Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling (ICAPS 2023)

Dynamic Weight Setting for Personnel Scheduling with Many Objectives

Lucas Kletzander, Nysret Musliu

Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling
DBAI, TU Wien, Karlsplatz 13, 1040 Vienna, Austria
{lucas.kletzander,nysret.musliu}@tuwien.ac.at

Abstract

When large sets of constraints and objectives are combined in a practical optimization problem, managing all these potentially conflicting goals can become very difficult and might require to solve an instance multiple times. First, an instance might be infeasible with the current constraints, in which case our system introduces a novel violation score to help identify the constraints that need to be relaxed for the next run. Second, multiple objectives are often combined using a linear combination with hand-crafted weights, which are very difficult to set such that the result matches the expectations regarding the balance between individual objectives. Instead, the user can tell our system particular thresholds for the expected changes in objectives, e.g., to reduce objective 1 by 10 % while not increasing objective 2 by more than 5 %. Dynamic weight setting automatically adapts the weights to reach these thresholds or uses the violation scores to explain reasons for not reaching thresholds. It can not only be used for soft constraints, but also to determine weights when hard constraints are internally represented as soft constraints in metaheuristics. While the methodology is general, we have implemented it in the context of a personnel scheduling framework of our industry partner and present a detailed evaluation on the domain of Bus Driver Scheduling, where its benefits can be seen in multiple scenarios.

Introduction

When dealing with scheduling problems in practice, a high variety of different objectives and constraints can be found in different application scenarios. This is especially true in the area of personnel scheduling, where a wide range of legal requirements, collective agreements, company agreements and company goals affect the generation of a schedule in various, often conflicting ways. Finding a solution that is both efficient and respects employee well-being is of crucial importance. Employees are usually one of the highest cost factors for companies, and on the other hand bad schedules can have a tremendous negative impact on the social life and health of the employees. In the current time of employee shortages these issues are even more important.

The wide range of different rules leads to the issue that often no two problems for different companies are exactly the same. While there are common constraints and objectives that are frequently reused, like the objective to minimize paid working time, other constraints are very specific. Further, the objective to minimize is typically a (usually linear) combination of various different goals.

In the optimization system of our industry partner, this led to the development of a modular optimization tool that allows to flexibly add or remove constraints and objectives. While this approach works well to deal with the large amount of variation, the growing system now suffers from new issues based on the interactions of an increasing number of configuration options.

First, some problems might not admit feasible solutions due to being over-constrained, however, the large amount of constraints makes it hard to identify the root cause of such infeasibilities. Second, the large number of objectives makes it very difficult to set appropriate weights by hand, especially since the same change in weight might produce different changes in the result depending on the objectives and their combination. Iterative manual refinement of constraints or weights is often necessary to reach the desired results.

This paper addresses these issues by introducing weight setting to this scheduling system, providing a systematic optimization of the weights internally used by the system, while thresholds allow easier interaction for the user. We further introduce a novel score for constraint violations, making it easier for the user to determine the cause of infeasibilities, and providing recommendations for the constraints that should be relaxed to resolve the conflicts.

The weight setting procedure is evaluated on the exemplary domain of Bus Driver Scheduling based on an Austrian collective agreement, since this domain provides a public benchmark with a range of complex hard and soft constraints. The evaluation shows that our system is able to adapt internal weights to provide feasible solutions even in highly constrained scenarios, and the capability to suggest the correct constraints for relaxation in case of infeasibilites. Evaluating the threshold mechanism shows that objectives can successfully be guided in the required direction without the need of directly manipulating weights.

Related Work

Personnel scheduling occurs in many practical scenarios in different variations, several reviews give an overview (Ernst

et al. 2004; Burke et al. 2004; Van den Bergh et al. 2013; De Bruecker et al. 2015; Özder, Özcan, and Eren 2020). However, it has often been noted that only a small part of the academic work is put into practice (Böðvarsdóttir et al. 2022). Identified issues include the fact that the full problem complexity needs to be modeled by the solver, and flexibility for changes in the specification is needed.

However, a wide range of constraints and objectives leads to the question how to integrate these different goals. The PhD thesis by Böðvarsdóttir (2021) contains a recent survey that shows that the majority of methods uses weighted sums to deal with multiple objectives. On the other hand, weights for such sums are known to be very hard to properly adjust (Gärtner et al. 2018), and can even lead to counter-intuitive results (Petrovic and Vanden Berghe 2012). Multi-objective methods that try to approximate a Pareto front, predominantly evolutionary methods (Coello et al. 2007), are usually not designed to deal with the large number of objectives.

Not much work has been done in the area of automatically setting the appropriate weights for each instance. There has been work using historic data for weight extraction (Mihaylov et al. 2016), and similar ideas are used in guided local search (Voudouris, Tsang, and Alsheddy 2010), but new ideas have only recently been introduced as Behind-the-Scenes Weight Tuning (Böðvarsdóttir, Smet, and Berghe 2020), based on a concept of acceptance thresholds (user-defined goals for soft constraints or combinations of soft constraints) that have been introduced in an earlier paper (Böðvarsdóttir et al. 2019) and also used in further work (Böðvarsdóttir et al. 2021).

While these authors present a general methodology that our work builds upon, our system provides the novel contributions to deal with meta-heuristics that need to internally treat hard constraints as soft constraints, to deal with infeasible instances in a way that gives adequate feedback to the user to determine the cause of infeasibility, and to extend the system to new real-world problem domains including the evaluation on public benchmark instances.

