Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)

Hyper-Heuristics for Personnel Scheduling Domains

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

In real-life applications problems can frequently change or require small adaptations. Manually creating and tuning algorithms for different problem domains or different versions of a problem can be cumbersome and time-consuming. In this paper we consider several important problems with high practical relevance, which are Bus Driver Scheduling, Rotating Workforce Scheduling, and Minimum Shift Design. Instead of designing very specific solution methods, we propose to use the more general approach based on hyper-heuristics which take a set of simpler low-level heuristics and combine them to automatically create a fitting heuristic for the problem at hand. This paper presents a major study on applying hyper-heuristics to these domains, which contributes in three different ways: First, it defines new low-level heuristics for these scheduling domains, allowing to apply hyper-heuristics to them for the first time. Second, it provides a comparison of several state-of-the-art hyper-heuristics on those domains. Third, new best solutions for several instances of the different problem domains are found. These results show that hyper-heuristics are able to perform well even on very complex practical problem domains in the area of scheduling and, while being more general and requiring less problem-specific adaptation, can in several cases compete with specialized algorithms for the specific problems. These results help to improve industrial systems in use for solving different scheduling scenarios by allowing faster and easier adaptation to new problem variants.

Introduction

Industrial applications of scheduling technology often have to deal with changing requirements and the need to provide good solutions to new problem variants or even new problem domains in short time. However, often solution methods are tailored to a specific application and are difficult to adapt to changing constraints or different problems.

In contrast, hyper-heuristics are designed to work in a domain-independent way to provide efficient solutions across different domains, just requiring a set of low-level heuristics (LLH) and comparably few or even no parameter inputs in order to create the appropriate heuristic for solving a particular problem instance on the fly based on the performance of the individual LLHs (Burke et al. 2013, 2019). As part of the Cross-Domain Heuristic Search Challenge (Burke et al. 2011) the framework HyFlex (Ochoa et al. 2012) was introduced for a uniform implementation of different hyper-heuristics.

This paper presents a deep investigation of hyper-heuristics to three challenging personnel scheduling domains of high practical relevance for the first time, which are Bus Driver Scheduling (BDS), Rotating Workforce Scheduling (RWS), and Minimum Shift Design (MSD). We first propose several new low-level heuristics for these domains. Then we present a major comparison of 10 different versions of state-of-the-art hyper-heuristics on all three domains.

We obtain very good results on these domains, including several new best known solutions, and including practically relevant optimization goals that have been excluded from previous work. This provides strong results to deploy hyper-heuristics in practice to allow for easier adaptation to changing requirements and new problem domains.

Problem Domains

There has been a lot of work on different types of personnel scheduling problems for several decades which is summarized in several surveys (Burke et al. 2004; Ernst et al. 2004; Van den Bergh et al. 2013; De Bruecker et al. 2015). We focus on three different optimization domains in this area with complex real-life constraints, which are used in practice by our industry partner XIMES GmbH[1], an Austrian company providing consulting and well-known software packages for various kinds of employee scheduling applications.

Bus Driver Scheduling

The first domain under consideration is Bus Driver Scheduling (BDS), which 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.

In contrast, our domain deals with a complex real-life Bus Driver Scheduling Problem that was recently introduced by Kletzander and Musliu (2020), who also provided challenging instances that are publicly available. The best known solutions for most instances were achieved by a highly special-

[1] www.ximes.com

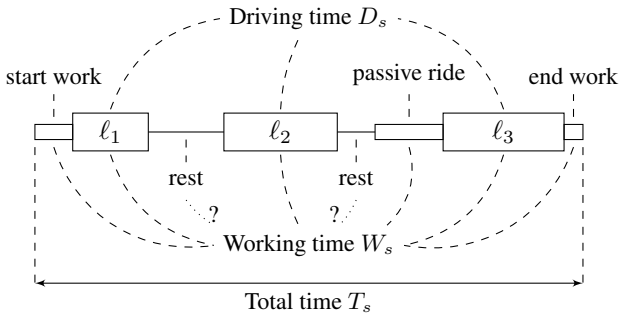


Figure 1: Example shift for BDS (Kletzander, Musliu, and Van Hentenryck 2021)

ized Branch and Price approach (Kletzander, Musliu, and Van Hentenryck 2021).

The domain deals with the assignment of bus drivers to vehicles that already have a predetermined route for one day. It encodes the real-life application of an Austrian collective agreement that includes a very complex set of constraints, and an objective function combining various cost and employee satisfaction goals to obtain practically useful shifts.

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 the leg is denoted by $drive_\ell = length_\ell = end_\ell - start_\ell$.

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. The constraints for each shift are summarized as follows, for details refer to Kletzander and Musliu (2020):

- No overlapping leg assignments and enough changing time in case of a tour change.
- Hard maximum for driving, working, and total time, additionally a soft minimum for working time.
- 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.
- Rest breaks of at least 30 minutes for shifts between 6 and 9 hours, and at least 45 minutes for shifts of more than 9 hours. Whether they are paid depends on their position within the shift.
- Shift splits are breaks of at least 3 hours which are always unpaid, but not rest breaks.

$$cost_s = 2 \cdot W'_s + T_s + ride_s + 30 \cdot ch_s + 180 \cdot split_s \quad (1)$$

The objective (1) combines paid working time W'_s as the main objective with the total time T_s, the passive ride time $ride_s$, the number of tour changes ch_s, and the number of shift splits $split_s$, and is based on the real-life requirements of balancing cost optimization with the need to create practically workable schedules for the employees.

