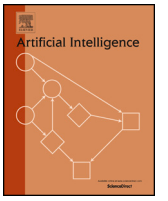
Artificial Intelligence 334 (2024) 104172

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint



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, Vienna, 1040, Austria

ARTICLE INFO

Keywords:
Hyper-heuristics
Personnel scheduling
Combinatorial optimization

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 Rotating Workforce Scheduling, Minimum Shift Design, and Bus Driver Scheduling. 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 four 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. Finally, a detailed investigation of the use of low-level heuristics by the hyper-heuristics gives insights in the way hyper-heuristics apply to different domains and the importance of different low-level heuristics. The 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. Several hyper-heuristics with very good performance across different real-life domains are identified. They can efficiently select low-level heuristics to apply for each domain, but for repeated application they benefit from evaluating and selecting the most useful subset of these heuristics. These results help to improve industrial systems in use for solving different scheduling scenarios by allowing faster and easier adaptation to new problem variants.

1. 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. Even if methods are more flexible, they often require selection of algorithm components and tuning of parameters to work properly in different settings.

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 (LLHs) and comparably few or even no parameter inputs in order to create the

* Corresponding author.
E-mail addresses: lucas.kletzander@tuwien.ac.at (L. Kletzander), nysret.musliu@tuwien.ac.at (N. Musliu).

https://doi.org/10.1016/j.artint.2024.104172
Received 26 June 2023; Received in revised form 8 April 2024; Accepted 11 June 2024

Available online 25 June 2024

Table 1
Example demand.

Shift type	Mon	Tue	Wed	Thu	Fri	Sat	Sun
D	1	1	1	1	1	1	1
A	1	1	1	1	1	1	0
N	1	1	1	1	1	1	1

appropriate heuristic for solving a particular problem instance on the fly based on the performance of the individual LLHs [1,2]. As part of the Cross-Domain Heuristic Search Challenge [3] the framework HyFlex [4] was introduced for a uniform implementation of different hyper-heuristics. There are several aspects that position hyper-heuristics within AI research. First, low-level heuristics and operators often mimic human problem-solving, which is crucial for obtaining good results with hyper-heuristic methods. Secondly, hyper-heuristic methods encompass various AI techniques, including machine learning techniques that involve sophisticated learning processes based on the characterization of the search process, as well as AI-based problem-solving techniques. Finally, hyper-heuristics offer possibilities to automate algorithm design and therefore provide techniques that often can only be produced with a lot of effort by highly skilled human experts in algorithm development. Therefore, such methods contribute to further automating the problem-solving capabilities of machines and clearly fall into the area of AI.

This paper is a significant extension of a previous conference paper [5], which won the Best Industry and Applications Track Paper Award in ICAPS 2022. We present a deep investigation of hyper-heuristics to three challenging personnel scheduling domains of high practical relevance for the first time, which are Rotating Workforce Scheduling (RWS), Minimum Shift Design (MSD), and Bus Driver Scheduling (BDS). We first propose several new low-level heuristics for these domains. Then we present a major comparison of 20 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. We identify which hyper-heuristics provide high-quality results across different domains, providing valuable information to select from the large available pool of hyper-heuristics.

Then we move to the investigation of the utility and usage of the low-level heuristics on the individual domains. We investigate the frequency of use for the different LLHs, as well as the spent runtime, identifying different patterns of usage by different hyper-heuristics as well as very different runtimes of the LLHs making it more challenging for the hyper-heuristics to evaluate the trade-off between runtime and solution improvement.

We further examine the patterns of consecutive LLHs and the importance of different classes of low-level heuristics for the different domains. Here we identify different important classes for the different domains, highlighting that depending on the structure of the problem, different approaches are needed to efficiently solve the problem. Besides providing good selections of LLHs for the individual domains and recommendations for the hyper-heuristics to apply, we show that in general it is a good practice to give a large set of low-level heuristics to the hyper-heuristics when solving an unknown problem, but this set should be evaluated and reduced to the most beneficial LLHs when repeatedly solving instances from the same domain for maximum efficiency.

2. 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 [6–9]. 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. We also aim at using the most practically relevant versions of the domains by considering additional optimization objectives with high practical relevance.

2.1. Rotating workforce scheduling

When shifts are already 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 [10]. Over the years several approaches were used to solve the problem, including use in commercial software by XIMES for almost 20 years [11]. A state-of-the-art complete method was introduced by Musliu et al. [12] and further extended by Kletzander et al. [13], 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 [14].

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.

[1] www.ximes.com.

Table 2
Solution for the example.

Employee	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 shows a simple demand matrix for the three shift types day shift (D), afternoon shift (A), and night shift (N), each requiring either 0 or 1 shifts of this type per day of the week. For this example, the goal is to find a cyclic schedule for 4 employees that fulfils the following constraints:

- 5 to 7 consecutive days of work
- 2 to 4 consecutive free days
- Consecutive shifts per type: 2 to 5 for D, 2 to 4 for A, 2 to 3 for N
- No D after A or N, no A after N

Table 2 shows a solution for the example using four employees (or four equal-sized groups of employees). Employees start their schedules 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 et al. [11,12]:

- 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. [13]. 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.

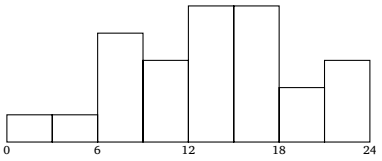
The actual optimization objectives are obtained by penalizing hard constraint violations v with a factor of 100 and adding the scaled optimization objective:

$$o_1 = 100 \cdot v + 0.1 \cdot (UB - F_1) \tag{1}$$

$$o_2 = 100 \cdot v + F_2 \tag{2}$$

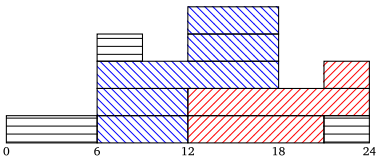
$$o_3 = 100 \cdot v + 0.0001 \cdot F_3 \tag{3}$$

Since F_1 is a maximization objective, but otherwise we only deal with minimization, $UB - F_1$ is used to turn o_1 into a minimization objective, where UB is an upper bound for F_1, ensuring that o_1 never gets negative. The very small scale for F_3 is due to the squares of distances getting large very fast.



Shift type	t_1	t_2
Min start	6:00	12:00
Max start	6:00	21:00
Min length	6:00	6:00
Max length	12:00	12:00

Fig. 1. Example instance.



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

Fig. 2. Solution for the example.

2.2. Minimum shift design

The shifts that are assigned in RWS can be obtained in several different ways. The second domain deals with one option, which is that 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 et al. [15] where a tabu search is proposed. This is the only work that also partially considers a fourth 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 [16] uses a network-flow based formulation to provide optimal solutions to all benchmark instances, however, without the fourth objective.

The specification is based on Musliu et al. [15]. The problem input is defined as follows:

- n consecutive timeslots $[a_1, a_2)$, $[a_2, a_3)$, …, $[a_n, a_{n+1})$, 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.

Fig. 1 shows a small example problem with demand for only one day and two available shift types.

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. Each combination of shift start time and shift length forms a shift class. A shift class is considered in use if there is any day in the planning period where employees are assigned to this class.
- 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.

The optimal solution for the example with no understaffing or overstaffing and three different shift classes in use is shown in Fig. 2.

2.3. Bus driver scheduling

The third domain under consideration is Bus Driver Scheduling (BDS), which is a part of crew scheduling in the process of operating bus transport systems [17]. It has been considered by many authors starting with Wren et al. [18], 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 et al. [16], who also provided challenging instances that are publicly available. The best known solutions for most instances were achieved by a highly specialized Branch-and-Price approach [19]. However, this solution approach is highly problem specific and not able to be adapted easily to similar problems with slightly different constraints, and it does not scale well to larger instances.

Fig. 3. Tours for an example problem.

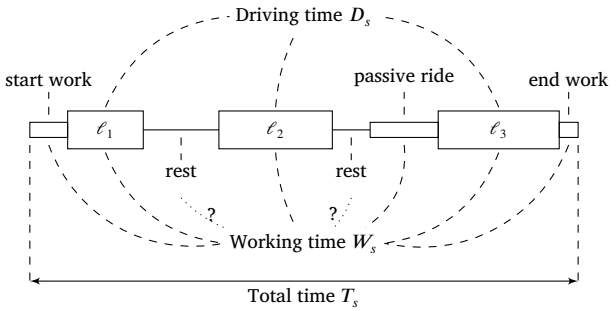


Fig. 4. Example shift for BDS [19].

An extension for application of this approach to larger instances has very recently been proposed using Large Neighbourhood Search [20]. There is also much attention on developing specialized heuristics for this problem, including Tabu Search [21] and Construct, Merge, Solve and Adapt (CSMA) [22] providing increasingly competitive heuristic solutions.

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.

Fig. 3 shows the tours for a small instance. Bus tours 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 Fig. 4. 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 et al. [16]:

- 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.

$$cost_s = 2 \cdot W'_s + T_s + ride_s + 30 \cdot ch_s + 180 \cdot split_s \tag{4}$$

The objective (4) 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. Fig. 5 shows the optimal solution according to this objective function for the example.

3. Hyper-heuristics setup

In this paper we investigate 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.

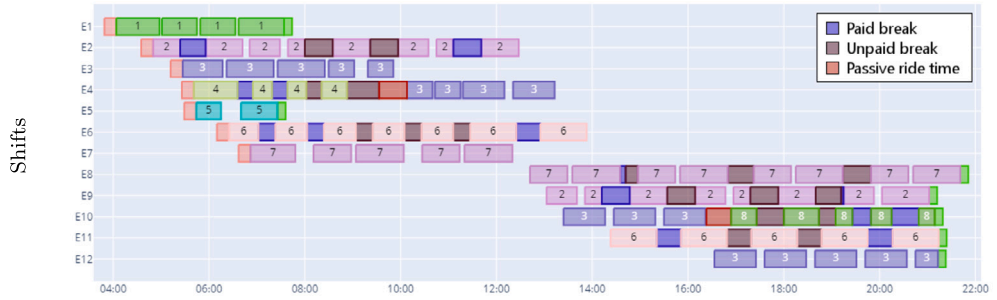


Fig. 5. Tours for an example problem. (For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.)

3.1. Low-level heuristics

For each of the problem domains we use three categories of low-level heuristics:

- Local Search: Improvement heuristics that are guaranteed to result in either better or same quality solutions.
- Mutations: They can change solution quality in any direction.
- Destroy and repair: These heuristics structurally change the solution by destroying and rebuilding a selected part of the solution.

These categories are based on three of the four possible categories in the HyFlex framework, where the last one would be crossover, which is not used since we rather focus on single-solution methods. Since local search and mutations 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.

When LLHs are used in combination with some parameters, different versions of the LLHs are created and presented to the hyper-heuristics to allow them to choose those with the more suiting parameter settings. Some deviations to prevent LLHs that take too long to execute are mentioned later regarding the individual domains.

3.1.1. 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 (F) with at most 1 or 10 steps to a better solution
- Best improvement (B) with at most 1 or 10 steps to a better solution
- Random improvement (R) with at most 1000 iterations (evaluations of a random neighbour) or 10 consecutive iterations without improvement

Mutations are based on two different sets of heuristics:

- Random improvement with metropolis acceptance criterion (M) 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):

$$p = \exp\left(\frac{-\Delta}{t}\right) \tag{5}$$

  Values for t are fixed to 1, 10, and 100.
- Random walk (W) is used for mutations which typically significantly worsen the solution. Randomly chosen moves are applied 1 or 10 times in a row.

Destroy-and-repair (DR) operators are specific to the domains.

3.1.2. Specific implementations for RWS

For the representation of this domain the shifts are already fixed to their columns in the solution matrix S 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 exchange 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 previously [23]. 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.

First and best improvement are used with 1 or 10 iterations of `SimpleSwap` (FS1, FS10, BS1, BS10). With the larger `Swap` neighbourhood they are only used with a single step since they take longer to execute (multiple swaps, labelled FM1, BM1). For the mutations only the full `Swap` neighbourhood is used, since it includes the smaller `SimpleSwap` anyway and mutations are fast. (RM for random improvement, M1, M10, M100 for Metropolis acceptance with different values for t, and W1, W10 for 1 or 10 iterations of random walk.)

Two new destroy-and-repair heuristics are used:

- `RemoveDay` (DD): 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` (DW): 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.

3.1.3. Specific implementations for MSD

Three different new neighbourhoods 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, previous work [15] uses 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.

Since still more moves than for the other domains are used, more options are available for the corresponding heuristics. `NewShift` and `RemoveShift` are used with 1 and 10 steps for first and best improvement (FN1, FN10, FR1, FR10, BN1, BN10), the larger `ChangeShift` only with one step (FC1, BC1). Additionally a best improvement heuristic with alternating `RemoveShift` and `NewShift` applications and at most 10 steps is used (A). It randomly starts with any of the two neighbourhoods. Further a version of this alternating search is included where only the combination of the two neighbourhoods needs to show an improvement in the solution value rather than each individual neighbourhood (AI).

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 of a randomly chosen shift type is generated. The different neighbourhoods are randomly selected with equal probabilities in each iteration of the randomized heuristics. RC labels random improvement, M1, M10, M100 represent Metropolis acceptance with different values for t, and W1, W10 represent random walk with 1 or 10 iterations. 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 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.

Therefore, the four combinations are labelled DCE, DCF, DDE, and DDF.

3.1.4. 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 the Simulated Annealing solution method 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.

