T \qquad (12)$	There should be one instructor at each timeslot who can be the chair.
$\sum_r x_{r,t} = 1 \quad r \in R, \forall t \in T \qquad (13)$	There should be one instructor at each timeslot who can be the secretary.
$\sum_m x_{m,t} \geq 1 \quad m \in M, \forall t \in T \qquad (14)$	There should be at least one instructor at each timeslot who can be the member. There are some situations when there must be more instructors in an exam who could be the member (e.g. the supervisor and the examiner could also be members and different people). That is the reason for the minimum criteria.
$\sum_m x_{m,t} \leq 2 \quad m \in M, \forall t \in T \qquad (15)$	There should be a maximum of two instructors in a timeslot who could be in the role of member.

$\sum_s x_{s,t} = 1 \quad s \in S,\ \forall t \in T$	(16)	There should be precisely one student in every timeslot.
$\sum_t x_{s,t} = 1 \quad t \in T,\ \forall s \in S$	(17)	Every student should be in exactly one timeslot.
$x_{s,t} - x_{u_s,t} \leq 0 \quad \forall t \in T\ ,\ \forall s \in S$	(18)	The supervisor of the student should be there on the exam when the student has the examination.
$x_{s,t} - \sum x_{a_{c_s},t} \leq 0 \quad \forall t \in T\ ,\ \forall s \in S$	(19)	An examiner from the student's course should be there on the exam when the student has the examination.

4.3.3 Constraints for further requirements

The model has to fulfil all the hard requirements which were defined before. These are listed in Table 2.

Table 2: Constraints for further hard requirements

$\sum_c x_{c,b} = 1 \quad c \in C,\ \forall b \in B$	(20)	There should be precisely one chair in every block.
$\sum_r x_{r,b} = 1 \quad r \in R,\ \forall b \in B$	(21)	There should be precisely one secretary in every block.
$\sum_c^{c \in C} \sum_t^{t \in T} (x_{c,t} * Cost_{c,t}) = 0$	(22)	Chairs should be scheduled if only if they are available. ($Cost_{c,t}$ is the penalty score for non-availability.)
$\sum_r^{r \in R} \sum_t^{t \in T} (x_{r,t} * Cost_{r,t}) = 0$	(23)	Secretaries should be scheduled if only if they are available. ($Cost_{r,t}$ is the penalty score for non-availability.)
$\sum_b^{b \in B} x_{c,b} \geq D_c^- \quad \forall c \in C$ $\sum_b^{b \in B} x_{c,b} \leq D_c^+ \quad \forall c \in C$	(24)	The workload of chairs should be held between limits. (D_c^- is the absolute minimum value of the workload of chairs, D_c^+ is the maximum.)
$\sum_b^{b \in B} x_{r,b} \geq D_r^- \quad \forall r \in R$ $\sum_b^{b \in B} x_{r,b} \leq D_r^+ \quad \forall r \in R$	(25)	The workload of secretaries should be held between limits.
$\sum_t^{t \in T} x_{m,t} \geq D_m^- \quad \forall m \in M$ $\sum_t^{t \in T} x_{m,t} \leq D_m^+ \quad \forall m \in M$	(26)	The workload of members should be held between limits.

5 Case study

The algorithm was tested on an actual test set, acquired from the Budapest University of Technology and Economics (BME), Department of Automation and Applied Informatics. The official regulation of BME is available online at [10] and [11]. The test set contains 100 students, 49 instructors and 12 courses. Just for these 100 students, the number of possible schedules is about 10462, according to equation 27.

$$100! \times 4^{100} \times 9^{100} \times 10^{100} \times 3^{100} \approx 10^{462} \tag{27}$$

where 100 is the number of timeslots, 4 is the number of chairs, 9 is the number of secretaries, 10 is the number of members, and 3 is the average number of instructors for a course.

As noted before, the Gurobi solver was used to solve the model.

There were 15106 decision variables, 260 constraints, and it runs for $1,3$ seconds on 8 threads. The minimized objective value was 40.

We analysed the solution by hand, and we found out that members and secretaries did not fulfil only requirements for equal workloads. The only reason for this was that these instructors have many more students than the others, and they should be there on more exams as supervisors.

6 Achieved results

We have made a scoring system in advance for requirements, which is practical for measuring the schedule's goodness. Furthermore, the schedules could be compared to each other based on the penalty points. We have made two different algorithms before for the final exam scheduling, a genetic algorithm-based [6] and a heuristic approach based on pair graphs and the Hungarian method [5].

First, we compared the results of MILP with our own earlier algorithms. The outcomes are illustrated in Figure 3. As it can be seen, the MILP-based algorithm was the best in every sight. MILP scores more than ten times better than the previous best-performing heuristic algorithm and performs better orders in runtime under similar conditions.

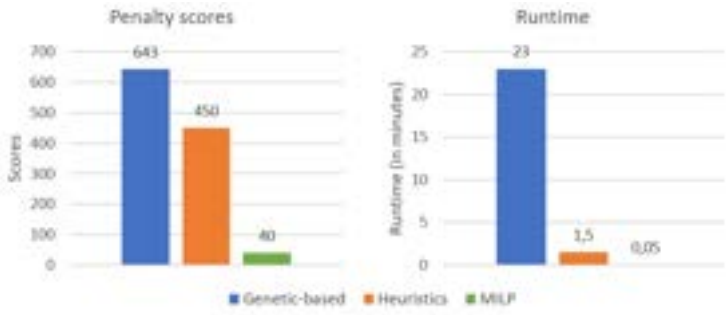


Fig. 3 The comparison of own scheduling algorithms

MILP models are often worth examining from a scalability point of view. In final exam scheduling, 100 students are the average number of students per student group. There may be some increase for popular courses (such as computer science, for which the test case was made). However, given the university's capacity, the maximum number of students per teaching base is 150. Taking this case into account, the performance of the algorithm was tested for both 125 and 150 students, and a fictitious set of 175 and 200 students were generated, which far exceeds the real numbers. The results of this are shown in Figure 4, which shows that for 125 students, there is little difference, while above 150 students, the runtime starts to increase. However, it can be seen that for a set of 200 students, the running time is significantly better than for any other algorithm on a set of 100 students.

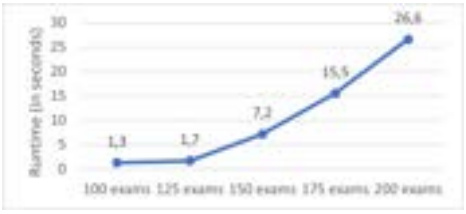


Fig. 4 Testing the scalability of the algorithm

Furthermore, the results are compared to some algorithms discussed in the literature (close to our problem's statement) based on comprehensive properties. The results are depicted in the Table 3, where ✓ means that this requirement is considered and fulfilled, and ✗ otherwise. ✓/✗ means that it is invented but not implemented yet.

Table 3 Algorithms compared along with their abilities

	requirements not equally important	equal workload of instructors	availabilities of people	block-based structure	parallel timeslots and collision handling	specific roles of instructors
own MILP-based algorithm	✓	✓	✓	✓	✓/✗	✓
Wijgers, Hoogeveen [12] (2007)	✗	✗	✓	✓	✗	✗
Al-Yakoob, Sherali, Al-Jazzaf [1] (2010)	✓	✓	✓	✗	✓	✗
Kochaniková, Rudová [9] (2013)	✓	✗	✓	✗	✓	✓
Bergmann, Fischer, Zurheide [3] (2014)	✓	✓	✓	✗	✓	✗
Ivancevic, Knezevic, Lukovic [7] (2014)	✗	✗	✗	✓	✓	✓
Aslan, Şîmşek, Karkacier [2] (2017)	✗	✓	✗	✗	✓	✗

According to these, it can be said that our algorithm gives a more comprehensive solution for more questions. It covers more abilities of final exam scheduling than the algorithms discussed before in the literature for similar problems.

7 Conclusion

Despite the popularity of scheduling, the topic of the final exam scheduling provides another exciting problem because it cannot be created in a ”normal” scheduling design methodology. It needs comprehensive solutions.

A model based on linear programming has been developed, which considers more aspects at the same time than similar solutions discussed in the literature. It also far outperforms algorithms previously developed for this problem.

With some improvements to our algorithm (like introducing parallel exams), there can be solutions, which are better and fairer than the manually compiled schedules.

Acknowledgements Project no. FIEK_16 – 1 – 2016 – 0007 has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the Centre for Higher Education and Industrial Cooperation - Research infrastructure development (FIEK_16) funding scheme.

References

1. Al-Yakoob, S., Sherali, H., Al-Jazzaf, M.: A mixed-integer mathematical modeling approach to exam timetabling. Computational Management Science **7**, 19–46 (2010). DOI 10.1007/s10287-007-0066-8
2. ASLAN, E., ŞİMŞEK, T., KARKACIER, A.: A binary integer programming model for exam scheduling problem with several departments. Bilgi Ekonomisi ve Yönetimi Dergisi **12**(2), 169–175 (2017)
3. Bergmann, L.K., Fischer, K., Zurheide, S.: A linear mixed-integer model for realistic examination timetabling problems. In: Proceedings of the 10th International Conference on the Practice and Theory of Automated Timetabling, pp. 82–101 (2014)
4. Dantzig, G.: Linear programming and extensions. Princeton university press (2016)
5. Erdos, S.: Algorithm based on Hungarian Method for Final Exam Scheduling. In: Automation and Applied Computer Science Workshop (2019)
6. Erdos, S., Kovari, B.: Genetic algorithm based solution for final exam scheduling. In: MultiScience - microCAD International Multidisciplinary Scientific Conference (2019)
7. Ivančević, V., Knežević, M., Luković, I.: A course exam scheduling approach based on data mining. Smart Digital Futures 2014 **262**, 132 (2014)
8. Kirkpatrick, D.G., Hell, P.: On the completeness of a generalized matching problem. In: Proceedings of the Tenth Annual ACM Symposium on Theory of Computing, STOC ’78, p. 240–245. Association for Computing Machinery, New York, NY, USA (1978). DOI 10.1145/800133.804353. URL https://doi.org/10.1145/800133.804353
9. Kochaniková, B., Rudová, H., et al.: Student scheduling for bachelor state examinations. In: Proceedings of the 6th Multidisciplinary International Scheduling Conference–MISTA, pp. 762–766. Citeseer (2013)
10. Rektori Kabinet Oktatási Igazgatóság: A Szenátus X./10./2015-2016. (2016. VII. 11.) számú határozata A BME TANULMÁNYI ÉS VIZSGASZABÁLYZATÁRÓL. http://www.kth.bme.hu/document/2061/original/BME (2016)
11. Sujbert, L.: BME VIK BSc szakdolgozat, záróvizsga, oklevél szabályzat. https://www.vik.bme.hu/document/1343 (2017)
12. Wijgers, R., Hoogeveen, J.: Solving the examination timetabling problem (2007)

Timetabling Round Robin Tournaments with the Consideration of Rest Durations

Tasbih Tuffaha · Burak Çavdaroğlu · Tankut Atan

Abstract Rest durations of opposing teams have recently emerged as a new fairness criterion for the timetabling of sports leagues. A rest difference is the difference between the rest durations of the opposing teams of each game. A problem, so-called rest difference problem, simultaneously schedules the games to the rounds and assigns the games of each round to the matchdays in order to minimize the total rest difference throughout a round robin tournament. In this study, we provide a mixed integer programming (MIP) formulation of the problem and propose a heuristic method which outperforms the results of the MIP on several problem instances.

Keywords sports timetabling · tournament fairness · rest differences · circle method · Vizing method

1 Introduction

Timetabling of round robin tournaments with respect to various fairness criteria is one of the most popular research topics in sports scheduling. The related literature has mainly focused on fairness issues such as balancing the carry-over effect (e.g. Russell (1980), Anderson (1997), Guedes and Ribeiro (2011))

T. Tuffaha
Department of Industrial Engineering, Kadir Has University, Cibali, İstanbul, Turkey, 34083
Tel.: +90-212-5336532
Fax: +90-212-5336515
E-mail: 20161107007@stu.khas.edu.tr

B. Çavdaroğlu
Department of Industrial Engineering, Kadir Has University, Cibali, İstanbul, Turkey, 34083
E-mail: burak.cavdaroglu@khas.edu.tr

T. Atan
Department of Industrial Engineering, Bahçeşehir University, Beşiktaş, İstanbul, Turkey, 34353 E-mail: sabritankut.atan@eng.bau.edu.tr

or the opponent's strength in a series of consecutive matches (e.g. Briskorn (2009), Briskorn and Knust (2010), Zeng and Mizuno (2013)), and minimizing the total number of breaks (e.g. de Werra (1981), Elf et al. (2003), Miyashiro and Matsui (2005), van't Hof et al. (2010)). However, fairness criteria regarding the rest durations between the consecutive games have not received much attention by researchers. In this study, we consider such a fairness criterion which aims to minimize the total number of rest differences between the opposing teams of each game in a compact round robin tournament. A *round* is composed of a set of games in which every team plays at most one game. In *compact round robin tournaments*, games are scheduled in a minimum number of rounds necessary for finishing all the games so that each team plays exactly one game in each round. Since each round usually consists of several days in practice, tournament organizers need to determine the *matchday* of each individual game in a compact round robin tournament. The problem, so-called the *rest difference problem (RDP)*, constructs a *timetable* which determines both the round and matchday of each game such that the total rest difference (the difference between the rest durations of two opposing teams in a game) throughout the tournament is minimized.

We now provide an illustrative example to describe the rest difference problem. In the example, we consider a single round robin (SSR) tournament in which each of $n = 10$ teams plays against each other exactly once. The tournament is composed of 9 rounds and each team plays exactly one game in each round (since the SSR tournament is assumed to be *compact*). The total number of games is $n(n-1)/2 = 45$. Assuming that each round is spread into 3 consecutive matchdays, the next round immediately starts the next day after the last matchday of the previous round. As a result, the tournament lasts for 27 days. Table 1 provides a feasible timetable for this example. The number of games distributed to first, second and third matchdays are selected to be 2, 2 and 1, respectively. Table 2 illustrates the rest durations of opposing teams in each game. One can observe that the opposing teams do not rest equal number of days in most of the games of this timetable. For example, before the game 13 (G13) between team 6 and team 2 in round 3, team 6 and team 2 play their games of previous round in the third and first matchdays, respectively. Therefore, team 6 has two days before its game in round 3, while team 2 has four days, which is 2 days more than the rest duration of team 6. The rest difference of 2 days between the teams in this game is considered as an unfairness that weighs against team 6 (or favors team 2). When we sum the rest differences in all games of Table 2 , the total rest difference in this tournament is found to be 38.

There exist a few studies regarding the relative rest durations of the opposing teams. Suksompong (2016) investigates three different fairness criteria, guaranteed rest time, games played difference index, and rest difference index, in asychronous round robin tournaments. Asychronous tournament is a special case of round robin tournaments in which each game is played at a distinct consecutive time (e.g. matchday). In particular, rest difference index bears a resemblance to the objective function of rest difference problem. The rest dif-

Table 1 A feasible timetable of RDP(10,3)

	First Matchday				Second Matchday				Third Matchday	
Round 1	**G1:**	1 vs 10	**G2:**	2 vs 9	**G3:**	3 vs 8	**G4:**	4 vs 7	**G5:**	5 vs 6
Round 2	**G6:**	10 vs 2	**G7:**	7 vs 5	**G8:**	8 vs 4	**G9:**	9 vs 3	**G10:**	1 vs 6
Round 3	**G11:**	1 vs 4	**G12:**	5 vs 3	**G13:**	6 vs 2	**G14:**	7 vs 10	**G15:**	8 vs 9
Round 4	**G16:**	2 vs 3	**G17:**	8 vs 6	**G81:**	9 vs 5	**G19:**	10 vs 4	**G20:**	1 vs 7
Round 5	**G21:**	4 vs 5	**G22:**	10 vs 8	**G23:**	2 vs 7	**G24:**	3 vs 6	**G25:**	1 vs 9
Round 6	**G26:**	1 vs 5	**G27:**	6 vs 4	**G28:**	7 vs 3	**G29:**	8 vs 2	**G30:**	9 vs 10
Round 7	**G31:**	1 vs 8	**G32:**	9 vs 7	**G33:**	10 vs 6	**G34:**	2 vs 5	**G35:**	3 vs 4
Round 8	**G36:**	4 vs 2	**G37:**	1 vs 3	**G38:**	5 vs 10	**G39:**	6 vs 9	**G40:**	7 vs 8
Round 9	**G41:**	1 vs 2	**G42:**	3 vs 10	**G43:**	4 vs 9	**G44:**	5 vs 8	**G45:**	6 vs 7

Table 2 Rest durations (in days) for the timetable given in Table 2

	First Matchday				Second Matchday				Third Matchday	
Round 1	**G1:**	–	**G2:**	–	**G3:**	–	**G4:**	–	**G5:**	–
Round 2	**G6:**	3 vs 3	**G7:**	2 vs 1	**G8:**	3 vs 3	**G9:**	4 vs 3	**G10:**	5 vs 3
Round 3	**G11:**	1 vs 2	**G12:**	3 vs 2	**G13:**	2 vs 4	**G14:**	4 vs 4	**G15:**	4 vs 4
Round 4	**G16:**	2 vs 3	**G17:**	1 vs 2	**G18:**	2 vs 4	**G19:**	3 vs 4	**G20:**	5 vs 4
Round 5	**G21:**	2 vs 2	**G22:**	2 vs 3	**G23:**	4 vs 2	**G24:**	4 vs 4	**G25:**	3 vs 4
Round 6	**G26:**	1 vs 3	**G27:**	2 vs 3	**G28:**	3 vs 3	**G29:**	4 vs 3	**G30:**	3 vs 5
Round 7	**G31:**	3 vs 2	**G32:**	1 vs 2	**G33:**	2 vs 4	**G34:**	3 vs 4	**G35:**	4 vs 5
Round 8	**G36:**	1 vs 2	**G37:**	3 vs 1	**G38:**	3 vs 3	**G39:**	3 vs 4	**G40:**	5 vs 5
Round 9	**G41:**	3 vs 3	**G42:**	3 vs 2	**G43:**	4 vs 3	**G44:**	3 vs 2	**G45:**	4 vs 3

ference index defined in the aforementioned study is equal to the maximum difference in rest durations of opposing teams among all games of a timetable, while the objective function of rest difference problem is the sum of rest differences in all games of a timetable. The study also shows that the lower bound for the rest difference index is 1, and a timetable with $n \geq 6$ teams constructed by the circle method always has the rest difference index value of 2.

Atan and Çavdaroğlu (2018) is another study concerning the rest durations of the opposing teams comparatively. In this study, they first define a fairness criterion called *rest mismatch* as the occurrence of a difference between the rest durations of two opposing teams in a game. It should be noted that a rest mismatch does not consider the magnitude of the difference in the rest durations of opposing teams. Next, they construct a timetable with both round and matchday assignments that aims to minimize the total number of rest mismatches in the tournament. The heuristic proposed in the study finds optimal results but only works for rest mismatch problems where the number of matchdays is restricted to 2 and the number of teams is a multiple of 8 (It finds near optimal results if the number of teams is a multiple of 4 but not 8).

Last but not least, Çavdaroğlu and Atan (2020) investigates the rest difference problem for given *opponent schedules*, i.e. schedules in which games have already been assigned to rounds. The study shows that the rest difference problem of a given schedule is decomposable into optimizing the rounds separately, and that each decomposed problem is an instance of the quadratic assignment problem. It also provides a polynomial-time exact algorithm for opponent schedules constructed by the circle method.

The organization of the rest of the paper is as follows. Section 2 first formally describes the problem of minimizing the sum of rest differences of teams by determining the round and matchday of each game. In Section 3, a heuristic method that decides the round and matchday of each game is described. The experiments and the comparison of the performance of the heuristic method with that of mixed integer programming formulation are also given in this section. Section 4 concludes the paper.

2 A Formal Description of the Problem

Suppose that there is a single round robin tournament (SRRT) for teams $i \in T = \{1, \ldots, n\}$ where n is even. In each round $r \in R = \{1, \ldots, n-1\}$, the games of that round have to be assigned to consecutive matchdays $d \in D = \{1, \ldots, p\}$. The *Rest Difference Problem* with n teams and p matchdays is prescribed as RDP(n, p). In RDP(n, p), we assume that games $g \in G = \{1, \ldots, n(n-1)/2\}$ are allocated to p matchdays as evenly as possible in each round. If an allocation with equal number of games in each matchday is not possible, then the numbers of games in the matchdays are assumed to be in descending order. For example, if $p = 3$ in an SRRT with $n = 10$ teams, the number of games in the matchdays should be $(2, 2, 1)$. Let $play_{g,i}$ get the value of 1 if team i plays in game g, 0 otherwise. The number of games to be played in matchday d of each round is denoted by $nGames_d$. We let M be a large positive number at least equal to $p-1$, the maximum possible difference in rest periods between two teams. The binary variable $x_{g,r,d}$ decides if game g is played in matchday d of round r or not. The binary variable $y_{g,r}$ represents the decision of whether game g is assigned to round r or not. The rest difference variable p_g^1 (p_g^2) denotes the number of days the first (second) team in game g had less rest than its opponent. Unless the opposing teams of game g rest for equal amount of time after their games in the previous round, a difference in rest durations occurs and either p_g^1 or p_g^2 gets a positive value. The following mixed integer program (`MIP1`) formulates RDP(n, p).

$$min \quad z = \sum_{g \in G} p_g^1 + p_g^2 \tag{1}$$

subject to:

$$\sum_{r \in R} \sum_{d \in D} x_{g,r,d} = 1 \quad \forall g \in G \tag{2}$$

$$\sum_{d \in D} x_{g,r,d} = 1 \quad \forall \{g \in G, i \in T : play_{g,i} = 1\}, \forall r \in R \tag{3}$$

$$\sum_{g \in G} x_{g,r,d} = nGames_d \quad \forall r \in R, \forall d \in D \tag{4}$$

$$\sum_{d \in D} x_{g,r,d} = y_{g,r} \quad \forall g \in G, \forall r \in R \tag{5}$$

$$\sum_{d \in D} d \cdot x_{g',r-1,d} - \sum_{d \in D} d \cdot x_{g'',r-1,d} + p_g^2 - p_g^1 \leq M \cdot (1 - y_{g,r})$$
$$\forall \{i, j \in T, g, g^{'}, g^{''} \in G : play_{g,i} = play_{g,j} = play_{g^{'},i} = play_{g^{''},j} = 1\}, \forall r \in R \setminus \{1\} \tag{6}$$

$$x_{g,r,d} \in \{0,1\} \quad \forall g \in G, \forall r \in R, \forall p \in P \tag{7}$$

$$y_{g,r} \in \{0,1\} \quad \forall g \in G, \forall r \in R \tag{8}$$

$$p_{g,r}^1, p_{g,r}^2 \geq 0 \quad \forall g \in G \tag{9}$$

The problem's objective is to generate a timetable that minimizes the total rest difference among the opposing teams of all $n(n-1)/2$ games in the tournament. Constraint 2 assigns each game to exactly one matchday in the timetable. Constraint 3 makes sure that each team plays exactly once in each round. Constraint 4 sets the number of games to be played in each matchday. The number of games in each matchday d is determined a priori with $nGames_d$. Constraint 5 determines in which round a game was set to be played by the model. Constraint 6 checks for the first and second team of each game g in each round, identifies the matchdays of the games g' and g'' played by these two teams in the previous round, and finds the time difference between these matchdays. This difference is equal to the rest difference of the opposing teams in game g. If the time difference is in favor of the first (second) team, then p_g^2 (p_g^1) gets a positive value while p_g^1 (p_g^2) is forced to be zero. Constraint 7 through Constraint 9 give the types of decision variables.