Weight Setting System

This section formally defines the weight setting methods as well as the computation of the violation score. Independent of the specific problem domain the constraints and thresholds are given as follows:

- A set of hard constraints $\mathcal{H} = \{h_1, \ldots, h_H\}$
- Hard constraint weights $w_{h_i} \in \mathbb{R}^+$ for $i \in 1, \ldots, H$
- A set of soft constraints $\mathcal{S} = \{s_1, \ldots, s_S\}$
- Soft constraint weights $w_{s_i} \in \mathbb{R}^+$ for $i \in 1, \ldots, S$
- A set of soft constraint thresholds $\mathcal{T} = \{t_1, \ldots, t_S\}$

The given problem has a set $\mathcal{H}$ of size H of hard constraints, each with a weight w_{h_i}. The reason for using a weight for hard constraints is the use of meta-heuristic solution methods that internally convert hard constraints to soft constraints, but require a weight for these constraints high enough that the total amount of violation $v_{h_i} \in \mathbb{R}^{\geq 0}$ for each hard constraint i is 0 when the meta-heuristic terminates.

For soft constraints a similar set $\mathcal{S}$ of size S is given, however, violations $v_{s_i} \in \mathbb{R}^{\geq 0}$ are permitted in a feasible solution. Note that this covers both simple objectives (violations of a bound of 0) and soft constraints in relations to some non-zero bound for a property of a solution.

$$obj = \sum_{i=1}^{S} w_{s_i} \cdot v_{s_i} \tag{1}$$

Equation (1) defines the typical objective function of the problem that needs to be minimized. However, when solving repeatedly, the user can provide soft constraint thresholds t_i, which tell the system to adapt the corresponding weight w_{s_i} such that $v_{s_i} \leq t_i$ for an acceptable solution.

Weight Setting Procedure

The system starts with the given set of weights and adapts them in a repeated cycle until the goals are met or the situation is detected to be unrecoverable. The approach in this section is based on previous work (Böðvarsdóttir, Smet, and Berghe 2020), but our paper provides several novel extensions including the treatment of hard constraints in meta-heuristic methods, several adaptations to keep weights from rising excessively, and the introduction of a novel violation score to guide the user towards reasons for infeasibility.

Algorithm 1 shows the core loop of the weight setting procedure. There are two manipulations of the weights during the initialization: First, the highest weights are limited to a factor of w_L times the lowest weight for both $w_{\mathcal{H}}$ and $w_{\mathcal{S}}$ in line 1 to prevent excessive divergence after multiple applications. Second, if a solution is already given, it can already be evaluated regarding the thresholds $\mathcal{T}$ in line 3:

$$w_{s_i} \leftarrow w_{s_i} \cdot \min\{1 + f \cdot (v_{s_i} - t_i);\ 10\} \quad \forall i \in 1, \ldots, S \text{ with } v_{s_i} > t_i \tag{2}$$

$$w_{s_i} \leftarrow w_{s_i} \cdot \max\left\{1 - \frac{t_i - v_{s_i}}{t_i};\ 0.1\right\} \quad \forall i \in 1, \ldots, S \text{ with } v_{s_i} < t_i \text{ and } t_i > 0 \tag{3}$$

Equation (2) shows the adaptation when the thresholds are not met by the initial solution, therefore the weights are immediately increased. f is an increase factor described later in this section. The maximum increase is by a factor of 10. Equation (3) shows the adaptation when there is room to the threshold, leading to a proportional decrease in weight (to at least on tenth of the previous value). These initial weight manipulations are the only ones that can reduce weights.

A fail counter is initialized in line 5 and used to determine when the thresholds cannot be met during the following loop. This main loop, starting at line 6, is otherwise continued until either successful termination or a timeout.

Depending on the current situation, the solution might be reset in line 7. If no solution is given so far, a new solution is initialized by a given initialization method. However, a reset is also performed if hard constraint violations are currently present in the solution. This is due to the fact that solutions with significant levels of hard constraint violations

Algorithm 1: Core weight setting structure

Input: Instance $inst$, hard constraint weights $w_{\mathcal{H}}$, soft constraint weights $w_{\mathcal{S}}$, weight limit w_L, timeout t, optional solution sol, optional thresholds $\mathcal{T}$

Result: Termination status, optimized solution sol, violation scores vs in case of a failure

```
1  w_H, w_S ← limitWeights(w_H, w_S, w_L);
2  if sol is given then
3  |  w_S ← initialWeightUpdate(sol, w_S);
4  end
5  fail_count ← 0;
6  while timeout t not reached do
7  |  sol ← checkReset(sol);
8  |  sol ← appAlg(inst, sol, w_H, w_S, t, fast);
9  |  v_hard, v_soft ← evalViolations(sol, T);
10 |  if no violation in v_hard and v_soft then
11 |  |  sol ← appAlg(inst, sol, w_H, w_S, t, ext);
12 |  |  return Success, sol;
13 |  end
14 |  if isImprovement(v_hard, v_soft) then
15 |  |  fail_count ← 0;
16 |  else
17 |  |  fail_count ← fail_count + 1;
18 |  |  if abort(fail_count) then
19 |  |  |  vs ← score(v_hard, v_soft);
20 |  |  |  return Failure, vs;
21 |  |  end
22 |  |  updateAlgSettings(v_hard, v_soft);
23 |  end
24 |  if no violation in v_hard then
25 |  |  w_S ← updateWeights(v_soft);
26 |  else
27 |  |  w_H ← updateWeights(v_hard);
28 |  end
29 end
30 return Timeout;
```

often have very different structure than those without such violations, and starting from scratch with adapted weights is often easier than changing an existing infeasible solution.