Individual swaps are used for 1 or 10 iterations of first and best improvement (FI1, FI10, BI1, BI10). Since the `SequenceSwap` neighbourhood for multiple swaps is rather large, it is only used with a single step (FM1, BM1). Random improvement is used with both neighbourhoods (RI, RM), while Metropolis acceptance is only used with `SequenceSwap` to prevent an excessive number of mutation heuristics (M1, M10, M100 for different values of t), and random walk is again used with both neighbourhoods for 1 or 10 iterations (WI1, WI10, WM1, WM10).

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` (DE): 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` (DT): 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.

3.2. Hyper-heuristics selection

We critically evaluate and compare 20 hyper-heuristics. The large selection includes both hyper-heuristics from the original CHeSC 2011, including the winner GIHH, as well as state-of-the-art hyper-heuristics using different approaches based on Iterated Local Search or Reinforcement Learning, which provided the best results on the competition instances in recent time. All used hyper-heuristics are implemented in HyFlex. This ensures that they are applied under the same conditions and adhere to the same ruleset.

Hyper-heuristics aim to select the best low-level heuristics to improve the objective of the instance that is given to them. For almost all methods, this is done by some kind of adaptive mechanism or learning, e.g., using reinforcement learning [24]. In the HyFlex setting that is also used for this work, domains and instances are unknown to the solution method, therefore learning needs to be done online while solving the instance. This requires the use of AI methods that are fast, can work with very stochastic data (the low-level heuristics typically use a high degree of randomness), and still discover meaningful patterns for the efficient application of low-level heuristics. Furthermore, the development of appropriate low-level heuristics and the utilization of relevant features to characterize the search process are crucial aspects in optimizing the search.

- ALNS: Self-adaptive large neighbourhood search based on Laborie et al. [25] with alternating perturbation and reconstruction moves, learning weights for the LLHs based on their performance according to the update rule $w_{t+1} = (1-\alpha)\cdot w_t + \alpha \cdot r$, where the reward r is calculated as cost improvement over time.
- BSW-ALNS: Bigram sliding window ALNS is a version of ALNS where the weights for the reconstruction moves are not learnt independently, but based on the preceding perturbation move. The moving time windows based on Thomas et al. [26] are supposed to deal with the fact that different LLHs take different amounts of time by taking this into account when calculating the weights $w(o) = (1-\lambda)\cdot L(o) + \lambda \cdot \frac{L}{T}\cdot T(o)$, where $L(o)$ is the local efficiency of operator o (improvement over time in a time window going back to the time of finding the current best solution minus a fixed value), $T(o)$ is the total efficiency of o (improvement over time since starting the algorithm), and L and T are the combined local and total efficiency of all operators.
- SW-ALNS: Sliding window ALNS extends ALNS only with the time windows, but not the bigram extension.
- CH-BI (Bigram): Chuang [27] 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 [28]. 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 occurred in successful chains, using the maximum likelihood estimate $P(h_i|h_{i-1}) = \frac{C(h_{i-1}h_i)}{\sum_{h\in H} C(h_{i-1}h)}$. $C(h_{i-1}h)$ denotes the number of occurrences of h after h_{i-1} in the successful sequences recorded in a warm-up period.
- CH-FR (Frequency): In this case the probability to choose a heuristic depends on the proportion of times it appeared in successful chains in the warm-up period. Two different models are used, one for sequences of length 1, and one for longer sequences.
- CH-PR (Pruning): Heuristics with bad performance (never part of an improving solution chain) 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 [29] 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, aiming for 100 phases in the given runtime. The metric is a weighted combination of, in decreasing order of importance, the number of new best solutions found in the current phase, the total improvement versus worsening using this heuristic in the current phase, and the total improvement versus worsening using this heuristic overall. Heuristics with a bad ranking or very slow runtime are excluded for some phases using a tabu mechanism, or eventually forever, if they are excluded too often. Relay hybridisation is used to learn effective pairs of heuristics based on a linear reward-inaction update scheme. A threshold accepting method is used as an adaptive diversification strategy.
- L-GIHH (Lean GIHH): This hyper-heuristic was obtained by performing Accidental Complexity Analysis on GIHH [30], and was reported to provide similar or even slightly better performance while greatly reducing the complexity of GIHH.
- HAHA: Lehrbaum et al. [31] proposed a hyper-heuristic which switches between working on a single solution and a pool of 7 perturbed solutions together with an adaptive strategy for selecting local search heuristics that evaluates the number of times the LLH resulted in an accepted solution over the total runtime of this heuristic. The local search heuristics are then applied in decreasing order of their quality score.
- SSHH: A sequence-based selection hyper-heuristic utilising a hidden Markov model to analyse the performance of, and construct, longer sequences of low level heuristics to solve difficult problems [32]. Each low-level heuristic is associated with three probability matrices, which handle acceptance strategy, parameters for the LLH, and the transition probabilities to other LLHs. These probabilities are updated when new best solutions are found.
- MS: This iterated multi-stage hyper-heuristic approach cycles through two interacting hyper-heuristics and operates based on the principle that not all low level heuristics for a problem domain would be useful at any point of the search process [33]. The first hyper-heuristic determines this subset using a greedy heuristic selection. The second hyper-heuristic only uses this subset with selection probabilities based on the scores from the first. Both hyper-heuristics use a variant of a threshold move acceptance method. Low-level heuristics are used both individually, and in pairs using relay hybridisation. The current implementation contains some slight changes to make it work in the used version of HyFlex.
- FS-ILS: Fair-Share Iterated Local Search by Adriaensen et al. [34] uses iterated local search with a conservative restart condition. Perturbations are picked depending on the success of their application followed by local search. Local search heuristics are used randomly, but with a tabu mechanism if not effective. After local search, acceptance of the new solution is decided by using a variant of the Metropolis condition.
- TS-ILS: This hyper-heuristic based on iterated local search by Adubi et al. [35] is based on FS-ILS, but applies a probabilistic learning technique (Thompson Sampling) to configure the behaviour of the ILS. It assumes to solve an online bandit problem to choose a configuration of perturbations to apply. TS-ILS reported very good results on the HyFlex domains.
- EA-ILS: Based on FS-ILS, this hyper-heuristic by Adubi et al. [36] uses an evolutionary algorithm to control the selection of perturbative LLHs. It chains up to 2 mutation or destroy-and-repair heuristics, and evolves these sequences using mutations. The perturbations are followed by local search selected based on a Hidden Markov Model. The method was reported to provide very good results on the HyFlex domains.
- QLEARN, SARSA, E-SARSA, MC: This set of methods by Mischek and Musliu [37] uses reinforcement learning to learn the next LLH to apply, together with solution chains based on the Luby sequence. The state is represented by the last heuristic, and the remaining number of steps in the chain (truncated at 5). They use an epsilon-greedy policy, and the reward is calculated based on the improvement over time. The different versions use different update rules, which are Q-learning, SARSA, expected SARSA, and Monte Carlo.
- LAST-RL: This hyper-heuristic by Kletzander and Musliu [38] also uses reinforcement learning, but with a more complex state representation based on 15 features including information about the last heuristic, its success, the total improvement and time, the progress in the current chain, and the improvement and time over the last 10 heuristic applications. Tile coding is used as value function approximation, and the epsilon-greedy policy is modified to use exploration probabilities based on iterated local search. LAST-RL reports high-quality results on several real-life scheduling domains.

4. Evaluation of hyper-heuristics and low-level heuristics

All problem domains and low-level heuristics were implemented in a general interval-based framework for personnel scheduling problems 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. 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. 2.90 GHz, 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. An extension of the result tables in this section containing the detailed results for all individual instances is available online.[2] The evaluation is done on publicly available benchmark instances for all domains to allow comparison with previous and future work.

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

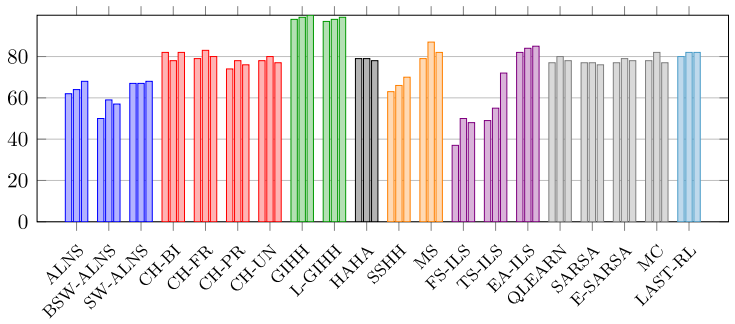


Fig. 6. Number of feasible solutions out of 100 per hyper-heuristic for each of the three optimization objectives.

4.1. Rotating workforce scheduling

This domain was evaluated on the original set of 20 real-life instances from the standard benchmark data set.[3] While there are more randomly generated instances available, we compare to previous results for the optimization objectives [13].

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. Fig. 6 shows the number of feasible results per method and objective (F_1, F_2, F_3 from left to right for each hyper-heuristic). 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. There is a clear plateau around 80 that is reached by most of the hyper-heuristics. This includes the CH heuristics, HAHA, and all the hyper-heuristics based on reinforcement learning. It is even slightly surpassed by MS and EA-ILS. The versions of ALNS can only solve around 60 instances, similar to SSHH, while FS-ILS and TS-ILS perform even worse. While for most hyper-heuristics there is no significant difference between the results for different objectives, the most notable difference is seen for TS-ILS regarding F_3. This objective has fewer plateaus in the solution landscape, which might have an impact on this particular hyper-heuristic.

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 3 out of 20 instances every method obtained the optimum in each of the 5 runs, for 2 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 3 shows a comparison of hyper-heuristic results with the previous solutions obtained using a MiniZinc solver-independent formulation [13] in combination with the solver lazy clause generation solver Chuffed only on those instances where GIHH, L-GIHH and MiniZinc did not all find the optimum. Note that objective F_1 is a maximization objective, while F_2 and F_3 are minimization objectives.

The recent approach by Becker [14] is able to solve the feasibility version of RWS very efficiently was used to provide optimal values for RWS with objective F_1 (Opt.). However, it would be much more difficult to adapt this method to F_2 and F_3 since these objectives require tracking the distance between free weekends. Including these objectives is much easier in the heuristic framework that is used in our evaluation and does not require any adaptation of the low-level heuristics or hyper-heuristics, which highlights the flexibility of our approach.

The results show that the hyper-heuristics can improve previous best known results for 12 out of 27 hard instances compared to MiniZinc (7 out of 20 excluding F_1). Solutions can also 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. MS also provides very good results across the instances, even if it sometimes struggles with feasibility. HAHA can provide one significantly better best known solution (F_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.

Figs. 7, 8, and 9 show the relative comparison of the performance of the hyper-heuristics based on the average gap among the 5 runs to the best solution found among the hyper-heuristics for each instance. For instances where a hyper-heuristic could never find a feasible solution, a gap of 100% (300% for Fig. 8 since feasible gaps might be beyond 100%) has been assumed.

This visualization confirms the superior performance of GIHH, which outperforms the other hyper-heuristics on the majority of instances, followed by L-GIHH. It also shows the great performance of MS, which also finds at least one feasible solution for each instance, even if not as reliable as GIHH and L-GIHH, and shows low average gaps. EA-ILS also shows lower gap values than most remaining hyper-heuristics. Reasonable performance is also shown by the CH variants, by HAHA, and by the RL-based methods. The ALNS versions, SSHH, FS-ILS and TS-ILS struggle to get good results.

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

Table 3
Best result comparison for hard RWS instances.

Obj.	Inst.	Opt.	MiniZinc	GIHH	L-GIHH	HAHA	MS
F_1	9	**35**	34	**35**	**35**	34	34
F_1	11	**7**	4	**6**	**6**	5	4
F_1	12	**8**	**8**	7	**8**	7	7
F_1	15	**19**	-	**15**	13	10	**15**
F_1	18	**23**	**23**	21	20	22	21
F_1	19	**35**	24	**26**	23	18	25
F_1	20	**43**	-	32	26	**37**	33
F_2	9	-	**2**	3	3	3	3
F_2	10	-	**3**	4	4	5	4
F_2	11	-	**5**	6	6	13	7
F_2	12	-	**4**	5	5	5	5
F_2	13	-	**4**	5	5	6	5
F_2	15	-	-	**7**	8	-	**7**
F_2	16	-	**4**	5	6	6	6
F_2	17	-	**4**	5	5	6	5
F_2	18	-	**4**	6	7	8	6
F_2	19	-	28	**8**	11	23	**8**
F_2	20	-	-	**11**	15	25	**11**
F_3	9	-	**26522**	**26522**	26524	26524	**26522**
F_3	10	-	**8772**	8784	8778	8778	8784
F_3	11	-	21710	**20783**	**20783**	23596	21714
F_3	12	-	**4836**	5243	**4836**	**4836**	5243
F_3	15	-	-	**196806**	200887	209213	204988
F_3	16	-	**16868**	16900	16888	16890	16900
F_3	18	-	**84414**	87218	87218	87218	87200
F_3	19	-	1456141	**1253063**	1267498	1339601	1267466
F_3	20	-	3855876	**3268396**	3295030	-	3321584

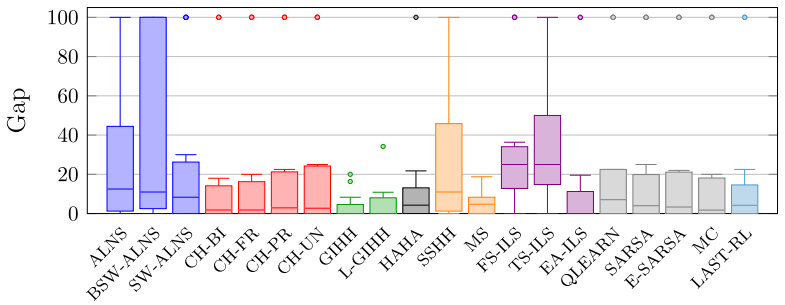


Fig. 7. Comparison of average gap for different hyper-heuristics on RWS (F_1).