Solving `MIP1` with commercial solvers cannot return feasible solutions for some RDP(n,p) instances within a reasonable amount of time particularly when n and p values are increased (refer to Section 3 for more details). Even though the computational complexity of RDP has not been proven yet, we conjecture that the problem is NP-hard.

3 A Heuristic Method and Experimental Results

In this section, we propose a heuristic method that can be applied to the RDP(n,p) where n is the (even) number of teams, and p is the number of matchdays. The heuristic method is conducted in two steps. First, we generate an initial opponent schedule for an SRRT with n teams using the well-known circle method. The circle methods is popular in sports scheduling particularly because it minimizes the number of breaks (the occurrence of consecutive home or away games) in round robin tournaments. (More details about how the circle method is applied to construct league schedules can be found in Çavdaroğlu and Atan (2020)). In this initial schedule, the games of each round are identified without assigning the games into matchdays. Second, the rounds of the

initial schedule are randomized and then the matchdays are determined for each randomized schedule using a mixed integer programming model which minimizes the total rest differences for a given opponent schedule. This MIP model is a modified version of `MIP1` presented in Section 2. The formulation of the modified MIP model (`MIP2`) can be found in Çavdaroğlu and Atan (2020). It is developed to solve the RDP(n, p) in which the opponent schedule (i.e. round assignment of games) is given a priori. Using `MIP2`, matchday assignments are found for each randomly generated schedule and the timetable (i.e. opponent schedule with matchday assignments) having the lowest total rest difference value is selected as the best solution found.

The same heuristic is also applied for a set of opponent schedules generated using a procedure known as *edge coloring* or *Vizing method* (Januario et al. (2016), Januario and Urrutia (2012)). Vizing method presents a framework for the construction of an arbitrary edge coloring of a complete graph K_n with $n-1$ colors where each one of n teams corresponds to a vertex, each game between teams i and j to an edge (i, j) of K_n, and each color to a distinct round. Thus, edges with the same color are the games played during the same round, and each arbitrary edge coloring of a complete graph K_n represents an opponent schedule for SRRT with n teams.

In Table 3, the first column provides the problem instances we considered in our experimental analysis. These instances span all RDP(n, p) instances having $n = 16, 18, 20$ teams and p matchdays ranging from 2 to $n/2$. As mentioned earlier in the formal description, we assume the games are allocated to matchdays as evenly as possible. This allocation for each instance is shown in the second column.

The computer runs were executed on an Intel Core i7-7600U CPU 2.9 GHz computer with 8GB of RAM. `MIP1` solutions are given in the third column along with their run times. They were obtained with GAMS using either Gurobi (Gurobi Optimization Inc., 2019) or CPLEX (IBM, 2019) solver. Note that these instances were solved by both solvers but only the best solutions were reported. For `MIP1` solutions, a time limit of 10 hours (36,000 seconds) was used. In all instances, we ran the solver by the end of the time limit, and reported the best solutions found. It can be noted that in some instances `MIP1` could not even find a feasible solution within the time limit. The results also show that `MIP1` performs poorer with increasing values of n and p since both the number of decision variables and the number of constraints are strictly increasing functions of n and p. Moreover, in none of the instances did `MIP1` find a lower bound better than 0.

To obtain timetables with total rest difference values better than that of the `MIP1` solutions, we generate $\lambda_{cm} = 1000$ random permutations of rounds of the initial opponent schedule that is constructed using the circle method. We also generate $\lambda_{vm} = 1000$ arbitrary edge colorings using the Vizing method. After solving `MIP2` model for each one of λ_{cm} (λ_{vm}) schedules of RDP(n, p) and selecting the schedule with the lowest total rest difference value, the heuristic using the circle method (the Vizing method) produces the results given in the

fourth (fifth) column of Table 3. For each given schedule of RDP(n, p), `MIP2` finds the optimal result in less than 10 seconds.

Table 3 Results

RDP (n, p)	# games in each matchday	`MIP1` `(Time)`	Heuristic with circle method	Heuristic with Vizing method
(16,2)	4,4	0 (66.75)*	28	16
(16,3)	3,3,2	28 (36000)	22*	30
(16,4)	2,2,2,2	44 (36000)*	72	60
(16,5)	2,2,2,1,1	84 (36000)*	98	86
(16,6)	2,2,1,1,1,1	12 8(36000)	104*	110
(16,7)	2,1,1,1,1,1,1	152 (36000)	136*	138
(16,8)	1,1,1,1,1,1,1,1	180 (36000)	160*	166
(18,2)	5,4	16 (36000)	32	14*
(18,3)	3,3,3	36 (36000)*	64	44
(18,4)	3,2,2,2	54 (36000)*	96	60
(18,5)	2,2,2,2,1	118 (36000)	128	90*
(18,6)	2,2,2,1,1,1	176 (36000)	160	124*
(18,8)	2,1,1,1,1,1,1,1	No Solution	224	186*
(18,9)	1,1,1,1,1,1,1,1,1	No Solution	256	218*
(20,2)	5,5	14 (36000)*	36	24
(20,3)	4,3,3	70 (36000)	72	44*
(20,4)	3,3,2,2	108 (36000)	108	72*
(20,5)	2,2,2,2,2	166 (36000)	144	106*
(20,6)	2,2,2,2,1,1	168 (36000)	180	148*
(20,7)	2,2,2,1,1,1,1	260 (36000)	216	172*
(20,8)	2,2,1,1,1,1,1,1	No Solution	252	206*
(20,9)	2,1,1,1,1,1,1,1,1	No Solution	288	244*
(20,10)	1,1,1,1,1,1,1,1,1,1	No Solution	324	274*

In Table 3, for each problem instance, the method with the lowest total rest difference value is marked with an asterisk (*). It can be stated that in most problem instances the heuristic approach with either circle method or Vizing method performs better than the commercial solvers running `MIP1` model. RDP(16,2), RDP(16,4), RDP(16,5), RDP(18,3), RDP(18,4), RDP(20,2) are the only cases where `MIP1` performs better than the heuristic approaches. On the otherhand, for the cases where $p \geq 6$, the heuristic approach always outperforms `MIP1`. Thus, one could arguably claim that with increased values of p the heurtic approach is more likely to give better results than the MIP model of the problem. Last but not least, in all instances where either $n = 18$ or $n = 20$, the heuristic with Vizing method outperforms the heuristic with the circle method.

4 Conclusion

In this study, we first introduce the rest difference problem which aims to minimize the total rest differences of opposing teams by determining the round and matchday of each game. We then present a mathematical formulation of the problem. The proposed heuristic is capable of finding timetables with better objective values than MIP formulation for most problem instances considered.

We believe that there is still some room for further improvement in the total rest difference value. The schedules obtained by swapping the rounds of a schedule generated by the circle method are isomorphic. Rather than using only a round swap, other neighborhood searches can be applied to the generated schedules potentially leading to more diverse schedules. In the second stage where we solve RDP(n, p) for a given opponent schedule, the MIP model would then be able to find a timetable with even further improved total rest difference values.

On the other hand, in this research, we assume that the games are allocated to matchdays as evenly as possible. For future work, one can consider different allocations of games. Changing allocations of games in the matchdays may lead to an improved or worsened total rest difference value. Furthermore, we provide another direction for future work regarding rest differences. Rather than minimizing the total rest difference in the tournament, one can investigate to balance the rest differences over the teams.

References

Anderson I (1997) Combinatorial designs and tournaments. Oxford University Press

Atan T, Çavdaroğlu B (2018) Minimization of rest mismatches in round robin tournaments. Computers and Operations Research 99:78–89

Briskorn D (2009) Combinatorial properties of strength groups in round robin tournaments. European Journal of Operational Research 192(3):744–754

Briskorn D, Knust S (2010) Constructing fair sports league schedules with regard to strength groups. Discrete Applied Mathematics 158(2):123–135

Çavdaroğlu B, Atan T (2020) Determining matchdays in sports league schedules to minimize rest differences. Operations Research Letters 28(3):209–216

Elf M, Jünger M, Rinaldi G (2003) Minimizing breaks by maximizing cuts. Operations Research Letters 31(5):343–349

Guedes AC, Ribeiro CC (2011) A heuristic for minimizing weighted carry-over effects in round robin tournaments. Journal of Scheduling 14(6):655–667

Gurobi Optimization Inc (2019) Gurobi Optimizer Reference Manual Version 8.1. URL http://www.gurobi.com

van't Hof P, Post G, Briskorn D (2010) Constructing fair round robin tournaments with a minimum number of breaks. Operations Research Letters 38(6):592–596

IBM (2019) Cplex Optimizer User Manual Version 12.8. URL https://www.ibm.com/support/knowledgecenter/SSSA5P_12.8.0/ilog.odms.studio.help/pdf/gscplex.pdf

Januario T, Urrutia S (2012) An edge coloring heuristic based on Vizing's theorem. In: Proceedings of the Brazilian Symposium on Operations Research, pp 3994–4002

Januario T, Urrutia S, Ribeiro CC, De Werra D (2016) Edge coloring: A natural model for sports scheduling. European Journal of Operational Research 254(1):1–8

Miyashiro R, Matsui T (2005) A polynomial-time algorithm to find an equitable home–away assignment. Operations Research Letters 33(3):235–241

Russell K (1980) Balancing carry-over effects in round robin tournaments. Biometrika 67(1):127–131

Suksompong W (2016) Scheduling asynchronous round-robin tournaments. Operations Research Letters 44(1):96–100

de Werra D (1981) Scheduling in sports. In: Hansen P (ed) Studies on Graphs and Discrete Programming, North-Holland, pp 381–395

Zeng L, Mizuno S (2013) Constructing fair single round robin tournaments regarding strength groups with a minimum number of breaks. Operations Research Letters 41(5):506–510

REDOSPLAT DSL for timetabling requirements

Razija Turčinhodžić Mulahasanović ·
Samir Ribić

Abstract This paper presents a declarative language called REDOSPLAT intended to describe timetabling problems. It is designed for textual entry. Unlike most timetable data formats, REDOSPLAT uses easy to understand syntax based on sentences which describe requirements. DSL is complemented with a wizard which defines ordinary sentences.

Keywords Timetabling · DSL · REDOSPLAT

1 Introduction

Different formats and languages were developed to formulate requirements for timetabling problems. Some of them are line oriented languages [1], others mimic object-oriented programming languages [2], and some use language formats for expert systems [3]. XML is also base for wide number of languages [4]. In addition, there are languages similar to spoken languages. They are intuitive when it comes to data entry and include UniLang [5] and REDOSPLAT [6]. REDOSPLAT is a language that has been developed for about 10 years. Its latest version was introduced in 2018, and it is the topic of this paper. It is primarily made for school timetabling, but it can also be used for university and examination timetabling. It is a domain-specific declarative language whose sentences resemble those of spoken language. In our examples, the focus will be on school timetabling problems.

University of Sarajevo
Zmaja od Bosne bb, Sarajevo, Bosnia Herzegovina
Tel.: +387 33 250 700
Fax: +387 33 250 725
E-mail: samir.ribic@etf.unsa.ba, razija.turcinhodzic@etf.unsa.ba

2 Language scope

REDOSPLAT is designed to express different requirements for timetabling in schools across the world. It is used to describe time, resources, events, constraints, and solving goals. Time is expressible as days, working shifts (with a list of time slots they belong to) and time slots (with a day they belong to). Resources can have short and long names and they include sites (for schools in different cities and buildings), rooms (with optional capacity, price or site), classes (with optional default shift and the number of students), sub-class groups (with an optional number of students or percentage of class they belong to), joined multiple classes, teachers and courses (which can be grouped in categories). Event descriptions may be complex sentences, because describing events vary from simple ones (where the single teacher teaches a course to dedicated class), to events that allow selection between teachers, multiple teachers at the same time, joined classes, choice between rooms, lessons in multiple rooms, fixing time slots, spreading over the week, double and triple lessons, simultaneous lessons and lesson ordering. Constraints include unavailable time slots or days (for teachers, classes and rooms), wishes for time slots (soft constraint for teachers, classes and rooms with weight), default rooms (for teachers and classes), daily limits (minimal or maximal number of events per teacher/class/course category can be held in a day or shift), a prohibition that two teachers, rooms or classes can not be allocated at the same time, idle time policy (soft or hard) and travel time between sites. Preferred goals may be adjusted to define the main goal of solving (reducing idle time, reducing travel time, satisfy timeslot wishes, or make a balance between course category). Most of the requirements are optional, and do not need to be entered, but if the underlying solving engine supports it, they are used in the solution.

3 Language syntax examples

In schools, time is represented through time slots that generally have the same length. Time slots may belong to days and to shifts. Shifts are sets of time slots, usually morning or afternoon slots. These sentences describe one day, one timeslot and one shift.

```
Monday is day number 1.
MO1 is time slot 1 on Sunday.
SH1 is a shift consisting of MO1, ...
```

Here is an example of possible sentences describing classes, subgroups and teachers.

```
I is a class.
II is a class from shift SH1.
I-a is a subgroup of I.
JohnFein is a teacher.
```

Courses can be divided by categories so that some special restrictions, such as favouring difficult courses to occur in the earlier time slots, can be defined as well.

```
Mathematics is a course. Chem is course of category 2.
FEE is a course called "Fundamentals of electrical engineering".
```

Rooms may have their own capacity (number of seats), a fixed price if rented, and locations if lessons are taught in multiple buildings.

```
R1 is a room.
R2 is room with 30 seats costs 18.
FS is a room called "Sports Hall".
SwimmingPool is a building.
SP is located in SwimmingPool.
```

It is possible to assign a room to a particular teacher, class or subject. It is also possible to assign a teacher to the class that he or she teaches.

```
JohnFein is located in R1.
II is located in R2.
```

The main sentence creates an event (lesson). The event can have a teacher, a class, a room (or more of each of these resource), and the way lessons are spread over the week (doubles, triples). It is also possible to fix lessons of a course at predefined time slots if necessary.

```
KatinaFein teaches English to the group II1, II2
3 times a week, separately, with 1 double, in the room R1.
```

Following these basic requirements, special requirements may be added which include:

- unavailability of teachers, classes and rooms in designated time slots (this is a hard constraint);

```
SportsHall is unavailable on MO10.
JohnFein is unavailable on Friday.
```

- expressing the desire for the class, teacher or room to be used or not used within a certain time slot, which also sets the coefficient weight (soft constraint);

```
JohnFein wishes time slots MO1 (200).
```

- requiring specific time slots by teachers, classes or rooms;

```
ValeriBrowning requests time slot MO2.
```

- restrictions on the minimum and maximum hours for classes and teachers (and rooms) during the day or shift;

```
Ia has between 5 and 7 lessons.
NencyGreen has a maximum of 0 lessons on Friday.
```

- limiting the number of time breaks for teachers and classes;

```
Pauses for I are forbidden.
Pauses for ValerieBrowning are limited to 1.
```

- regulating special time slot overlaps;

```
MO2 overlaps with MO1.
```

- regulation of special prohibition of simultaneous lessons for some teachers, classes and rooms;

```
II1 conflicts with II2.
```

- class aggregation;

```
CompetitionMath is composed of I-a, II1.
```

- travel time between locations;

```
Travel time between SwimmingPool and MainBuilding is 2 timeslots.
```

4 IDE, wizzard, solving and usage

Although easy to read, typing in multiple similar sentences can be boring. To simmplify entry of sentences, a wizzard-based GUI was created allowing quick generation of basic constraints (time, teachers, classes, courses, rooms, capacities, workloads).

However, less used constrains are not covered by GUI, because they will tend to clutter and slow down the entry. Specific requirements and exceptions to the default values are entered subsequently by modifying the generated sentences and entering new ones. The editor includes syntax highlighting.

After selecting the algorithm and all other parameters needed to solve it, a timetable is generated in the case of syntactically correct instances. The solution can be presented in HTML, TABLE and XHSTT format. Conversion from XHSTT to REDOSPLAT is also possible. The solution includes lists

of all requirements that are not met. REDOSPLAT is integrated with KHE engine [7] based solvers [8] or linear programming based solvers [9] and multiple algorithms for automatic timetable generation.

REDOSPLAT was tested at 16 local schools, of which 13 schools have shifts. Within the selected schools we have 7 primary schools, 3 grammar schools, 4 mixed secondary schools, 1 technical secondary school and 1 university department. They vary in effort required to express the demands. The easiest school has 20 classes with about 500 events, and works in 2 shifts. The most difficult school has 56 classes and about 2500 events and it works in 4 shifts (2 for students and 2 for teachers). The university department has about 150 courses and 1900 events. Schools were visited 2-3 times, the first and second time data were taken and the language was presented, and the second or third visit was used to present a solution. The data entry for smaller schools lasted 2-3 hours, including verification of entered data. The entry for larger schools took 2-3 days, but the duration of data entry is not longer than the time that schools normally need.

5 Concluding remark

REDOSPLAT is GPL licensed tool and it is available at

https://sourceforge.net/projects/redosplat/

We invite the interested reader to use and contribute to our software: by adding better algorithms, speeding up the engine, suggesting new syntax elements to describe requirements that exist in some countries etc.

The simplified syntax may attract researchers to contribute with more real-life examples for time table requirements.

References

1. L. Di Gaspero B. McCollum A. Schaerf., The second international timetabling competition (ITC-2007): Curriculum based course timetabling (track 3), Technical Report QUB/IEEE/Tech/,,2007
2. E.K.Burke J.H.Kingston P.A.Pepper, A standard data format for timetabling instances, International Conference on the Practice and Theory of Automated Timetabling,213-222, 1997
3. L. F. Lai LF C. C. Wu et. al. , An artificial intelligence approach to course timetabling, International Journal on Artificial Intelligence Tools, 17(01):223-240, 2008.
4. G. Post G S. Ahmadi et. al, An XML format for benchmarks in high school timetabling, Annals of Operations Research, 194(1), 385-397, 2012
5. L. P. Reis E. Oliveira , Language for specifying complete timetabling problems, International Conference on the Practice and Theory of Automated Timetabling, 322-341, 2000
6. S. Ribić R Turčinhodžić, A. Muratović-Ribić, T. Kosar,REDOSPLAT: A readable domain-specific language for timetabling requirements definition, Computer Languages Systems and Structures, 54, 252-272, 2018
7. H. Kingston, The khe high school timetabling engine,, 2010
8. G. H. Fonseca H. G. Santos E. G. Carrano, Integrating matheuristics and metaheuristics for timetabling, Computers and Operations Research, 74, 108-117,2016
9. S. Ribić R. Turčinhožić A. Muratović-Ribić, Modelling constraints in school timetabling using integer linear programming, 2015 XXV International Conference on Information Communication and Automation Technologies (ICAT), (pp. 1-6), 2015

MILP. Try. Repeat.

Computing solutions to the ITC 2021 instances by repeated massive parallel MILP computations

Timo Berthold · Thorsten Koch · Yuji Shinano

Abstract This article describes how we solved most instances of the ITC-2021 as mixed-integer linear optimization problems (MILP). Our goal was to contend in this competition without developing specialized algorithms, i.e., only using existing MILP solvers. This path was challenging but provided feasible solutions for 40 out of 45 instances. Four of these instances where the best solutions overall and two of them were even proven optimal.

First, we will present how we modeled the problems as MILPs, discussing a few variations. Next, we describe how we combined methods to compute better solutions increasingly by restarting different MILP solvers and running the distributed massively parallel ParaXpress solver on HPC computers. Additionally, we computed a particular objective function based on the analytic center and either used this directly or with a newly developed variant of the feasibility pump heuristic. In the end, we also added a simulating annealing heuristic from the literature.

Keywords ITC2021 · MILP · Timetabling · Sports Scheduling · Massive parallel · Analytic Center

Mathematics Subject Classification (2010) 90C09 · 90-08 · 68R05

Timo Berthold
Fair Isaac Germany GmbH
Stubenwald-Allee 19, 64625 Bensheim, Germany
E-mail: timo.berthold@fico.com, OrcID: 0000-0002-6320-8154

Thorsten Koch
TU Berlin, Chair of Software and Algorithms for Discrete Optimization
Str. des 17. Juni 135, 10623 Berlin, Germany
E-mail: koch@zib.de, OrcID: 0000-0002-1967-0077

Yuji Shinano
Applied Algorithmic Intelligence Methods department, Zuse Institute Berlin
Takustr.7, 14195 Berlin
E-mail: shinano@zib.de, OrcID: 0000-0002-2902-882X

1 Introduction

Our goal was to participate in the International Timetabling Competition on Sports Timetabling ITC-2021[1] without developing specialized algorithms, i.e., only by using existing software and mainly working on the modeling and combining existing MILP solving approaches. In the following, we will describe how we came up with a MILP-based approach that made us reach the finals without special-purpose algorithms.