Line 8 contains the main target algorithm call. It performs the underlying solution method, which is a black box as far as weight setting is concerned, until it terminates by its internal criterion. Parameters are chosen to run the target algorithm in reasonable time for repeated execution (argument $fast$). Afterwards, the violations of hard constraints and thresholds are extracted from the solution in line 9.

If no violation is found, weight setting was successful and the algorithm can be run again (or continue running) with the final weights, but different parameters like more runtime or more steps until termination (extended run, argument ext). This is done to save time during repeated weight setting runs, but give the opportunity to improve the final results further. If any hard constraint weights were changed during weight setting, these are doubled for the final phase to provide a safety margin against reintroducing violations.

The call `isImprovement` in line 14 is supposed to decide whether the overall amount of violations is moving in the right direction. While it is expected that several weight adaptations are necessary to reach all thresholds, a promising approach should at least get closer to this goal over time according to some measure, otherwise the weight setting process might be stuck, e.g., by just alternating violations between different constraints. The most general choice is to count the total (unweighted) violation of all constraints which is also used in the current system. An improvement is only made if a new best total violation is found. Note, however, that in the case of constraints with violations on very different scales (e.g. one unit of violation for constraint 1 is about as hard to remove as 10000 units of violation for constraint 2), it might be necessary to adapt this decision.

If the violation level is an improvement according to the previous definition, the fail counter is reset, otherwise, it is increased. In this case, in line 18 a choice on potentially aborting the process is made. It is expected that even in a successful weight setting process, sometimes violation levels will briefly increase, however, not for too many consecutive steps. Therefore, if the system cannot reach a new best violation level for 5 consecutive tries (up to 10 when the level always stays constant), the process is aborted as a failure, and the violation score vs is provided as feedback.

If there is no improvement, but no abort yet, an optional algorithm update is performed in line 22. This update might change the parameters of the algorithm to allow a more thorough optimization in the next loop. This update is meant to resolve issues where the initial parameter setting of the algorithm might not be well suited for the given instance.

Finally, weights are adapted based on the current violations in lines 24 to 28. This is first done for the hard constraints, and for the soft constraints only once the hard constraint violations are resolved. This is again due to structural issues that a solution might have in case of severe hard constraint violations. When these violations are resolved, the solution might look very different anyway, making it less useful to adapt the soft constraint weights beforehand.

In any case, both hard and soft constraint weights are adapted according to the same scheme, which is based on Böðvarsdóttir, Smet, and Berghe 2020:

$$w_{h_i} \leftarrow w_{h_i} \cdot \min\{\max\{1 + f \cdot v_{h_i};\ 2\};\ 100\} \qquad \forall i \in 1, \ldots, H \text{ with } v_{h_i} > 0 \quad (4)$$

$$w_{s_i} \leftarrow w_{s_i} \cdot \min\{\max\{1 + f \cdot (v_{s_i} - t_i);\ 2\};\ 10\} \qquad \forall i \in 1, \ldots, S \text{ with } v_{s_i} > t_i \quad (5)$$

Equation (4) shows the update for hard constraints, Equation (5) for soft constraints. The only differences are that for hard constraints, their value is directly equal to the amount of violation, while for the soft constraints the difference to the threshold is used, and a different maximum.

The value f, according to Böðvarsdóttir, Smet, and Berghe 2020, is used to balance the speed of setting the weights for different constraints where the amount of violation is measured on very different scales. On the other

hand, this may add many additional parameters to the configuration of the system. Therefore, after some testing, we decided to use $f = 0.01$ for all constraints in the current system, which has the advantage of additional stabilization due to the lower and upper bounds. However, if it occurs for any problem domain that weights for individual constraints are not adapted properly, this might need to be changed.

Another adaptation in relation to previous work is the introduction of a lower and upper bound for the adaptation factor of the weight. A lower bound of 2 is introduced to speed up very slow convergence, e.g., in case a constraint has a very low violation like 1, but requires a significant increase in weight to get rid of this violation. The upper bound of 100 for hard constraint weights is to prevent excessive weight growth in case there is a large amount of violation. A factor of 100 typically already changes the result in very significant ways, even higher growth showed to lead to very huge weights on larger instances. The factor for soft constraints is chosen even lower at 10, mainly to prevent introducing too many hard constraint violations that are otherwise caused by very large increases of soft constraint weights.

Violation Score

Getting a result that conforms to given thresholds without manually specifying weights is a very useful feature, but it is equally important to deal with the case when no set of weights can be found that can fulfill all requirements. In Böðvarsdóttir, Smet, and Berghe 2020, the termination criterion is when a weight reaches a certain factor compared to its original weight. In our system, we use the fail counter as described before, as the time it takes weights to reach such a factor can be very different otherwise. However, so far none of these methods help too much with identifying where the process got stuck, and therefore help the user to figure out what might be changed to resolve the infeasibility.

An option would be to look at the amount of violation given when the decision of aborting is made. However, this is just a momentary snapshot, e.g., if violation oscillates between two different constraints, this method might only capture one of them. In fact, one can think of several different ways to get an idea about the main contributors:

- All constraints violated at the evaluation with the lowest overall amount of violation (V_L).
- All constraints violated in any smallest subset of violated constraints (e.g., if always at least 2 constraints are violated, choose all constraints that are violated in any evaluation with exactly two violated constraints, V_s).
- Count the number of times since the evaluation with the lowest overall amount of violations that a particular constraint was violated - the more often, the more this constraint contributes to infeasibility. All constraints with the highest value are captured by V_r (recent violations).
- In contrast, constraints that were never violated since the evaluation with the lowest overall amount of violation are very unlikely to contribute to infeasibility (V_0).