Empl.	Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	D	D	D	D	N	N	-
2	-	-	A	A	A	A	N
3	N	N	-	-	D	D	D
4	A	A	N	N	-	-	-

Table 1: Example rotating workforce schedule

Rotating Workforce Scheduling

Once shifts are fixed, they have to be assigned to employees according to several constraints dealing with allowed sequences of shifts and limitations to the consecutive shift assignments. In many applications, a rotating schedule, where each employee rotates through the same sequence of shifts with different offsets, is a preferred way of scheduling.

The Rotating Workforce Scheduling (RWS) problem can be classified as a single-activity tour scheduling problem with non-overlapping shifts and rotation constraints (Baker 1976). Over the years several approaches were used to solve the problem, including use in commercial software by XIMES for almost 20 years (Musliu, Gärtner, and Slany 2002). A recent state-of-the-art complete method was introduced by Musliu, Schutt, and Stuckey (2018) and further extended by Kletzander et al. (2019), in particular by introducing several optimization goals from practice into the previous satisfaction problem. A decomposition approach for classical RWS was recently introduced by Becker (2020).

Formally, a rotating workforce schedule consists of the assignment of shifts or days off to each day across several weeks for a certain number of employees. Table 1 shows an example for four employees (or four equal-sized groups of employees), assigning the three shift types day shift (D), afternoon shift (A), and night shift (N). Each employee starts their schedule in a different row, moving from row i to row $i \bmod n + 1$ (where n is the number of employees) in the following week.

The satisfaction version of RWS is based on definitions and notation by Musliu, Gärtner, and Slany (2002) and Musliu, Schutt, and Stuckey (2018):

- n: Number of employees.
- $w = 7$: Length of the schedule. The total length of the planning period is $n \cdot w$, as each employee rotates through all n rows.
- $\mathbf{A}$: Set of work shifts (activities), enumerated from 1 to m, where m is the number of shifts. A day off is denoted by a special activity O with numerical value 0, $\mathbf{A}^+ = \mathbf{A} \cup \{O\}$.

- R: Temporal requirements matrix, an $m \times w$-matrix where each element $R_{i,j}$ corresponds to the number of employees that need to be assigned shift $i \in \mathbf{A}$ at day j.
- ℓ_w and u_w: Minimal and maximal length of blocks of consecutive work shifts.
- ℓ_s and u_s: Minimal and maximal lengths of blocks of consecutive assignments of shift s for each $s \in \mathbf{A}^+$.
- Forbidden sequences of shifts: Any sequences of shifts (like $N\ D$, a night shift followed by a day shift) that are not allowed in the schedule. This is typically required due to legal or safety concerns. The given instances use forbidden sequences of lengths 2 ($\mathbf{F}_2$) and 3 ($\mathbf{F}_3$).

The task is to construct a cyclic schedule S, represented as an $n \times w$-matrix, where each $S_{i,j} \in \mathbf{A}^+$ denotes the shift or day off that employee i is assigned during day j in the first period of the cycle.

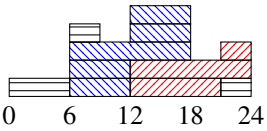
The problem has mostly been treated as a satisfaction problem, often providing a large number of solutions making it difficult to choose the most suitable schedule. Instead, in practice it is beneficial to include the choice criteria into the problem, therefore, we focus on the optimization variants of the problem introduced by Kletzander et al. (2019). Since employees in shift work often have difficulties aligning their free time with friends and family with more regular working times, the three different objectives deal with the optimization of free weekends, where a free weekend is defined as having both Saturday and Sunday off:

- F_1: Maximize the number of free weekends f.
- F_2: Minimize the maximum distance d_m between consecutive free weekends ($d_m = n + 1$ if no weekend is free).
- F_3: Minimize the sum of squared distances between weekends $d = \sum_{i=1}^{n} \widehat{Dist}_i^2$ where $\widehat{Dist}_i$ is the distance to the next free weekend if weekend i is free, and n otherwise.

Minimum Shift Design

The third domain describes a very different approach to designing shifts compared to BDS. Here, a certain demand is given for each time of the planning period, and the objective is to construct shifts within given limits to cover the demand as well as possible. However, since a large number of very different shifts is more difficult to handle, another objective is to minimize the number of distinct shifts that are introduced.

Minimum Shift Design (MSD) was introduced by Musliu, Schaerf, and Slany (2004) where a tabu search is proposed. This is the only work that also partially considers a forth objective that is not used further in literature, but often very relevant in practice. It sets limits for the average number of shifts per week that will be assigned to an employee. If this objective is not included, it might not be possible to find a good or even feasible assignment regarding sequences of consecutive shifts or the total number of shifts when assigning the shifts to individual employees. While this objective has been dropped in other work on this problem, we deal with the full practical problem formulation. The state-of-the-art method (Kletzander and Musliu 2020) uses a network-flow based formulation to provide optimal solutions to all benchmark instances, however, without the forth objective.