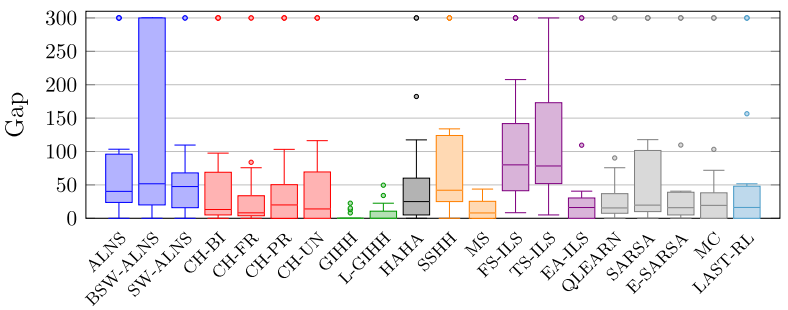


Fig. 8. Comparison of average gap for different hyper-heuristics on RWS (F_2).

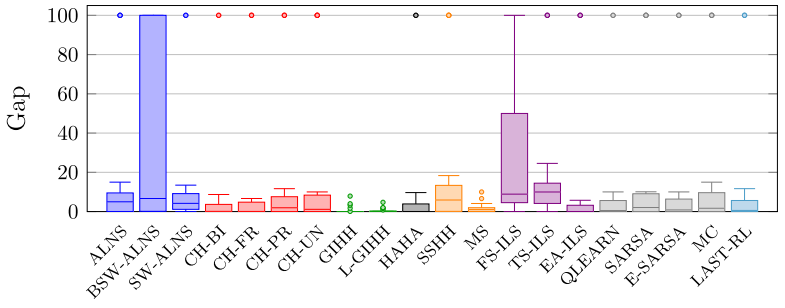


Fig. 9. Comparison of average gap for different hyper-heuristics on RWS (F_3).

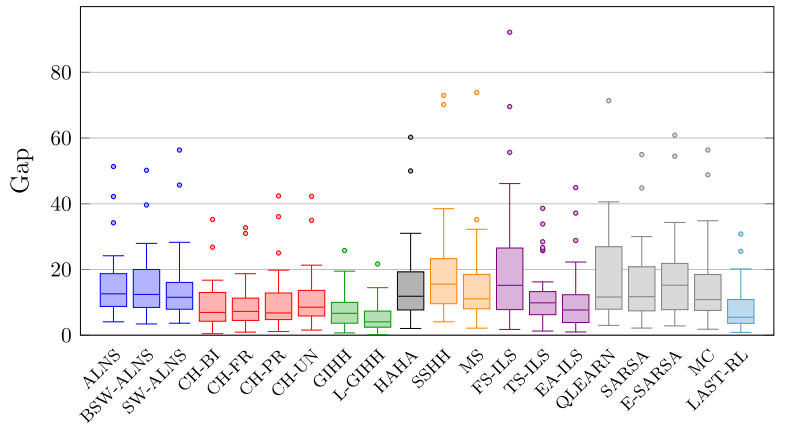


Fig. 10. Comparison of average gap for different hyper-heuristics on reduced MSD.

4.2. Minimum shift design

For this domain a large benchmark data set is available.[4] In particular we use 93 instances in four data sets, but focus on sets 3 (30 realistic instances) and 4 (3 real-life or realistic 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 fourth objective was only considered by one previous work [15], 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, Fig. 10 provides the average gap among 5 runs to the optimal results [39] 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 other domains, however, GIHH shows a larger gap to L-GIHH. Based on Q1 and median, LAST-RL actually outperforms GIHH in this comparison. Good results are also obtained by the CH variants, and by TS-ILS and EA-ILS. The ALNS variants show mediocre performance, while the remaining hyper-heuristics show significant weaknesses.

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 7%. While the exact method is clearly stronger on the reduced problem formulation, it would be much more difficult to adapt to the fourth objective. For the hyper-heuristics setup, however, this is very easy to achieve and we can provide benchmark results on all instances with the fourth objective enabled.

Fig. 11 shows the average gap among 5 runs for MSD with the fourth objective, compared to the best solution provided by any of the hyper-heuristics (since now there are no known optimal solutions). Overall, the relative performance of the individual hyper-heuristics is very similar to before. However, the clear winner is now LAST-RL, with a median gap of 0.09%, showing that it can find the best solution among the hyper-heuristics for around half of the instances, with mostly very small gaps on the others. Still, L-GIHH can secure the second place, followed by CH-BI in third place. Good performance is also shown by the other CH variants, GIHH, and EA-ILS. Mediocre performance is seen by the ALNS variants, HAHA, and TS-ILS. The remaining hyper-heuristics struggle with the problem domain.

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

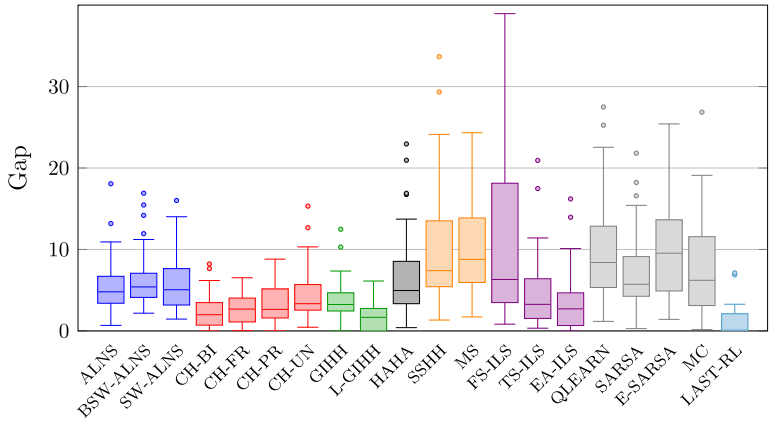


Fig. 11. Comparison of average gap for different hyper-heuristics on full MSD.

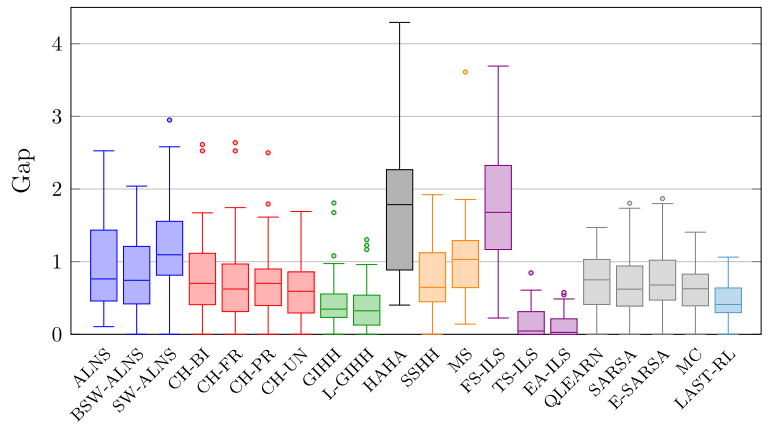


Fig. 12. Comparison of average gap for different hyper-heuristics for BDS.

4.3. Bus driver scheduling

This domain was evaluated on the set of 50 benchmark instances used by previous work.[5] They span 10 size categories from around 10 tours (70 legs) to around 100 tours (1000 legs) based on real-life demand distributions.

Fig. 12 shows the comparison of the performance between different hyper-heuristics, again based on the average gap over 5 runs for the benchmark instances. The results show that on this domain EA-ILS, tightly followed by TS-ILS, provide the best results. Good performance is also achieved by L-GIHH, GIHH, and LAST-RL. Mediocre performance is shown by the other RL-based methods, and the CH variants. The ALNS variants, as well as SSHH and MS show even higher gaps, while HAHA and FS-ILS show much higher gaps.

Table 4 shows the average results per instance category for the hyper-heuristics in comparison to the results from Branch and Price (BP) or Large Neighbourhood Search (LNS), the best average among previous heuristic methods (Simulated Annealing, Hill-climber, Tabu Search), and the recent method CSMA. The entry in bold highlights the best result among the previous heuristic methods and the hyper-heuristics, the value in italics the best overall result.

The best hyper-heuristics shown in the table can reliably outperform the best of the previous heuristics in all but the smallest category, again showing their great utility on this complex problem. Branch and Price provides near-optimal solutions for most instances except the largest ones in the data set. However, it has two major disadvantages. First, it has difficulty scaling to larger instances. This is mitigated by using Large Neighbourhood Search, which internally uses B&P for its sub-problems. Second, compared to the rule-specific implementation of the sub-problem in B&P, hyper-heuristics can be adapted to different rule-sets easily, which is often important in practice. CSMA is a very recent heuristic solution method and the only one that can outperform the hyper-heuristics slightly, despite being more problem-specific and using the same runtime on a much faster processor.

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

Table 4
Average results for BSD with different solution methods.

Instance	BP / LNS	Prev. Heuristics	CSMA	GIHH	L-GIHH	TS-ILS	EA-ILS	LAST-RL
10	*14709.2*	**14739.6**	14879.7	14847.4	14810.6	14827.4	14775.9	14837.4
20	*30294.8*	30970.8	30745.9	30892.2	30810.8	30723.1	**30654.3**	30778.1
30	*49846.4*	51257.8	50817.2	51059.4	51037.6	50901.0	**50862.7**	51129.5
40	*67000.4*	69379.9	68499.9	68988.4	69022.2	**68552.6**	68603.2	68849.8
50	*84341.0*	87262.5	86389.2	87184.4	87145.2	**86831.6**	86832.1	87083.0
60	*99727.0*	104296.2	102822.9	103491.6	103467.3	**103008.8**	103100.2	103447.6
70	*118524.2*	123225.6	121141.9	122198.6	122321.8	**121977.0**	121993.8	122372.3
80	*134513.8*	140482.4	138760.3	139648.2	139551.9	139353.8	**139353.4**	140074.6
90	*150370.8*	156296.4	155078.3	155560.8	155649.6	**155263.5**	155638.3	155633.0
100	*167081.1*	172909.0	171768.7	171879.8	**171833.5**	172210.6	172166.8	172564.6

Table 5
Number of wins for each hyper-heuristic.

	F_1		F_2		F_3		MSD		MSD+T_4		BDS	
Method	best	avg	best	avg	best	avg	best	avg	best	avg	best	avg
ALNS	10	5	7	4	7	5	0	0	0	0	2	0
BSW-ALNS	9	5	6	3	5	4	0	0	0	0	2	1
SW-ALNS	10	6	6	4	7	3	0	0	0	0	1	1
CH-BI	12	10	11	5	10	6	5	1	5	3	1	2
CH-FR	13	10	13	5	8	7	3	0	4	0	3	2
CH-PR	12	10	13	6	8	6	2	0	3	2	4	2
CH-UN	12	11	12	6	9	6	0	0	0	0	1	1
GIHH	**17**	**17**	**20**	**16**	**16**	**15**	5	0	1	0	4	3
L-GIHH	16	13	15	12	15	11	**24**	**29**	6	9	5	5
HAHA	13	9	8	5	11	10	0	0	0	0	1	0
SSHH	10	5	6	3	6	5	0	0	0	0	2	2
MS	13	9	16	8	10	6	0	0	0	0	1	0
FS-ILS	7	4	4	0	2	1	0	0	1	0	1	0
TS-ILS	6	3	4	0	7	2	0	0	1	0	15	17
EA-ILS	13	12	11	7	9	9	2	1	4	3	**22**	**21**
QLEARN	12	8	12	5	8	7	0	0	0	0	4	1
SARSA	11	9	11	5	9	6	0	0	0	0	2	1
E-SARSA	12	9	12	5	9	6	0	0	0	0	2	1
MC	13	9	13	7	8	6	0	0	0	0	6	1
LAST-RL	12	9	12	6	10	8	7	2	**17**	**16**	4	2

4.4. Comparison of hyper-heuristics

Table 5 shows the number of times a hyper-heuristic obtained the best result out of all 20 hyper-heuristics in comparison on an instance (ties are awarded to all hyper-heuristics involved in the tie), both for best and average results.

This final comparison sums up the picture from the previous sections regarding the relative performances of the hyper-heuristics: There is not one winner across domains, but different hyper-heuristics show their strengths and weaknesses on different domains. GIHH wins on RWS, L-GIHH on reduced MSD, LAST-RL on full MSD, and EA-ILS on BDS. This picture is consistent both regarding the best results among 5 runs, and regarding the average. RWS has many easy instances, resulting in many ties, while the other domains produce a clear winner on almost all instances.

However, the table does not say anything about the performance on domains where these hyper-heuristics did not win. Based on the combination of the previous results, the conclusion is that these four hyper-heuristics GIHH, L-GIHH, LAST-RL, and EA-ILS showed very strong performance on all the domains under evaluation, making them not only the winner in particular cases, but also very reliable hyper-heuristics across different domains, which is a key requirement for successful hyper-heuristics. It is especially of note that, despite being outperformed by more recent hyper-heuristics in the CHeSC competition domains, the original winner GIHH (and its modified version L-GIHH) are still excellent choices for complex real-life domains more than a decade after its initial introduction.