The challenge in ITC-2021 is to construct time-constrained double round-robin tournament with 16 to 20 teams for 45 different scenarios. There are hard constraints, which have to be respected, and soft constraints, which might be violated, but whose violation will result in a penalty. The goal is to minimize the penalties, which makes it a classical optimization problem. For details, see [19].

Section 2 will present the MILP formulation we used and its variants. In Section 3 we will describe our basic approach of repeated restarts. Then, as some of the instances – expectedly – turned out to be difficult, we looked into existing approaches for finding feasible MILP solutions and combined them in a new manner: The analytic center objective and the feasibility pump heuristics, described in Section 4. One of our original ideas was to model the problem instances as MILPs and then solve them on a supercomputer. We sketch the setup and outcome in Section 5. Regarding the general state-of-the art in sports timetabling, we refer to [12,13].

2 The MILP model

We modelled the problem as a mixed-integer linear program (MILP) as follows: First we define the set of teams: $T := \{0, \ldots, \text{teams} - 1\}$, the set of slots $S := \{0, \ldots, \text{slots} - 1\}$, and $S_0 := S \setminus \{0\}$, and $S_9 := S \setminus \{\text{slots} - 1\}$. The sets of slots of the 1st and 2nd half of the season are represented as : $S_1 := \{0, \ldots, \text{slots}/2 - 1\}$, $S_2 := S \setminus S_1$. Correspondingly, the sets of matches are called $M := \{(i, j) \mid i, j \in T, i \neq j\}$. We introduce binary variables x_{ijs} with $(i, j) \in M$ and $s \in S$. $x_{ijs} = 1$ indicating whether team i is playing home against team j away during slot s.

The following constraints ensure the basic requirements: each match gets assigned exactly one slot: $\sum_{s \in S} x_{ijs} = 1$ for all $i, j \in M$; in each slot the number of matches equals half the number of teams: $\sum_{i,j \in M} x_{ijs} = 1/2|T|$ for all $s \in S$; and each team only plays once in each slot: $\sum_{t,j \in M} x_{tjs} + \sum_{i,t \in M} x_{its} = 1$ for all $s \in S, t \in T$. We also added the (redundant) equation $\sum_{s \in S,(i,j) \in M} x_{ijs} = 1/2|S||T|$. Furthermore, in case we have a phased tournament, each pair of teams can only play once per half season: $\sum_{s \in S_1}(x_{ijs} + x_{jis}) = 1$ for all $(i, j) \in M, i < j$.

Now we introduce binary variables b^h_{ts}, and b^a_{ts}, $t \in T$, $s \in S_0$ indicating whether team t has a home or away break during slot s, respectively. Addition-

[1] `https://www.sportscheduling.ugent.be/ITC2021`

ally, we need the following constraints: $\sum_{a\in T\setminus\{t\}} x_{ta,s-1} + \sum_{a\in T\setminus\{t\}} x_{ta,s} \leq 1 + b^h_{ts}$ for all $t \in T, s \in S_0$ and similarly for b^a_{ts}.

Next, we introduce inequalities to enforce the different types of constraints that can appear in the instances. Please see [19] for the parameters of the particular constraints: `min, max, intp` $\in \mathbb{N}$, `teams` $\subseteq T$, `slots` $\subseteq S$, `meetings` $\subseteq T^2$. For CA2, CA3, CA4, and BR1 we only show the `mode="HA"` case, for the two cases the respective x-variables have to be removed accordingly. Note that the violation counter $v_* \geq 0$ has to be fixed to zero in case of a hard constraint. This gives rise to the following constraints:

- CA1(`max`): $\sum_{n\in T\setminus\{t\}} x_{tns} \leq \texttt{max} + v^{\text{ca1}}$ for all $t \in$ `teams`, $s \in$ `slots` for `mode="H"`, and x_{nts} in case `mode="A"`.
- CA1(`max`): $\sum_{n\in T\setminus\{t\}} x_{tns} \leq \texttt{max} + v^{\text{ca1}}$ for all $t \in$ `teams`, $s \in$ `slots` for `mode="H"`, and x_{nts} in case `mode="A"`.
- CA2(`max,teams`): $\sum_{s\in\texttt{slots}} \sum_{n\in\texttt{teams2}} (x_{tns} + x_{nts}) \leq \texttt{max} + v^{\text{ca2}}$ for all $t \in$ `teams`.
- CA3(`max,intp,teams`): $\sum_{n\in\texttt{teams2}} \sum_{r\in\{s-\texttt{intp}+1,\ldots,s\}} (x_{tnr} + x_{ntr}) \leq \texttt{max} + v^{\text{ca3}}_s$ for all $t \in$ `teams`, $s \in S \setminus \{0, \ldots, \texttt{intp} - 2\}$.
- CA4(`max,slots,teams1,teams2`): $\sum_{(i,j)\in\texttt{teams1}\times\texttt{teams2}, i\neq j} (x_{ijs} + x_{jis}) \leq \texttt{max} + v^{\text{ca4}}_s$ for all $s \in$ `slots`.
- GA1(`min,max,slots,meetings`): $\texttt{min} + v^{\text{ga1}}_n \leq \sum_{s\in\texttt{slots}} \sum_{(i,j)\in\texttt{meet}} x_{ijs} \leq \texttt{max} + v^{\text{ga1}}$.
- BR1(`max,slots,teams`): $\sum_{s\in\texttt{slots}} (b^h_{ts} + b^a_{ts}) \leq \texttt{max} + v^{\text{ba1}}$ for all $t \in$ `teams`.
- BR2(`max,slots,teams`): $\sum_{s\in\texttt{slots}} \sum_{t\in\texttt{teams}} (b^h_{ts} + b^a_{ts}) \leq \texttt{max} + v^{\text{ba2}}$.
- FA2(`intp,slots,teams`): $\sum_{a,p\in T\setminus\{t\}} \sum_{p\in\{0,\ldots,s\}} x_{tap} = h_{ts}$ for all $t \in T$, $s \in S$; $h_{is} - h_{js} \leq \texttt{intp} + v^{\text{fa2}}_{ij}$ and $h_{js} - h_{is} \leq \texttt{intp} + v^{\text{fa2}}_{ij}$ for all $s \in$ `slots`, $(i,j) \in \texttt{teams}^2, i < j$.
- SE1(`min,teams`): $|\sum_{s\in S}((s+1)x_{ijs} - (s+1)x_{jis})| \geq 1 + \texttt{min} + v^{\text{se1}}_{ij}$ for all $(i,j) \in \texttt{teams}^2, i < j$.

We wrote a python program to convert the XML description of the ITC-2021 instances into the above model formulated in the modeling language Zimpl [9], which then can generate an LP or MPS file. Zimpl automatically reformulates the absolute value needed for SE1 into an integer linear formulation.

As an objective, we minimize the sum over all violation counters multiplied with their respective penalty values. This objective function is precisely the one computed by the validator of the ITC-2021.

Variations: We experimented (to a minimal extent) with some variations of the above model. We added an objective cutoff with the objective value of the best known solution so far to improve the location of the analytic center as described in Section 4. As far as we can tell, it didn't hurt.

We also experimented with adding parts or even all of the odd-set constraints to push up the lower bound or increase feasibility. Apart from the

LP solves getting slow due to the huge number of constraints, there was no observable effect. Hence we discarded this approach quickly.

Another modeling variant we explored was to remove all soft constraints to get a feasibility instance. We assumed that this might be useful to find initial solutions, as due to the reduced number of constraints, we expected the problems to solve faster. Unfortunately, the effect was not as pronounced as we hoped for. We were only able to generate few initial solutions this way.

Furthermore, we also tried relaxing feasibility by changing the basic constraints from *equal to one* to *less equal to one* and penalized a non-sufficient number of variables set to one in the objective. Also CPLEX [8] and Xpress [5] offer such a procedure out-of-the-box, it is available via the `feasopt` and `repairinfeas` command, respectively. This approach did not provide additional solutions.

We believe the last two ideas had little effect for the following reasons: Most heuristics in a MILP solver are guided by the current LP solution. The LP solution depends on the search tree, and the search tree is built depending on the objective function. We observed that it was most effective to restart the solution process frequently. Apparently, the objective was not guiding the tree search towards better feasible solutions. Removing the objective as in the two experiments above left the solver completely clueless in what direction to go. Since the instances have comparatively few feasible solutions, hitting one by chance proved to be unlikely.

Additionally, we observed that increasing the running time of an instance only had a limited effect, i.e., only minor local improvements were happening, and then the primal solution was stuck in a local optimum. A substantial amount of change in the solution would be required for a major improvement beyond this local optimum, i.e., a distant part of the search tree needed to be explored. At the same time, MILP tree search algorithms tend to explore the vicinity of the current search area first and avoid big jumps. Hence, restarting worked reasonably well to overcome this behavior.

3 Repeated restarts and simulated annealing

In an initial step, we attempted to solve the problem formulation as given in Section 2 using standard MILP solvers. We employed various solvers and solver settings. This is for two reasons: Firstly, different solvers have different strengths, so while some models might solve better with one solver, another solver might be superior in another instance. Secondly, we aim at exploiting the effect of performance variability [10].

Using our cluster facility, we ran each instance up to 25 times, using either CPLEX, Gurobi [7], SCIP [1,6] or Xpress as a solver, using a time limit of up to 12 hours. We varied the parameters between the runs, usually increasing the number of heuristic runs, setting an emphasis for feasible solutions, reducing the amount of cutting plane generation, etc. Further, we altered the LP solver used for the initial relaxation. When available, we

used a solution polishing mode after a certain time. We also changed the random seed parameter between runs. As an example, a typical setting for Gurobi would be `Seed=131313 TimeLimit=30000 ImproveStartTime=15000 Cuts=0 Method=3 Presolve=2 Heuristics=0.5`

This way, we could gradually improve the best-known solution. Finally, there were a few notoriously hard problems for which we needed better heuristic methods to find a first feasible solution. We discuss this in the next section.

Note that we never used any specific settings for any of the instances. In principle, they all went through the same loop of repeated solving by various different solvers. The amount of loops depended on the solution quality. Similarly, whether we applied heuristics methods to ignite the search depended on whether our initial MILP searches came up with a feasible solution, but not on any problem characteristics.

There were five instances where we failed to find any feasible solution despite all efforts. After the contest, the ITC team provided us a sample solution to one of these problems. With the knowledge of this initial solution, we were immediately able to improve it.

Towards the end, we implemented a basic ad-hoc version of the simulating annealing heuristics as described in [2]. Again, we used the best-known solution as a start. The heuristic was able to improve this solution in several cases. We used the resulting solution as starting solution for further MILP solver runs.

4 Analytic Center Objective and Feasibility Pump

Some of the strongest primal heuristics implemented inside MILP solvers are improvement heuristics and depend on the knowledge of an initial solution for the problem at hand. To a certain extent, this also holds for the main branch-and-bound search.

In many cases, getting any feasible solution for a MILP problem is not too hard; the complexity of the problem primarily lies in finding the optimal solution and, even more, proving its optimality. However, the ITC-2021 instances, as typical for sport scheduling problems, are such that even getting an initial feasible solution can be very challenging.

Therefore, we decided to employ two special, expensive start heuristic procedures. The first one is making use of an Analytic Center objective as described in [4]. It replaces the objective function of a MILP by coefficients that correspond to the analytic center of the polyhedron associated to the MILP. Furthermore, we searched for initial solutions using a Feasibility Pump algorithm, see, e.g. [3]. The variant we employed uses multiple integer reference vectors in the projection step and is described in the thesis of Mexi [11]. Note that both methods are available out-of-the-box for general MILPs. The Feasibility Pump is available in source code on GitHub, while Analytic Center Search can be run via a special setting of Xpress.

Using our MILP solver portfolio and the two described dedicated start heuristics, we could find solutions for almost all ITC instances. The few with-

out a feasible solution and others for which we only saw a low-quality solution were forward to the next step in our problem-solving scheme: massively parallel MILP solving.

5 Massively parallel computations

As a third step, next to running a MILP solver portfolio and employing MILP start heuristics, we utilized massive parallel MILP solving on a supercomputer. Therefore, we used the massively parallel MILP solver *ParaXpress* [17] which has been developed by using *Ubiquity Generator(UG) framework* [14]. It builds on the FICO-Xpress Optimizer. From a massive parallelization point of view, the latest version of ParaXpress has almost the same features as that of ParaSCIP [16].

We ran ParaXpress on two HLRN IV supercomputers, Lisa and Emmy. In HLRN IV, each compute node employs two sockets of an Intel Xeon Platinum Processor 9242 and 362 GB of Memory, and each job uses 128 or 256 nodes (12288 or 24576 cores, respectively). ParaXpress runs a single controller process called LoadController and ten of thousands of solver processes that solve (sub-)MILPs, each running on a single core. We executed each job with a time limit of 12 hours. Jobs could be interrupted. Altogether, we spent more than seven million core hours.

We show the principal procedure in Algorithm 1. Note that initially, the set $\mathcal{I}$ contained selected instances that we thought were most suitable for a supercomputing approach.

More precisely, we conducted a large scale racing [18,15], in which all Solvers in ParaXpress run Xpress with different parameter settings independently, but incumbent solutions found are shared to cut off search trees.

When there is a new incumbent solution for an instance, a new job to restart the solving process with the incumbent solution is created. We submitted almost all jobs with 12,287 Solvers; only a few ran with 24,576 Solvers. We also tried to solve instances that could not find improved solutions with different ways of running ParaXpress, such as a general parallel Branch-and-bound method provided by UG, rather than a racing search. Note that ParaXpress could solve some of the instances to proven optimality.

6 Results and Conclusion

Table 1 shows our final results. For two instances, we were able to prove optimality of the solution. We could show that 36 of the 45 instances need to violate some of the soft constraints, given that the MILP solver proved a positive lower bound for them. Our results are not exhaustive. We mainly stopped due to the deadline. However, the progress noticeably slowed down.

The catch of our approach is that it did not use any special purpose scheduling algorithm. All methodology is general MILP technology, available indepen-

Algorithm 1: Massive parallel computation procedure

Data: Let $\mathcal{I}$ be the set of pairs of instances and its incumbent solution $(I, \mathbf{x})$.
Result: Updated $\mathcal{I}$.

$NoImprovedSol := \emptyset$. // indicate instances with no improved solutions
$SubmittedJob := \emptyset$. // indicate instances for submitted jobs
while *The due date has not reached* **do**
  while $\mathcal{I} \neq \emptyset$ **do**
    Select $(I, \mathbf{x})$ and $\mathcal{I} := \mathcal{I} \setminus (I, \mathbf{x})$.
    Submit a job to solve I by ParaXpress with MIP start $\mathbf{x}$
      (Method: large scale racing).
    $SubmittedJob := SubmittedJob \cup (I, \mathbf{x})$.
  while *terminated job exists* **do**
    Remove $(I, \mathbf{x})$ from $SubmittedJob := SubmittedJob \setminus (I, \mathbf{x})$.
    if *Improved soluiton* x^* *was found for* $(I, \mathbf{x})$ **then**
      $\mathcal{I} := \mathcal{I} \cup (I, \mathbf{x}^*)$
    else
      $NoImprovedSol := NoImprovedSol \cup (I, \mathbf{x})$
  while *Enough computing resources are available and* $NoImprovedSol \neq \emptyset$ **do**
    Select $(I, \mathbf{x})$ and $NoImprovedSol := NoImprovedSol \setminus (I, \mathbf{x})$.
    Submit a job to solve I by ParaXpress with MIP start $\mathbf{x}$
      (Method: large scale parallel Branch-and-Bound).
    $SubmittedJob := SubmittedJob \cup (I, \mathbf{x})$.

Table 1 Submitted results. Bold indicates the objective value is proven to be optimal. Italic indicates a zero lower bound. For all other instances a positive lower bound was proven.

Instance no.	Early	Mid	Late
01	804	—	2068
02	*402*	—	5525
03	1246	9700	3069
04	*764*	**7**	**0**
05	—	413	—
06	5506	2270	1212
07	6881	2991	2525
08	1409	174	1454
09	*122*	*810*	790
10	—	1813	2544
11	6843	3367	*305*
12	*1025*	1538	5669
13	360	*1051*	3877
14	25	*1679*	1433
15	4616	1614	*40*

dent of this work. Our main contribution was modeling the ITC-2021 problems and the way to combine the different MILP-solving methods.

It became apparent that a dedicated scheduling heuristic to find initial solutions would have been a helpful extension since finding an initial solution was causing severe troubles for some of the models. Once we knew an initial solution, the MILP technology could improve it to a satisfactory level in most

cases. We conclude that our methodology could probably be well combined with more problem-specific approaches. It would be interesting to see how well such an integrated approach performed in practice.

For our research, the competition showed direct impact, as it was a great test case for a new version of the feasibility pump heuristic [11], which could compute feasible solutions for 33 of the 45 instances. Furthermore, given that the number of cores available on single machines is constantly increasing, it might be worthwhile to incorporate a restart mechanism similar to the one we scripted directly into the solvers. There are certain classes of problems where this seems to work well. In a sense, this is similar to the idea of racing ramp-up, which we employed for the massive parallel computations.

We will make the code to generate the MILP model instances publicly available through the ITC organizers.

Acknowledgements The work for this article has been conducted within the Research Campus MODAL funded by the German Federal Ministry of Education and Research (fund number 05M14ZAM). The work was supported by the National High Performance Computing Center at the Zuse Institute Berlin (NHR@ZIB). We are grateful to the supercomputer staff, especially Matthias Läuter and Tobias Watermann. Thank to the organizers of the ITC 2021 for providing such nice instances and doing a great job.

References

1. Achterberg, T., Berthold, T., Koch, T., Wolter, K.: Constraint integer programming: A new approach to integrate CP and MIP. In: L. Perron, M.A. Trick (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, pp. 6–20. Springer (2008). DOI 10.1007/978-3-540-68155-7_4
2. Anagnostopoulos, A., Michel, L., Hentenryck, P.V., Vergados, Y.: A simulated annealing approach to the traveling tournament problem. Journal of Scheduling **9**, 177–193 (2006). DOI 10.1007/s10951-006-7187-8
3. Berthold, T., Lodi, A., Salvagnin, D.: Ten years of feasibility pump, and counting. EURO Journal on Computational Optimization **7**(1), 1–14 (2019). DOI 10.1007/s13675-018-0109-7
4. Berthold, T., Perregaard, M., Mészáros, C.: Four good reasons to use an interior point solver within a MIP solver. In: N. Kliewer, J.F. Ehmke, R. Borndörfer (eds.) Operations Research Proceedings 2017, pp. 159–164. Springer (2018). DOI 10.1007/978-3-319-89920-6_22
5. FICO: Xpress Optimization. URL `https://www.fico.com/fico-xpress-optimization/docs/latest/overview.html`
6. Gamrath, G., Anderson, D., Bestuzheva, K., Chen, W.K., Eifler, L., Gasse, M., Gemander, P., Gleixner, A., Gottwald, L., Halbig, K., Hendel, G., Hojny, C., Koch, T., Le Bodic, P., Maher, S.J., Matter, F., Miltenberger, M., Mühmer, E., Müller, B., Pfetsch, M.E., Schlösser, F., Serrano, F., Shinano, Y., Tawfik, C., Vigerske, S., Wegscheider, F., Weninger, D., Witzig, J.: The SCIP Optimization Suite 7.0. ZIB-Report 20-10, Zuse Institute Berlin (2020). URL `http://nbn-resolving.de/urn:nbn:de:0297-zib-78023`
7. Gurobi: Optimizer version 9.1. URL `https://www.gurobi.com`
8. IBM: IBM ILOG CPLEX optimization studio 12.1. URL `https://www.ibm.com/products/ilog-cplex-optimization-studio`
9. Koch, T.: Rapid mathematical programming. Ph.D. thesis, Technische Universität Berlin (2004). URL `https://zimpl.zib.de`
10. Lodi, A., Tramontani, A.: Performance variability in mixed-integer programming. In: Theory Driven by Influential Applications, pp. 1–12. INFORMS (2013). DOI 10.1287/educ.2013.0112

11. Mexi, G.: New ideas for the feasibility pump: Using multiple reference vectors. Master's thesis, Technische Universität Berlin (2021). To appear
12. Rasmussen, R.V., Trick, M.A.: Round robin scheduling - a survey. Tech. rep., Carnegie Mellon University (2006). DOI 10.1184/R1/6707867.v1
13. Ribeiro, C.C.: Sports scheduling: Problems and applications. International Transactions in Operational Research **19**(1-2), 201–226 (2012). DOI 10.1111/j.1475-3995.2011.00819.x
14. Shinano, Y.: The ubiquity generator framework: 7 years of progress in parallelizing branch-and-bound. In: N. Kliewer, J.F. Ehmke, R. Borndörfer (eds.) Operations Research Proceedings 2017, pp. 143–149. Springer (2018)
15. Shinano, Y., Achterberg, T., Berthold, T., Heinz, S., Koch, T., Winkler, M.: Solving hard MIPLIB2003 problems with ParaSCIP on supercomputers: An update. In: Parallel Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International, pp. 1552–1561 (2014). DOI 10.1109/IPDPSW.2014.174
16. Shinano, Y., Achterberg, T., Berthold, T., Heinz, S., Koch, T., Winkler, M.: Solving open MIP instances with ParaSCIP on supercomputers using up to 80,000 cores. In: 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 770–779. IEEE Computer Society (2016)
17. Shinano, Y., Berthold, T., Heinz, S.: ParaXpress: an experimental extension of the FICO Xpress-Optimizer to solve hard MIPs on supercomputers. Optimization Methods and Software **33**(3), 530–539 (2018). DOI 10.1080/10556788.2018.1428602
18. Shinano, Y., Heinz, S., Vigerske, S., Winkler, M.: FiberSCIP—a shared memory parallelization of SCIP. INFORMS Journal on Computing **30**(1), 11–30 (2018). DOI 10.1287/ijoc.2017.0762
19. Van Bulck, David and Dries Goossens and Jeroen Beliën and Morteza Davari: The fifth international timetabling competition (ITC 2021): Sports timetabling. Proceedings of MathSport International 2021 Conference, MathSport pp. 117–122 (2021)

A Hybrid Model to Find Schedules for Double Round Robin Tournaments With Side Constraints

Jasper van Doornmalen · Christopher Hojny · Roel Lambers · Frits Spieksma

Abstract We describe a method for finding solutions to the instances provided by the International Timetabling Competition, edition 2021.