Now all these scores capture some potential aspects that might contribute, and one can think of situations where one of them helps, but another does not. On the other hand, giving a complex score with several components to a user is a bad idea as well, since it can be hard to figure out all details of the different ratings. While it should still be available for expert users, a much simpler score is needed for regular use. This is done by combining the above criteria into one score per constraint as follows:

- High (3): Constraint is part of V_L, V_s, and V_r, resolving the violation without addressing this hard constraint or threshold is unlikely.
- Moderate (2): Constraint is part of V_L, V_s, or V_r, but not all of them, it is significantly involved in the violations.
- Low (1): Not in any other category. Constraint shows limited interaction with the cause of infeasibility.
- None (0): Constraint is part of V_0, but not V_s. This constraint shows no significant interaction.

This system allows to give one simple feedback per constraint, can easily be integrated into the UI, e.g., by color-coding, and provides a quick and intuitive visualization of the most likely reasons for infeasibility. In case of soft constraint thresholds, these can then be made less strict, while for some hard constraints it might be possible to adapt some bounds to allow more feasible solutions.

Simulated Annealing

While the optimization algorithm in use is not the main focus of this paper, and the weight setting procedure does not depend on the specific algorithm chosen (it just assumes that the algorithm internally uses weights to transform hard constraints to soft constraints), this subsection is intended to give a brief overview of the Simulated Annealing algorithm used in our system, which is very similar to Kletzander and Musliu 2020b, and its parameters and interactions with the weight setting process.

Simulated Annealing is run until a given number of consecutive iterations without improvement or a given timeout t_{SA}, whichever is reached first. This number of iterations is 10 during weight setting, increased by 10 % in `updateAlgSettings`, and 100 in the final extended run.

Moves are selected randomly from a given set of moves that depends on the problem domain. As usual, improving moves are always accepted, while worsening moves are accepted with probability $\exp\left(-\frac{change}{T}\right)$ based on the change in solution value $change$ and the current temperature T.

The starting temperature $T_{start} = 100$ is reduced by multiplication with a cooling factor T_f after each 1000 move evaluations. $T_f = 0.9$ during weight setting, reducing the distance to 1 by 10 % in `updateAlgSettings`, and $T_f = 0.999$ for the final extended run. The algorithm returns the best solution encountered during the whole search.

Application of the System

This section will present a detailed evaluation of different usage scenarios both regarding hard and soft constraints in feasible and infeasible applications. The evaluation was done on an exemplary problem domain from practice, and

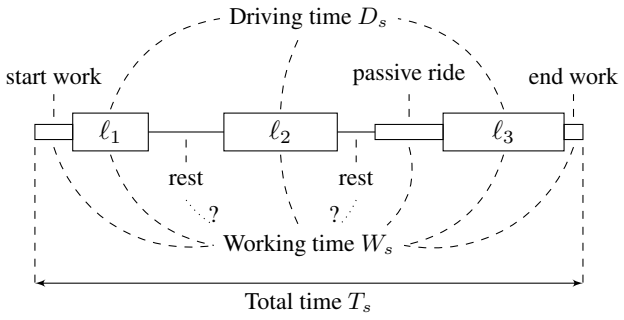

Figure 1: Example shift for BDS (Kletzander, Musliu, and Van Hentenryck 2021)

executed on a computing cluster running Ubuntu 16.04.1 LTS with Intel Xeon CPUs E5-2650 v4 (max. 2.90GHz, 12 physical cores, no hyperthreading), but each individual execution was performed single-threaded. For the evaluation timeouts are set to 10 minutes for Simulated Annealing during weight setting, one hour for the final extended run, and at most three hours for a full weight setting procedure.

While the system is not limited to a particular domain, we use the Bus Driver Scheduling problem based on the Austrian collective agreement (Kletzander and Musliu 2020b; Kletzander 2022) as an exemplary domain. It is well suited since it represents a complex problem formulation with multiple, partly contradicting, hard constraints and objectives, and has a publicly available benchmark data set.

Bus Driver Scheduling (BDS) is a part of crew scheduling in the process of operating bus transport systems (Ibarra-Rojas et al. 2015). It has been considered by many authors starting with Wren and Rousseau (1995), but mostly focused only on cost only. A problem variant with slightly different constraints from practice was previously tackled by a goal-oriented method where weights were adapted in a manual tuning process (Kletzander and Musliu 2020a).

The domain deals with the assignment of bus drivers to vehicles that already have a predetermined route for one day. Bus routes are given as a set $\mathbf{L}$ of individual bus legs, each leg $\ell \in \mathbf{L}$ is associated with a tour $tour_\ell$ (corresponding to a particular vehicle), a start time $start_\ell$, an end time end_ℓ, a starting position $startPos_\ell$, and an end position $endPos_\ell$. The driving time for leg ℓ is $drive_\ell = end_\ell - start_\ell$.

A tour change occurs when a driver has an assignment of two consecutive bus legs i and j with $tour_i \neq tour_j$. The time it takes to change from position p to position q when not actively driving a bus (passive ride time), is $d_{p,q}$ for $p \neq q$. $d_{p,p}$ represents the time it takes to switch tour at the same position, but is not considered passive ride time. Each position p is further associated with an amount of working time for starting a shift ($startWork_p$) and ending a shift ($endWork_p$) at that position.

A solution to the problem is an assignment of exactly one driver to each bus leg. A schematic example shift is shown in Figure 1. It shows the three main measures of time that are relevant for evaluating a shift: driving, working and total time. For details of the problem constraints refer to Kletzander and Musliu (2020b). The following definitions use E as the set of all shifts (employees) of a potential solution.