While MS showed excellent results on RWS, and TS-ILS on BDS, those hyper-heuristics could not generalize their performance over the different real-life domains. HAHA showed some promising results on RWS as well, but overall got very unstable results, often leading to large gaps as well. The CH variants showed some decent results and no very bad outliers, but could also not excel in any domain. The RL variants also showed reasonable overall behaviour, with most deviation on MSD. This is interesting since LAST-RL showed excellent behaviour in this domain. The remaining hyper-heuristics, which are the ALNS variants, SSHH, and FS-ILS, could not show any strengths on our real-life domains.

A partial reason for worse behaviour might be inadequate parameters. E.g., for ALNS, while several options were tried before the comparison for its parameters (5 minutes warmup, 1 minute window length), performance might increase with more careful

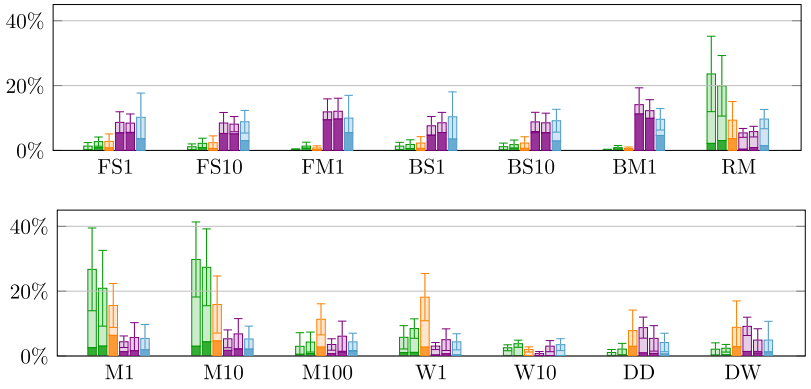


Fig. 13. Low-level heuristic frequencies for RWS (F_1).

tuning. However, no tuning was required for the winning hyper-heuristics, they either do not have parameters, or were used with their default parameters. 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). Since high-quality results have been achieved on different domains, one of the winning hyper-heuristics, L-GIHH is in the process of deployment by our industrial partner to improve their current scheduling systems, especially in the context of new and changing requirements.

5. Detailed evaluation of low-level heuristics

The previous section showed that in all three real-life domains the use of hyper-heuristics leads to very good results, even when compared to problem-specific solution methods. This section will now go into details on how the different adaptive and learning mechanisms utilize the given low-level heuristics to achieve high-quality results. We tracked the number of heuristic applications for each heuristic and each run and found some interesting patterns based on a detailed analysis: First, different hyper-heuristics show very different LLH preferences, which also differ by domain. Second, the same hyper-heuristic usually favours similar LLHs for different instances of the same domain, only some heuristics are actually swapped depending on the instance.

While the hyper-heuristics are designed to select the best combinations of low-level heuristics from the set that is presented to them, more information on the effect and usage of individual low-level heuristics will be drawn from a more detailed evaluation in this section. In order to go into more detail on the individual low-level heuristics, we will limit this part of the evaluation to the four most promising candidates among the hyper-heuristics (GIHH, L-GIHH, EA-ILS, LAST-RL), and two hyper-heuristics that showed very good performance on individual domains, which are MS and TS-ILS.

5.1. Relative frequencies and runtimes

The first part of this evaluation will focus on the usage of low-level heuristics by the different hyper-heuristics, showing both the frequencies of the application as well as the time that is spent on these low-level heuristics with various different time characteristics.

Figs. 13 and 14 show this evaluation for objective F_1 of RWS. First we will give a general description of the structure of these graphs: Each graph shows the range of low-level heuristics in two or three subgraphs, and their relative frequency or time separately for each of the six hyper-heuristics in this evaluation. From left to right, the hyper-heuristics are GIHH, L-GIHH (green), MS (orange), TS-ILS, EA-ILS (violet), and LAST-RL (cyan). Note that the percentage axis is always the same for the different subgraphs displaying frequency for the same problem, while the subgraphs for time are on different scales since the more thorough local search heuristics typically take way more runtime per execution than the other low-level heuristics.

For each individual bar, the solid part represents applications of the low-level heuristic that improved the current solution. Note that this does not directly measure any quality of the LLH, e.g., for local search heuristics improvements would clearly be expected to occur more often than for LLHs used for perturbation. Further, this value depends on when a hyper-heuristic chooses to use an LLH, e.g., a local search that is always used after a perturbation might have a higher rate of improvement than one that is always used after another local search. Still, the comparison within the same category of LLH and among different hyper-heuristics might add some value. Finally, the error bars show the standard deviation across the instances of the benchmark data set.

An immediate observation from Figs. 13 and 14, as well as Figs. 15 and 16 (F_2), and Figs. 17 and 18 (F_3) that will also hold for further comparisons is that the different hyper-heuristics have very different strategies for selecting low-level heuristics.

GIHH and L-GIHH (green) show similar behaviour (as expected), and for all three versions of RWS, they show a strong preference for the randomized local search (RM) and the mutations that are more inclined towards improving solutions (M1 and M10), with around two thirds of all applications in these categories. Among the remaining LLHs, the other mutation heuristics are the most

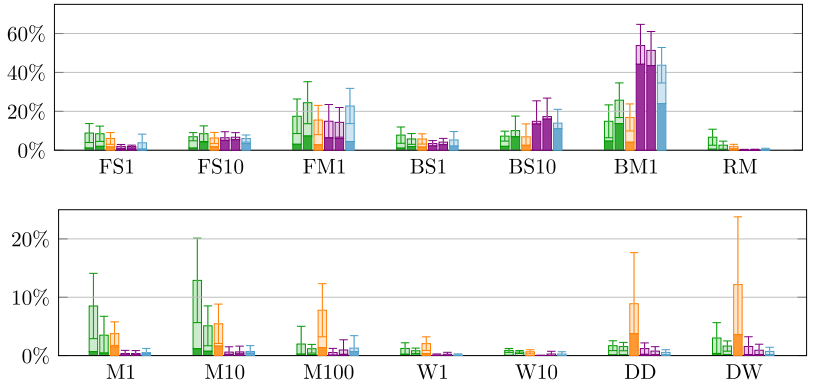


Fig. 14. Low-low heuristic execution times for RWS (F_1).

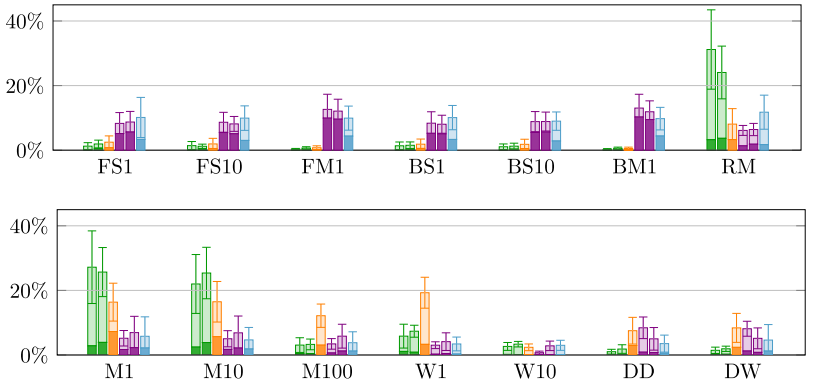


Fig. 15. Low-level heuristic frequencies for RWS (F_2).

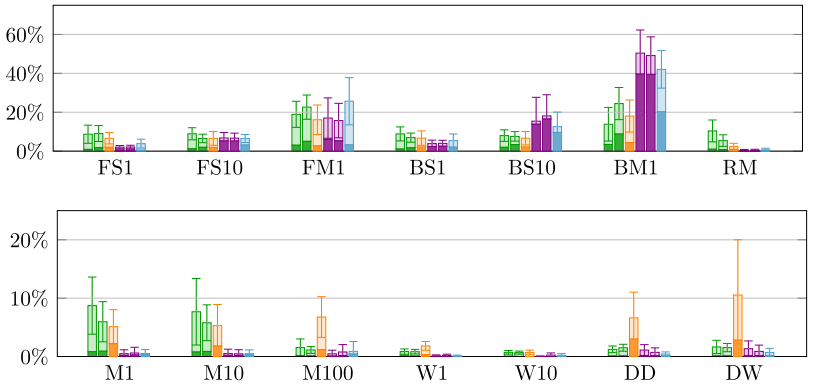


Fig. 16. Low-level heuristic execution times for RWS (F_2).

frequent, followed by destroy-and-repair, and lastly by the remaining local search heuristics. There is significant variation, but this trend, especially regarding the three most used LLHs, is very clear across the different instances.

MS (orange) also shows high usage of randomized local search and the mutations M1 and M10, but also for the more diversifying mutations M100 and W1, as well as for the destroy-and-repair operators which do not see much use by GIHH and L-GIHH.

The three remaining hyper-heuristics TS-ILS and EA-ILS (violet), as well as LAST-RL (cyan), show a similar pattern which includes much more usage of local search heuristics, and a moderate use of mutations as well as destroy-and-repair. TS-ILS uses a higher fraction of destroy-and-repair, while EA-ILS has slightly more mutation usage, and LAST-RL favours RM more than the other two. Of note is that despite the very similar pattern, TS-ILS performs much worse than EA-ILS on this domain, which will be discussed further based on other comparisons. It is also interesting that LAST-RL has a much lower success rate of local search heuristics compared to the other two (solid filling of the bars), indicating different usage patterns of the heuristics.

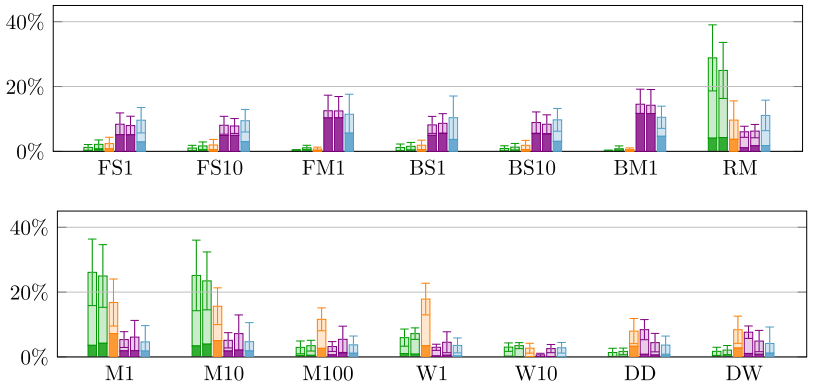


Fig. 17. Low-level heuristic frequencies for RWS (F_3).

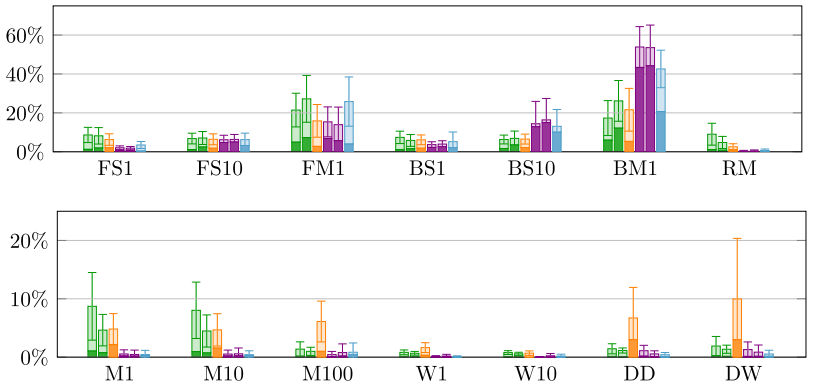


Fig. 18. Low-level heuristic execution times for RWS (F_3).

When comparing the runtimes, starting with Fig. 14, it is clear that the systematic local search heuristics take a lot of the overall runtime, especially when using the larger neighbourhoods in FM1 and BM1. Note that even for hyper-heuristics that choose not to use these LLHs frequently (GIHH and L-GIHH), they spend a major part of their runtime on the small percentage of calls to these LLHs.

For most hyper-heuristics, the mutation and destroy-and-repair LLHs do not take a large portion of their runtime. For TS-ILS, EA-ILS, and LAST-RL, this can account for more than half the runtime for BM1 alone. While still small compared to the local search heuristics, GIHH and L-GIHH spend significant time on their frequently used mutation heuristics (M1 and M10). The destroy-and-repair heuristics are most notably used by MS regarding runtime. Note that despite frequent use for many hyper-heuristics, RM only uses up very limited runtime, so despite the low success rate compared to other local search heuristics, it can easily be applied very often in the same time it takes to apply other local search heuristics with higher success rate.

In comparison, Figs. 16 and 18 show very similar patterns and no significant changes caused by the different objectives.

Based on the results, it seems that the strategy to predominately use fast randomized local search and mutations that have a higher chance for positive (or at least limited negative) change is most successful for this domain. This makes it hard for many hyper-heuristics, since a considerable focus on local search heuristics is often built into the hyper-heuristics by design. Still, the major differences between the results of TS-ILS and EA-ILS despite very similar usage frequencies show that the pattern of usage has great impact as well. GIHH and L-GIHH mainly differ based on their usage of more destructive mutations and destroy-and-repair operators, which both seem to result in good strategies for this domain.

Figs. 19 and 20 show the frequencies and runtimes for the low-level heuristics used on the reduced MSD problem. MSD is the problem domain with the largest set of low-level heuristics, 22 in total. The first two subgraphs show the wide range of 13 local search heuristics, the last subgraph shows the mutations and destroy-and-repair heuristics.