Keywords Sport scheduling · Mixed Integer Programming · Matheuristics · Fix and Optimise · International Timetabling Competition 2021

1 Introduction

The International Timetabling Competition (ITC) is a competition where participants are asked to solve instances of a particular type of timetabling problems. In the 2021 edition [4], the instances originate from the domain of sport scheduling. The aim of this note is to describe the method that we have used to find solutions for the instances provided by the 2021 edition of the ITC.

There are many types of scheduling problems within the realm of sport scheduling, however the ITC focused solely on the scheduling of so-called *Double Round Robin (DRR)* tournaments. In a DRR tournament each team plays each other team twice, once at home and once away. The matches are distributed over different *rounds* or *slots*, such that each team plays at most one match per round. When the number of teams (N) is even, it is possible to schedule a DRR using $2N - 2$ rounds, implying that each team plays exactly once in each round. The resulting schedule is then called *compact*, as it uses the minimum number of rounds needed to schedule all the matches. A compact DRR is a very popular format used in many sports; for instance, most national soccer leagues are organised as a compact DRR. As the instances provided by

Eindhoven University of Technology
Combinatorial Optimization Group
PO Box 513
5600 MB Eindhoven, The Netherlands
E-mail: {m.j.v.doornmalen, c.hojny, r.lambers, f.c.r.spieksma}@tue.nl

the ITC ask for compact DRRs, we focus here exclusively on finding compact schedules.

Literature Review

There is a growing amount of papers dealing with all kinds of problems in sport scheduling. Results on the complexity of finding round robin schedules can be traced back to Easton [5], see also Briskorn et al. [3], and Van Bulck and Goossens [12]. Integer programming formulations have been studied, among others, in Briskorn and Drexl [2]. For more general information concerning sport scheduling, we refer to well-known surveys by Rasmussen and Trick [11] and Kendall et al. [8]; a bibliography is maintained by Knust [9]. When scheduling a league in practice, many different real-life aspects may turn up, see Goossens and Spieksma [6] for an overview devoted to European soccer leagues. The instances provided by the ITC feature many practical side-constraints; in fact, we discuss the properties of a schedule as required by the 2021 ITC in Section 2, and we call the problem of finding such a schedule ITC-DRR. In Section 3 we describe our method, and in Section 4 we give the results.

2 Requirements of a ITC-DRR Schedule

Ultimately, the quality of a schedule depends on the preferences of the organisers. It is a fact, however, that there are reoccurring themes that often need to be taken into account when devising a high-quality compact DRR schedule. In Section 2.1, we specify our notation and phrase the problem, and in Section 2.2 we discuss the type of hard and soft constraints present in the instances of the ITC.

2.1 Notation

We use $\mathcal{T}$ for the set of teams, $\mathcal{S}$ for the set of rounds; thus we have $|\mathcal{T}| = N$ and $|\mathcal{S}| = 2N - 2$. A match is an ordered pair $(i, j) \in \mathcal{T} \times \mathcal{T}, i \neq j$, where team i plays Home and j plays Away. For each match, there should be a round $r \in \mathcal{S}$ where this specific match occurs and in each $r \in \mathcal{S}$, team $i \in \mathcal{T}$ should play exactly one match.

All other constraints that occur, tend to be specific for a team or a subset of teams, and applicable to a subset of the rounds, see Section 2.2. Constraints are either so called *hard constraints*, or are *soft constraints*. The hard constraints need to be satisfied; for the soft constraints, a penalty for every (unit of) violation is given. The objective is to find a schedule that satisfies all the hard constraints and minimises the total sum of the penalties induced by the soft constraints; we refer to this problem as ITC-DRR.

2.2 On the variety of constraints in the ITC instances

We describe different types of constraints that are present in the ITC instances.

2.2.1 The type of DRR

It is quite natural to organise the DRR such that each pair of teams meets once in the first half of the schedule (i.e., in the first $N-1$ rounds), and once in the second half of the schedule (i.e., in the last $N-1$ rounds); this is called *phased*. A phased competition can be regarded as two consecutive Single Round Robins (SRR).

Apart from being phased or not, most competitions require that the rounds where the match (i,j) and its return (j,i) for $i,j \in \mathcal{T}$ is played, are separated by a given minimum number of rounds. When this minimum number of rounds equals $N-1$ for all the matches, the second half of the season is the complement of the first half of the season, and the resulting DRR is called *mirrored*.

2.2.2 Constraints concerning Home Away Patterns (HAPs)

Consider, for a given team, the series of $2N-2$ home and away matches: we call the resulting pattern the Home Away Pattern (or HAP) of that team. When a team plays two matches in consecutive rounds either both home or both away, this is called a *break*. In many applications, breaks are seen as unfavourable for the team and their occurrence should be minimised.

More generally, it is quite common to demand that the home-away pattern according to which a team plays is balanced for certain sets of rounds. For instance, it is very common to demand that each team starts its first two rounds with a home match and an away match, and also plays in its last two (and even four) matches once at home and once away (twice at home and twice away). In addition, even when the DRR is not phased, one can demand that after $N-1$ rounds the number of home matches played by each team is either $\lfloor \frac{N-1}{2} \rfloor$ or $\lceil \frac{N-1}{2} \rceil$.

To ensure favourable break-patterns, schedulers often use a so-called First-Break-Then-Schedule approach, which goes back to Nemhauser and Trick [10]. In such an approach, first the home away patterns are fixed, and then the matches are determined consistent with the given HAPs.

2.2.3 Constraints concerning matches

In practice, not all matches are treated equally: matches between top teams are subject to much more attention than other matches. As a consequence, finding the appropriate round for such a match can be an important factor in the quality of a schedule. For instance, many leagues feature so-called Super Sundays: rounds where 4 or 6 top teams play matches amongst them. Finding the ideal round for such a Super Sunday is important. In addition, some matches should not be played in certain rounds: a match between top teams

should not be played in the last round (for risk of not being relevant anymore). Further, one can imagine, in the context of soccer leagues, that qualification for other tournaments have an impact on the rounds where particular matches should be avoided.

It is also very common to strive for a balance in the strength of consecutive opponents, especially in the first 4 or 6 rounds. Indeed, it is considered quite unfavourable to have to play against 3 top teams in a row, whether or not at home or away.

3 Solution process

In principle, the ITC-DRR problem can be modelled as a compact mixed-integer program (MIP), which can be solved by a black-box MIP solver. In practice, however, state-of-the-art solvers fail to solve instances even with a small number of teams within weeks. For this reason, we suggest a heuristic approach to find solutions satisfying all hard constraints of ITC-DRR and that aims to minimise the violation of soft constraints. Our approach consists of three phases:

1. constructing a schedule S for a compact phased double-round robin tournament;
2. turning S into a schedule S' adhering to all hard constraints;
3. starting from S', searching for a schedule that violates less soft constraints while still satisfying all hard constraints.

We use an explicit rule to construct the initial schedule S in Phase 1, whereas Phases 2 and 3 make use of procedure that alternates between solving a MIP and using a local search algorithm to find better solutions. In the following, we describe the details of these three steps.

3.1 Finding an Initial Schedule

To construct the initial schedule S, we use the circle method [1] to create a schedule for a compact single-round robin tournament. This tournament is then duplicated where the home-away status of each match is flipped. By combining these two single-round robin tournaments a compact and phased double-round robin tournament is found.

Note that S is only guaranteed to satisfy the compactness constraint of ITC-DRR. If S violates some hard constraint, we enter Phase 2; otherwise, we directly continue with Phase 3.

3.2 Finding Improving Schedules

In both Phase 2 and 3, our aim is to find schedules that improve on the initial schedule S or S', respectively. To quantify the quality of a solution,

we introduce a function f that measures the violation of hard constraints (Phase 2) or soft constraints (Phase 3) by a schedule. Moreover, violations of hard constraints are not permitted in Phase 3. Our goal is to find a schedule that minimises the value of f. As mentioned above, we search for such a schedule by an alternating procedure whose components are described next.

3.2.1 Local Search

Januario et al. [7] discuss how schedules for compact single-round robin tournaments can be encoded in an edge-coloured complete undirected graph. The vertices of the complete graph correspond to the teams and edges encode possible matches. Rounds of the matches are distinguished by assigning each edge a colour, i.e. the edges of each colour class form a perfect matching.

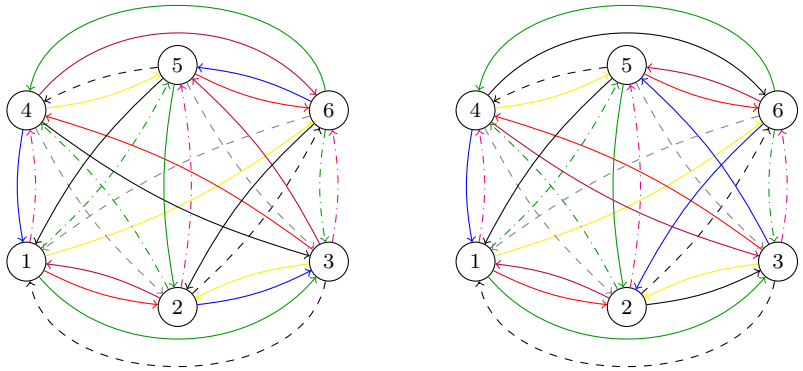
To find better schedules, they discuss a local search algorithm on such colourings. The idea of this algorithm is that one schedule can be turned into another by the following procedure: Starting with two vertices (teams) $i, j \in \mathcal{T}$ one can select a set of colours (rounds) $c_0, c_1, \ldots, c_{p-1} \in \mathcal{S}$ that induce p disjoint paths $C_0, \ldots, C_{p-1}$ where path C_k is a path from i to j alternating over the coloured edges c_k and $c_{k+1 \pmod p}$. Then, one can re-colour the edges by interchanging the alternating colours on each path. If the paths are disjoint, then the obtained edge-coloured graph is again a valid encoding for a schedule.

The concept of this neighbourhood for single-round robin tournaments can be generalised to double-round robin tournaments by replacing every edge in the graph by two anti-parallel arcs that can be coloured independently. That gives an arc-coloured complete directed graph, where the vertices correspond to the teams and the arc colours correspond to the rounds. The set of disjoint paths in the single-round robin case now translates to a set of arc-disjoint paths where it is permitted to traverse arcs in either direction, see Figure 1 for an example.

In the local search part of our procedure, we iterate over the possible disjoint paths, starting by checking small neighbourhoods: the sets of colours (rounds) of size 2. Once all possibilities are checked for all pairs of teams, this number of colours is increased. In each iteration we check the change of the objective, and based on this change we decide to accept the solution and restart, or to reject the solution and continue to the next iteration. We do this in a simulated annealing fashion, i.e. we also allow to continue with worse solutions with a certain probability. To ensure feasibility in Phase 3, solutions that violate hard constraints are never accepted. Finally, we terminate the procedure after a certain time limit and continue with the MIP approach.

3.2.2 MIP Approach

To improve on the schedule found by local search, we use a MIP model. Its main variables are $x_{i,j,r} \in \{0,1\}$, where $i, j \in \mathcal{T}$ $(i \neq j)$ and $r \in \mathcal{S}$. This variable indicates that match (i, j) takes place in round r. The variables $hb_{i,r}, ab_{i,r} \in \{0,1\}$ represent the home-break or away-break status of team i on slot $r \in \mathcal{S}$,



(a) Initial graph. (b) After re-colouring.

Fig. 1: An example of two neighbouring solutions using the colour-cycle (purple, blue, black) between team 3 and 6. The alternating paths are: purple–blue: $(3,5),(6,5)$; blue–black: $(2,3),(6,2)$; and black–purple: $(4,3),(4,6)$.

respectively. These are used to model the constraints related to breaks. For soft constraint class $c \in \mathcal{C}$, we denote by $d_{c,i}$ the penality of violating the i-th constraint in class c. The constraint penalty is denoted by p_c, and the model objective is to minimise the total penalty given by $\sum_{c\in\mathcal{C}} \sum_{i\in c} p_c d_{c,i}$. Each constraint is modelled by a set of linear constraints in a straightforward fashion, such that the feasible region matches the description of the constraint classes in the competition.

We use a fix-and-optimise matheuristic where we fix a (uniformly) random subset of rounds to the current schedule, and optimise for the remaining rounds. Given an initial feasible schedule, a random subset $\hat{\mathcal{S}} \subseteq \mathcal{S}$ is selected. Next, the variables x_{ijr} for $r \in \mathcal{S} \setminus \hat{\mathcal{S}}$ are fixed to the value corresponding to the initial solution, and the model is optimised. We have two parameters that change depending on the behaviour of the model: A time limit and a sample size. The same time limit parameter is used in the local search approach to ensure that the time spent in both approaches is about equal. If the time limit is reached while no improved solution is found, we increase the time limit and possibly reduce the sample size. If an optimal solution is found within the time limit, the random sample size is increased and the time limit is slightly decreased. Moreover, a solution limit is imposed. If multiple improved solutions are found within the time limit, we stop the optimisation and continue with another sample of rounds. In general, if after solving the MIP an improvement is found, we restart the MIP-approach with a different sample. Otherwise we go back to local search using the best schedule found so far as initialisation.

4 Computational Experience

For the implementation we have used Python 3.8. The MIPs are solved using Gurobi 9.1.0 with the `gurobipy` package. Table 1 compares the result of our solution approach (LS+MIP) with the approach where the compact MIP of Section 3.2.2 (MIP) is solved. The test set comprises the early test instances provided by [4]. Each instance has a time limit of 24 hours. A dash denotes that no feasible solution is found within the time limit. Observe that in most cases the solution process with local search finds a better primal solution than the compact MIP formulation. These results are found by using 4 cores on machines with an Intel Xeon Platinum 8260 with 42.7GB of RAM per instance.

Table 1: Computational results for the early instances comparing our method (LS+MIP) to solving the compact mixed-integer programming formulation (MIP) with a time limit of 24 hours per instance. Shown are the best primal solutions, dual bounds, the time per phase of LS+MIP and the number of iterations in LS and MIP.

	LS+MIP							MIP	
	Phase 2			Phase 3					
		Iterations			Iterations				
Instance	Time(s)	LS	MIP	Time(s)	LS	MIP	Primal	Primal	Dual
Early 1	822	6	15	85578	72	75	619	1588	1.00
Early 2	453	4	12	85947	57	94	369	606	0.00
Early 3	490	4	15	85910	50	69	1212	1701	55.19
Early 4	86400	60	98	–	–	–	–	–	0.00
Early 5	86400	65	101	–	–	–	–	–	258.88
Early 6	46127	60	82	40273	35	60	4682	–	633.64
Early 7	2049	12	25	84351	51	89	7306	10153	1271.64
Early 8	10	1	0	86390	60	91	1588	1615	210.40
Early 9	54	1	0	86346	82	121	443	206	0.00
Early 10	86400	50	79	–	–	–	–	–	319.77
Early 11	1647	17	26	84753	61	94	6772	10841	329.43
Early 12	61723	36	77	24677	31	41	1130	1850	0.00
Early 13	525	4	15	85875	53	81	372	498	2.00
Early 14	76	1	5	86324	54	114	24	42	1.00
Early 15	577	5	13	85823	59	78	4779	6021	514.48

References

1. Anderson, I.: Combinatorial designs and tournaments. 6. Oxford University Press (1997)
2. Briskorn, D., Drexl, A.: IP models for round robin tournaments. Computers & Operations Research **36**(3), 837–852 (2009)
3. Briskorn, D., Drexl, A., Spieksma, F.C.R.: Round robin tournaments and three index assignments. 4OR **8**(4), 365–374 (2010)
4. van Bulck, D., Goossens, D., Beliën, J., Davari, M.: The fifth international timetabling competition (ITC 2021): Sports timetabling. In: Proceedings of MathSport International 2021 Conference, MathSport, pp. 117–122 (2021)

5. Easton, K.K.: Using integer programming and constraint programming to solve sports scheduling problems. Ph.D. thesis, Georgia Institute of Technology (2003)
6. Goossens, D.R., Spieksma, F.C.R.: Soccer schedules in europe: an overview. Journal of scheduling **15**(5), 641–651 (2012)
7. Januario, T., Urrutia, S., Ribeiro, C.C., De Werra, D.: Edge coloring: A natural model for sports scheduling. European Journal of Operational Research **254**(1), 1–8 (2016)
8. Kendall, G., Knust, S., Ribeiro, C.C., Urrutia, S.: Scheduling in sports: An annotated bibliography. Computers & Operations Research **37**(1), 1–19 (2010)
9. Knust, S.: Classification of literature on sports scheduling. http://www.inf.uos.de/knust/sportssched/sportlit_class/. Accessed: June 2021
10. Nemhauser, G.L., Trick, M.A.: Scheduling a major college basketball conference. Operations research **46**(1), 1–8 (1998)
11. Rasmussen, R.V., Trick, M.A.: Round robin scheduling–a survey. European Journal of Operational Research **188**(3), 617–636 (2008)
12. Van Bulck, D., Goossens, D.: On the complexity of pattern feasibility problems in time-relaxed sports timetabling. Operations Research Letters **48**(4), 452–459 (2020)

MILP Based Approaches for Scheduling Double Round-Robin Tournaments

Daniil Sumin · Ivan Rodin

Abstract This paper presents some mixed-integer linear programming (MILP) based models for solving the International Timetabling Competition on Sports Timetabling (ITC2021). The approach explained here came third in the competition, and it found the best solution in 16 out of the 45 instances.

Keywords International Timetabling Competition 2021 · Mixed-Integer Linear Programming · Sports Timetabling · Home-Away Patterns · Decomposition

1 Introduction

This paper presents the approaches used to solve the ITC2021 timetabling problems [2]. The article is organized as follows. Problem description is given in Section 2. Section 3 introduces the Baseline model that is MILP monolithic formulation for scheduling problem. Section 4 describes the Patterns model consisting of two optimization parts: possible patterns generation and assigning patterns to the teams. Section 5 proposes the Patterns Mirrored model that is based on the mirroring format of the some real-life competitions. Section 6 presents the most applicable model called the 2-Phased model that decomposes the problem into two consecutive parts: the first and the second round of the competition. A combination of the Patterns and the 2-Phased approaches is described in Section 7. Proposed models' applicability for different types of instances is considered in Section 8.

D. Sumin
Higher School of Economics, Russia, Moscow
E-mail: danilsum21@gmail.com

I. Rodin
Moscow State University, Russia, Moscow
E-mail: del.ris@yandex.ru

2 Problem description

The input data is a set of settings and constraints of optimization problems specified in the RobinX format [4] (although not all features and constraints of this format were included in ITC2021 instances). All constraints are divided into two types: hard and soft. Hard constraints can never be violated, while maximum number of the soft constraints should be satisfied with respect to their importance. The main goal of the work is to develop an algorithm for scheduling a double round-robin tournament in which all hard constraints are satisfied and the penalty for soft constraints is minimized.

On the other hand, constraints can be divided into the following groups: capacity constraints (CA1,CA2,CA3,CA4), game constraints (GA1), break constraints (BR1,BR2), fairness constraints (FA2) and separation constraints (SE1). Capacity constraints regulate when teams play home or away in a specific group of slots. Game constraints are used to forbid or to force playing specific games in certain slots. Break is a situation when a team plays two consecutive games with the same home or away status. Break constraints are used to manage the quantity of such situations. Nurmi et al. in [3] proposed a measure of fairness of a sports competition called "k-balancedness" which requires the difference in played home and away games to be smaller than k at any point of the season. There is no sign of FA2 constraints in all proposed models, because FA2 requires a lot of additional variables and constraints. FA2 constraints are implicitly optimized by BR2 and hard CA3. As a result, there are almost no violations of soft FA2 constraints in all instances. Separation constraints are used to control the interval between two games with the same opponents.

3 Baseline model

Baseline model is a full MILP formulation of the ITC2021 problem. This model considers a binary decision variable z_{s,t_1,t_2} that is equal to 1 if team t_1 plays at home against team t_2 on time slot s and 0 otherwise. The model is based on [4], where authors mathematically formulated all constraints following the RobinX data format. Some constraints in [4] were nonlinear, therefore, standard linearization techniques were used in the Baseline model.

This approach is developed for instances with the following properties:

1. small-sized problems;
2. problems without hard BR2 constraints;
3. problems without any type of SE1 constraints.

4 Patterns model

Breaks are one of the bottlenecks of the Baseline model, because they require a lot of linearization constraints and decision variables, so we propose the

Patterns model. The sequence of home and away games is called team's home-away pattern (HAP). The Patterns model consists of two optimization stages.

At the first stage we minimize the number of breaks in a competition using $h_{s,\,t}$ binary decision variables which indicate that team t plays at home on time slot s. Some constraints can be taken into account at this stage: total number of home games for each team, total number of home teams on each time slot, hard CA1 constraints, hard CA3 constraints and a subset of hard GA1 constraints. The set of possible patterns contains the solution of the first stage and all other patterns with zero or one break.

The second stage is similar to the Baseline model, apart from dealing with breaks. It has additional binary decision variables $q_{t,\,p}$ which indicate that team t follows HAP p from the set of possible patterns.

This approach is developed for instances with the following properties:

1. small-sized problems;
2. problems without any type of SE1 constraints;
3. problems where the biggest contribution to the objective function is made by soft BR2 constraints.

5 Patterns Mirrored model

Some European football leagues use a mirrored competition format, where the second half of the competition is identical to the first one with an inverted home advantage. The Patterns Mirrored model uses both stages of the Patterns model along with mirroring constraints. Mirrored format allows us to reduce the problem size significantly, because the schedule of the second round is completely defined by the schedule of the first. But with the mirrored scheme, we considerably reduce the feasible region, so some instances can be infeasible with this approach. Mirrored format automatically satisfied both separation and phased constraints. Moreover, we should take into account the lower bound on the number of breaks in the mirrored double round-robin tournament $3T-6$ [1], where T is the number of teams in competition.