Start	End	E.
6:00	18:00	3
12:00	24:00	2
21:00	9:00	1

Figure 2: Example demand and solution for MSD

The specification is based on Musliu, Schaerf, and Slany (2004). The problem input is defined as follows:

- n consecutive timeslots $[a_1, a_2), [a_2, a_3), \ldots, [a_n, a_{n+1})$, each slot $[a_i, a_{i+1})$ with the same slotlength sl in minutes and with a requirement for workers R_i indicating the optimal number of employees required at that timeslot.
- s shift types $t_1, \ldots, t_s$, each of them associated with a minimum and maximum start time $smin_s$ and $smax_s$, given in timeslots of the current day, as well as minimum and maximum length $lmin_s$ and $lmax_s$ in timeslots. Shifts can extend to the following day, shifts extending beyond a_{n+1} continue from a_1.

A feasible solution only uses shifts within the time windows specified by the shift types. Among feasible solutions, the following criteria have to be minimized as a weighed sum:

- T_1: Excess of workers (minute scale).
- T_2: Shortage of workers (minute scale).
- T_3: Number of shift classes in use. A shift class is considered in use if there is any day in the planning period where employees are assigned to this shift.
- T_4: Deviation of the average shift length from the defined window. This window is derived from a minimum and maximum number of shifts per week that are supposed to be assigned to each employee in average and the number of working hour per week per employee.

A small example demand with a solution with no understaffing or overstaffing and three different shifts in use is shown in Figure 2.

Hyper-heuristics Setup

In this paper we investigate for the first time hyper-heuristics for the three scheduling domains described in the previous section. We provide an extensive comparison of several different hyper-heuristics on existing benchmark instances. For each of the domains we propose new low-level heuristics which are an essential part of hyper-heuristics.

Low-level Heuristics

For each of the problem domains we use three categories of low-level heuristics (LLH), which are improvement heuristics that are guaranteed to result in either better or same quality solutions, mutations which can change solution quality in any direction, and destroy-and-repair heuristics that are supposed to structurally change the solution. These categories

are based on three of the four possible categories in the HyFlex framework. Since the first two of these are similarly implemented for all three domains, we describe the common heuristics first, and the used moves of the individual domains as well as the much more specific destroy-and-repair heuristics afterwards.

Improvement and Mutation Heuristics. Improvement heuristics are various versions of hill climbers. While each domain uses different moves, the algorithms in use are:

- First improvement with at most 1 or 10 steps
- Best improvement with at most 1 or 10 steps
- Random improvement with at most 1000 iterations or 10 consecutive iterations without improvement

Mutations are based on two different sets of heuristics:

- Random improvement with metropolis acceptance criterion for mutations with a tendency to move towards improving or slightly worsening solutions, with at most 1000 iterations or 10 consecutive iterations without improvement and the following acceptance probability for worsening solutions (where Δ is the change in solution value):
$$p = \exp\left(\frac{-\Delta}{t}\right) \tag{2}$$
Values for t are fixed to 1, 10, and 100.
- Random walk is used for mutations which typically significantly worsen the solution. Randomly chosen moves are applied 1 or 10 times in a row.

Specific implementations for BDS. The heuristic for the initial solution assigns bus legs to the shifts where they cause the least cost increase or to a new shift if cheaper. The following two moves based on Kletzander and Musliu (2020) are used for improvement and mutation heuristics:

- `Swap`: Choose one bus leg ℓ from one employee e_1 and a second employee e_2. Move ℓ from e_1 to e_2 and move back all bus legs in e_2 which now overlap with ℓ to e_1.
- `SequenceSwap`: Choose a consecutive sub-sequence $\ell_i, \ldots, \ell_j$ from one employee e_1 and a second employee e_2. Then move the whole sub-sequence from e_1 to e_2 and move back all bus legs in e_2 which now overlap with any bus leg $\ell_i, \ldots, \ell_j$ to e_1.

Since the `SequenceSwap` neighborhood is rather large, first and best improvement are only used with a single step. The random improvement mutations are only used with `SequenceSwap` to prevent an excessive number of mutation heuristics.

The new destroy-and-repair heuristics focus on removing one or multiple employees from the solution and rebuilding it using the construction heuristic. They are particularly important as they are the only LLHs for this domain that can change the number of employees in the solution, which can be very important for solution quality:

- `RemoveEmployee`: An employee is chosen randomly and removed from the solution. All bus legs previously assigned to this employee are reassigned by the construction heuristic. An employee might be fully removed if it is possible to reassign all bus legs to other employees.
- `RemoveTour`: A tour is randomly chosen from the instance, all employees assigned to any part of this tour are removed from the solution. The tour might be arranged in a less fragmented way during reassignment.