The problem has the following set of hard constraints $\mathcal{H}$:

- `Overlap` (h_1): No overlapping leg assignments and enough changing time in case of a tour change.
- `Max span` (h_2): Hard maximum $T_{max} = 14$ hours for the total span of a shift.
- `Max drive` (h_3): Hard maximum $D_{max} = 9$ hours for the total driving time per shift.
- `Driving breaks` (h_4): Driving breaks after at most 4 hours of driving time, with the options of one break of at least 30 minutes, two breaks of at least 20 minutes each, or three breaks of at least 15 minutes each.
- `Max work` (h_5): Hard maximum $W_{max} = 10$ hours for the total paid working time per shift.
- `Rest breaks` (h_6): Rest breaks of at least 30 minutes are required for shifts between 6 and 9 hours, and of at least 45 minutes for shifts of more than 9 hours.

Note that there are three hard constraints (`Overlap`, `Driving breaks`, `Rest breaks`), that deal with the structure of a shift without depending on a parameter and have to be fulfilled no matter what. In case they are violated, a numeric penalty is computed that indicates the amount of violation in minutes (v_{h_i}). The other three hard constraints (`Max span`, `Max drive`, `Max work`) depend on a parameter that could be changed if the surrounding rules were to be remodeled. This assumption will be made in some of the following experiments. Again the amount of violation in minutes is used to compute the penalty (v_{h_i}).

There is an additional hard constraint that is treated in a different way: A maximum number of shifts can be specified. Since all other constraints are evaluated for each shift, this one is implemented in a different way, providing a limit of shifts for the algorithm to use at any time. Therefore it can never be violated, on the other hand, every violation of hard constraints can usually be reduced by allowing more shifts. The limitation of treating this constraint differently comes from Simulated Annealing struggling with removing full shifts (intermediate states with short shifts are expensive), not from the weight setting procedure.

BDS further uses the following soft constraints $\mathcal{S}$:

- `Working time` (s_1): The total amount of paid working time $v_{s_1} = \sum_{e \in E} W_e$ excluding additional paid working time used to fill up shifts below $W_{min} = 6$ hours.
- `Min working time` (s_2): The additional amount of paid working time to fill up shifts below W_{min}, obtained by $v_{s_2} = \sum_{e \in E} \max\{W_{min} - W_e;\ 0\}$.
- `Span` (s_3): The sum of all total spans $v_{s_3} = \sum_{e \in E} T_e$.
- `Passive ride` (s_4): The sum of passive ride times $v_{s_4} = \sum_{e \in E} ride_e$.
- `Tour changes` (s_5): The total number of tour changes $v_{s_5} = \sum_{e \in E} ch_e$.
- `Shift splits` (s_6): The total number of shift splits (breaks of at least 3 hours which are always unpaid, but not rest breaks) $v_{s_6} = \sum_{e \in E} split_e$.

	Simulated Annealing				Weight Setting				
Size	Avg time	Best res	Avg res	Std dev	Avg time	Best res	Avg res	Std dev	Adapt.
10	601.0	15224.6	15224.6	0.0	731.1	**14929.0**	15038.4	115.2	4.3
20	587.8	31147.0	31228.4	86.6	845.1	**30854.4**	31309.4	419.7	5.3
30	757.7	51055.4	51247.6	159.2	940.4	**51002.8**	51908.2	851.9	5.7
40	862.3	69065.0	69225.9	148.9	1102.1	**68517.6**	69860.4	1528.3	4.4
50	911.8	87219.0	87450.0	236.9	1248.2	**86381.8**	87599.1	1193.3	5.2
60	1020.0	103265.2	103653.2	299.0	1409.2	**103112.2**	104545.2	1748.0	5.6
70	1097.0	**121523.4**	121947.5	355.0	1597.0	121959.8	123341.5	1365.9	5.0
80	1148.8	139620.8	140179.5	429.9	1804.9	**139578.0**	140666.5	1431.3	5.6
90	1235.9	**155293.0**	155883.3	520.0	2063.9	155548.8	156710.3	1181.9	4.9
100	1372.3	**171766.0**	172644.1	572.5	2327.2	171891.2	173458.9	1674.8	5.6

Table 1: Results for hard constraint weight setting

These six objectives can be combined to the overall objective function of the problem according to Equation (1). The objective function used in previous work defined the following cost for each shift:

$$cost_e = 2 \cdot W'_e + T_e + ride_e + 30 \cdot ch_e + 180 \cdot split_e \quad (6)$$

Equation (6) uses weights based on real-life requirements of balancing cost optimization with the need to create practically workable schedules for the employees. It combines `Working time` and `Min working time` using W'_e. While this set of weights is derived from practice, it is not sufficient for all application scenarios and frequently needs to be changed when priorities for the optimization shift.

The standard benchmark data set for this problem (Kletzander and Musliu 2020b) contains 50 instances distributed in 10 size categories with 5 instances each. These reach from around 10 tours (70 legs) to around 100 tours (1000 legs) based on real-life demand distributions. The moves used for Simulated Annealing either exchange individual legs or consecutive legs between pairs of employees.

Weight Setting for Hard Constraints

For the first experiment we want to come up with useful hard constraint weights from scratch. Obviously very high weights can always be used to ensure that hard constraints are not violated, but when the scale of soft constraint weights changes (e.g. because a user would enter very high weights for some soft constraints), previously working weights might not be enough. On the other hand, allowing meta-heuristics to also explore the infeasible space by using hard constraint weights that are not too high might benefit the solution process, but makes it more difficult to ensure no hard constraints are violated in the end.