Again, the differences between different hyper-heuristics are immediately obvious. GIHH and L-GIHH (green) show a similar pattern to RWS, where they especially prefer randomized local search and the mutations inclined towards improving solutions (M1 and M10). Of note is that like for RWS GIHH avoids the longest-running LLHs even more than L-GIHH (FC1 and BC1 in this case), while RC, M1, and M10 are even more preferred by GIHH. However, the strategy applied by L-GIHH seems to be more successful based on the results. Of all the non-randomized local search heuristics, the alternating options (A and AI) are the most preferred. Regarding destroy-and-repair, a clear preference for DCE and DCF over DDE and DDF is visible.

MS (orange), now on a domain where it does not do so well, again shows a similar pattern as for RWS. It uses most local search operators slightly more than GIHH and L-GIHH, except RC, with A, AI, and RC as the favourites. Mutations are used quite often

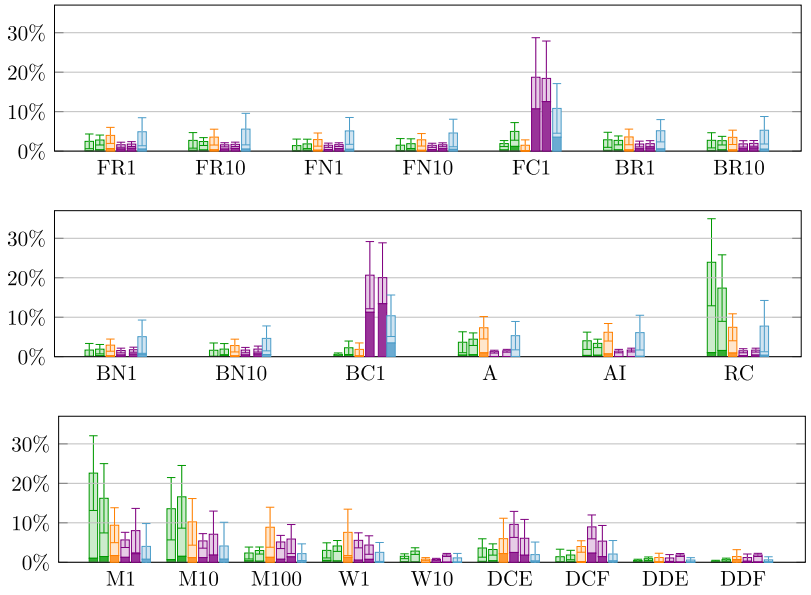


Fig. 19. Low-level heuristic frequencies for reduced MSD.

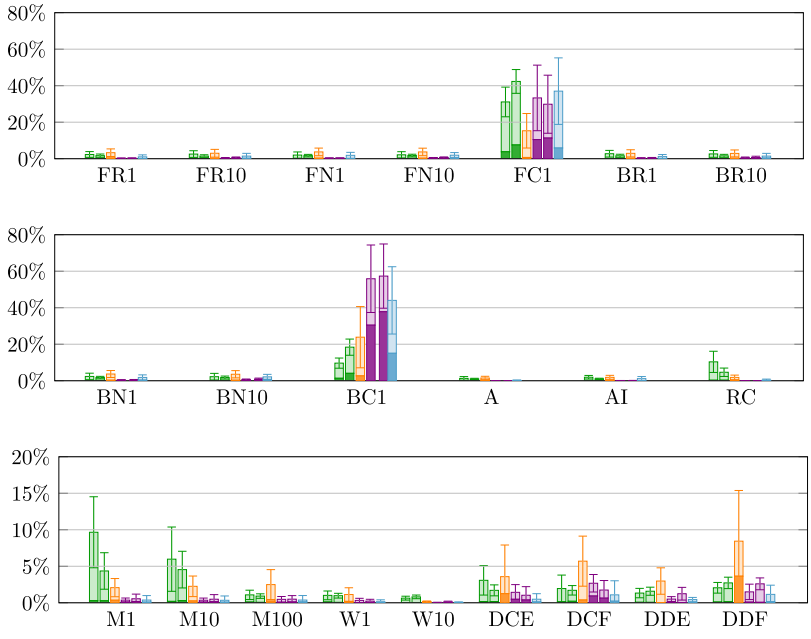


Fig. 20. Low-level heuristic execution times for reduced MSD.

again, just like for RWS with a more even spread across the different options except the most destructive one (W10). DCE and DCF are used more often than by GIHH and L-GIHH.

TS-ILS and EA-ILS (violet) again show a stronger preference for local search, however, this time with a very clear focus on FC1 and BC1, the heuristics with the largest neighbourhood, while other local search heuristics, both exhaustive and randomized, receive very little attention. While this does lead to good results, it is inferior to the strategy by GIHH and L-GIHH. Just like for RWS, there is moderate use of mutations (avoiding W10), and moderate use of DCE and DCF.

On this domain, LAST-RL (cyan) behaves differently to the ILS variants, which mostly seems to pay off regarding the results. Preference regarding local search heuristics is again given to FC1 and BC1, however, less overwhelming compared to TS-ILS and EA-ILS. Other local search heuristics are used moderately, with slightly higher usage for RC (but still far below GIHH and L-GIHH). Mutations and destroy-and-repair are not used as often as by the other hyper-heuristics, which is again consistent with RWS.

Regarding runtime in Fig. 20, the most time-consuming LLHs are immediately identified as those doing systematic search on the change neighbourhood (FC1, BC1). In comparison, the other systematic local search heuristics are relatively fast. In general it is of

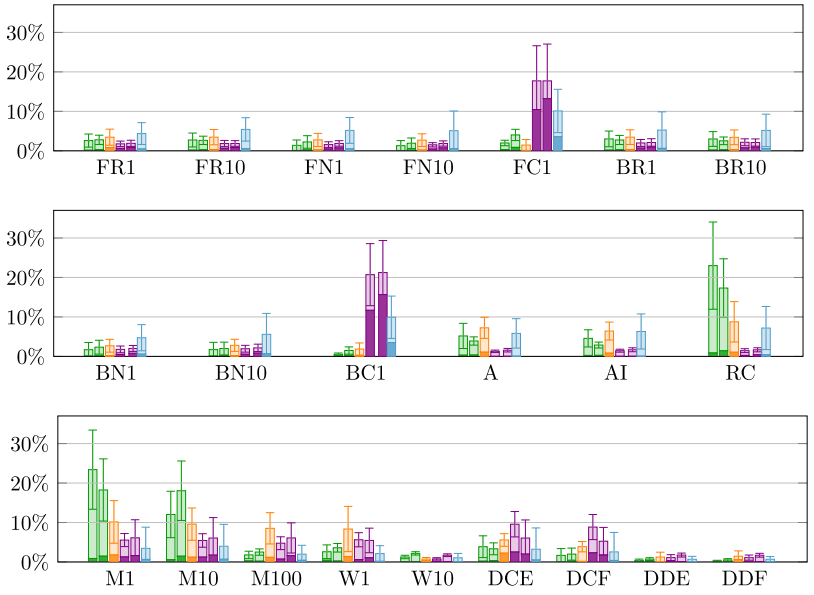


Fig. 21. Low-level heuristic frequencies for full MSD.

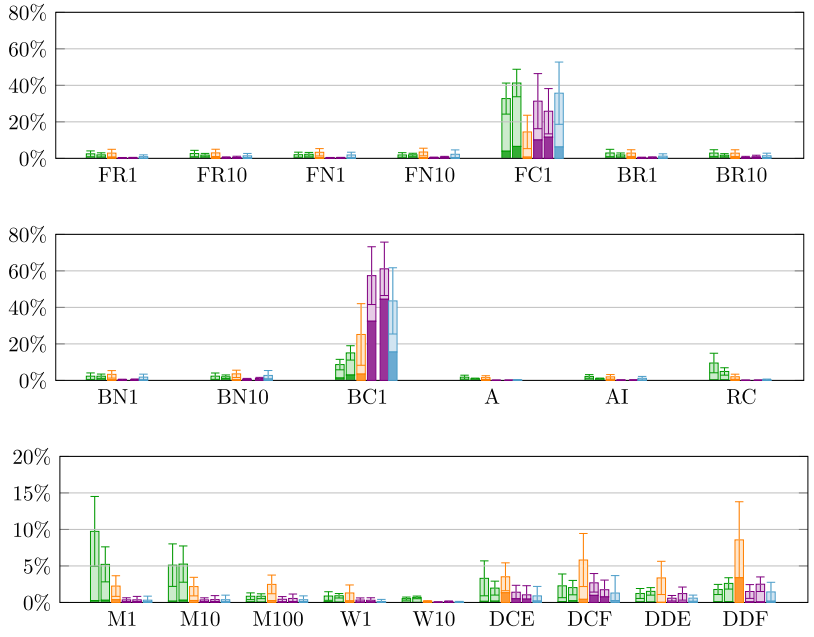


Fig. 22. Low-level heuristic execution times for full MSD.

note that for MSD, compared to RWS, the percentage of improving applications is much smaller. Indeed, only the very slow LLHs FC1 and BC1 have a significant percentage of improvements, while even all the other local search heuristics only improve rarely. However, based on the results, it seems that fast heuristics with rare improvements seem to be better overall on this domain.

Finally, regarding mutations and destroy-and-repair heuristics, runtime results are not surprising. Just like for RWS, especially GIHH, but also L-GIHH spend significant time on the mutations M1 and M10, while MS spends more time on the destroy-and-repair heuristics. These are more time-intensive compared to RWS since the systematic search used in the repair procedures is more involved. This is especially true for DCF and DDF (as the repair has more options), further DDE and DDF take more time than DCE and DCF relative to the frequencies which indicates that a larger part of the solution is destroyed. The results clearly show that the slower destroy-and-repair heuristics are used with much lower frequency, indicating that their runtime does not pay off.

Figs. 21 and 22 show the frequencies and time for full MSD. Again, the results are very similar to reduced MSD. This shows that the additional objective does not have significant influence on the strategies of the hyper-heuristics (which is similar to RWS),

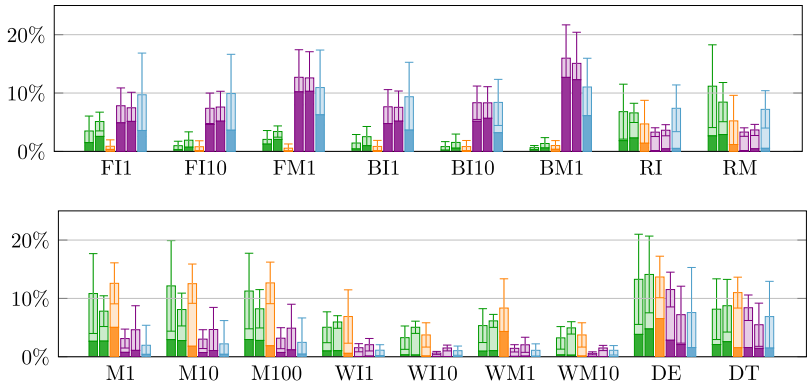


Fig. 23. Low-level heuristic frequencies for BDS.

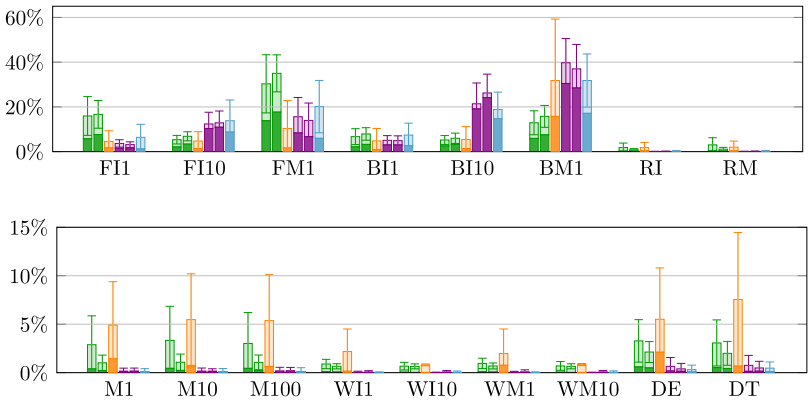


Fig. 24. Low-level heuristic execution times for BDS.

however, in this case the influence on the results is larger, as the strategy employed by LAST-RL outperforms L-GIHH on full MSD, while L-GIHH outperforms on reduced MSD.

Finally, Figs. 23 and 24 show the frequencies and runtimes for BDS. Regarding GIHH and L-GIHH (green), they show their typical behaviour to favour randomized local search and mutations inclined towards improving solutions, however, in this case not only M1 and M10, but also M100. Additionally, however, random walk heuristics are used more than in the previous domains, and especially the destroy-and-repair heuristics are used very frequently for BDS, in case of DE actually more than any other LLH.

MS (orange) again shows preference for randomized local search, most of the mutations with slightly less preference for the more destructive ones, and high usage of destroy-and-repair. M1, M10, and M100 are actually used more often by MS than by L-GIHH, with similar pattern to GIHH. However, based on the results this is not the best strategy for this domain.