This approach is developed for instances with the following properties:

1. large-sized problems;
2. problems with any type of SE1 constraints.

6 2-Phased model

2-Phased model is the decomposition approach for the ITC2021 problem. The idea is to divide the solution into two consecutive stages. At the first stage we build the schedule of the first phase of competition. At the second stage we schedule the second phase with respect to the solution of the first stage. Some of the constraints can be independently divided into two rounds: CA3, BR2 and SE1. In Phase 1, we tighten mutual constraints (CA1, CA2, CA4, GA1,

BR1) for having an opportunity to satisfy them while solving Phase 2. For these constraints, *max* constant from the respective constraint is distributed proportionally between two phases with respect to the number of slots of a certain phase in constraint. Example for the competition with 18 teams:

Initial constraint	CA2 (max = 2) with $slots = \{0; 1; 20; 21\}$
Generated the first phase constraint	CA2 (max = 1) with $slots = \{0; 1\}$
Generated the second phase constraint	CA2 (max = 1) with $slots = \{20; 21\}$

Besides, two special constraints are included in the first stage: if we have a mandatory game between team 1 and team 2 in the second round of the competition from GA1, then we know about their home-away status in the first round and the set of possible first round slots for their game for satisfying the SE1 constraint.

This approach is developed for instances with the following properties:

1. large-sized problems;
2. problems with any type of SE1 constraints;
3. problems, where the Patterns Mirrored model is infeasible.

7 Patterns 2-Phased model

The 2-Phased model could have trouble with satisfying the hard BR2 constraints, and may also incur high penalties in the objective function due to the soft BR2 constraints. Combining the Patterns model and the 2-Phased model into Patterns 2-Phased model can improve the results. The combined model consists of all the features of the two models: generation of possible patterns, and application of two consecutive phase models for assigning patterns to the teams.

This approach is developed for instances with the following properties:

1. large-sized problems;
2. problems with any type of SE1 constraints;
3. problems with strong influence of break constraints.

8 Applicability

We can say that the model applicability area is a class of problems, where this approach can find feasible solution in a reasonable time. Each model was developed to handle certain types of instances, but the scope of the approach can be much larger. Models applicability is shown in Fig. 1. The Baseline model is not applicable to large problems, instances with hard BR2 constraints, and instances with SE1 constraints. Patterns model can be applied to problems with hard BR2 constraints. Other models can be applied to any type of problem.

The general algorithm for solving each of the instances in the competition can be described as follows:

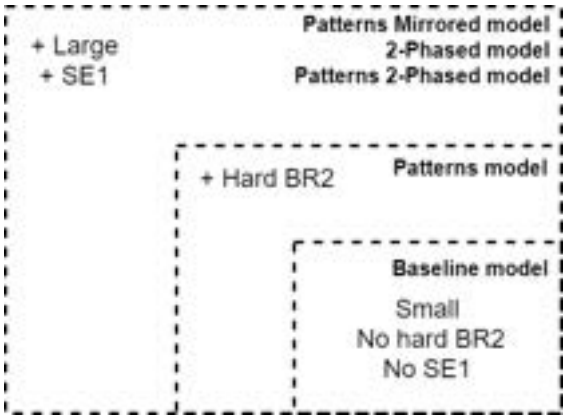


Fig. 1 Models' applicability

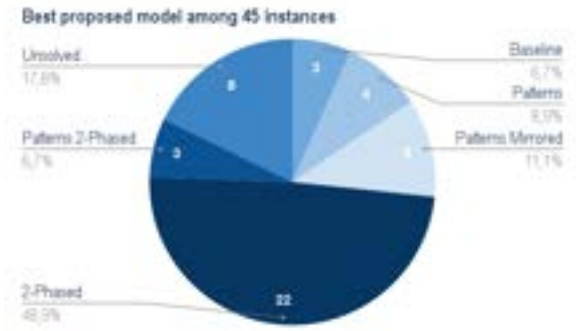


Fig. 2 Best proposed model among 45 instances

1. Choose appropriate formulations to apply to the problem instance to be solved depending on which constraints are present in the instance.
2. Solve the problem instance using the formulations selected in Step 1.
3. Select the best solution from the set of all produced solutions.

All instances from ITC2021 were solved according to the algorithm described above. Fig. 2 shows the distribution of the best solutions provided by the different formulations. It can be seen that the 2-Phased model is the most applicable approach for solving ITC2021 instances. In 7 out of 8 unsolved instances, the hard BR2 constraints could not be satisfied that is motivation for future research.

References

1. De Werra, D. (1981). Scheduling in sports. Studies on graphs and discrete programming, 11, 381-395.

2. Van Bulck, D., Goossens, D., Beliën, J., and Davari, M. (2021). The Fifth International Timetabling Competition (ITC 2021): Sports Timetabling. Proceedings of MathSport International 2021 Conference, MathSport, pp. 117-122.
3. Nurmi K, Goossens D, Bartsch T, Bonomo F, Briskorn D, Duran G, et al. A framework for scheduling professional sports leagues. In: Ao S, Chan A, Katagiri H, Xu L, editors. AIP Conference Proceedings. AMER INST PHYSICS; 2010. p. 14–28.
4. Van Bulck D, Goossens D, Schönberger J, Guajardo M. RobinX : a three-field classification and unified data format for round-robin sports timetabling. EUROPEAN JOURNAL OF OPERATIONAL RESEARCH. 2020;280(2):568–80.

An adaptive large neighbourhood search matheuristic for the ITC2021 Sports Timetabling Competition

Antony E. Phillips · Michael O'Sullivan · Cameron Walker

1 Introduction

We outline a model and method for solving the International Timetabling Competition on Sports Timetabling (Van Bulck et al., 2021b). Our algorithm generates a starting solution, and improves it using an adaptive large neighbourhood search (ALNS) using several neighbourhood types. Each neighbourhood subproblem is solved using integer programming, classifying the overall approach as a matheuristic. Similar methods have been effective on other classes of constrained scheduling problems (e.g. Pisinger and Røpke, 2007; Lindahl et al., 2018).

During the competition we generated 3 best-known solutions, and another 17 best-known solutions afterwards by improving other contestants solutions.

2 Modelling & Solution Approach

Our approach initially defines a monolithic integer program (IP) model which fully encodes the ITC2021 problem specification. The primary decision variables define the binary choice of assignment of a single game (ordered pair of teams) to a slot in the double round robin.

These variables are sufficient to prescribe a solution, however to enable modelling of the specified hard and soft constraints, we use two sets of auxiliary variables. The first set represents whether a break has occurred for each team, in each slot, for each game mode. The second set represents the magnitude of violation for each soft constraint, which are used to define the objective function.

Antony E. Phillips (aphi.xc@gmail.com)
Phillips Data Science Ltd

Michael O'Sullivan, Cameron Walker
Department of Engineering Science, University of Auckland

Neighbourhood	Description
Slots	Game-slot variables are free within a subset of slots
Teams	Game-slot variables are free if both teams are within a subset of teams
Teams+	Game-slot variables are free if either team is within a subset of teams
Order	Game-slot variables are free if this game or the reversed game is assigned to this slot in the current solution

Table 1 Neighbourhood types

This simple formulation is unlike most mathematical programs in the sports timetabling literature, which tend to use a multi-stage approach (Rasmussen and Trick, 2008). For example, an initial stage to select allowable home-away patterns, and a latter stage to build the full schedule.

For the instances in ITC2021, the monolithic IP we define can easily be constructed in memory, with the largest instance (Early 15) represented with 28,171 variables, 23,842 constraints and 11.1 million nonzeros. However, the problem structure makes this model intractable to solve to optimality.

2.1 Starting solution

To find a starting solution, we first attempt to solve the monolithic IP (terminating after 8 hours), which may find a feasible solution. If not, we construct a starting solution using a "canonical factorization" from de Werra (1981), which minimises the number of breaks. This solution is guaranteed to satisfy the challenging "BR2" constraint (maximum total number of breaks), but is likely to violate other hard constraints and thus not be feasible. Using the number of hard constraint violations as an objective function to be minimised, we then employ a hill climbing heuristic. Specifically we try all pairs of swaps between teams and slots (e.g. every assigned game for two teams are swapped).

2.2 Improvement Phase

From a starting solution, we iteratively apply an adaptive large neighbourhood search (ALNS), where part of the solution is allowed to be modified while the rest remains fixed. An IP is solved within this neighbourhood subproblem which aims to minimise the number of hard or soft constraint violations, depending whether the current solution is infeasible or feasible respectively. The types of neighbourhood are shown in Table 1.

The selection of which neighbourhood type to use is treated as a multi-armed bandit problem, and addressed using the Upper Confidence Bound (UCB) method. Based on the results from all previous iterations on this instance, the next neighbourhood type ("arm") is chosen as that with the greatest optimistic upper bound on its expected probability of improving the solution ("reward"); see formula (2.10) from Sutton and Barto (2018). This

balances between exploring options and exploiting those which have performed well in previous iterations.

The size of the neighbourhood is also adaptive, based on the outcome in the last iteration for this neighbourhood type. Except for the fixed-size "Order" neighbourhood, the size is increased or decreased by 1 unit (i.e. a slot or team) if the last iteration was solved within 5 minutes or if it terminated on a 30 minute time limit respectively. Searching a larger number of small neighbourhoods was found to be more effective than the opposite.

Within each iteration, the specific subset of slots or teams chosen for the neighbourhood is determined based on a randomly selected unsatisfied constraint (hard or soft). This allows the neighbourhood to focus on assignments which contribute to the total penalty of the solution. If the constraint is defined over more slots or teams than required for the neighbourhood, a random subset are chosen. In the converse situation, additional slots or teams are selected at random.

2.3 Results

To solve instances in parallel, we used Google Cloud Platform both to run our algorithm (Compute Engine virtual machines), and to maintain an online database of solutions and attempts (BigQuery).

Over several days, we used 4 "c2-standard-30" virtual machine instances, for a total of 10,686 vCPU hours (Google Cloud, 2021). Therefore, each of the 45 competition instances received an average of 237 vCPU hours, terminating on the total time elapsed. This is approximately similar to 1-2 days of execution on a standard consumer CPU (with 4 to 8 physical cores). All IPs were solved using Gurobi 9.1.1, with the "MIPFocus" parameter set to 1.

Our full set of results are shown in Table 2. The objective values of our best solutions during the competition are given in column "Us-ITC", and the best solutions from all teams in column "Best-ITC". We were able to find a feasible solution to 37 out of 45 instances, of which 3 solutions were the best-known from all submissions (as marked with an asterisk). However, most of our solutions have a notably higher objective than the best-known solutions.

After the competition, we tested the ALNS algorithm on the best-known solutions from all teams, each for 4 hours on a consumer CPU (AMD Ryzen 5900HX). In 17 cases we were able to generate a new best-known solution, with objective shown in column "Us-Post" of Table 2. These solutions are available on the competition website (Van Bulck et al., 2021a). Of the 17 highly optimised starting solutions, 12 were provided by Team Saturn, 3 by Team UoS, 1 by Team Udine and 1 by Team GOAL.

Finally, Figure 1 demonstrates the ALNS algorithm on the 'Early 15' instance. Starting from a feasible solution obtained by solving the monolithic model (with objective of 7,504), the ALNS algorithm reduces the objective value to 4,667 over 1100 iterations. Only the 91 successful iterations and the associated neighbourhood type are shown.

Instance	Objective		
	Us-ITC	Best-ITC	Us-Post
Early 1	666	362	
Early 2	379	160	145
Early 3	1171	1012	992
Early 4	–	512	507
Early 5	–	3127	
Early 6	4821	3352	3325
Early 7	7208	4763	
Early 8	1191	1114	1074
Early 9	447	108	
Early 10	–	3400	
Early 11	6713	4436	4426
Early 12	925	380	
Early 13	382	121	
Early 14	106	4	
Early 15	4667	3368	3362
Middle 1	–	5177	
Middle 2	–	7381	
Middle 3	11235	9701	
Middle 4	7*	7	
Middle 5	681	413	
Middle 6	2026	1125	1120
Middle 7	3317	1784	1783
Middle 8	277	129	
Middle 9	1315	450	
Middle 10	2370	1250	
Middle 11	3143	2511	2446
Middle 12	911*	911	
Middle 13	1044	253	252
Middle 14	1704	1172	
Middle 15	1401	495	485
Late 1	2406	1969	1922
Late 2	–	5400	
Late 3	2900	2369	
Late 4	0*	0	
Late 5	–	1939	1923
Late 6	1310	923	
Late 7	2805	1558	
Late 8	1252	934	
Late 9	1343	563	
Late 10	–	1988	1945
Late 11	376	207	202
Late 12	5542	3689	3428
Late 13	3099	1820	
Late 14	1714	1206	
Late 15	80	20	

Table 2 Full Results

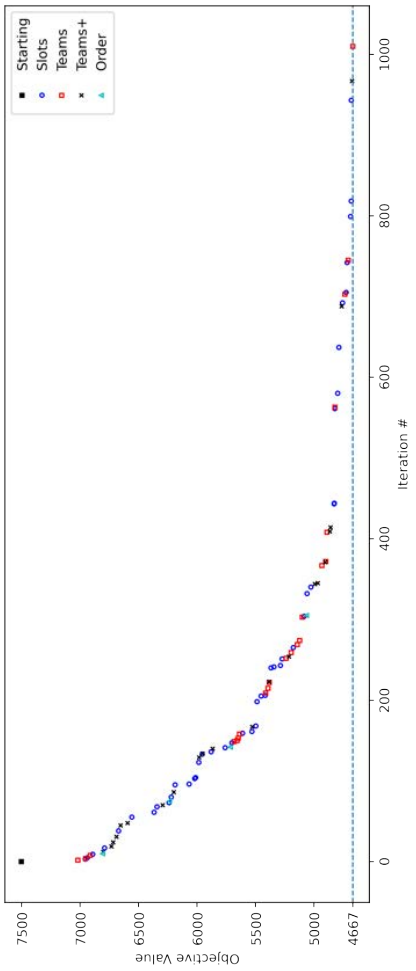


Fig. 1 ALNS Iterations for Early 15

3 Conclusion

Our algorithm performed adequately given its relative simplicity and modest execution time. The method to generate a starting solution was particularly simple, and consequently often could not provide a feasible solution to the ALNS algorithm. Notably, the 8 unsolved instances all include a phased tournament and a hard "BR2" constraint, which add dependencies across the entire solution and are hard to control with the defined ALNS neighbourhood types.

However, when operating on feasible solutions, the ALNS method was able to rapidly improve most solutions an appreciable amount. This led to us finding 3 best-known solutions during the competition, and 17 more after the competition, by improving the solutions from other teams.

The ALNS algorithm could likely be further sped up, as less than 10% of neighbourhoods found an improved solution. This suggests an opportunity for a more targeted choice of neighbourhoods, whether derived analytically or with online learning. The ALNS method could additionally be hybridized with the conventional decomposition approaches in sports timetabling, which add structured home-away patterns and multiple starting solutions.

References

de Werra, D. (1981). Scheduling in Sports. In Hansen, P., editor, *Annals of Discrete Mathematics (11)*, volume 59 of *North-Holland Mathematics Studies*, pages 381–395. North-Holland.

Google Cloud (2021). Machine families — Compute Engine Documentation — Google Cloud. https://cloud.google.com/compute/docs/machine-types. Accessed 2021-07-05.

Lindahl, M., Sørensen, M., and Stidsen, T. R. (2018). A Fix-and-Optimize Matheuristic for University Timetabling. *Journal of Heuristics*, 24(4):645–665.

Pisinger, D. and Røpke, S. (2007). A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403–2435.

Rasmussen, R. and Trick, M. (2008). Round robin scheduling – a survey. *European Journal of Operational Research*, 188:617–636.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Van Bulck, D., Goossens, D., Beliën, J., and Davari, M. (2021a). International Timetabling Competition 2021: Sports Timetabling – website. https://www.sportscheduling.ugent.be/ITC2021.

Van Bulck, D., Goossens, D., Beliën, J., and Davari, M. (2021b). The Fifth International Timetabling Competition (ITC 2021): Sports Timetabling. In *Proceedings of MathSport International 2021 Conference*, MathSport, pages 117–122.

A Fix-and-Optimize Heuristic for the ITC2021 Sports Timetabling Problem

George H.G. Fonseca · Túlio A.M. Toffolo

1 Introduction

This extended abstract briefly describes a fix-and-optimize heuristic for the Fifth International Timetabling Competition (ITC2021), which considered a challenging and realistic Sports Timetabling Problem. The ITC2021 problem consists basically of the assignment of games to rounds in a double round-robin tournament considering many constraints. An even number N of teams is considered, meaning there is a total of $2N-2$ rounds with every team playing exactly once at each round. In total, the ITC2021 problem imposes up to nine different constraints which represent common situations in the real-world. Two types of constraints are considered: hard constraints (H), which must be satisfied at all times, and soft constraints (S), whose violation is penalized in the objective function. The nine different constraints were categorized into five groups by ITC2021 organizers and described by Van Bulck et al. (2021) as:

1. **Capacity constraints** (CA): force a team to play home or away and regulate the total number of games played by a team or group of teams.
2. **Game constraints** (GA): enforce or forbid specific assignments of a game to rounds.
3. **Fairness constraints** (FA): prevent an unbalanced timetable concerning home games, travel distances, etc.

George H. G. Fonseca
Department of Computing and Systems – DECSI
Federal University of Ouro Preto, João Monlevade/MG, Brazil
E-mail: george@ufop.edu.br

Túlio A. M. Toffolo
Department of Computing
Federal University of Ouro Preto, Ouro Preto/MG, Brazil
E-mail: tulio@toffolo.com.br

4. **Break constraints** (BR): regulate the frequency and timing of breaks in a competition; we say that a team has a break if it has two consecutive home games, or two consecutive away games.
5. **Separation constraints** (SE): regulate the number of rounds between consecutive games involving the same teams.

As with most international challenges, a diverse set of benchmark instances was proposed. This is a particularly relevant contribution in the field since most papers addressing similar problems report case studies, resulting in limited comparison among different authors. The availability of benchmark instances may reduce this issue. For further details concerning the ITC2021 problem and the proposed instances, we refer the reader to Van Bulck et al. (2021).

2 Proposed algorithm

We initially formulated the ITC2021 problem as an integer program with three-indexed decision variables $x_{i,j,k}$, which take value 1 if game (i, j) is assigned to round k and 0 otherwise. However, as expected, most of the benchmark instances resulted in models which commercial solvers were not capable of solving within the runtime limit of 10 hours. While this result clearly motivated us to employ heuristics, the proposed integer programming formulation remained as one of the main components of our proposed algorithm, which may be categorized as a matheuristic.

Matheuristics are heuristics that take advantage of the power of mathematical programming (MP) solvers to tackle hard combinatorial optimization problems. More specifically, fix-and-optimize algorithms are matheuristics that iteratively employ a mathematical programming solver to optimize a small sub-problem while the remainder of the problem is fixed.

Algorithm 1 presents the proposed approach. Note that this algorithm is executed twice: first to obtain a feasible solution and then to optimize (improve) this solution. In the first execution, an initial solution is given by the the Polygon Method (Ribeiro and Urrutia, 2007). For the second execution, the feasible solution obtained in the first execution is given as input. Lines 1 and 2 load the MP model, initial solution and decision variables. Note that hard constraints are modelled as soft constraints in the first execution. Stopping criteria consist of a time limit or proven optimality (line 3). The solution is optimal if it has zero cost or if its sub-problem size matches the size of the problem and solver status is optimal. Lines 4 to 17 select the variables to be optimized at each iteration. We considered two ways of releasing variables: neighborhood $\mathcal{N}^R$, which randomly selects n rounds to be optimized (lines 6 to 11), and neighborhood $\mathcal{N}^T$, which randomly selects n teams to be optimized (lines 13 to 17). To allow venue exchange, in neighborhood $\mathcal{N}^R$ we also release the variable related to the inverse venue game of games that occur in one of the selected rounds (line 10). Line 18 fixes the non-selected variables to their current value. Line 19 solves the MP model while line 20 releases the variables for the next iteration. Finally, lines 21 to 24 adjust the sub-problem size.

Algorithm 1: GOAL Solver

Input: (i) Problem instance $\mathbb{P}$; (ii) Initial solution s_0; (iii) Sub-problem size n; (iv) Time limit t_{max}; (v) Iteration time limit t_{it}
Output: (i) Best solution s found.

$\mathbb{M} \leftarrow$ Load mathematical model for $\mathbb{P}$
$\mathcal{X} \leftarrow$ Load variables of $\mathbb{M}$ with solution s_0
while elapsed time $\leq t_{max}$ **and** optimal solution has not been found **do**
  $\mathcal{V} \leftarrow \emptyset$
  if *Random()* ≤ 0.5 **then**
    $rounds \leftarrow 0$
    while $rounds < n$ **do**
      $r \leftarrow$ Randomly select a non-selected round from $\mathbb{P}$
      $\mathcal{V} \leftarrow \mathcal{V} \cup$ Variables related to round r
      $\mathcal{V} \leftarrow \mathcal{V} \cup$ Variable related to the inversed venue game of the ones that occur in round r
      $rounds \leftarrow rounds + 1$
  else
    $teams \leftarrow 0$
    while $teams < \lfloor n/2 \rfloor$ **do**
      $t \leftarrow$ Randomly select a non-selected team from $\mathbb{P}$
      $\mathcal{V} \leftarrow \mathcal{V} \cup$ Variables related to team t
      $teams \leftarrow teams + 1$
  Fix variables $\mathcal{X} \setminus \mathcal{V}$ to their current value
  $(s, status) \leftarrow$ Solve $\mathbb{M}$ with time limit t_{it}
  Release fixed variables in $\mathbb{M}$
  if $status = Optimal$ **then**
    $n \leftarrow n + 1$
  else
    $n \leftarrow n - 1$
return s

3 Preliminary experiments

The proposed approach was implemented in Java 16. Gurobi 9.1 (Gurobi Optimization, LLC, 2021) was employed to solve sub-problem formulations. The computational experiments were executed on an Intel® Xeon E5620 2.40GHz with 120GB RAM running CentOS Linux 7. Each sub-problem runtime limit was set to 100 seconds, while initial sub-problem size n was set to 10, meaning 10 rounds for $\mathcal{N}^R$ and 5 teams for $\mathcal{N}^T$. We run the experiments with a 24-hour time limit. Table 1 presents the best solutions found by our solver along with the best known solutions (BKS) among all submitted by the 13 teams that participated in ITC2021[1]. For instances in which we could not find feasible solutions, hard (H) and soft (S) costs are displayed as H_S. Solutions marked with a ⊛ are proven optimal.