Specific implementations for RWS. For the representation of this domain the shifts are already fixed to their columns corresponding to all workforce requirements already fulfilled, resulting in the following moves:

- `SimpleSwap`: Choose a column c and two rows r_1 and r_2, and replace the corresponding two shifts.
- `Swap`: Choose a column c, two rows r_1 and r_2, and a length $\ell \leq 7$, and perform a simple swap for ℓ consecutive cells starting from column c (moving to the next rows and first column when going past the last column). This exchanges up to a week of consecutive work assignments.

The moves are similar to those used by Musliu (2006). The initialization is more sophisticated than in previous heuristics for the problem. Instead of just randomly assigning shifts within each column, shifts are assigned according to the following algorithm: For each column, for each shift that needs to be assigned to the column, assign it to the row with the least increase in objective value. This already forms useful sequences of shifts during the construction process.

Again, first and best improvement with the larger `Swap` neighborhood are only used with a single step. For the mutations only the full `Swap` neighborhood is used, since it includes the smaller `SimpleSwap` anyway.

Two new destroy-and-repair heuristics are used:

- `RemoveDay`: A random column c is chosen, all assignments for this day are removed and reassigned using the construction heuristic. The significance is that all rows are affected at once with this heuristic.
- `RemoveWeeks`: A random range of up to 10 consecutive weeks is chosen, all assignments from these weeks are removed and reassigned using the construction heuristic. This allows to reorganize larger consecutive blocks of shifts.

Specific implementations for MSD. Three different new neighborhoods are used for this domain:

- `NewShift`: A day d and shift type t are chosen. Then, a shift within the bounds set by the chosen shift type t is added to the schedule on day d.
- `RemoveShift`: A shift s is chosen and removed from the schedule.
- `ChangeShift`: A shift s is chosen and removed from the schedule. Then, a shift type t is chosen and a new shift within the bounds set by t is created and added to the schedule on day $d(s)$, the previously assigned day of shift s. This combines the previous two moves without the need to deal with a potential situation of low cover in-between.

Compared to these moves, Musliu, Schaerf, and Slany (2004) use a larger set of different moves more focused on shift classes, while we use fewer moves focused on individual shifts, to prevent dealing with an excessive number of

LLHs. NewShift and RemoveShift are used with 1 and 10 steps, the larger ChangeShift only with one step for first and best improvement. Additionally a best improvement heuristic with alternating RemoveShift and NewShift applications and at most 10 steps is used. It randomly starts with any of the two neighborhoods. Further a version of this alternating search is included where only the combination of the two neighborhoods needs to show an improvement in the solution value rather than each individual neighborhood.

Due to the objective for reducing the number of different shift classes, it is often more beneficial to try to assign another shift to an already existing shift class instead of a new one. Therefore, when shifts are randomly generated for random improvement and random walk, the following procedure is applied: With 50% probability, an already existing shift class is randomly chosen and the corresponding shift returned, else any shift within the bounds a randomly chosen shift type is generated. The different neighborhoods are randomly selected with equal probabilities in each iteration of the randomized heuristics. The initialization heuristic uses random improvement with this setting for up to 10000 iterations or 100 iterations without improvement.

The new destroy-and-repair heuristics we propose for the MSD combine two different remove operators with two different repair operators resulting in four heuristics. The remove operators are:

- RemoveClass: A shift class is randomly chosen and all shifts within this class are removed. This is supposed to help getting rid of potentially less useful shift classes.
- RemoveDay: All shifts on a particular day are removed, allowing to reschedule this day at once while leaving the other days untouched.

The repair operators are based on best improvement to focus on the best replacement shifts rather than introducing additional shift classes by using randomized repair operators:

- RepairExisting: Use only shifts from shift classes already in use to repair the schedule, filling the gaps as well as possible without introducing new shift classes.
- RepairFull: Use full best first search, setting the focus more on covering the demand .

Hyper-heuristics

We critically evaluate and compare ten state of the art hyper-heuristics including approaches that provided very good results in the hyper-heuristic competition and recent approaches that were implemented by Mischek and Musliu (2021). All used hyper-heuristics are implemented in HyFlex. This ensures that they are applied under the same conditions.

- ALNS: Self-adaptive large neighborhood search based on Laborie and Godard (2007) with alternating perturbation and reconstruction moves, learning weights for the LLHs based on their performance.
- BSW-ALNS: Bigram sliding window ALNS is a version of ALNS where the weights for the reconstruction moves are not learned independently, but based on the preceding perturbation move. The moving time windows based on Thomas and Schaus (2018) are supposed to deal with the fact that different LLHs take different amounts of time by taking this into account when calculating the weights.
- SW-ALNS: Sliding window ALNS extends ALNS only with the time windows, but not the bigram extension.
- CH-BI (Bigram): Chuang (2020) describes several methods in his thesis based on solution chains. If any solution in the chain is better than the starting solution, it is accepted and the chain stopped, otherwise the whole chain is discarded. Chain lengths are chosen according to the Luby sequence. The way to choose heuristics is different for the four variants in this comparison. For CH-BI the probability to select a heuristic depends on the previous heuristic in the chain and the number of times this pair of heuristics occured in successful chains.
- CH-FR (Frequency): In this case the probability to choose a heuristic depends on the number of times it appeared in successful chains.
- CH-PR (Pruning): Heuristics with bad performance are pruned after a warm-up period, otherwise uniform random selection.
- CH-UN (Uniform): Here the heuristics are chosen according to a uniform random distribution.
- GIHH: This approach by Misir et al. (2011) was the winner of CHeSC 2011. It uses an adaptive dynamic heuristic set to monitor the performance of each heuristic to select heuristics in different phases, finds effective pairs of heuristics, and uses a threshold accepting method.
- L-GIHH (Lean GIHH): This hyper-heuristic was obtained by performing Accidental Complexity Analysis on GIHH by Adriaensen and Nowé (2016), and was reported to provide similar or even slightly better performance while greatly reducing the complexity of GIHH.
- HAHA: Lehrbaum and Musliu (2012) proposed a hyper-heuristic which switches between working on a single solution and a pool of solutions together with an adaptive strategy for selecting LLHs.