We start all hard constraint weights at 1, using weight setting to raise them to values that ensure no violations remain in the solution. We compare the results to using Simulated Annealing in exactly the same configuration as used in the final extended run directly with preset hard constraint weights ($w_{h_i} = 1000$ for all).

Table 1 shows a summary of these results where each row represents the average across the instances of the given size.

First, the results show that, while starting from very low hard constraint weights, weight setting can effectively find

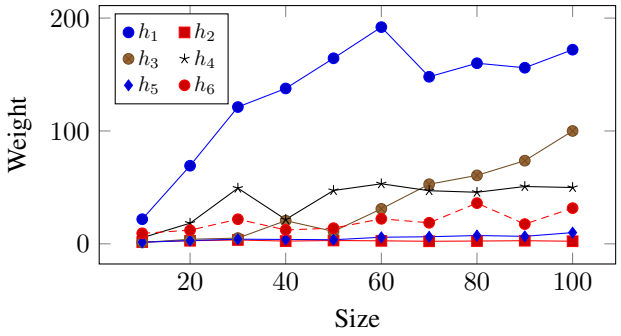

Figure 2: Final weights for different sizes

weights such that feasible solutions are reliably found. The process was successful for all 250 runs (5 per instance), and the number of weight iterations, shown in the very right column, was very moderate and consistent across all instances. In never took more than 10 steps for any run, and the overall average is 5.15 with a standard deviation of 1.54. The overhead in runtime is in the average 43.9 %. Since weight setting does a full Simulated Annealing run at the end of the procedure, this means that despite starting from obviously too low weights, the majority of the runtime is still dedicated to the final optimization run with the appropriate weights.

Regarding the solution quality, differences are in general low, after all the same algorithm is employed, however a slight but clear pattern emerges that shows better results for weight setting for smaller instances and pure SA for larger instances. This is most likely due to lower final weights allowing SA to also explore parts of the infeasible space. This can be very beneficial for small instances where the search space is small and good parts might be separated by infeasible regions, while for large instances an already too large search space in enlarged even more.

On the positive side this effect can especially be seen for instances 1, 5 (size 10), and 6 (size 20), where both the best and average results are each more than 3.6 % better with weight setting compared to pure SA, which is a very good improvement for this kind of problem. On the negative side, with weight setting the standard deviation rose very significantly, making the results for larger instances less stable.

Sh.	Violation scores Run 1	Run 2	Run 3	Run 4	Final cost
12					14417
11	130111				14598
10	230011	130011			14590
9	230100	130111			14969
8	230110	230020			15548
7	231221	331231			15798
6	333330	322220	331130	320020	311010

Table 2: Squeezing instance 10_1 until infeasibility

Figure 2 shows the average value of the weights for each size of instances. This comparison shows that in general weights stayed much smaller than the presets of 1000. In particular, for BDS most weights can stay below 100 (except h_1), and h_2 and h_5 mostly did not need to exceed 10.

Further weights stayed smaller for smaller instances, and then grew with increasing instance size up to a certain point, however, for most weights (except h_3), the required weights seem to grow more and more slowly or even reach a plateau. These plateau values could then be used with a safety margin for unseen instances.

For the next two experiments, we will artificially make it more and more difficult to find a feasible solution by reducing the maximum number of shifts, which is enforced by a hard boundary. Eventually the maximum for span, driving time, or working time per shift would need to be extended to fit the legs into even fewer shifts. Note that raising these limits would not be a goal in practice, but it allows a very controlled and systematic way to reduce the feasible space.

The first feasibility experiment is performed exemplary on instance 10_1. Table 2 shows one line for each maximum number of shifts (Sh.) that was tested. The next columns show violation scores for consecutive runs for the hard constraints in order of $\mathcal{H}$ (e.g., 130111 means a violation score of 1 for h_1, 3 for h_2, 0 for h_3 and so on). In case `Max span` (h_2), `Max drive` (h_3), or `Max work` (h_5) were among the ones with the highest violation score, the corresponding limits were increased by one hour and weight setting retried. The first empty column marks reaching a feasible solution.

Obviously, h_2 (`Max span`) seems to be the most violated constraint using fewer shifts. In all cases with 11 to 7 shifts, span is correctly identified as the major contributor. For 11 shifts, setting $T_{max} = 15$ is enough for feasibility, while for the others span is again correctly identified as the major contributor in run 2. For cases with 10 to 8 shifts $T_{max} = 16$ resolves the infeasibility. For 7 shifts, it correctly identifies both span and h_5 (`Max work`) as major contributors (together with h_1 that has no parameter). Indeed, only raising both T_{max} to 16 and W_{max} to 11 resolves the infeasibility.

For 6 shifts, there is no hope of a feasible solution, since 7 tours are active at the same time. The first result shows a massively over-constrained problem (most constraints have very high score), already the second result hints at the unrecoverable infeasibility by showing the highest score for h_1 (`Overlap`). Still, the attempt to increase parameters of constraints with at least score 2 is executed for several further steps to make sure. Finally, h_1 is the only constraints with a score > 1, making it clear that no further changes to the other constraints will resolve this situation.

C.	Reduct.	Incr.	Feas.	Avg. t.	W. fac.
s_1	11.7	483.7	72.2	973.5	1178.8
s_2	94.8	215.4	100.0	986.4	5.9
s_3	4.5	27.7	0.0	1255.1	526056
s_4	94.4	25.9	100.0	1149.2	25.0
s_5	73.5	4.8	100.0	1012.4	7.5
s_6	100.0	0.0	100.0	1929.0	60.4

Table 3: Reducing different soft constraint values

Overall, this experiment showed that our violation score works very well to identify the causes for infeasibility and resolve them, as well as to identify infeasibility that cannot be resolved.