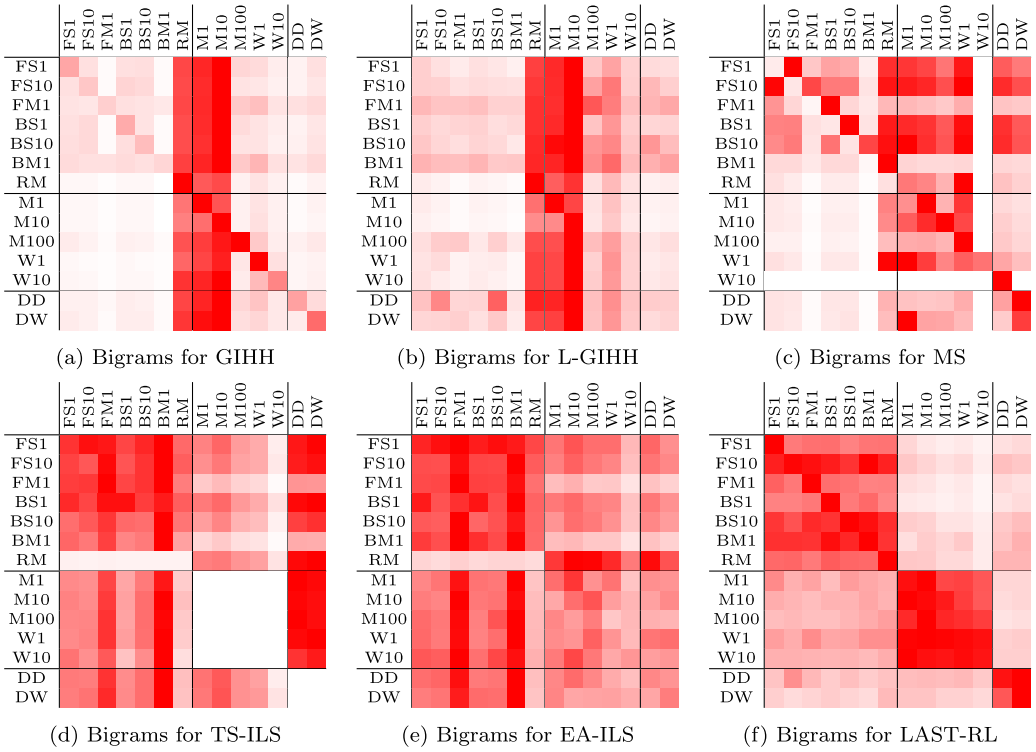
TS-ILS and EA-ILS, the most successful hyper-heuristics on this domain, show their usual preference for local search, especially with larger neighbourhood (FM1 and BM1). The pattern is similar to RWS, with high usage of most local search heuristics except for the randomized versions.

LAST-RL (cyan) also shows its typical behaviour with a more uniform preference for the different local search heuristics, with slightly less preference for the randomized versions. Mutations are again used rarely, while destroy-and-repair is used at a comparable rate to the local search heuristics.

Regarding runtime, Fig. 24 shows the expected pattern that the systematic local search heuristics take most of the runtime. Unsurprisingly, using the larger neighbourhood (FM1 and BM1) takes longer than searching the smaller neighbourhood. Of note is that MS spends more of its relative runtime on destroy-and-repair compared to the other hyper-heuristics despite similar usage frequencies. However, it avoids slower LLHs more rigorously, allowing a larger number of LLH applications overall. Also note that MS has especially large error bars, showing that its behaviour fluctuates a lot.

An interesting observation compared to the other domains is a higher success rate of the operators, e.g., local search very often leads to improvements, and even mutations and destroy-and-repair frequently improve the solution. This high success rate might help to explain the good performance of the ILS variants, and also the low performance of MS, since there seems to be a correlation between performance and frequency of local search application.

Comparing results across all domains, it is very clear that the different hyper-heuristics have very different strategies which they then consistently apply on the different domains. Depending on the domain, this leads to different performance depending on how



(a) Bigrams for GIHH (b) Bigrams for L-GIHH (c) Bigrams for MS

(d) Bigrams for TS-ILS (e) Bigrams for EA-ILS (f) Bigrams for LAST-RL

Fig. 25. Bigrams for RWS (F_1).

well the strategy fits the domain. However, it is also clear that the strategies adapt to the different domains, as it is evident from the different frequencies of LLHs in the same category as a result of these adaptive mechanisms.

Regarding the comparison of GIHH and L-GIHH across all domains, both heuristics always favour randomized local search and mutations with low level of destruction. The results show that they act in a very similar way, which is expected since L-GIHH is closely derived from GIHH. The details regarding their difference in LLH selection can constantly be seen in all domains as an increased preference of GIHH for the randomized local search and improving mutation heuristics (the only notable difference is M10 for full MSD where L-GIHH uses M10 significantly more often), while L-GIHH allocates slightly more calls for the other LLHs.

MS shows more focus on destroy-and-repair, as well as moderately destructive mutations, while TS-ILS, EA-ILS, and LAST-RL show much more focus on local search. TS-ILS and EA-ILS especially concentrate on more sophisticated, but slower local search, while LAST-RL uses local search more evenly. TS-ILS has higher preference for destroy-and-repair, while EA-ILS has higher preference for mutations.

Most notable differences were in the selection of local search heuristics, with a more even spread for RWS and BDS, and a very narrow focus on few LLHs for MSD. Randomized local search was notably less used for BDS, while the frequency and pattern for destroy-and-repair also changed significantly, which especially high usage for BDS.

5.2. Combinations of low-level heuristics

While the frequencies of individual low-level heuristics can already give several conclusions about the way the different hyper-heuristics work, the LLHs are not applied in isolation, but in sequence. Therefore, the next step is to analyze how LLHs are applied relative to each other.

Fig. 25 shows a corresponding graph for each of the hyper-heuristics representing the frequencies of bigrams, which are pairs of consecutive low-level heuristics. All frequencies are averaged across all runs on all instances of a domain. Each pixel in a subgraph represents the frequency of a bigram (ℓ_1, ℓ_2), where ℓ_1 is given by the row as the first LLH in the bigram, and ℓ_2 is given by the column and represents the second LLH in the bigram. Since LLH frequencies vary widely, each row is scaled according to its maximum value, e.g., in the first row (FS1) of Fig. 25a, the bigram with the largest frequency (FS1, M10) is scaled to 100%, and all other entries in this row are scaled relative to this frequency.

It is immediately clear that just as in the previous evaluation, each hyper-heuristic shows a very different pattern. GIHH (Fig. 25a) shows an unsurprising preference for RS, M1, and M10, since these LLHs were the most frequently used in the previous part of the investigation. What this part shows is that the preferred LLHs are frequently chosen after all other LLHs, not just in particular combinations. Of further interest is that fact that the main diagonal is very strong, indicating frequent repetitions of the same LLH. Local search heuristics except RM mainly appear only after others of the same group.

Fig. 25b shows that L-GIHH shares the preference to use RS, M1, and M10 after all other LLHs, but it barely shows any preference for the main diagonal. A significant preference for local search heuristics after other local search heuristics can be seen again, but also to follow on local search with other mutation and destroy-and-repair operators. Of note is a strong preference of FS10 and BS10 after DD.

MS (Fig. 25c) shows a very interesting, and very different pattern. Instead of the main diagonal, it favours a shifted diagonal, indicating that it likes to iterate through the LLHs, moving on to the next in the given order. Local search except BM1 and RM is often followed by many different LLHs. Mutations often lead to other mutations and RM, while destroy-and-repair is mostly followed by more destroy-and-repair (especially DW), but DW also often leads to M1. FM1 and BM1 are rarely chosen at all, and W10 only follows W1, and is only followed by DD.

TS-ILS (Fig. 25d) again shows a very unique pattern, never using multiple mutations or destroy-and-repair operators in a row. Otherwise, there is a more uniform distribution with some exceptions. W10 is rarely chosen, while the favourites FM1 and BM1 are often chosen after every other LLH except RS. These two are also mostly followed by more local search, while all other local search and mutation heuristics frequently lead to destroy-and-repair heuristics.

EA-ILS (Fig. 25e) does not share the limitation of mutations and destroy-and-repair operators, but otherwise shows a similar distribution. Again, FM1 and BM1 are the most frequent LLHs after all other LLHs except RS. RS is notable since it rarely follows after mutation or destroy-and-repair, and rarely leads to more local search. Again, lower use of W10 is visible. Some scattered preferences among mutations and destroy-and-repair are visible without a clear pattern.

LAST-RL (Fig. 25f) shows a clear preference to stay within the same category of LLH, which is true for all three categories. Especially for local search, the main diagonal is clearly visible. It is of note that some local search heuristics show a preference to be repeated (FS1, FM1, BS1, RS), while the others rather lead to a more uniform distribution of succeeding local search heuristics. A few other pairwise preferences are faintly visible, but without a clear pattern.

Note that results are similar for the other RWS objectives, therefore only F_1 is shown.

Fig. 26 shows the patterns for full MSD. The picture for GIHH is very similar, with a strong preference for the main diagonal, and for RC, M1, and M10 after all other LLHs. L-GIHH again shares the selection of RC, M1, and M10 (M10 more uniformly than GIHH). This time the main diagonal is more visible for L-GIHH as well, but still much less than for GIHH. Faint preferences are also seen for FC1, A, W1, and DCE compared to the remaining LLHs, the remaining pattern is very uniform.

MS again shows the shifted diagonal, indicating that LLHs are often applied in their given order. In contrast to RWS, this diagonal dominates the picture with few other frequent combinations. There is a sub-block among local search heuristics A, AI, RC, and the mutations except W10, and there are strong preferences for FR1 after DTN, and M1 after DPN. The preference for the first mutation after the last destroy-and-repair operator is shared with RWS. Based on the performance, it might be that the large number of LLHs combined with the preference to iterate through them in order lead to worse performance for this domain compared to RWS.

TS-ILS now shows a very different picture compared to RWS. The mutations and destroy-and-repair operators are now mostly only followed by the favourite local search heuristics FC1 and BC1, which are also repeated frequently. All other local search heuristics lead to the mutations except W10, and to DTE and DTN. These two destroy-and-repair operators also frequently follow after mutations. EA-ILS shows a very similar picture, mainly only reducing the preference for DTE and DTN after mutations.

LAST-RL now shows a very diverse picture, while still overall preferring to chose heuristics from the same category again, it also learns the preference for FC1 and BC1, in this case mostly after all kinds of local search, M1, M10, DTE, and DTN. The rarely used more destructive mutations, DPE, and DPN are more often followed by other LLHs of their corresponding subgroup. Of note is also a preference for RC after most local search heuristics.

Fig. 27 finally shows the bigrams for BDS. This domain shows patterns in-between the other domains. While there is not such a strong focus on a few individual LLHs as in MSD, there is a more diversified picture than for RWS. The main patterns for each hyper-heuristic are again repeated. Of note is a much stronger preference for destroy-and-repair, especially for GIHH, L-GIHH, and TS-ILS. GIHH and L-GIHH show a more diverse selection of LLHs, and especially GIHH shows a more diverse picture for different rows besides the main diagonal. Again, TS-ILS, EA-ILS, and to a lesser degree LAST-RL share a preference for FM1 and BM1. LAST-RL also shows a preference for MI1 and BI1 after the less destructive mutations, and especially low selection of mutations after local search.

Overall, a core finding repeats from the previous evaluation: Each hyper-heuristic shows a characteristic pattern that is visible for each of the individual domains, but the hyper-heuristics also adapt their pattern to each of the domains. Several hyper-heuristics like to repeat LLHs, in particular GIHH and LAST-RL, while MS likes to iterate through the set of LLHs. GIHH and L-GIHH again showed their focus for fast randomized changes, while TS-ILS and EA-ILS show their preference for a focus on a few local search heuristics, interleaved with mutation or destroy-and-repair. Less uniform pattern were shown by MS for RWS, GIHH for BDS, and LAST-RL for MSD and BDS. This might hint at more complex relations being learnt for individual pairs of low-level heuristics and fits with the learning approach that is used in LAST-RL.

5.3. Importance of low-level heuristics for solution quality

The previous part of the evaluation investigated in detail how often and how long different low-level heuristics are executed by different hyper-heuristics. However, keep in mind that this does not necessarily mean that an LLH that is used rarely does not have a significant impact on solution quality, e.g., a perturbation might only be used rarely, but still be very important and effective to escape local optima. In this part, different subsets of LLHs will be excluded from usage to investigate the effect on the solution quality.