Evaluation

All problem domains and low-level heuristics were implemented in Python and run using PyPy 7.3.5 for adequate speed. The hyper-heuristics were implemented in the Java HyFlex framework and run using OpenJDK 8u292. A general interface that allows hyper-heuristics to run on domains outside of HyFlex is implemented and used to connect the different parts of the software. All evaluations have been performed with one hour of runtime, which was also the maximum runtime for compared methods except when stated otherwise. This includes both PyPy and Java execution, but not the transmission times of the interface to transparently be able to compare hyper-heuristics implemented in HyFlex or externally. The experiments were run on a computing cluster with Intel Xeon CPUs E5-2650 v4 (max. 2.90GHz, 12 physical cores, no hyperthreading), but each individual run was performed single-threaded. Each method was run on each instance 5 times to account for random variations. For

Instance	SA	HC	BP	SW-ALNS	CH-PR	GIHH	L-GIHH	HAHA
10	**14717.4**	14904.4	*14709.2*	14875.6	14805.6	14787.0	14773.6	14966.6
20	30860.6	30931.4	*30290.3*	30941.4	**30671.2**	30731.6	30694.0	31222.4
30	50947.4	51544.2	*49846.4*	51164.2	50903.6	**50765.8**	50854.2	51642.6
40	69119.8	69533.6	*67000.4*	69360.2	68847.6	**68639.6**	68645.4	69866.8
50	87013.2	**86718.6**	*84341.0*	87375.8	87034.0	86762.0	86729.8	88062.8
60	103967.6	103780.0	*99727.0*	103876.8	103464.8	**103138.8**	103149.8	104434.4
70	122753.6	122912.8	*118524.2*	122597.8	122025.6	121671.8	**121660.6**	122688.2
80	140482.4	139765.2	*134513.8*	139747.2	139209.2	139123.0	**139041.6**	140504.4
90	156385.0	156239.4	*150370.8*	155374.4	**154972.4**	155093.8	155113.2	156052.4
100	173524.0	172327.8	172582.2	172406.4	***171182.4***	171278.2	171325.4	172318.0

Table 2: Best results for BSD with different solution methods

brevity, this section shows the most important results, full result tables are available online[2]. The evaluation uses publicly available benchmarks, which include randomly generated, but also real-world examples, to allow comparison with previous and future work.

Bus Driver Scheduling

This domain was evaluated on the set of 50 benchmark instances used by previous work[3]. They span 10 size categories from around 10 tours (70 legs) to around 100 tours (1000 legs) based on real-life demand distributions.

Table 2 shows the average of the best results per instance category for the hyper-heuristics (only best version for ALNS and CH) in comparison to the previous best results (Simulated Annealing, Hill-climber, Branch and Price). The entry in bold highlights the best heuristic result (all methods except BP), the value in italics the best overall result. All five included hyper-heuristics can outperform previous heuristic results on at least 3 categories, with CH-PR, GIHH, and L-GIHH outperforming them in 8 categories.

Branch and Price is an exact method that provides near-optimal solutions for most instances except the largest ones in the data set. However, it has two major disadvantages. First, it has problems scaling to larger instances. Real-life instances are often even larger than the largest benchmark instances. However, even on the largest size of the given instances the hyper-heuristics provide better results than BP, despite BP using twice as much time for these instances. In fact, several methods improve the previous best known solutions for two out of the five instances in the largest category for this problem. Therefore, hyper-heuristics can be considered the new state-of-the-art method for very large instances. Second, compared to the rule-specific implementation of the sub-problem in BP, hyper-heuristics can be adapted to different rule-sets easily, which is often important in practice.

Rotating Workforce Scheduling

This domain was evaluated on the original set of 20 real-life benchmark instances[4]. While there are more randomly generated instances available, we compare to previous results for the optimization objectives (Kletzander et al. 2019).

In contrast to the other domains this domain has a much stronger emphasis on feasibility. For several instances, it is already difficult to find a feasible solution. We used a constant weight of 100 for hard constraint violations and scaled the optimization objectives to be small enough to reliably receive feasible solutions. Table 3 shows the number of feasible results per method and objective. GIHH and L-GIHH obtain feasible results on all instances, with only few exceptions in some individual runs. The other methods, however, are not able to solve some of the more challenging instances. The versions of CH and HAHA perform in a similar way, ALNS can solve even fewer instances.