In the final experiment regarding hard constraints we extended this approach for the whole set of 50 instances. The initial number of shifts was repeatedly reduced by 10 % until the instance was infeasible, then the recommended hard constraint parameter was increased by one hour just as before. As it turns out, for all instances this should be h_2 (`Max span`). Out of the 250 total runs, only for 8 runs additional violations were reported where other runs showed that the given setting should already be feasible. Out of these, 7 runs ended after the next (not necessary) parameter adaptation, one reported the most violation for h_4 (`Driving break`).

Out of the 272 violations scores excluding those spurious 8 (most instances needed one round of adaptations, some two), only 6 did not report h_2 in the high category. For all of them, h_2 was in the modest category. In 109 cases h_1 was in the highest category, which makes sense as the choice is either to violate the maximum span or assign overlapping bus legs. Out of the 6 cases with h_2 getting moderate score, for four of them h_1 got high score, for two of them h_1 got moderate score. Only at two other occasions one of the other constraints got a high score, which was h_5 with a high score, but no actual need to increase W_{max}, and both occurred on the same instance (20_8).

Overall, this experiment showed that across different instances of very different sizes, and multiple repetitions, our system is very reliable and consistent in reporting the most involved constraints regarding infeasibility.

Weight Setting for Soft Constraints

The final part of the experiments deals with weight setting for soft constraints. The concept of weight setting itself stays the same, internally the difference is only whether the violation is obtained directly from the value of the constraints or by calculating the difference to a given threshold.

This section presents two different experiments: Initially, in each experiment the first run is done without any thresholds. Then, in the first experiment, a threshold is set to lower the value for one of the soft constraints by 10 % ($t_i = 0.9 \cdot v_{s_i}$) without any restrictions on the remaining constraints (keeping their original weights, no thresholds), while in the second experiment, a maximum worsening of 5

Thresholds						Values						Weights					
work	m.w.	span	r	c	s	work	m.w.	span	r	c	s	work	m.w.	span	ride	change	split
						4129	1053	4546	31	1	0	2	2	1	1	30	180
4335	948	4773	33	2	1	4194	975	4601	31	6	0	62	72	55	52	480	18
4335	948	4773	33	3	1	4197	909	4700	0	1	0	471	456	493	4628	31027	1.8
												236	228	493	4628	1034	0.01

Table 4: Balanced reduction of `Min working time`

% for any other soft constraint ($t_i = 1.05 \cdot v_{s_i}$) is used.

Table 3 shows the results of the first experiment, where each instance was assigned to one of the soft constraints with the goal of a 10 % reduction (equally distributed across all sizes). Each row shows the summary for instances where one particular soft constraint was chosen. Reduct. shows the percentage of reduction for the threshold value, Incr. shows the average percentage increase for all other soft constraint values (a raise from 0 is counted like 100 % increase). The next columns shows the percentage of runs that did not report violations, the average runtime in seconds, and the average factor of the weight increase for the target constraint.

As expected, the results show very different findings for the different soft constraints. This highlights how individual the reactions to weight changes can be. For s_1 (`Working time`), we see indeed an average reduction by slightly above 10 %, however, a very strong increase for the other objectives with an average of more than 480 %. A reduction of working time by 10 % is actually a very challenging task (also seen by the weight factor of over 1000), and the only way to manage such a strong reduction is by greatly increasing the use of shift splits, which is the highest contributor to this increase. In several occasions, the threshold could not be achieved, but the soft constraint threshold is reliably identified as the cause by the violation score in these cases.

s_2 (`Min working time`), s_4 (`Passive ride`), s_5 (`Tour changes`), and s_6 (`Shift splits`), despite a much smaller weight factor, lead to a tremendous decrease in the respective constraint value, far beyond the original goal. Especially the last three are constraints with often few violations, where each decrease is very close to eliminating the whole soft constraint violation. Note, however, that this comes with much smaller increase in other soft constraints, except for s_2, since the reduction of minimum working time typically requires to chain short shifts with other parts of the schedule by using more tour changes. s_5, on the other hand, is below 5 %, and s_6 can be eliminated while other changes average out to 0.

Finally, s_3 (`Span`) can clearly not be reduced by 10 % without getting infeasible. Despite a massive increase in weight, an average reduction of only 4.5 % could be reached, and the violation score clearly identifies the threshold for s_3 as the cause of the problems for all runs.

In the second experiment, the use of thresholds for all soft constraints is demonstrated on the example of instance 10_1 again. Table 4 shows the process. The initial values (first line), obtained with the initial weights, are adapted to obtain the thresholds in the second line: A reduction of 10 % for `Min working time` and a maximum increase of 5 % (at least 1) for the other soft constraints. First the number of tour changes rises beyond the limit, while s_2 is still above the goal. After a threshold increase, however, the second round of adaptations manages to find a differently structured solution that only increases span and working time within the bounds, while the others stay on the original value or even decrease.

Note, however, the completely counter-intuitive changes in weights that were necessary to obtain this result, as shown by the increase factors in the last line. The highest increase was necessary for ride time, followed by tour changes, while the actual goal had the second lowest increase. With weight setting, this could be achieved easily, with only one interactive step, while figuring out these weights manually would have taken considerable effort and time.

Conclusion

Overall, we presented a weight setting system that dynamically adapts weights to reach various optimization goals without exposing the user to tedious tuning of weights. While very general regarding both algorithms and application domains, we evaluated it in detail on a personnel scheduling domain with Simulated Annealing.