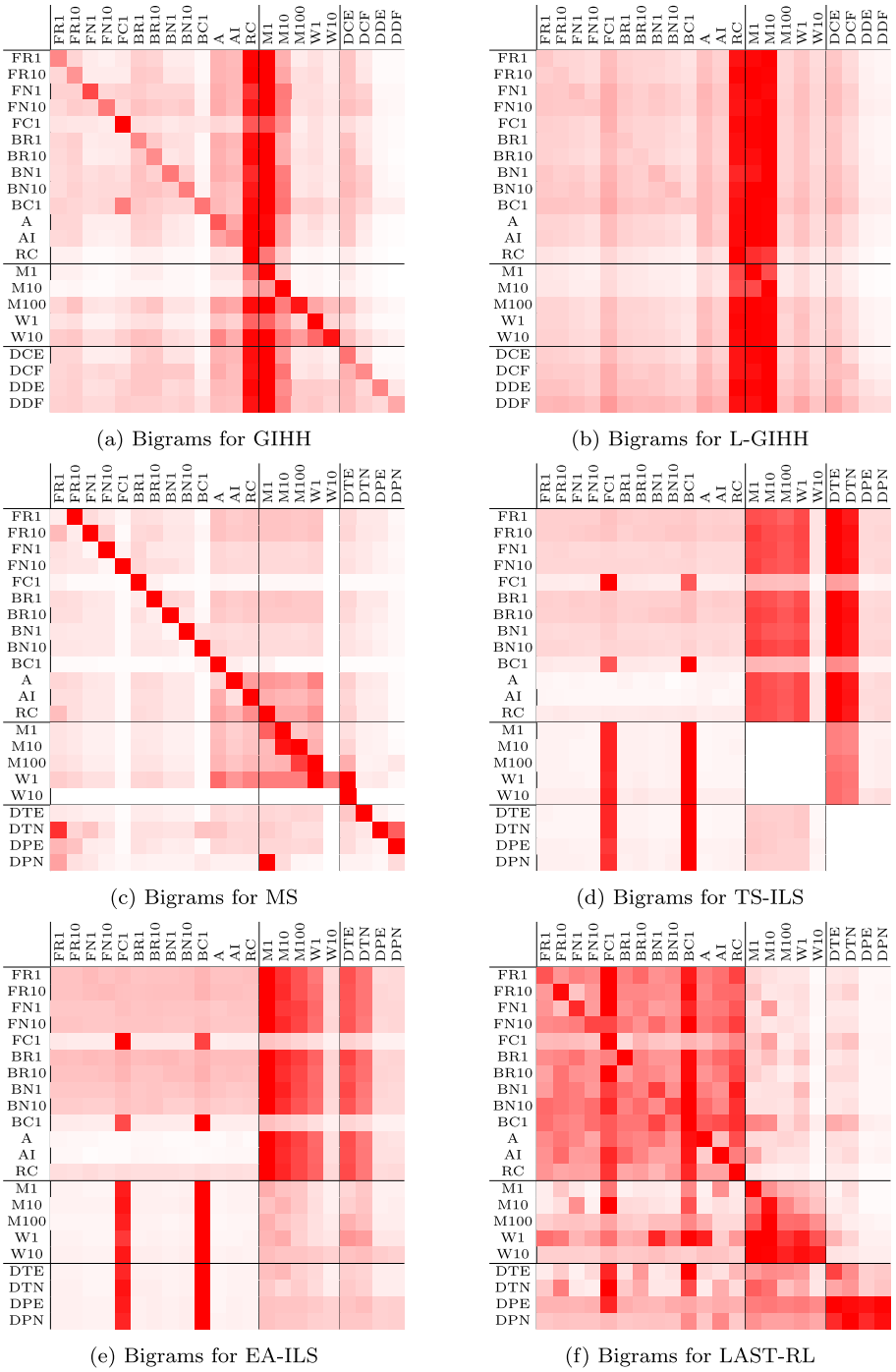
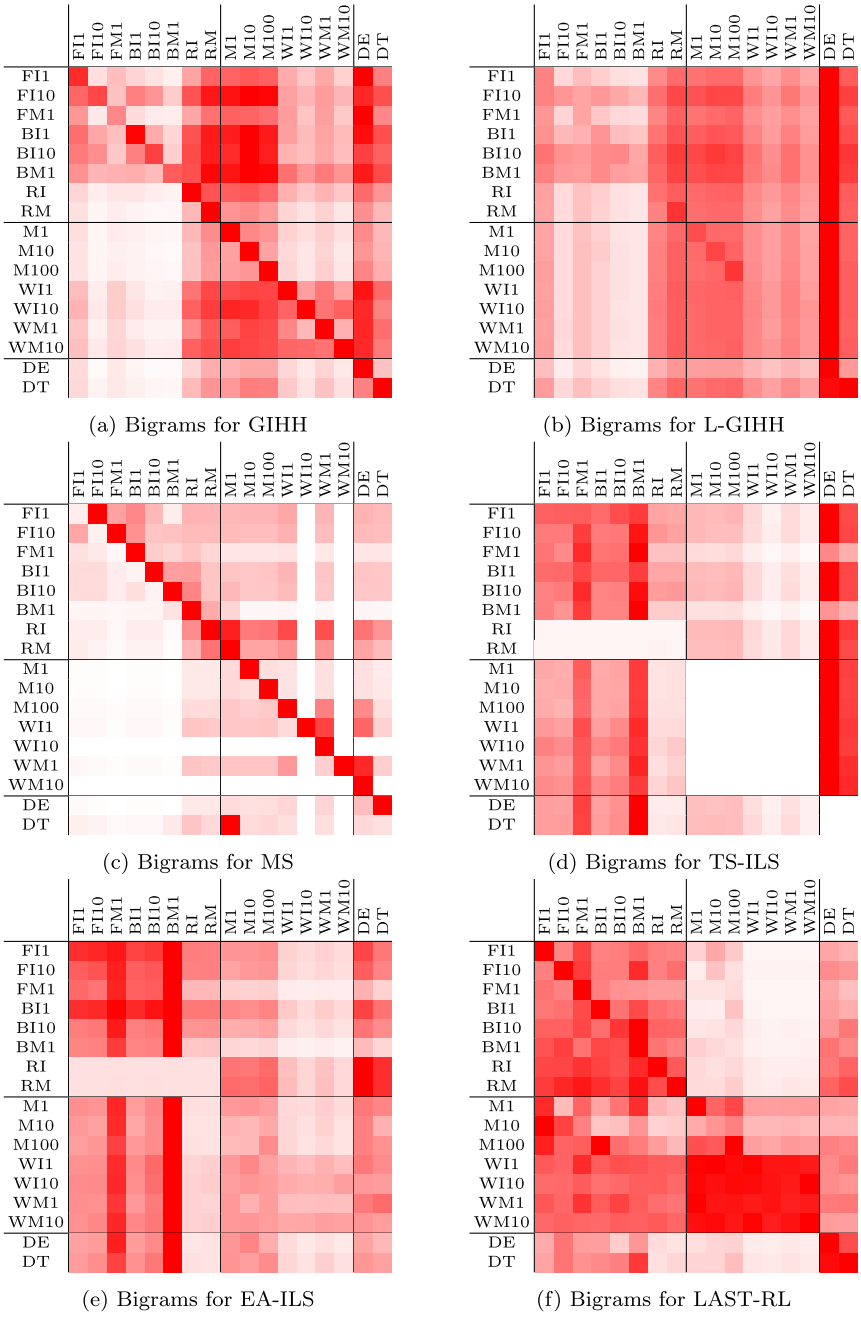


Fig. 26. Bigrams for full MSD.

The expected result is to see a decrease in solution quality (higher objective) when removing important LLHs, and a small improvement in quality (lower objective) when removing LLHs that are less useful for solving the problem. Good hyper-heuristics should be able to exclude these LLHs on their own, but they will still spend some time evaluating the performance of the LLHs that might be spent better on other LLHs.

Table 6 shows the first set of experiments for RWS using objective F_1. The structure is the same for this and the following tables. Each column represents a hyper-heuristic, each row a change in the set of low-level heuristics. The first three rows are the same for all sets of experiments and represent the drastic step of removing all LLHs of a particular type (No LS is without local search, No M is without mutations, and No DR is without destroy-and-repair heuristics). The following rows each restrict the use of one of the three



(a) Bigrams for GIHH

(b) Bigrams for L-GIHH

(c) Bigrams for MS

(d) Bigrams for TS-ILS

(e) Bigrams for EA-ILS

(f) Bigrams for LAST-RL

Fig. 27. Bigrams for BDS.

main types to a subset of the LLHs, in case of RWS these are local search only by first improvement (Only F), local search only by best improvement (Only B), local search only by random improvement (Only R), mutation only using LLHs with Metropolis acceptance criterion (Only M), mutation only using random walk (Only W), destroy-and-repair only by using the day-based removal (Only DD), and destroy-and-repair only using the week-based removal (Only DW).

The values in the tables show the average change of the objective in percent, where a negative value is an improvement and a positive value represents a worse objective. Note that the magnitude of the change varies a lot between the tables; this is mainly due to the different scale of objective values for different domains and objectives. E.g., for F_1 in RWS, objective values are very small, therefore each change represents a large percentage. Also, for RWS feasibility plays an important role, and infeasible solutions have an objective value that can be orders of magnitude higher than feasible solutions. Note that $F_1 = 0$ is possible for some instances. In case the original result was 0, any change is evaluated as +100% for this set of experiments. Overall, we are interested in comparing trends between heuristics for the same objective rather than the magnitude of the change between different objectives.

Table 6
Objective changes (%) when removing different subsets of LLHs (RWS F_1).

Change	GIHH	L-GIHH	MS	TS-ILS	EA-ILS	LAST-RL
No LS	−35	−29	–	–	–	3681
No M	41583	12507	15373	–	64	60
No DR	20	29	32024	–	53	97
Only F	−10	−16	155	13	45	49
Only B	−21	276	17944	17	69	89
Only R	−32	−36	19849	2788	20	2463
Only M	16	266	1080	−21	57	53
Only W	11493	4063	5519	48	75	76
Only DD	35	23	6336	5615	70	1245
Only DW	122	169	29164	−17	65	48

Table 7
Objective changes (%) when removing different subsets of LLHs (RWS F_2).

Change	GIHH	L-GIHH	MS	TS-ILS	EA-ILS	LAST-RL
No LS	−11.5	−12.6	–	–	–	47.8
No M	4.8	6.0	1336.5	–	44.1	−7.6
No DR	−7.7	19.8	−16.9	–	2.5	32.9
Only F	−4.4	−1.2	224.2	−21.8	−12.3	−7.7
Only B	−8.4	0.4	823.6	32.8	12.6	59.2
Only R	−11.6	−14.1	1717.8	−47.8	−22.2	5.5
Only M	1.2	1.3	131.1	−9.0	1.6	−11.7
Only W	−5.1	−1.0	1071.1	55.9	17.9	27.4
Only DD	3.9	1.9	299.9	10.3	8.9	5.0
Only DW	−3.2	8.1	247.6	−19.3	−0.3	−8.5

The first aspect to note is that some hyper-heuristics rely on certain categories of LLHs and do not work if no LLH in that category is available. This is the case for many hyper-heuristics regarding local search, as most of them use this category as a fundamental part of their solution process, in particular for MS, TS-ILS, and EA-ILS. TS-ILS actually assumes to have access to mutations and destroy-and-repair as well, providing no result for the first three experiments.

Table 6 shows different trends for the different hyper-heuristics depending on their strategies. GIHH and L-GIHH are very much focused on mutations, as they actually show a lower objective without local search, a small decrease in quality without destroy-and-repair, but a huge increase in objective (showing a large increase in infeasibilities) when removing mutations. Within local search, the most positive effect is achieved using only randomized local search, within mutations it is clear that the mutations using the Metropolis acceptance are the most important, but using only M also leads to slightly worse solutions. Regarding destroy-and-repair, only DW performs worse than only DD.

MS actually shows greatly worse solutions whenever subsets of LLHs are removed, possibly related to the preference to iterate through the different LLHs. For TS-ILS, EA-ILS, and LAST-RL, almost all trends for removal of subsets are also towards an increase in objective, even though not as bad as for MS. TS-ILS and LAST-RL show very similar trends, with problems when removing local search. The high value for only R indicates that those hyper-heuristics especially rely on systematic local search to get better results. They also both need DW for good results. It seems that DD works better in combination with the mutation strategy for GIHH and L-GIHH, while DW works better with the local search strategy for TS-ILS and LAST-RL. Regarding mutations, results are slightly better using only M than only W for the ILS variants as well as LAST-RL. EA-ILS, while not performing best on this domain, is actually the most stable regarding elimination of subsets, with only minor increases.

Table 7 shows the results for RWS with objective F_2. The overall magnitude of change for F_2 and F_3 is much lower since the objective values are not as close to 0. Further, results rarely get more infeasible in this set of experiments. Some trends persist from F_1, however, some effects are different.

The picture for GIHH and L-GIHH is similar, again with a positive effect of removing local search, or at least all local search except randomized. Interestingly there is now a positive effect of using only W, and only DW for GIHH, potentially due to the different objective. MS keeps the trend of getting much worse, often infeasible, with the removal of most subsets. No DR shows a positive change, just like GIHH, but the trend here is not clear.

For the ILS variants and LAST-RL, local search now shows better results with only F or only R, but clearly worse results with only B, indicating that the extra runtime for best first search is not justified in combination with this objective. Regarding mutations, they now show a trend towards only M, which is in contrast to GIHH and L-GIHH. DW is preferred over DD like before. EA-ILS shows much worse results without any mutations, while LAST-RL shows much worse results without local search or destroy-and-repair, but does not seem to rely on mutations. EA-ILS is not as stable any more as it seemed for F_1.

Table 8 shows the results for RWS with objective F_3. Results are again similar, but with some slight changes. For GIHH and L-GIHH, only removing all local search, or all except randomized local search has a positive impact, while especially removing all mutations has very negative impact. MS shows the usual huge increase in objectives. The results of TS-ILS are not consistent (but TS-ILS performed badly on RWS, so the results might contain a lot of random fluctuations based on this bad performance). Interestingly,

Table 8
Objective changes (%) when removing different subsets of LLHs (RWS F_3).

Change	GIHH	L-GIHH	MS	TS-ILS	EA-ILS	LAST-RL
No LS	−8.0	−8.1	–	–	–	24.8
No M	1735.7	586.8	15329.9	–	311.9	9.9
No DR	14.3	0.2	14572.3	–	214.6	2.4
Only F	7.1	21.4	108.7	1276.4	102.1	−18.5
Only B	14.1	14.6	681.0	−7.1	333.2	21.1
Only R	−8.1	−8.0	1663.6	−44.8	314.9	−8.7
Only M	21.7	6.9	1800.4	−8.0	−5.2	−16.3
Only W	14.4	8.1	18040.1	−0.9	242.4	28.8
Only DD	294.6	14.2	3217.8	441.7	103.7	26.4
Only DW	21.7	14.4	11249.0	−18.1	215.4	−14.9

Table 9
Objective changes (%) when removing different subsets of LLHs (reduced MSD).