Looking at the optimization results, there is a strong division between easy and hard instances that was already visible in previous work: For several instances, all methods reliably reach the optimum on every single run. E.g., for F_1, for 5 out of 20 instances every method obtained the optimum in each of the 5 runs, for 3 further instances at least one of the 5 runs reached the optimum for each method. On the other hand, for several of the hard instances previously no result was obtained in the timeout of one hour.

Table 4 shows a comparison of hyper-heuristic results with the previous Minizinc solutions (Kletzander et al. 2019) only on those instances where GIHH, L-GIHH and MiniZinc did not all find the optimum. Instances are shown as *objective-instance*. Note that objective 1 is a maximization objective, while 2 and 3 are minimization objectives. The results show that the hyper-heuristics can improve previous best known results for 12 out of 27 hard instances. In particular, solutions can be provided for all 5 instances which previously ran into timeout. On the other hand, for those instances where the optimum could not be reached, the distance to the optimum is often very small.

In comparison regarding the optimization, GIHH is able to provide more best solutions than L-GIHH which is still the second best method. HAHA can provide one significantly better best known solution (1-20), showing also for other instances that if it can provide feasible solutions, those are often of good quality, but those values are not reached reliably. While the other hyper-heuristics are less competitive, they can still improve some of the results compared to Minizinc on the very difficult instances.

[2]https://cdlab-artis.dbai.tuwien.ac.at/papers/hyper-heuristics-personnel/

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

[4]https://www.dbai.tuwien.ac.at/staff/musliu/benchmarks/

Objective	ALNS	BSW-ALNS	SW-ALNS	CH-BI	CH-FR	CH-PR	CH-UN	GIHH	L-GIHH	HAHA
F_1	62	50	67	82	79	74	78	**98**	97	79
F_2	64	59	67	78	83	78	80	**99**	98	79
F_3	68	57	68	82	80	76	77	**100**	99	78

Table 3: Number of feasible solutions out of 100 per hyper-heuristic

Obj-Inst	MiniZinc	GIHH	L-GIHH	HAHA
1-9	34	**35**	**35**	34
1-11	4	**6**	**6**	5
1-12	**8**	7	**8**	7
1-15	-	**15**	13	10
1-18	**23**	21	20	22
1-19	24	**26**	23	18
1-20	-	32	26	**37**
2-9	**2**	3	3	3
2-10	**3**	4	4	5
2-11	**5**	6	6	13
2-12	**4**	5	5	5
2-13	**4**	5	5	6
2-15	-	**7**	8	-
2-16	**4**	5	6	6
2-17	**4**	5	5	6
2-18	**4**	6	7	8
2-19	28	**8**	11	23
2-20	-	**11**	15	25
3-9	**26522**	**26522**	26524	26524
3-10	**8772**	8784	8778	8778
3-11	21710	**20783**	**20783**	23596
3-12	**4836**	5243	**4836**	**4836**
3-15	-	**196806**	200887	209213
3-16	**16868**	16900	16888	16890
3-18	**84414**	87218	87218	87218
3-19	1456141	**1253063**	1267498	1339601
3-20	3855876	**3268396**	3295030	-

Table 4: Best result comparison for hard RWS instances

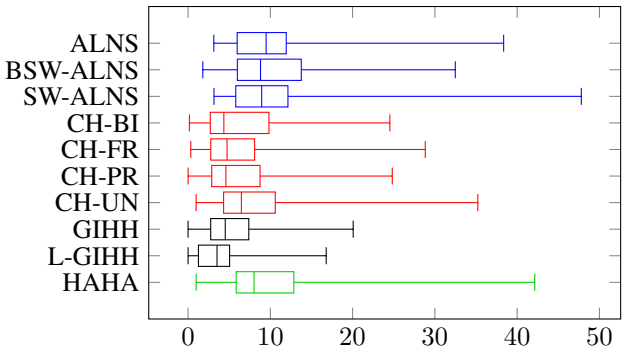


Figure 3: Best result deviations for reduced MSD

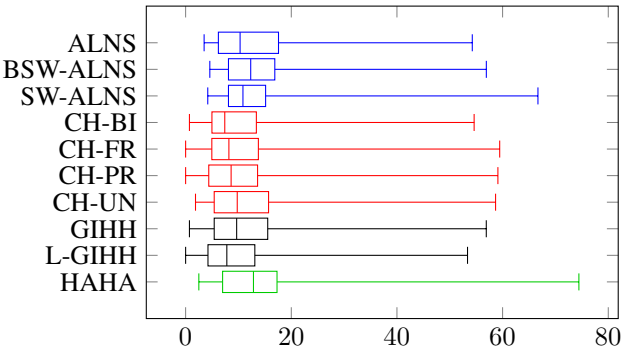


Figure 4: Best result deviations for full MSD

Minimum Shift Design

For this domain a large set of instances is available[5]. In particular we use 93 instances in four data sets, but focus on sets 3 (30 realistic instances) and 4 (3 real-life instances), where demand and shifts will not align perfectly, since the first sets are artificially created to allow an exact cover which is unrealistic in practice.