[1] Reported at https://www.sportscheduling.ugent.be/ITC2021/instances.php

Table 1 Best solutions found by the proposed approach for ITC2021.

Instance	Our Best	BKS	Instance	Our Best	BKS	Instance	Our Best	BKS
Early1	421	362	Middle1	5_6039	5177	Late1	2073	1969
Early2	309	160	Middle2	13_7232	7381	Late2	4_6346	5400
Early3	1146	1012	Middle3	9837	9701	Late3	2474	2369
Early4	2_1738	512	Middle4	⊛ **7**	7	Late4	⊛ **0**	0
Early5	13_4631	3127	Middle5	543	413	Late5	12_2402	1939
Early6	4088	3352	Middle6	1630	1125	Late6	1082	923
Early7	6434	4763	Middle7	2394	1784	Late7	2333	1558
Early8	**1064**	1064	Middle8	200	129	Late8	1165	934
Early9	538	108	Middle9	1050	450	Late9	1219	563
Early10	7_4963	3400	Middle10	1537	1250	Late10	13_3559	1988
Early11	5127	4436	Middle11	2798	2511	Late11	361	207
Early12	890	380	Middle12	1007	911	Late12	4786	3689
Early13	331	121	Middle13	430	253	Late13	**1820**	1820
Early14	84	4	Middle14	1682	1172	Late14	1562	1206
Early15	4196	3368	Middle15	1089	495	Late15	160	20

4 Conclusions

We briefly presented a two-neighborhood fix-and-optimize approach for the ITC2021 Sports Timetabling Problem. Limited attention has been given to fix-and-optimize methods for Sports Scheduling in the literature, despite the strong results obtained by such methodology when considering other scheduling problems. Preliminary experiments resulted in feasible solutions for 37 out of 45 instances. We found the best overall solution for 4 instances and proved optimality for 2 of them. These are encouraging results given the difficulty of the problem: even finding feasible solutions is already a challenge for some instances.

We believe there is still room for improvement in the proposed approach. Smarter ways of selecting sub-problems may be proposed; integration with usual heuristic neighborhoods can be explored; and parameter tuning may improve the algorithm's overall performance. Moreover, this approach can heavily benefit from improved formulations. First-break-then-schedule or first-schedule-then-break decompositions may be incorporated as well.

References

Gurobi Optimization, LLC (2021). Gurobi Optimizer Reference Manual.

Ribeiro, C. C. and S. Urrutia (2007). Heuristics for the mirrored traveling tournament problem. *European Journal of Operational Research 179*(3), 775–787.

Van Bulck, D., D. Goossens, J. Beliën, and M. Davari (2021). The fifth international timetabling competition (ITC 2021): Sports timetabling. In *Proceedings of MathSport International 2021 Conference, MathSport, in press.*, pp. 117–122.

Scheduling Double Round-Robin Sports Tournaments

Carlos Lamas-Fernandez · Antonio Martinez-Sykora · Chris N Potts

Abstract This paper considers the problem of assigning matches to time slots in a double-round robin sports tournament. An integer linear programming (ILP) model is developed which includes a variety of hard and soft constraint that are likely to be encountered when scheduling professional football/soccer league fixtures. The solution methodology used is a matheuristic that fixes a large number of variables in the ILP model at each iteration to enable a solution to be generated relatively quickly. In this fix-and-relax approach, different methods are used to determine which variables are to be fixed. Computational results are given for the instances having 16, 18 and 20 teams that form an international timetabling competition on sports timetabling (ITC2021). The main findings are that the matheuristic finds solutions for most ITC2021 instances relatively quickly with all hard constraints satisfied, and generates many best-known solutions for these instances.

Keywords Sports Tournament Scheduling · Double Round-Robin · Matheuristic

Mathematics Subject Classification (2010) 90

Carlos Lamas-Fernandez
CORMSIS (Centre for Operational Research, Management Science & Information Systems)
Southampton Business School
University of Southampton
E-mail: C.Lamas-Fernandez@soton.ac.uk

Antonio Martinez-Sykora
CORMSIS (Centre for Operational Research, Management Science & Information Systems)
Southampton Business School
University of Southampton
E-mail: A.Martinez-Sykora@soton.ac.uk

Chris N Potts
CORMSIS (Centre for Operational Research, Management Science & Information Systems)
School of Mathematical Sciences
University of Southampton
E-mail: C.N.Potts@soton.ac.uk

1 Introduction

Research on designing algorithms for scheduling sports competitions has taken place for almost fifty years, although interest in this research topic has increased during the last twenty years. For an overview of the major contributions, we refer to Knust [5]. In this paper, we address the scheduling of double round-robin tournaments of the type used in many European and South American football/soccer leagues where each team in the competition plays one home game and one away game against every other team. Goossens and Spieksma [4] provide an overview of the structure of the main football leagues in Europe. As pointed out by Van Bulck et al. [6], most publications focus on designing an algorithm that is suited to the constraints imposed by a particular league. As a consequence, computational work assessing the relative performance of such algorithms is scarce. This has motivated a unified data format for round-robin sports timetabling by Van Bulck et al. [7] that facilitates the comparison of algorithmic approaches. It has also motivated the International Timetabling Competition on Sports Timetabling, named ITC2021 [8], that "aims to stimulate the development of solvers for the construction of round-robin timetables". This paper reports on the solver developed by the authors for the double round-robin sports scheduling problem (DRRSSP), and provides computational results in the form of objective function values for the 45 instances on which the result of ITC2021 is based.

The double round-robin tournaments of ITC2021 are *compact*, meaning that in each time slot every team has exactly one match. We distinguish between *phased* and *unphased* double round-robin tournaments. In a phased tournament, the matches played in the first half of the slots comprise a single round-robin tournament, and similarly for the matches in the second half of the slots, whereas an unphased tournament has no such constraints on the matches. A *break* occurs if a team plays at home in some slot having played at home in its previous slot (*home break*), or if a team plays away in some slot having played away in its previous slot (*away break*).

There are different classes of constraints on the DRRSSP. These arise due to the interests of multiple stakeholders. The football clubs competing in a tournament are concerned about maximizing revenue by having their home matches being played in slots when more fans are likely to be able to watch the game. They also prefer schedules in which home and away matches alternate, thereby reducing the number of breaks. Teams that use the same stadium clearly create constraints, and other events occurring within the vicinity of the stadium in certain time slots may prevent a home match being assigned to these slots. The police are responsible for safety outside of the ground, which may impose constraints on the number of teams playing in the same city in the same slot. TV companies invest significantly in gain broadcasting rights to the matches, and may prefer schedules having the most high-profile matches spread throughout the season.

This aim of our study is to propose a new algorithm for creating schedules for the DRRSSP, and to evaluate this approach on the 45 instances of

the ITC2021 competition. In Section 2, we provide a formal description of the DRRSSP that we are addressing, and Section 3 contains an integer programming formulation of the problem. Section 4 describes our proposed matheuristic, which takes the form of a fix-and-relax procedure. Computational results obtained by applying our matheuristic to the 45 instances provided within ITC2021 are presented and discussed in Section 5. Lastly, Section 6 contains some concluding remarks.

2 Problem Description

For the DRRSSP, it is required to design a round-robin tournament that allocates matches to time slots. However, Van Bulck et al. [6] observe from various studies reported in the literature that there can be various constraints that affect how the matches are scheduled.

Let n denote the number of teams competing in the double round-robin tournament, where n is even. Further, let $T = \{1, \ldots, n\}$ be the set of all teams. For each pair of teams $i, j \in T$, where $i < j$, there is match (i, j) for which team i plays a home game against team j and a match (j, i) for which team i plays an away match against team j. Thus, each team plays $n - 1$ home games at its own venue and $n - 1$ away games at its opponent's venue. The tournament has a set S of time slots for the matches. We assume that the minimum number of time slots is used so that $S = \{1, \ldots, 2n - 2\}$, which produces a *compact* tournament. If the matches that are played in slots $1, \ldots, n - 1$ define a single round-robin tournament, then the matches in slots $n, \ldots, 2n-2$ also define a single round-robin tournament, and such a structure is *phased* tournament. For a phased tournament, we define $S' = \{1, \ldots, n-1\}$ to be the set of slots used for the first phase.

There are structural constraints that ensure that the matches assigned to time slots satisfy the conditions of a double round-robin tournament, with additional constraints added if the tournament is phased. However, there are many other types of constraints within ITC2021, as listed below, which can either be hard or soft.

Capacity constraints: within a given set of time slots, a team is forced to play at home or away, and the total number of matches played by a team or by a set of teams has an upper limit.

Game constraints: given a set of time slots and a set of matches, the number of matches assigned to these time slots has an upper limit and a lower limit.

Break constraints: within a given set of time slots, there is an upper limit on the total number of breaks for a given set of teams.

Fairness constraints: within each of a given set of slots, there is an upper limit on the largest difference in the number of home games played between each pair of teams within a given set.

Separation constraints: for a given set of teams, there is a lower limit on the gap between home and away matches between each pair of teams in this set.

A solution of the DRRSSP has, for each soft constraint, a non-negative deviation that specifies the number of units of violation of the constraint. Also, each soft constraint has an associated weight that represents the penalty per unit violation of the constraint. The objective of the problem is to design a double round-robin tournament in which all hard constraints are satisfied so that the sum of weighted deviations for all soft constraints is minimized.

3 Integer Linear Programming Model

Our integer linear programming (ILP) model has variables and structural constraints that are widely used in round-robin sports scheduling, such as in the study of Durán et al. [3]. The key parameters used in our ILP are the set of teams T and the set of slots S, as defined in Section 2, with S' representing the set containing the first half of the slots.

The variables in our integer programming model that define the structure of the tournament are

$$x_{ijs} = \begin{cases} 1 & \text{if match } (i,j) \text{ is played in slot } s, \\ 0 & \text{otherwise.} \end{cases} \qquad \forall i,j \in T,\ \forall s \in S$$

However, some tournament specifications impose constraints on numbers of breaks that are allowed. Thus, we introduce additional variables

$$b_{is}^{\mathrm{HA}} = \begin{cases} 1 & \text{if } i \text{ has a home or away break in slot } s, \\ 0 & \text{otherwise.} \end{cases} \qquad \forall i \in T,\ \forall s \in S \setminus \{1\}$$

In some cases, it is necessary to differentiate between a home break and an away break. Thus, if $b_{is}^{\mathrm{HA}} = 1$, then $b_{is}^{\mathrm{H}} = 1$ or $b_{is}^{\mathrm{A}} = 1$ depending on whether a home break or an away break occurs for team i in slot s. Also, there is a relationship $b_{is}^{\mathrm{HA}} = b_{is}^{\mathrm{H}} + b_{is}^{\mathrm{A}}$ for all $i \in T$ and $s \in S \setminus \{1\}$. Also, when there are constraints on the separation in terms of the number of slots between the two matches played by a pair of teams i and j, it is useful to introduce the variables

$$y_{ij} = \begin{cases} 1 & \text{if match } (i,j) \text{ occurs before match } (j,i), \\ 0 & \text{otherwise.} \end{cases} \qquad \forall i,j \in T$$

The following subsections provide the classes of constraints that are included in the model. There are some structural constraints given below in Section 3.1 which must be satisfied in a double round-robin tournament. The remaining constraints are non-structural.

Each non-structural constraint has an index by which it is identified. Associated with most constraints c is a threshold value t_c, which is the maximum value that some linear combination of the x_{ijs}, b_{is}^{H}, b_{is}^{A}, b_{is}^{HA} and y_{ij} variables can achieve without incurring a penalty. In such cases, the right-hand side of

the constraint is set to $t_c + d_c$, which d_c is an integer deviation variable for constraint c. The remaining constraints c have a threshold value t_c, which is the minimum value that some linear combination of the x_{ijs}, b^{H}_{is}, b^{A}_{is}, b^{HA}_{is} and t_{ij} variables can achieve without incurring a penalty. For these constraints, the right-hand side of the constraint is set to $t_c - d_c$, which d_c is an integer deviation variable for constraint c. If a constraint c of either type is hard, we introduce $d_c = 0$ as a further constraint.

3.1 Structural constraints

The structural constraints on a double round-robin tournament ensure that the variables are assigned values that create a valid tournament. The structural constraints are all hard. The first set of constraints below ensure that within each slot s team i either has a home game or an away game against some other team j. The second constraints impose the condition that a match (i, j) for each pair of teams i and j appears in exactly one slot. The third set of constraints, which are applied only when the DRRSSP is phased, force all pairs of teams i and j to play exactly one match in the first half of the time slots S', and consequently exactly one match in the second half of the time slots $S \setminus S'$.

$$\sum_{j \in T \setminus \{i\}} (x_{ijs} + x_{jis}) = 1 \qquad \forall i \in T,\ \forall s \in S \quad (1)$$

$$\sum_{s \in S} x_{ijs} = 1 \qquad \forall i, j \in T \quad (2)$$

$$\sum_{s \in S'} (x_{ijs} + x_{jis}) = 1 \qquad \forall i, j \in T \quad (3)$$

The constraints linking the b^{H}_{is} and b^{A}_{is} variables with the x_{ijs} variables are

$$\sum_{j \in T} (x_{ijs} + x_{i,j,s-1)}) \leq b^{\mathrm{H}}_{is} + 1 \qquad \forall i \in T, s \in S \setminus \{1\} \quad (4)$$

$$\sum_{j \in T} (x_{jis} + x_{j,i,s-1}) \leq b^{\mathrm{A}}_{is} + 1 \qquad \forall i \in T, s \in S \setminus \{1\} \quad (5)$$

Further, the constraints linking y_{ij} with the x_{ijs} variables are

$$\sum_{s \in S} s(x_{jis} - x_{ijs}) \leq M y_{ij} \qquad \forall i, j \in T \quad (6)$$

$$\sum_{s \in S} s(x_{ijs} - x_{jis}) \leq M(1 - y_{ij}) \qquad \forall i, j \in T \quad (7)$$

where M is a constant that satisfies the condition $M \geq |S| - 1$.

3.2 Capacity constraints

The DRRSSP has four types of capacity constraints. These sets of constraints are denoted by CA_1, CA_2, CA_3 and CA_4.

There is a set $\mathrm{CA}_1 = \mathrm{CA}_1^{\mathrm{H}} \cup \mathrm{CA}_1^{\mathrm{A}}$ of home and away capacity constraints of type 1. Each constraint c of CA_1 is specified by a team i_c, a set of slots S_c and a threshold value t_c. The home capacity constraint is

$$\sum_{j \in T} \sum_{s \in S_c} x_{ijs} \leq t_c + d_{ci} \qquad \forall c \in \mathrm{CA}_1^{\mathrm{H}},\ i = i_c \qquad (8)$$

while the corresponding constraints for $c \in \mathrm{CA}_1^{\mathrm{A}}$ are identical except that x_{ijs} is replaced by x_{jis}.

The capacity constraints c of types 2 and 3 are each specified by a team i_c, a set of teams T_c and a set of slots S_c for type 2 constraints. Also, there are constraints that refer to home games, away games, and home-away games that are index by H, A and HA. Thus, the constraint sets are $\mathrm{CA}_t = \mathrm{CA}_t^{\mathrm{H}} \cup \mathrm{CA}_t^{\mathrm{A}} \cup \mathrm{CA}_t^{\mathrm{HA}}$ for types $t = 2$ and $t = 3$. The home constraints for type 2 and type 3 are

$$\sum_{j \in T_c} \sum_{s \in S_c} x_{ijs} \leq t_c + d_{ci} \qquad \forall c \in \mathrm{CA}_2^{\mathrm{H}},\ i = i_c \qquad (9)$$

$$\sum_{j \in T_c} \sum_{s=k+1}^{k+I_c} x_{ijs} \leq t_c + d_{ci} \qquad \forall c \in \mathrm{CA}_3^{\mathrm{H}},\ i = i_c,\ \forall k \in K \qquad (10)$$

where $K = \{0, \ldots, |S| - I_c\}$ and I_c is the length of an interval defining the slots to be considered for type 3 constraints. The corresponding constraints for $c \in \mathrm{CA}_t^{\mathrm{A}}$ and $c \in \mathrm{CA}_t^{\mathrm{HA}}$ for $t = 2$ and $t = 3$ are identical except that x_{ijs} is replaced by x_{jis} for away constraints and are x_{ijs} is replaced by $x_{ijs} + x_{jis}$ for home-away constraints.

Type 4 capacity constraints are specified by two sets of teams T_{c1} and T_{c2}, and a set of slots S_c. The constraint set is $\mathrm{CA}_4 = \mathrm{CA}_4^{\mathrm{H}} \cup \mathrm{CA}_4^{\mathrm{A}} \cup \mathrm{CA}_4^{\mathrm{HA}}$. Moreover, the home constraint set comprises$\mathrm{CA}_4^{\mathrm{H}} = \mathrm{CA}_{4a}^{\mathrm{H}} \cup \mathrm{CA}_{4b}^{\mathrm{H}}$ where $\mathrm{CA}_{4a}^{\mathrm{H}}$ and $\mathrm{CA}_{4b}^{\mathrm{H}}$ are defined by

$$\sum_{i \in T_{c1}} \sum_{j \in T_{c2}} \sum_{s \in S_c} x_{ijs} \leq t_c + d_c \qquad \forall c \in \mathrm{CA}_{4a}^{\mathrm{H}} \qquad (11)$$

$$\sum_{i \in T_{c1}} \sum_{j \in T_{c2}} x_{ijs} \leq t_c + d_c \qquad \forall c \in \mathrm{CA}_{4b}^{\mathrm{H}},\ \forall s \in S_c \qquad (12)$$

The away, and home-away constraints are similarly partitioned, with the replacement of x_{ijs} by x_{jis} and $x_{ijs}+x_{jis}$, respectively, providing the constraints.

3.3 Game constraints

The set of game constraints is denoted by GA. For each constraint $c \in \text{GA}$, a set S_c of slots and a set G_c of games are specified. A game in which team i plays at home against team j is denoted by (i, j). The constraints are

$$\sum_{(i,j)\in G_c} \sum_{s\in S_c} x_{ijs} \leq t_c + d_c \qquad \forall c \in \text{GA} \qquad (13)$$

$$\sum_{(i,j)\in G_c} \sum_{s\in S_c} x_{ijs} \geq t'_c - d_c \qquad \forall c \in \text{GA} \qquad (14)$$

where t'_c is a lower limit on the number of games from G_c that are played in the slots of S_c. For each $c \in \text{GA}$, there are lower and upper bound constraints, although a positive d_c value for one of these constraints implies that the other constraint is satisfied as a strict inequality.

3.4 Break constraints

There are two types of constraints that limit numbers of breaks. The set of type 1 breaks is denoted by $\text{BR}_1 = \text{BR}_1^{\text{H}} \cup \text{BR}_1^{\text{A}} \cup \text{BR}_1^{\text{HA}}$, where home breaks, away breaks and home-away breaks are considered. Each type 1 break constraint c specifies a team i_c and a set of slots S_c. The home break constraints of type 1 are

$$\sum_{s\in S_c} b_{is}^{\text{H}} \leq y_c + d_c \qquad \forall c \in \text{BR}_1^{\text{H}}, i = i_c \qquad (15)$$

while the away and home-away break constraints of BR_2^{A} and BR_3^{HA} are of a similar form.

A set BR_2 specifies the type 2 break constraints. Each constraint $c \in \text{BR}_2$ specifies sets S_c of slots T_c of teams. These type 2 break constraints are

$$\sum_{i\in T_c} \sum_{s\in S_c} b_{is}^{\text{HA}} \leq t_c + d_c \qquad \forall c \in \text{BR}_2 \qquad (16)$$

3.5 Fairness constraints

Fairness constraints aim to ensure that pairs of teams play approximately the same number of home games at the end of selected slots. The set of fairness constraints is denoted by FA. Each constraint $c \in \text{FA}$ specifies a pair of teams i_c and i'_c and a set of slots S_c. The constraints are

$$\sum_{j\in T} \sum_{s=1}^{\hat{s}} (x_{ijs} - x_{i'js}) \leq t_c + d_c \qquad \forall c \in FA,\ i = i_c,\ i' = i'_c,\ \forall \hat{s} \in S_c \qquad (17)$$

3.6 Separation constraints

A set SE of separation constraints aims at avoiding the two matches between a pair of teams being too close together. Each constraint $c \in \mathrm{SE}$ specifies a set T_c from which the pairs of teams are selected. The constraints are

$$\sum_{s \in S} s(x_{ijs} - x_{jis}) \geq t_c + 1 - d_c - My_{ij} \qquad \forall c \in SE,\ \forall i, j \in T_c \qquad (18)$$

3.7 Objective function

Let $C = \mathrm{CA}_1 \cup \mathrm{CA}_2 \cup \mathrm{CA}_3 \cup \mathrm{CA}_4 \cup \mathrm{GA} \cup \mathrm{BR}_1 \cup \mathrm{BR}_2 \cup \mathrm{FA} \cup \mathrm{SE}$ be the set of all the constraints in the problem, excluding the structural constraints defined in Section 3.1. Let $C = \tilde{C} \cup \bar{C}$ where $\tilde{C}$ is the set of soft and $\bar{C}$ is the set of hard constraints. Let w_c be the given unit penalty for constraint $c \in \tilde{C}$. Then the objective function can be written as

$$\text{Min} \sum_{c \in \tilde{C}} w_c d_c \qquad (19)$$

Recall that we set $d_c = 0$ for all $c \in \bar{C}$ to guarantee feasibility.

4 Matheuristic

Our Matheuristic algorithm relies on the ILP model described in Section 3. The instances created for ITC2021 are very challenging, and the ILP model is unable to solve them with Gurobi (version 9.0) using reasonable computational effort, mainly because of the size in terms of numbers of teams, soft constraints and hard constraints. Therefore, the rationale in this study is to solve smaller problems so that we can expect a competitive behaviour from the solvers that are currently available.