The system includes the transfer from hard constraints to soft constraints for use with various meta-heuristics and showed to be very efficient in providing good weights for hard constraints. When instances are infeasible, a violation score combines multiple indicators to a simple, yet very effective score that the user can easily use to adapt constraint parameters to resolve the infeasibility. Finally, instead of manually adapting the combination of weights to get solutions with different properties, soft constraint thresholds can be used to modify solutions in a controlled way. The evaluation showed the very counter-intuitive way weights affect the resulting values, and that our weight setting system is a very valuable tool to perform such changes.

Note that other methods to deal with the thresholds are possible, including non-linear options like only penalizing values above the threshold within the algorithm. However, our system allows application on any optimization algorithm using a linear combination of objectives without any internal modification, and it allows to obtain weights that might be very useful for other instances of the same problem.

Beside other weighting options, future work might include user studies evaluating in detail how users interact with the system, as well as methodological extensions like the inclusion of priorities, more fine-grained weight adjustments to prevent overshooting thresholds, or learning from the results of weight adaptations.

Acknowledgments

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

References

Böðvarsdóttir, E. B. 2021. *Addressing real-world challenges in nurse rostering*. Ph.D. thesis, Technical University of Denmark.

Böðvarsdóttir, E. B.; Bagger, N.-C. F.; Høffner, L. E.; and Stidsen, T. J. 2022. A flexible mixed integer programming-based system for real-world nurse rostering. *Journal of Scheduling*, 25(1): 59–88.

Böðvarsdóttir, E. B.; Smet, P.; and Berghe, G. V. 2020. Behind-the-Scenes Weight Tuning for applied nurse rostering. *Operations Research for Health Care*, 26: 100265.

Böðvarsdóttir, E. B.; Smet, P.; Berghe, G. V.; and Stidsen, T. J. 2021. Achieving compromise solutions in nurse rostering by using automatically estimated acceptance thresholds. *European Journal of Operational Research*, 292(3): 980–995.

Böðvarsdóttir, E. B.; Smet, P.; Vanden Berghe, G.; and Stidsen, T. 2019. A modeling methodology to support nurse rostering practitioners. In *Proceedings of the 9th Multidisciplinary International Conference on Scheduling: Theory and Applications*, 141–155. MISTA.

Burke, E. K.; De Causmaecker, P.; Vanden Berghe, G.; and Van Landeghem, H. 2004. The State of the Art of Nurse Rostering. *Journal of Scheduling*, 7(6): 441–499.

Coello, C. A. C.; Lamont, G. B.; Van Veldhuizen, D. A.; et al. 2007. *Evolutionary algorithms for solving multi-objective problems*, volume 5. Springer.

De Bruecker, P.; Van den Bergh, J.; Beliën, J.; and Demeulemeester, E. 2015. Workforce planning incorporating skills: State of the art. *European Journal of Operational Research*, 243(1): 1–16.

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

Gärtner, J.; Bohle, P.; Arlinghaus, A.; Schafhauser, W.; Krennwallner, T.; and Widl, M. 2018. Scheduling matters–Some potential requirements for future rostering competitions from a practitioner's view. In *12th International Conference of the Practice and Theory of Automated Timetabling*, 33–42.

Ibarra-Rojas, O.; Delgado, F.; Giesen, R.; and Muñoz, J. 2015. Planning, operation, and control of bus transport systems: A literature review. *Transportation Research Part B: Methodological*, 77: 38–75.

Kletzander, L. 2022. *Automated Solution Methods for Complex Real-life Personnel Scheduling Problems*. Ph.D. thesis, TU Wien.

Kletzander, L.; and Musliu, N. 2020a. Scheduling Bus Drivers in Real-Life Multi-Objective Scenarios with Break Constraints. In *Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling-PATAT 2021*, volume 1, 34–40.

Kletzander, L.; and Musliu, N. 2020b. Solving Large Real-Life Bus Driver Scheduling Problems with Complex Break Constraints. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, 421–429.

Kletzander, L.; Musliu, N.; and Van Hentenryck, P. 2021. Branch and Price for Bus Driver Scheduling with Complex Break Constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 11853–11861.

Mihaylov, M.; Smet, P.; Van Den Noortgate, W.; and Vanden Berghe, G. 2016. Facilitating the transition from manual to automated nurse rostering. *Health Systems*, 5(2): 120–131.

Özder, E. H.; Özcan, E.; and Eren, T. 2020. A Systematic Literature Review for Personnel Scheduling Problems. *Int. J. Inf. Technol. Decis. Mak.*, 19(6): 1695–1735.

Petrovic, S.; and Vanden Berghe, G. 2012. A comparison of two approaches to nurse rostering problems. *Annals of Operations Research*, 194(1): 365–384.

Van den Bergh, J.; Beliën, J.; De Bruecker, P.; Demeulemeester, E.; and De Boeck, L. 2013. Personnel scheduling: A literature review. *European Journal of Operational Research*, 226(3): 367–385.

Voudouris, C.; Tsang, E. P.; and Alsheddy, A. 2010. Guided local search. In *Handbook of metaheuristics*, 321–361. Springer.

Wren, A.; and Rousseau, J.-M. 1995. Bus Driver Scheduling — An Overview. In Fandel, G.; Trockel, W.; Daduna, J. R.; Branco, I.; and Paixão, J. M. P., eds., *Computer-Aided Transit Scheduling*, volume 430, 173–187. Berlin, Heidelberg: Springer Berlin Heidelberg.