However, the forth objective was only considered by Musliu, Schaerf, and Slany (2004), but with evaluation only on the first data set. Furthermore, the weight of T_4 is set to 0 in the instance generator as soon as the objective interferes with the perfect cover. Therefore, either T_4 is not considered or does not have an effect anyway. In our evaluation, we use the bounds in the provided instances, but with a fixed weight of 1000 instead of the weight from the instances which is either 1000 or 0.

To have a baseline comparison, Figure 3 provides the deviation from the optimal results by Kletzander and Musliu (2019) in percent for each hyper-heuristic on the realistic data sets 3 and 4 without using T_4. This provides two conclusions. First, it shows the differences in the hyper-heuristics. Here, L-GIHH clearly provides the best performance. In contrast to the previous domains, however, GIHH shows a larger gap to L-GIHH. While regarding the worst deviation GIHH is still on the second place, the CH methods (except CH-UN) provide very good results in the average case and CH-BI outperforms GIHH in the median as well as upper and lower quartile. ALNS and HAHA perform significantly worse.

Second, the results show that especially L-GIHH can provide high quality results on those real-life instances, with the upper quartile at a gap of 5 %. While the exact method is clearly stronger on the reduced problem formulation, it would be much more difficult to adapt to the forth objective. For the hyper-heuristics setup, however, this is very easy to achieve and we can for the first time provide benchmark results on all instances with the forth objective enabled.

[5]https://www.dbai.tuwien.ac.at/proj/Rota/benchmarks.html

Method	BDS	F_1	F_2	F_3	MSD	$+T_4$
ALNS	3	10	7	7	10	11
BSW-ALNS	4	9	6	5	16	16
SW-ALNS	5	11	6	7	16	18
CH-BI	4	12	11	11	28	32
CH-FR	9	13	13	8	25	30
CH-PR	11	12	13	8	25	38
CH-UN	5	12	12	9	18	19
GIHH	13	**17**	**20**	**16**	28	25
L-GIHH	**15**	16	15	15	**76**	**46**
HAHA	1	13	8	11	9	14

Table 5: Number of wins for each hyper-heuristic

Figure 4 shows the deviations for MSD with the forth objective compared to the optimal solutions for reduced MSD. These are now only a lower bound that is not reachable for most instances, but it allows the same relative comparison which shows that the relative performance of the individual hyper-heuristics is very similar to before. However, this time the lowest median is actually achieved by CH-BI, while the best quartiles are still achieved by L-GIHH, but less distinct. It also shows that for a majority of instances, including this additional objective will raise the objective value by a few percent, while for a few instances it raises quite significantly.

Comparison of Hyper-heuristics

Table 5 shows the number of times a hyper-heuristic obtained the best result out of all 10 hyper-heuristics in comparison on an instance (ties are awarded to all hyper-heuristics involved in the tie). This final comparison sums up the picture from the previous sections regarding the relative performances of the hyper-heuristics: L-GIHH is the best heuristic on two out of the three domains, especially clearly in MSD, and only beaten by the original GIHH on the RWS domain. The CH versions (except CH-UN) perform well on all domains, sometimes even outperforming GIHH. HAHA shows reasonably good performance for RWS, but struggles with the other domains.

The versions of ALNS did not perform well. While several options were tried before the comparison for its parameters (5 minutes warmup, 1 minute window length), performance might increase with more careful tuning. However, no tuning was required for the other hyper-heuristics. Note that even though the LLHs have several parameters as well, different options of LLHs with different parameter settings were successfully used across all three domains to prevent the need for detailed tuning.

While the quality of the best results seems to depend on the set of available LLHs, the winning hyper-heuristics show to be very efficient at selecting a good mix relative to their competitors in different scenarios (even in filtering out ill-behaving LLHs when a bug occurred for one of the domains). We tracked the number of heuristic applications for each heuristic and each run and found some interesting patterns: First, different hyper-heuristics show several different LLH preferences. Second, the same hyper-heuristic usually favors similar LLHs for different instances of the same domain, but some heuristics are actually swapped depending on the instance.

Specifically looking at the destroy-and-repair heuristics shows that for most of the instances and hyper-heuristics in the different domains, the destroy-and-repair heuristics are frequently used. Further, some additional experiments showed that without any destroy-and-repair heuristics, e.g., on BDS the winning method L-GIHH looses its ability to outperform the results from literature on all but one instance, clearly showing that these are relevant to the performance. They seem similarly important for MSD, while less important for RWS.

Conclusion

In this paper, we provided a major study on using various hyper-heuristics to solve different domains in personnel scheduling which include complex constraints and combined optimization goals from practical applications. We provided new low-level heuristics for three different scheduling domains used by our industry partner, applying hyper-heuristics to them for the first time. We compared several versions of state-of-the-art hyper-heuristics and provided multiple comparisons for the relative performance of the hyper-heuristics on the individual domains. The results showed that especially Lean GIHH, but also the original GIHH are very well suited to efficiently solve the different scheduling domains.