The fix-and-relax matheuristic approach (also known as relax-and-fix) provides a framework for producing solutions for ILPs by solving a series of smaller problems. These smaller problems are created by fixing many of the variables and then solving the ILP for the variables that are not fixed. The early research on fix-and-relax concentrated on lot-sizing problems and was initiated by Dillenberger et al. [2]. More recently, fix-and-relax has been applied in sports scheduling to the traveling umpire problem (TUP) by de Oliveira et al. [1]. In the traveling umpire problem, matches in a round-robin tournament are specified as an input, and it is required to allocate an umpire to each match so that the total distance traveled by the umpires is minimized. In the study of de Oliveira et al., there are binary assignment variables that define the allocation of umpires. The high-quality of the solutions obtained for the TUP suggests that a fix-and-relax approach could be successful for our DRRSSP.

One obvious strategy is to consider as a first step only the hard constraints and check whether the ILP in which the soft constraints are ignored is solvable with a reasonable amount of computational effort. However, we have identified that this relaxation is not sufficient for the instances provided in the ITC2021 competition because there are no instances for which a feasible solution can be obtained. Consequently, our proposed strategy is based on fixing a subset of the variables to specific values, where the method of variable fixing takes into account the features of the current solution. The other variables will remain as decision variables, thereby producing a fix-and-relax approach. Based on this idea, we introduce five different neighbourhoods to be used at the two stages of our algorithm. In the first stage, we aim to find a feasible solution ignoring the soft constraints, while the second stage considers the full model. In the first stage we propose a Variable Neighbourhood Search (VNS), and in the second stage we propose a Variable Neighbourhood Descent (VND) using the same neighbouring structure but adding a multi-start feature.

We define ILP_c to the constrained new ILP model in which we fixed some of the variables to predefined values. We also denote by $\hat{x}_{ijs}$ by the value of each variable x_{ijs} in the current solution. In Section 4.1 we define the neighbourhoods, while Sections 4.2 and 4.3 then explain the VNS and VND approaches that are used in the corresponding stages of the algorithm.

4.1 Neighbourhoods

We use the following neighbourhoods (N1-N5). All of these neighbourhoods are based on a fix-and-relax approach using the ILP, where several ILP models are solved by Gurobi with a time limit imposed (in our experiments, we ran tests with 30, 60 and 600 second per model).

- N_1 **Slots**. In this neighbourhood we select a subset of slots $\bar{S} \subseteq S$ and then we fix all the other slots as in the current solution, i.e, $x_{ijs} = \hat{x}_{ijs}$ for all $i, j \in T$ and $s \in S \setminus \bar{S}$. This neighbourhood requires a new parameter for the algorithm, n_1, which is the number of slots to be selected in S. We perform as many iterations as constraints we have violated in the model, and for each constraint violated we select the slots where there is any match scheduled that contributes to the Left Hand Side (LHS) of the constraint. We then randomly add other slots until n_1 slots are chosen.
- N_2 **Teams**. In this neighbourhood, we select a subset of teams $\bar{T} \subseteq T$ and then we fix all the matches for all pairs of teams, except the pairs $i, j \in \bar{T}$, which remain as variables. Therefore, $x_{ijs} = \hat{x}_{ijs}$ and $x_{jis} = \hat{x}_{jis}$, $\forall i \in T, j \in T \setminus \bar{T},\ s \in S$. As in N1, this neighbourhood also requires a new parameter, n_{2s}, which is the number of teams that will be considered in $\bar{T}$. Similarly to N1, for each constraint we select in $\bar{T}$ the teams that are contributing to the left-hand side of the inequalities violated in the current iteration. We then randomly add other teams until n_2 teams are chosen.
- N_3 **Rows and Columns**. We select a subset of slots $\bar{S} \subseteq S$ and a subset of teams $\bar{T} \subseteq T$, and then we fix all the matches that are not scheduled on

any slot in $\bar{S}$ and both teams are not in $\bar{T}$, i.e, $x_{ijs} = \hat{x}_{ijs}, \forall i, j \in T \setminus \bar{T},\ s \in S \setminus \bar{S}$. Slots in $\bar{S}$ are selected in a similar way as in N1, but we select only $n_1/2$ slots, and also we always select 2 teams in $\bar{T}$, and these two teams are always the ones contribute more towards the violation of the current violated constraint being analysed.

N_4 **Phased** (only for phased tournaments). We fix one half of the competition and we optimise the other half, i.e, we solve first the model where $x_{ijs} = \hat{x}_{ijs}$, $\forall s \in \{1, \ldots, |S|/2\}$, and then the model in which $x_{ijs} = \hat{x}_{ijs}$, $\forall s \in \{|S|/2+1, \ldots, |S|\}$. We solve only two models every time we use this neighbourhood.

N_5 **Home and away**. In this neighbourhood, we allow home and away matches between the same pair of teams to be swapped. Specifically, we fix $x_{ijs} = 0$, for all $i, j \in T$ and $s \in S$ such that $\hat{x}_{ijs} + \hat{x}_{jis} = 0$. This neighbourhood has no random selection, but the resulting model is generally challenging. Thus, in our computer experiments, Gurobi rarely proves optimality, and generally stops due to the time limit condition.

4.2 Checking feasibility - VNS (Hard only)

In this stage we consider the ILP where all the soft constraints except the BR2 are ignored. If a feasible solution is found, it is then used for the second stage (Section 4.3). A high-level pseudo-code is presented in Algorithm 1.

An initial solution where many of the constraints are violated is found by solving the ILP model only with the structural constraints (see Section 3.1) and ignoring all the other constraints. We refer to that solution as Sol_0, which generally has many other constraints that are violated. We then compute the value of total deviation that corresponds to solution Sol_0, and set F_0 be that value (line 2 Algorithm 1).

In line 7 we apply all the neighbourhoods until no improvement is found, where we regard the solution as being a local optimum. In line 11, if the solution is still not feasible, we then increase the coefficients in the objective function of all the d_c variables, with the exception of the BR2 (break 2 constraints), where we increase the coefficients of the b_{is} variables in the objective function if $\hat{b}_{is} = 1$. In lines 12-13 we check whether the solution is feasible.

For any d_c variables that take a value of 0 but have a positive coefficient in the objective function due to the previous iterations, we decrease their weight by taking into account the slack of the corresponding constraint in the model. If there is a positive slack then we decrease the current weight by one unit.

4.3 Optimising with soft constraints

In this stage we consider the full ILP model (with both hard and soft constraints), and the input is the final solution obtained by algorithm described in Section 4.2. A pseudo-code is provided in Algorithm 2. We first reset all the

Algorithm 1 VNS (Hard only)

```
1: procedure VNS(n1, n2)                    ▷ n1 and n2 do not change in this part
2:     Sol0 ← Compute initial solution (with objective function F0)
3:     i ← 0
4:     while Improvement do
5:         i ← i + 1
6:         Soli ← Sol0
7:         Apply N1-N5 in Soli
8:         if Improvement found then        ▷ with the current objective function being used
9:             Update Soli
10:        else
11:            Change weights on the objective function
12:        if Solution is feasible then
13:            Stop and return feasible solution Soli
```

weights on the objective function as follows. The weight on the d_c variables, where $c \in \tilde{C}$ is given by the penalty of the constraints, and the weight of all the other d_c variables related to hard constraints is set to 100. We identified that this value is sufficiently large to converge to better solutions without violating too many hard constraints, which may lead to difficulties on recovering feasibility. However, this approach indeed allows to visit infeasible solutions during the search process. Therefore, it is important to check that the solution we check in line 10 of Algorithm 2 is indeed feasible.

We start with the objective function defined by the solution obtained by the VNS (hard-only) algorithm, which corresponds to the actual penalty of the solution. Next we apply the N1-N5 neighbourhoods until we obtain a local optimum, at which stage we increase the two parameters of the algorithm, n_1 and n_2 (see Section 4.1) by one, until Gurobi cannot solve the model efficiently. In some instances where there are not many constraints, the resulting ILP models are easier to solve and, therefore, higher values for n_1 and n_2 are explored (up to $n_1 = 30$ and $n_2 = 14$ in the best cases). For the larger or more complex instances resulting in more challenging ILPs, the model becomes intractable with $n_1 = 18$ and $n_2 = 8$.

After performing some computational experiments and exploring other strategies, we found that developing a multi-start algorithm is able to produce the best quality solutions. We execute multiple runs (n_s) with the same initial solution (by introducing randomness in N1-N3) or by different initial solutions obtained by the VNS (hard-only) algorithm.

5 Computational experiments

For the computational experiments we have used 2.6GHz Intel Sandybridge processors, and each run was performed by 4 CPUs with 16GB of memory. We have used Gurobi (version 9.0) to solve the ILP models and we run the algorithm three times with 30, 60 and 600 second per model. In stage 1 of the algorithm (VNS), we run the algorithm until a feasible solution is found.

Algorithm 2 VND (hard + soft) - multistart

```
1: procedure VND(n_s)                          ▷ n_s is the number of muti-start runs
2:     for i = 1..., n_s do
3:         Sol_i ← One random solution obtained from VNS (Hard only) (Algorithm 1).
4:         n_1 = 5 and n_2 = 4
5:         exploring = True
6:         while exploring do
7:             Apply N1-N5 in Sol_i
8:             if Improvement found then          ▷ Objective function does not change
9:                 Update Sol_i
10:                if best solution improved then
11:                    Update best solution
12:            n_1 = n_1 + 1
13:            if Gurobi cannot find feasible solution of the resulting ILP then
14:                n_1 = 5
15:                n_2 = n_2 + 1
16:                if Gurobi cannot find feasible solution of the resulting ILP then
17:                    exploring = False
18:    Return best solution
```

At this stage we also considered adding the soft BR2 (break 2) constraints in order to begin stage 2 with a solution that already has a low number of breaks.

The computation time needed to find a feasible solution is shown in the first three columns of Table 1. It is worth highlighting that for almost all instances it is relatively quick to find a feasible solution, although in some instances it is more challenging and many iterations of the VNS are required to find a feasible solution, especially the instance Middle 2.

In stage 2 of the algorithm (Section 4.3), for each run of the algorithm we set $n_s = 60$ (multi-start runs), and we initially set $n_1 = 8$ and $n_2 = 6$. The computation time of one single run of the algorithm strongly depends on the instance that we are solving, but the largest amount of time on a single run was up to 6 days. Our best solutions obtained are reported in Table 1. The instances solved are divided into three sets of 15 instances each (Early, Middle and Late), which were released at different times during the ITC2021 competition.

Table 1 Best results obtained by the proposed algorithm (VNS+VND) for the ITC2021 instances. The second column (TTF) represents the time to achieve feasibility (in hours) and the third column (ITC2021) presents the best solution known for the instance at the end of the competition. The last column presents the gap from the best VNS+VND solution to the best known.

Instance		TTF (h)	VNS+VND	ITC2021	Gap
Early					
	1	< 1	362	362	0%
	2	< 1	222	160	28%
	3	< 1	1052	1012	4%
	4	< 24	536	512	4%
	5	< 24	3127	3127	0%
	6	< 1	3714	3352	10%
	7	< 1	4763	4763	0%
	8	< 1	1114	1114	0%
	9	< 1	108	108	0%
	10	< 72	3400	3400	0%
	11	< 1	4436	4436	0%
	12	< 1	510	380	25%
	13	< 1	121	121	0%
	14	< 1	47	4	91%
	15	< 1	3368	3368	0%
Middle					
	1	< 24	5177	5177	0%
	2	> 100	7381	7381	0%
	3	< 24	9800	9701	1%
	4	< 1	7	7	0%
	5	< 1	494	413	16%
	6	< 1	1275	1125	12%
	7	< 1	2049	1784	13%
	8	< 1	129	129	0%
	9	< 1	450	450	0%
	10	< 1	1250	1250	0%
	11	< 1	2608	2511	4%
	12	< 1	923	911	1%
	13	< 1	282	253	10%
	14	< 1	1323	1172	11%
	15	< 1	965	495	49%
Late					
	1	< 1	1969	1969	0%
	2	< 1	5400	5400	0%
	3	< 1	2369	2369	0%
	4	< 1	0	0	-
	5	< 3	2218	1939	13%
	6	< 1	923	923	0%
	7	< 1	1652	1558	6%
	8	< 1	934	934	0%
	9	< 1	563	563	0%
	10	< 3	2031	1988	2%
	11	< 1	226	207	8%
	12	< 1	3912	3689	6%
	13	< 1	2110	1820	14%
	14	< 1	1363	1206	12%
	15	< 1	40	20	50%

The proposed algorithm obtained the best results in 22 out of 45 instances by the end of competition, and in further 7 instances the best known result is within 6% of the solution obtained by the proposed algorithm.

6 Concluding remarks

In this study, we propose a novel matheuristic algorithm having two stages to solve the DRRSSP. In the first stage, we address the feasibility problem by using a VNS framework, where a second stage we optimizes the soft constraint violations by using a multi-start algorithm with a VND. Both the VNS and the VND use the same neighbourhood structure that combines five different neighbourhoods. The results obtained shows that both stages of the algorithm perform well, being able to prove optimality in one instance (Late 4, with an objective value of 0) and finding a feasible solution in all 45 instances of the ITC2021 challenge competition.

Acknowledgements The authors acknowledge the use of the IRIDIS High Performance Computing Facility, and associated support services at the University of Southampton, in the completion of this work. Useful comments by two anonymous referees on an earlier draft of our paper have helped in creating this improved version.

Conflict of interest

The authors declare that they have no conflict of interest.

References

1. de Oliveira L, de Souza CC, Yunes T: Improved bounds for the traveling umpire problem: A stronger formulation and a relax-and-fix heuristic. *European Journal of Operational Research* **236** 592–60 (2014)
2. Dillenberger C, Escudero LF, Wollensak A, Zhang W: On practical resource allocation for production planning and scheduling with period overlapping setups. *European Journal of Operational Research* **75** 275-286 (1994)
3. Durán G, Guajardo M, Miranda J, Sauré D, Souyris S, Weintraub A, Wolf R: Scheduling the Chilean soccer league by integer programming. *Interfaces* **37(6) 539–552 (2007)**
4. **Goossens DR, Spieksma FCR: Soccer schedules in Europe: An overview.** *Journal of Scheduling* **15 641-651 (2011)**
5. **Knust, S: Classification of sports scheduling literature. http://www2.informatik.uni-osnabrueck.de/knust/sportssched/sportlit_class/. Accessed 07/06/2021.**
6. **Van Bulck D, Goossens D, Schönberger J, Guajardo M: RobinX: A three-field classification and unified data format for round-robin sports timetabling.** *European Journal of Operational Research* **280(2), 568-580 (2020)**
7. **Van Bulck D, Goossens D, Beliën, J, Davari, M: ITC—Sports Timetabling Problem Description and File Format. https://www.sportscheduling.ugent.be/ITC2021/ (2021)**
8. **Van Bulck D, Goossens D, Beliën, J, Davari, M: The Fifth International Timetabling Competition (ITC 2021): Sports Timetabling.** *Proceedings of MathSport International 2021 Conference, MathSport* **pp. 117-122. (2021)**

Multi-Neighborhood Simulated Annealing for the Sport Timetabling Competition ITC2021

Roberto Maria Rosati · Matteo Petris · Luca Di Gaspero · Andrea Schaerf

1 Introduction

Sport timetabling is an active research field, mainly due to the commercial interest in the maximization of fan attendance (in person or remotely) to sport events. Among the various possible structures for sport competitions, the round-robin tournament is the most frequently used for most team sports.

We describe in this paper the solver that we developed for the Sport Timetabling Competition ITC2021, a three-stage Simulated Annealing approach, that makes use of a portfolio of six different neighborhoods. Five of them are classical ones, already proposed in the literature, whereas the sixth one, named *PartialSwapTeamsPhased*, is a variant of one of them that we specifically designed to deal with phased instances. Our solver has many parameters and it has been tuned using the F-RACE procedure (Birattari et al., 2010), upon a set of experimental configurations designed using the Hammersley point set (Hammersley and Handscomb, 1964).

Overall, the final outcome is that the three-stage Simulated Annealing solver is able to find a feasible solution on 44 out of 45 instances and ranked second in both the first competition milestone and the final round.

2 Related Work

Interest in Sport Timetabling started growing from the 70s, with initial research by Gelling (1973), Russell (1980), Wallis (1983), de Werra (1981), and de Werra et al. (1990). Due to its complexity, (Rosa and Wallis, 1982; Dinitz et al., 1994), several metaheuristic and heuristic algorithms have been proposed for the Sport Timetabling Problem throughout the years. During the years 2000s, new neighborhoods for local-search-based metaheuristics were developed by Ribeiro and Urrutia (2004), Anagnostopoulos et al. (2006) and Di Gaspero and Schaerf (2007), as a consequence of rising interest in the Traveling Tournament Problem (Easton et al., 2001). More recent contributions to (meta)heuristic methods for Sport Timetabling are found in Lewis and Thompson (2011), Costa et al. (2012) and Januario and Urrutia (2016). Finally, Van Bulck et al. (2020b) proposed a unified data format for the round-robin sports timetabling, named RobinX, also employed in the Sport Timetabling Competition ITC2021 (Van Bulck et al., 2021). For a more complete bibliographic revision for sport timetabling we redirect the reader to Rasmussen and Trick (2008) and Kendall et al. (2010).

Roberto Maria Rosati, Luca Di Gaspero, and Andrea Schaerf
DPIA, University of Udine, via delle Scienze 206, 33100 Udine, Italy
E-mail: {robertomaria.rosati,luca.digaspero,andrea.schaerf}@uniud.it

Matteo Petris
Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
E-mail: matteo.petris@inria.fr

3 Problem Description

Many variants of the round-robin tournament problem have been discussed in the literature. We consider here the version proposed for the International Timetabling Competition ITC2021 (Van Bulck et al., 2021), which takes into account five types of constraints collected from real-world cases: capacity constraints, game constraints, break constraints, fairness constraints and separation constraints. This formulation has the peculiarity that every single specific constraint can be stated as either hard or soft, as they may express fundamental properties of the timetable and must be satisfied (*hard* version), or they may express preferences and can be violated (*soft* version). Another characteristic of the ITC2021 formulation is that it has abandoned the classical *mirrored* structure in which the second leg is identical to the first one, with home and away positions swapped. That is, the structure of ITC2021 instances is either completely free or *phased*, meaning that a team has to match all other teams in each leg (but not in the same order).

4 Solution Method

We designed a three-stage multi-neighborhood Simulated Annealing for the solution of the problem. As search space we consider the set of all two-leg round-robin timetables. The multi-neighborhood is a hexamodal neighborhood made up by a portfolio of six different local search neighborhoods, which are specifically tailored for the sport timetabling problem. Five of them, called *SwapHomes*, *SwapTeams*, *SwapRounds*, *PartialSwapTeams*, and *PartialSwapRounds*, are adaptations of classical ones from Ribeiro and Urrutia (2004), Anagnostopoulos et al. (2006), and Di Gaspero and Schaerf (2007). The sixth one is a novel neighborhood called *PartialSwapTeamsPhased*, specifically designed to deal with phased instances. It is based on the concept of *mixed phase*, which is a partition of the timetable in two subsets, named mixed legs, where each couple of teams play together, respectively, for the first and for the second time. This definition is independent from the current satisfaction of the phase constraint. The move considers two teams and a set of rounds and swaps the positions of the two teams in the matches in the given set of rounds. A prerequisite is that the matches involved in the move must all belong to the same mixed leg. In this way, the move *PartialSwapTeamsPhased* swaps a subset of teams inside one of the two mixed legs and it is invariant with respect to the phase.

The metaheuristic employed is basically the classical Simulated Annealing defined by Kirkpatrick et al. (1983). The search is executed is three distinct sequential stages. Specifically, the first stage starts its search either from a random or from a greedy solution, the second and the third stages are warm-started with the output of the previous stage. The differences between the stages consist in the restrictions applied to the search space and in the exclusion or inclusion of certain constraints.

5 Experimental Results

Our code was developed in C++ and compiled with GNU g++ version 9.3.0 on Ubuntu 20.04.2 LTS. The tuning phase was partially performed on a cluster of virtual machines provided by the CINECA consortium. All the other experiments presented in this section were run on a machine equipped with AMD Ryzen Threadripper PRO 3975WX processor with 32 cores, hyper-threaded to 64 virtual cores, with base clock frequency of 3.5 GHz, and 64 GB of RAM. In both settings, one single virtual core is used for each experiment.