With these methods we were able to provide several advances for the individual domains. For BDS, we were able to outperform previous heuristic approaches and find two new best known solutions for the benchmark data set. For RWS, we could provide several new best known solutions for the different optimization goals including several solutions for instances previously running into timeouts without a solution. For MSD, for the first time we provided comprehensive results using the forth objective that has high practical relevance, hopefully fuelling more research into this more realistic version of the problem.

In practice, these results are very useful in many ways. Besides the specific improved solutions, the results show that using hyper-heuristics for complex practical problems is a very useful approach to provide high-quality solutions without spending too much time for highly problem-specific solution methods. In real-life applications there are often different variations of these problems, frequently including customer-specific constraints or adapted optimization goals. However, the LLHs are independent from the additional constraints or different objectives, so they do not need to be adapted to different customers, allowing rapid adaptation for changing requirements. Therefore, deployment of the leading hyper-heuristics in the industrial software of our partner will be targeted next.

In future work we plan to investigate the effects of the different LLHs in more detail, including the analysis of the chosen combinations of LLHs and the importance of specific sets of these heuristics for the solution quality.

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

Adriaensen, S.; and Nowé, A. 2016. Case study: An analysis of accidental complexity in a state-of-the-art hyper-heuristic for HyFlex. In *2016 IEEE Congress on Evolutionary Computation (CEC)*, 1485–1492. IEEE.

Baker, K. R. 1976. Workforce allocation in cyclical scheduling problems: A survey. *Journal of the Operational Research Society*, 27(1): 155–167.

Becker, T. 2020. A decomposition heuristic for rotational workforce scheduling. *Journal of Scheduling*, 23(5): 539–554.

Burke, E. K.; De Causmaecker, P.; Berghe, G. V.; and Van Landeghem, H. 2004. The state of the art of nurse rostering. *Journal of scheduling*, 7(6): 441–499.

Burke, E. K.; Gendreau, M.; Hyde, M.; Kendall, G.; McCollum, B.; Ochoa, G.; Parkes, A. J.; and Petrovic, S. 2011. The cross-domain heuristic search challenge–an international research competition. In *International Conference on Learning and Intelligent Optimization*, 631–634. Springer.

Burke, E. K.; Gendreau, M.; Hyde, M. R.; Kendall, G.; Ochoa, G.; Özcan, E.; and Qu, R. 2013. Hyper-heuristics: a survey of the state of the art. *JORS*, 64(12): 1695–1724.

Burke, E. K.; Hyde, M. R.; Kendall, G.; Ochoa, G.; Özcan, E.; and Woodward, J. R. 2019. A classification of hyper-heuristic approaches: Revisited. In *Handbook of meta-heuristics*, 453–477. Springer.

Chuang, C.-Y. 2020. *Combining Multiple Heuristics: Studies on Neighborhood-base Heuristics and Sampling-based Heuristics*. Ph.D. thesis, Carnegie Mellon University.

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.

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.; and Musliu, N. 2019. Modelling and Solving the Minimum Shift Design Problem. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 391–408. Springer.

Kletzander, L.; and Musliu, N. 2020. 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.; Gärtner, J.; Krennwallner, T.; and Schafhauser, W. 2019. Exact methods for extended rotating workforce scheduling problems. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, 519–527.

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.

Laborie, P.; and Godard, D. 2007. Self-adapting large neighborhood search: Application to single-mode scheduling problems. *Proceedings MISTA-07, Paris*, 8.

Lehrbaum, A.; and Musliu, N. 2012. A new hyperheuristic algorithm for cross-domain search problems. In *International Conference on Learning and Intelligent Optimization*, 437–442. Springer.

Mischek, F.; and Musliu, N. 2021. A collection of hyper-heuristics for the HyFlex framework. Technical Report CD-TR 2021/1, TU Wien.

Misir, M.; De Causmaecker, P.; Vanden Berghe, G.; and Verbeeck, K. 2011. An adaptive hyper-heuristic for CHeSC 2011. In *OR53 Annual Conference, Date: 2011/09/06-2011/09/08, Location: Nottingham, UK*.

Musliu, N. 2006. Heuristic methods for automatic rotating workforce scheduling. *International Journal of Computational Intelligence Research*, 2(4): 309–326.

Musliu, N.; Gärtner, J.; and Slany, W. 2002. Efficient generation of rotating workforce schedules. *Discrete Applied Mathematics*, 118(1-2): 85–98.

Musliu, N.; Schaerf, A.; and Slany, W. 2004. Local search for shift design. *European Journal of Operational Research*, 153(1): 51–64.

Musliu, N.; Schutt, A.; and Stuckey, P. J. 2018. Solver Independent Rotating Workforce Scheduling. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 429–445. Springer.

Ochoa, G.; Hyde, M.; Curtois, T.; Vazquez-Rodriguez, J. A.; Walker, J.; Gendreau, M.; Kendall, G.; McCollum, B.; Parkes, A. J.; Petrovic, S.; et al. 2012. Hyflex: A benchmark framework for cross-domain heuristic search. In *European Conference on Evolutionary Computation in Combinatorial Optimization*, 136–147. Springer.

Thomas, C.; and Schaus, P. 2018. Revisiting the self-adaptive large neighborhood search. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 557–566. Springer.

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.

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.