Table 1 reports the results obtained by the solver. The column *Best solution found* reports the best solution that our solver was able to find in all experiments. Some of these values are those that we submitted to the ITC2021 competition, others have been found in later experiments. When no feasible solution has been found, the number of hard violations followed by a letter H is reported. Next columns, labeled *Average values*, report the data obtained in a set of experiments that we run independently from the competition, in order to extract information on the average behavior of the algorithm in its final configuration. At least 48 runs per instance were performed to collect these data. Columns *Cost* and *Time* report, respectively, the average values of the objective function and the average time needed for a complete run of the three stages. Regarding the average cost, the value is computed only on feasible solutions. Column *Feasible* reports the ratio between feasible solutions and total runs. Finally, column *Best known cost* contains the best known results at the moment this

Instance	Best solution found	Average values			Best known cost
		Cost	Time (s)	Feasible	
Early_1	423	540.7	5667	1.00	362
Early_2	318	384.6	14843	1.00	145
Early_3	1068	1176.5	12194	1.00	992
Early_4	556	1007.8	8759	0.56	507
Early_5	4117	-	28517	0.00	3127
Early_6	3927	4543.0	35161	1.00	3325
Early_7	5205	6721.7	37486	1.00	4763
Early_8	**1051**	1151.9	21394	1.00	1051
Early_9	132	228.7	10324	1.00	108
Early_10	4986	-	35856	0.00	3400
Early_11	4526	5784.5	43692	1.00	4426
Early_12	1010	1200.2	14726	1.00	380
Early_13	173	233.8	19675	1.00	121
Early_14	63	82.3	5616	1.00	4
Early_15	3556	3945.8	46714	1.00	3362
Middle_1	5657	6075.0	26290	0.06	5177
Middle_2	5H	-	26890	0.00	7381
Middle_3	**9542**	11403.1	44748	0.23	9542
Middle_4	16	33.0	5660	1.00	7
Middle_5	510	624.4	6223	1.00	413
Middle_6	1701	2186.3	21350	1.00	1120
Middle_7	2203	2452.7	16303	1.00	1783
Middle_8	136	196.6	19717	1.00	129
Middle_9	640	772.1	17610	1.00	450
Middle_10	1357	1687.5	14432	1.00	1250
Middle_11	2696	2996.5	43876	1.00	2446
Middle_12	950	1054.2	14599	1.00	911
Middle_13	362	479.3	15687	1.00	252
Middle_14	**1172**	1304.6	37483	1.00	1172
Middle_15	985	1099.7	8704	1.00	485
Late_1	2021	2372.7	20242	1.00	1922
Late_2	5715	6085.5	41432	0.49	5400
Late_3	2457	2718.0	18327	1.00	2369
Late_4	**0**	0.0	2354	1.00	0
Late_5	2341	-	9190	0.00	1923
Late_6	930	1121.3	7121	1.00	923
Late_7	1765	2226.5	22959	1.00	1558
Late_8	997	1155.3	11285	1.00	934
Late_9	715	881.2	25963	1.00	563
Late_10	2571	3527.3	32511	0.05	1945
Late_11	207	289.3	15891	1.00	202
Late_12	3944	4830.6	35513	1.00	3428
Late_13	1868	2285.5	21006	1.00	1820
Late_14	**1202**	1326.3	39160	1.00	1202
Late_15	60	82.8	6434	1.00	20

Table 1: Best and average results

article is written, according to data published on the website of the competition (Van Bulck et al., 2020a). When the current known best was determined by our solver, the value in the corresponding column is marked in bold. Overall, our solver could find at least one feasible solution on 44 out of 45 instances. According to data, in its final configuration it manages to determine very easily a feasible solution on 36 instances, which are characterized by a feasibility ratio of 1.00, as it can be observed in column *Feasible* of Table 1. The other instances appear to be harder to solve for the algorithm. In particular, instances Early_5, Early_10, Middle_2, and Late_5 result to be considerably challenging, as feasible solutions are found just sporadically.

We also assessed the impact of the new neighborhood *PartialSwapTeamsPhased*. To do so, we run an additional set of experiments on phased instances with and without making use of *PartialSwapTeamsPhased*. We highlight that employing the new neighborhood *PartialSwapTeamsPhased* brings benefit to the majority of the 22 phased instances: in 17 of these the average cost improvement is 4.24%. One of the remaining five was solved to feasibility only in the configuration that employs *PartialSwapTeamsPhased*. The other four are not solved by any of the two configurations in the given number of runs.

6 Conclusions

In this study, we considered the version of the Sport Timetabling Problem proposed for the ITC2021 competition. We tackled the problem employing a three-stage multi-neighborhood Simulated Annealing approach, which makes use of six different neighborhoods. In particular, the neighborhood that we named PartialSwapTeamsPhased is a novel contribution. Finally, we performed a parameter tuning for the solver using the F-RACE procedure that allowed us to find a set of parameters values for this problem.

This approach managed to find a feasible solution for 44 out of the 45 instances proposed by the competition. Feasible solutions were found rather easily for most of the instances, however the metaheuristic struggled to produce feasible solutions for certain instances, even in long execution times. The results obtained by the Simulated Annealing approach allowed us to rank second out of 13 participants in the final ranking of the competition.

Future work will be devoted to improve the results and performances on both the considered instances and on other benchmark instances for round-robin tournament. We think that relevant advancements can be achieved through a wider study and application of the PartialSwapTeamsPhased neighborhood on a larger set of instances. Possible research directions may also include the definition and integration of new neighborhoods in the Simulated Annealing algorithm, and the implementation and evaluation of new greedy techniques to generate different initial solutions, not restricted to the canonical pattern. Further research may also be committed to develop a matheuristic approach, such as Large Neighborhood Search (LNS), which embeds exact methods in our Simulated Annealing algorithm.

Acknowledgments.

The authors thank the CINECA consortium for the access to the Cloud Computing resources under the Italian Super Computing Resource Allocation Project grant IA4ITC_C.

The authors are also grateful to the ITC2021 organizers David Van Bulck, Dries Goossens, Jeroen Beliën, and Morteza Davari, and to the anonymous reviewers for their very helpful and detailed comments.

References

Anagnostopoulos A, Michel L, Van Hentenryck P, Vergados Y (2006) A simulated annealing approach to the traveling tournament problem. Journal of Scheduling 9(2):177–193

Birattari M, Yuan Z, Balaprakash P, Stützle T (2010) F-race and iterated F-race: An overview. In: Experimental methods for the analysis of optimization algorithms, Springer, Berlin, pp 311–336

Costa FN, Urrutia S, Ribeiro CC (2012) An ils heuristic for the traveling tournament problem with predefined venues. Annals of Operations Research 194(1):137–150

Di Gaspero L, Schaerf A (2007) A composite-neighborhood tabu search approach to the traveling tournament problem. Journal of Heuristics 13(2):189–207

Dinitz JH, Garnick DK, McKay BD (1994) There are 526,915,620 nonisomorphic one-factorizations of k_{12}. Journal of Combinatorial Design 2:273–285

Easton K, Nemhauser G, Trick M (2001) The traveling tournament problem description and benchmarks. In: Seventh International Conference on the Principles and Practice of Constraint Programming (CP 99), Springer-Verlag, LNCS, vol 2239, pp 580–589

Gelling EN (1973) On 1-factorizations of the complete graph and the relationship to round robin schedules. PhD thesis

Hammersley JM, Handscomb DC (1964) Monte Carlo methods. Chapman and Hall, London

Januario T, Urrutia S (2016) A new neighborhood structure for round robin scheduling problems. Computers & Operations Research 70:127–139

Kendall G, Knust S, Ribeiro C, Urrutia S (2010) Scheduling in sports: An annotated bibliography. Computers and Operations Research 37(1):1–19

Kirkpatrick S, Gelatt D, Vecchi M (1983) Optimization by simulated annealing. Science 220:671–680

Lewis R, Thompson J (2011) On the application of graph colouring techniques in round-robin sports scheduling. Computers & Operations Research 38(1):190–204

Rasmussen RV, Trick MA (2008) Round robin scheduling–a survey. European Journal of Operational Research 188(3):617–636

Ribeiro CC, Urrutia S (2004) Heuristics for the mirrored traveling tournament problem. In: Proc. of the 5th Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT-2004), pp 323–342

Rosa A, Wallis WD (1982) Premature sets of 1-factors or how not to schedule round robin tournaments. Discrete Applied Mathematics 4:291–297

Russell KG (1980) Balancing carry-over effects in round robin tournaments. Biometrika 67(1):127–131

Van Bulck D, Goossens D, Beliën J, Davari M (2020a) Website of the fifth international timetabling competition (itc 2021): Sports timetabling. `https://www.sportscheduling.ugent.be/ITC2021/`, last accessed: 16/11/2021

Van Bulck D, Goossens D, Schönberger J, Guajardo M (2020b) Robinx: A three-field classification and unified data format for round-robin sports timetabling. European Journal of Operational Research 280(2):568–580

Van Bulck D, Goossens D, Beliën J, Davari M (2021) The fifth international timetabling competition (itc 2021): Sports timetabling. In: MathSport International 2021 Conference, pp 117–122

Wallis WD (1983) A tournament problem. Journal of the Australian Mathematical Society Series B 24:289–291

de Werra D (1981) Scheduling in sports. In: Hansen P (ed) Studies on Graphs and Discrete Programming, North Holland, pp 381–395

de Werra D, Jacot-Descombes L, Masson P (1990) A constrained sports scheduling problem. Discrete Applied Mathematics 26:41–49

Reprobate at ITC 2021

Pseudoboolean Optimisation for RobinX Sports Timetabling

Martin Mariusz Lester

Abstract We report on the development of *Reprobate*, a tool for solving sports timetabling problems in the RobinX format. The main approach used by the tool is to encode a timetabling problem using pseudoboolean (PB) constraints and solve it using existing solvers. Initially, it uses a monolithic encoding that attempts to satisfy all constraints simultaneously. If this finds a feasible solution, the tool can improve it using a separate encoding that tunes only the home/away pattern while fixing the pairings of teams. Furthermore, *Reprobate* employs a small portfolio of different solvers and encoding variations and returns the best solution found by any of them.

We entered *Reprobate* in the International Timetabling Competition 2021. It was able to find feasible solutions for the majority of instances, although it struggled to handle large break constraints. For those instances where it could initially find a solution, the combination of tuning, use of a portfolio of solvers, and variations in encoding yielded an average reduction in solution cost of 23%.

Keywords pseudoboolean constraints · sports timetabling

1 Introduction

The 2021 edition of the International Timetabling Competition (ITC) was based around solving sports timetabling problems presented in a restricted version of the RobinX format [5]. The competition was limited to time-constrained double round-robin (2RR) tournaments. In such a tournament, each of n teams plays every other team exactly twice over $2(n-1)$ slots: once at home and once away. The RobinX format allows a wide range of other constraints to be specified, for example concerning the slots in which particular matches may occur, or limiting the number of breaks where a team has consecutive home or consecutive away games [4]. Some

M. M. Lester
Department of Computer Science
University of Reading
United Kingdom
E-mail: m.lester@reading.ac.uk

of these constraints are hard, meaning that they must be satisfied. Others are soft, meaning that they may be violated, but there is a cost for doing so. The goal is to find a solution that minimises the sum of costs of violated constraints. See the competition report for a full description [3].

We developed the tool *Reprobate* for the competition. In Section 2 we outline the approach used by the tool and some of the optimisations we developed. We discuss its performance in Section 3, before concluding in Section 4.

Reprobate is implemented as a series of Perl scripts that invoke existing pseudoboolean (PB) solvers such as *clasp* [6] and *Sat4J* [1]. The tool [8] and solvers are freely online under an open source licence.

2 Approach

At its core, *Reprobate* extracts the constraints from a RobinX timetabling problem and encodes them monolithically as a pseudoboolean (PB) optimisation problem, specifically a Weighted Boolean Optimisation (WBO) problem. Then, it uses an existing PB solver to solve the problem. If the solver is successful, *Reprobate* extracts the timetable from the solution. To make the system more competitive, we implemented three optimisations: a portfolio of solvers, a tuning process, and some variations on the encoding.

2.1 Pseudoboolean Constraints

The pseudoboolean constraint satisfaction (PBS) [10] problem is a generalisation of the well-known boolean satisfiability (SAT) problem that makes it easy to express cardinality constraints. In a weighted boolean optimisation (WBO) problem, these constraints can be given costs, with the goal being to minimise the sum of costs. SAT-based approaches to sports timetabling have been considered before [11,7], but are relatively uncommon.

Our encoding uses the following sets of boolean variables:

1. $M_{t_1,t_2,s}$ — true just if team t_1 plays home against team t_2 in slot s;
2. $H_{t,s}$ — true just if team t plays home in slot s;
3. $B_{t,s,h}$ — true if team t has a home break ($h = 0$)/an away break ($h = 1$) in slot s, with $s > 0$.

The timetable is determined by the M variables; the remaining auxiliary variables make it easier to express certain constraints.

Linear PB constraints are equivalent to 0-1 Integer Linear Programming (0-1 ILP), which is itself a restriction of Mixed Integer Programming (MIP). However, there is an important practical difference between PB solvers and 0-1 ILP solvers. PB solvers tend to use techniques from SAT solvers, such as clause-driven conflict learning (CDCL). In contrast, 0-1 ILP solvers tend to use techniques from linear programming (LP). According to Berthold and others [2], "feasibility problems with many constraints that have 0/1 coefficients only" tend to work best with PB solvers, but "instances with many inequalities with arbitrary coefficients" tend to work best with MIP solvers. Our encoding uses only +/- 1 coefficients, so all constraints are either pure SAT constraints or cardinality constraints. We investigated

the possibility of using commercial MIP solvers such as Gurobi and CPLEX with our encoding, but found that they performed very poorly.

2.2 Portfolio of Solvers

It is well-known that different SAT solvers perform well on different SAT instances. Therefore, if one wishes to solve a particular instance, it is most effective to run several different solvers in parallel and see which one (if any) finishes first. This *portfolio* approach is used by the most competitive SAT solvers, although it is banned from the main track of the SAT competition, as portfolio solvers tend not to contribute towards the development of new techniques. The same is true for PB instances, so in *Reprobate*, we use 2 solvers (*clasp* and *Sat4J*) with a range of options.

2.3 Portfolio of Encodings

SAT and PB solvers are often sensitive to the exact encoding of the constraints in a problem. For this reason, we implemented 6 variations on our initial encoding, configurable using command-line *switches*. As the number of variations is small, by default *Reprobate* simply tries all of them individually and picks the best solution found by any of them.

2.4 Tuning Process

Many PB solvers are complete, in the sense that they are guaranteed eventually to find an optimal solution if one exists. However, in practice, this often takes infeasibly long, and even if a solver finds a feasible solution, it may not be optimal. In previous work on generating tournament timetables for the 4-player game mahjong [9], we found that we could often improve a timetable using a separate encoding that fixes the groupings of players in a particular round. It is possible that the optimal solution to the original problem is no longer a solution to this modified encoding, but in practice this is not a problem, as it always produces at least as good a solution as we could find without it.

Reprobate uses a similar technique. After it has found a feasible solution, it generates a separate encoding of the timetabling problem in which pairings of teams in each slot are fixed, but not the choice of which team plays home and which team plays away. Again, it solves this using a PB solver and extracts a timetable from the solution. If the *tuned* timetable is an improvement on the best timetable produced by the monolithic encoding, *Reprobate* returns that; otherwise, it returns the original solution.

3 Results

Using the default encoding and the best solver from our portfolio (*clasp* with the *crafty* preset) with a timeout of 600 s, *Reprobate* was able to find feasible

	01	02	03	04	05	06	07	08
Early	*	*	4884			*		5584
Middle				99	2901	3235	8563	1189
Late	3683		7784	0		3678	*	4583

	09	10	11	12	13	14	15
Early	4858		*	2070	*	3489	7443
Middle	3530		4263	4950	6199	*	7590
Late	2940			*	8564	3712	5910

Table 1 Baseline performance of *Reprobate* on ITC 2021 instances. Figures show objective score with default encoding, *clasp (crafty)* PB solver, a timeout of 600 s and no tuning phase. Lower is better. Instances marked with * can be solved using other settings. All results were generated on a machine running Debian Linux 10 with a 3.4 GHz Intel Core i5-7500 CPU and 64 GB of RAM.

solutions for 25 out of 45 problem instances (56%) in the ITC 2021. Table 1 shows the corresponding objective scores, which we adopt as our baseline. Applying the portfolio of solvers and encodings during the ITC 2021, we increased this to 29 out of 45 (64%). The addition of another encoding variation after the competition [7] increased this to 33 (73%).

This was not competitive in the ITC 2021, where it did not place in the top half of the results. However, according to the competition report, "for most problem instances, a straightforward integer programming formulation could not even generate a feasible solution", so it is better than that. Of the instances *Reprobate* could not solve during the competition, all except Middle 3 contained a large, hard BR2 constraint that limited the number of breaks permitted in the timetable.

Focusing now just on the 25 instances in our baseline, Figure 1 shows the relative improvement made by our optimisations, as well as the best scores submitted during the competition; Table 2 shows the raw numbers. In combination, the optimisations we made yielded an average decrease in objective of 23%, although this is still some way off the best solutions found for most instances.

4 Conclusion

We have developed *Reprobate*, the first PB-based tool for solving RobinX sports timetabling problems. To the best of our knowledge, this is the first general-purpose sports timetabling tool that uses the OPB file format (for PBS/WBO problems) and its associated solvers. While *Reprobate* is effective for many timetabling problems, it struggles to handle large break constraints. This is a known limitation of SAT-based approaches, for which he have implemented some existing mitigations, but more work is needed to investigate how best to handle these.

References

1. Berre, D.L., Parrain, A.: The sat4j library, release 2.2. J. Satisf. Boolean Model. Comput. **7**(2-3), 59–6 (2010). DOI 10.3233/sat190075. URL https://doi.org/10.3233/sat190075
2. Berthold, T., Heinz, S., Pfetsch, M.: Solving pseudo-boolean problems with scip. Tech. Rep. 08-12, ZIB, Takustr. 7, 14195 Berlin (2008)

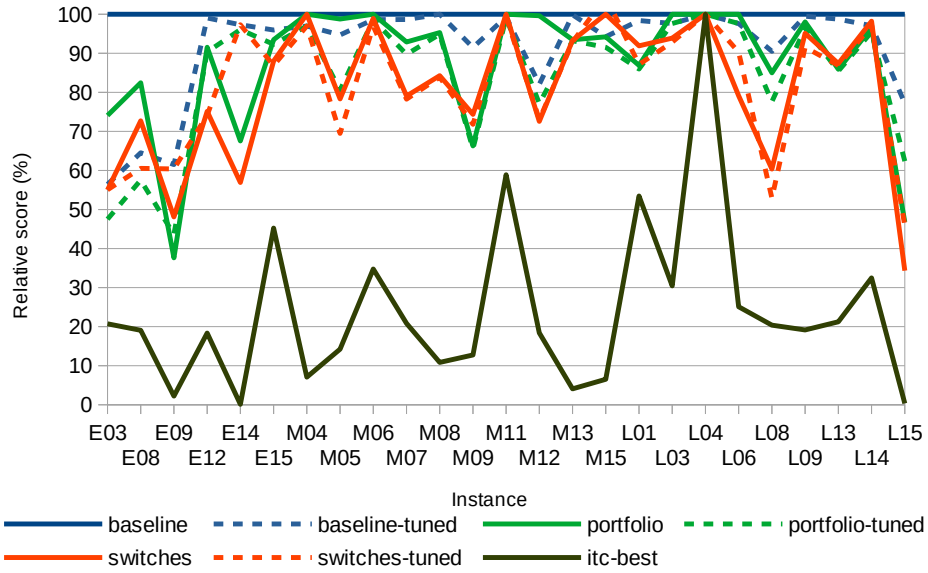


Fig. 1 Effect of tuning on objective, for the baseline, the portfolio and different switches. ITC best solutions included for comparison.

instance	baseline	baseline-tuned	portfolio	portfolio-tuned	switches	switches-tuned	itc-best
E03	4884	2757	3618	2321	2686	2689	1012
E08	5584	3603	4604	3212	4060	3380	1064
E09	4858	2988	1828	2118	2337	2933	108
E12	2070	2050	1895	1870	1555	1535	380
E14	3489	3395	2357	3351	1986	3395	4
E15	7443	7142	6957	6870	6551	6466	3368
M04	99	96	99	96	99	96	7
M05	2901	2747	2865	2337	2272	2016	413
M06	3235	3190	3235	3190	3200	3135	1125
M07	8563	8447	7954	7681	6769	6701	1784
M08	1189	1189	1133	1128	1002	997	129
M09	3530	3230	2340	2315	2625	2535	450
M11	4263	4233	4263	4223	4263	4233	2511
M12	4950	4061	4931	3817	3592	3614	911
M13	6199	6190	5793	5778	5785	5758	253
M15	7590	7153	7151	6947	7590	7931	495
L01	3683	3623	3201	3166	3385	3209	1969
L03	7784	7599	7784	7599	7308	7228	2369
L04	0	0	0	0	0	0	0
L06	3678	3590	3678	3590	2910	3320	923
L08	4583	4148	3895	3557	2766	2424	934
L09	2940	2925	2882	2837	2796	2691	563
L13	8564	8456	7357	7317	7482	7441	1820
L14	3712	3602	3583	3543	3646	3602	1206
L15	5910	4560	2765	3685	2030	2710	20

Table 2 Effect of tuning on objective, for the baseline, the portfolio and different switches. ITC best solutions included for comparison.

3. Bulck, D.V., Goossens, D.R., Beliën, J., Davari, M.: The fifth international timetabling competition (itc 2021): Sports timetabling. Proceedings of MathSport International 2021 Conference pp. 117–122
4. Bulck, D.V., Goossens, D.R., Beliën, J., Davari, M.: Itc2021 — sports timetabling problem description and file format (2020)
5. Bulck, D.V., Goossens, D.R., Schönberger, J., Guajardo, M.: Robinx: A three-field classification and unified data format for round-robin sports timetabling. Eur. J. Oper. Res. **280**(2), 568–580 (2020). DOI 10.1016/j.ejor.2019.07.023. URL https://doi.org/10.1016/j.ejor.2019.07.023
6. Gebser, M., Kaufmann, B., Schaub, T.: Conflict-driven answer set solving: From theory to practice. Artif. Intell. **187**, 52–89 (2012). DOI 10.1016/j.artint.2012.04.001. URL https://doi.org/10.1016/j.artint.2012.04.001
7. Horbach, A., Bartsch, T., Briskorn, D.: Using a sat-solver to schedule sports leagues. J. Sched. **15**(1), 117–125 (2012). DOI 10.1007/s10951-010-0194-9. URL https://doi.org/10.1007/s10951-010-0194-9
8. Lester, M.: Reprobate. URL https://doi.org/10.5281/zenodo.5084254
9. Lester, M.M.: Scheduling reach mahjong tournaments using pseudoboolean constraints. In: C.M. Li, F. Manyà (eds.) Theory and Applications of Satisfiability Testing – SAT 2021, pp. 349–358. Springer International Publishing, Cham (2021). URL https://doi.org/10.1007/978-3-030-80223-3_24
10. Manquinho, V.M., Roussel, O.: The first evaluation of pseudo-boolean solvers (pb'05). J. Satisf. Boolean Model. Comput. **2**(1-4), 103–143 (2006). DOI 10.3233/sat190018. URL https://doi.org/10.3233/sat190018
11. Zhang, H., Li, D., Shen, H.: A SAT based scheduler for tournament schedules. In: SAT 2004 - The Seventh International Conference on Theory and Applications of Satisfiability Testing, 10-13 May 2004, Vancouver, BC, Canada, Online Proceedings (2004). URL http://www.satisfiability.org/SAT04/programme/74.pdf