Change	GIHH	L-GIHH	MS	EA-ILS	TS-ILS	LAST-RL
No LS	−1.30	−1.24	–	–	–	5.69
No M	2.67	5.55	9.03	–	17.44	4.16
No DR	−0.27	0.43	−3.98	–	0.05	−0.70
Only F	−0.88	−0.09	−0.78	3.29	2.57	1.50
Only B	−1.14	−1.23	−0.80	2.33	1.13	−0.59
Only A	−1.39	−1.31	−2.90	−2.10	0.10	2.45
Only R	−1.04	−1.44	−3.69	−3.47	−1.63	2.09
Only M	0.81	0.60	−0.78	−0.90	−0.36	0.14
Only W	0.77	1.88	5.40	1.17	3.60	1.95
Only DT	−0.73	−0.14	−2.27	0.17	−0.52	−0.36
Only DP	−0.02	−0.32	−0.37	−0.44	0.23	−0.12

Table 10
Objective changes (%) when removing different subsets of LLHs (full MSD).

Change	GIHH	L-GIHH	MS	EA-ILS	TS-ILS	LAST-RL
No LS	−1.34	−1.94	–	–	–	4.40
No M	5.44	7.58	13.54	–	16.13	3.96
No DR	−0.36	−0.37	−3.35	–	−0.44	−0.68
Only F	−0.96	−0.53	−1.03	1.92	1.48	1.19
Only B	−1.54	−1.36	1.01	0.47	0.70	−1.42
Only A	−1.37	−1.48	−3.05	−2.05	1.27	1.42
Only R	−1.08	−1.71	−2.70	−2.90	−0.13	2.55
Only M	0.40	0.78	0.20	−0.52	0.07	−0.29
Only W	1.59	3.47	8.70	0.72	2.61	−0.16
Only DT	−0.14	−0.31	−2.22	−0.34	−0.43	0.01
Only DP	−0.69	−0.15	1.13	−0.56	0.23	0.12

EA-ILS is consistent in very uniform changes independent of which subset is removed, but this time changes are actually much worse than for most other hyper-heuristics in comparison. Only M is the only exception. LAST-RL again shows that it needs local search, that only B is the worst choice regarding local search, and that only M and only DW are the preferred subsets in their category.

While these results are very aggregated (each entry in a table represents multiple runs on all benchmark instances for RWS), they show some very interesting trends that can be confirmed in combination with all objectives and complete the picture regarding LLH usage from the previous sections. Overall, for this domain it seems that the time-consuming best improvement heuristics take too long compared to their benefit, and keeping only the randomized local search would actually benefit the search for the best-performing hyper-heuristics GIHH and L-GIHH. Even though the winning hyper-heuristics GIHH and L-GIHH did not use first and best improvement frequently, previous evaluation (e.g., Fig. 14) shows that a significant portion of time is spent collecting some data on these LLHs. For the ILS variants and LAST-RL, removing the best improvement heuristics might also be beneficial.

In contrast, mutations and destroy-and-repair heuristics have a significant positive impact on solution quality, since even the removal of subsets of these shows a trend toward worse solutions in most cases. This is also interesting to see in comparison with the previous experiments (e.g., Fig. 13), which shows that GIHH and L-GIHH use randomized local search and the mutations based on the Metropolis acceptance criterion frequently, but random walk and destroy-and-repair less frequently. The current experiments show that even though these are not used frequently, they seem to play an important role in diversification of the solution and escaping local optima, and are therefore critical for solution quality.

Table 9 shows the changes for the reduced version of MSD, Table 10 for the full version of MSD. Note that the magnitude of changes is in general smaller than for RWS. Both tables show very similar values, therefore, they will be discussed together.

Table 11
Objective changes (%) when removing different subsets of LLHs (BDS).

Change	GIHH	L-GIHH	MS	EA-ILS	TS-ILS	LAST-RL
No LS	0.47	0.78	–	–	–	0.48
No M	0.22	0.21	0.76	–	−0.13	−0.06
No DR	1.28	1.43	1.28	–	1.87	1.50
Only F	−0.03	−0.07	−0.12	0.57	1.15	0.58
Only B	−0.22	−0.18	1.97	0.12	0.27	0.08
Only R	0.45	0.70	0.12	0.35	0.16	0.21
Only M	0.11	0.08	−0.01	−0.07	−0.15	−0.10
Only W	0.04	0.05	0.35	−0.06	0.09	0.06
Only DE	0.13	0.15	0.06	0.09	0.25	0.20
Only DT	0.15	0.27	1.73	0.09	0.16	0.08

GIHH and L-GIHH again show improvements when removing local search, and also consistently for all subsets of local search, with the smallest using Only F, and not much difference between the others or no local search at all. Removal of mutations again makes results significantly worse, while the removal of destroy-and-repair improves results except for No DR and L-GIHH on reduced MSD. Regarding mutations, all subset removals make the results worse, but in most cases Only W is worse than Only M.

MS also shows benefits for removal of most subsets of local search, especially when the remaining subset is Only A or Only R, it shows even stronger deterioration when removing mutations, especially those using the Metropolis acceptance, and it shows strong improvement removing destroy-and-repair, but with worse results using Only DP for full MSD. The stronger effect of changes to destroy-and-repair heuristics might be caused by the higher prevalence to use these.

TS-ILS gets worse using Only F or Only B, but confirms the benefits of Only A and Only R. It also confirms the need for mutations benefiting from Only M, but getting worse using Only W. Only DP is beneficial, Only DT depends on the version of MSD. For EA-ILS the results are very similar, showing the same trends for local search except worsening using Only A, the same trend regarding mutations, but here Only DT is more beneficial than Only DP. The results regarding local search are surprising, since the most used LLHs are first and best improvement heuristics. It seems that their increased runtime is actually diminishing the value of their improvements for the ILS variants.

LAST-RL, the winning method, clearly needs local search and mutations, but could do without destroy-and-repair. Interestingly, Only B is the best option regarding local search, so this method can actually use the slower method in a beneficial way. While mutations are crucial, any of the two subsets would do. Using only a subset of destroy-and-repair does not change much compared to using both.

Overall, regarding mutations, the results are again very clear in showing that these are very important for solution quality, as the overall removal consistently leads to worse solutions, and for many hyper-heuristics both subsets seem to be important. For destroy-and-repair there is a smaller trend towards better solutions when removing subsets or all heuristics. Regarding local search, the results are different depending on the preferences of the hyper-heuristics. GIHH and L-GIHH would benefit from greatly reducing local search heuristics, and most hyper-heuristics show very good results using Only R, many of them also using Only A. However, the winning method LAST-RL wins with a different strategy, where Only B would actually be the best choice. This shows that the balance between using slow, but more comprehensive local search versus fast, randomized local search can be very difficult to achieve for this domain, and different local search heuristics can be the best choice for different hyper-heuristics. However, just like for RWS, reducing the set of slow local search heuristics would be recommended, e.g., in this case by removing first improvement heuristics.

Table 11 shows the results for BDS. Here, the magnitude of change is the smallest since objective values are large and a large portion of them cannot be changed by the solution (all legs need to be covered in any solution). This time, the pattern from the previous domains showing mutations as the most important heuristics is not repeated.

Removing local search now shows clear negative impact for all hyper-heuristics, and in contrast to especially RWS, Only R consistently makes solutions worse, while Only F and Only B provide small improvements or degradations depending on the hyper-heuristic. This shows that for BDS, a systematic search is more beneficial than a randomized search, even though previous experiments again showed that it takes significantly longer runtime per execution. However, for BDS it seems that this additional time is well spent regarding solution quality.

Mutations, on the other hand, are much less important. The removal of mutations or their subsets shows diverging results, with slight worsening of the solutions for GIHH, L-GIHH, and MS, and slight improvements for some experiments regarding the other hyper-heuristics. However, most effects are not strong.

Destroy-and-repair, on the other hand, clearly seems to be the most important category for this domain. Removing all of these heuristics has a clear negative impact on all hyper-heuristics. While keeping only one of the destroy-and-repair heuristics reduces the loss of quality a lot, still all experiments show worse solutions. Therefore, both destroy-and-repair operators seem to contribute in a similar way.

Overall, already the previous experiments (Fig. 23 and Fig. 24) showed that the distribution of frequencies and runtimes across the low-level heuristics was more even than for the other domains. These experiments now confirm that almost all LLHs provide a positive impact on the solution quality and should be kept, especially the more problem-specific destroy-and-repair heuristics. The only LLH that does not seem to contribute well is randomized local search, which is actually used quite often by the hyper-heuristics, but also runs very fast.

These results are in contrast to the results on the other domains and show very well that for different domains different sets of low-level heuristics are important. Overall, there was no category that never had a positive impact, but also no category that was always very important: Systematic local search worked well for BDS, while randomized local search provided good results for RWS. Mutations were very important for RWS and MSD, but not very important for BDS. Destroy-and-repair was important for RWS, very important for BDS, but did not have much effect for MSD.

This shows that there is no global solution to select the best subset of low-level heuristics for a new domain. However, the results also show that good hyper-heuristics can provide very good results when presented with a large set of LLHs with mixed effect on solution quality. While it takes them some time to evaluate solution quality and exclude LLHs which do not provide a good enough impact for their runtime, they are still very effective overall, as shown by the good results across different domains also in comparison with problem-specific methods.

However, this evaluation also shows that results can be improved if a domain is frequently solved. It is not efficient to continuously redo the evaluation of the individual LLHs when it is known that certain LLHs do not have a positive effect across most instances of a domain. Hyper-heuristics can still adapt the use of LLHs for individual instances, but restricting the set of LLHs to those that are more likely to provide good results can streamline the solution process and allow better results in the given runtime.

Overall, the recommendation is therefore to use a rich set of various low-level heuristics for a new domain to get very good results very fast, and then evaluate the set of low-level heuristics to exclude those that do not contribute to good solutions once a domain is repeatedly solved.

6. Conclusion and future research directions

In this paper, we provided a major study on using various hyper-heuristics to solve very important problems in different domains of 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 several hyper-heuristics that are able to provide high-quality results across different real-life domains. These include Lean GIHH, but also the original GIHH, which despite their age are still highly competitive in these applications. EA-ILS, based on Iterated Local Search with evolutionary strategies, and LAST-RL, based on Reinforcement Learning, showed the best results among the recent state-of-the-art hyper-heuristics.

With these methods we were able to provide several advances for the individual domains. 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 fourth objective that has high practical relevance, hopefully fuelling more research into this more realistic version of the problem. For BDS, we were able to outperform previous heuristic approaches and come very close to the results of the most recent problem-specific heuristic solution method despite being problem independent and using a slower processor.

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 hyper-heuristics in the industrial software of our partner is in progress.

Finally, we provided a detailed investigation into the utility and usage of the individual low-level heuristics when applying different hyper-heuristics. The results showed which LLHs are used more frequently and take up more time depending on the hyper-heuristic and the domain, how chains of low-level heuristics are build, and which categories of LLHs are important for solution quality on which domain. We showed that different LLHs are important depending on the domain, providing both concrete results for the practical domains under investigation and general guidance for applying hyper-heuristics to new domains or applying them repeatedly on the same domain, which can further be helpful when considering new applications for hyper-heuristics.

To conclude, this paper contributes to AI research in two distinct directions. Firstly, it advances solution techniques for challenging personnel scheduling problems by introducing novel approaches for these domains. Secondly, by performing an extensive evaluation and analysis, it offers important insights into automated problem-solving techniques based on state-of-the-art hyper-heuristics that use various AI techniques to adapt for different domains.

There are several directions for future research. Regarding low-level heuristics for the personnel scheduling domains in this article, developing good crossover heuristics is still open and might give an advantage to hyper-heuristics that work on multiple solutions. There is also the option to define a parameter called *intensity of mutation* for mutations, and *depth of search* for local search, which can be adapted by hyper-heuristics as well. Not all hyper-heuristics change these parameters, but some might improve their results given the option. Regarding individual problem domains, the results from the detailed evaluation might help to improve problem-specific heuristics, e.g., by looking at the different impact of particular types of low-level heuristics, and trying to include the most helpful types into problem-specific methods.

Based on the findings that LLH selection was reasonably consistent over different instances of the same domain, it could also be interesting to include some domain-specific offline learning, if instances of a particular domain will be solved frequently. In this case online learning can still adapt the usage to the specific characteristics of the current instance, but domain experience can provide a better starting point for the hyper-heuristic compared to forgetting all information from previous instances. Finally, applying the

large set of hyper-heuristics to further real-life domains would be of interest to further test the general applicability of the winning hyper-heuristics, as well as collect more data on the usage and usefulness of low-level heuristics for new domains.

CRediT authorship contribution statement

Lucas Kletzander: Conceptualization, Investigation, Methodology, Software, Visualization, Writing – original draft. **Nysret Musliu:** Conceptualization, Methodology, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Lucas Kletzander reports financial support was provided by Christian Doppler Research Association. Nysret Musliu reports financial support was provided by Christian Doppler Research Association.

Data availability

Detailed result tables are available as supplementary material and on our webpage.

Acknowledgements

The financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development and the Christian Doppler Research Association is gratefully acknowledged. The authors acknowledge TU Wien Bibliothek for financial support through its Open Access Funding Programme.

Appendix A. Supplementary material

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.artint.2024.104172.

References

[1] E.K. Burke, M. Gendreau, M.R. Hyde, G. Kendall, G. Ochoa, E. Özcan, R. Qu, Hyper-heuristics: a survey of the state of the art, J. Oper. Res. Soc. 64 (12) (2013) 1695–1724, https://doi.org/10.1057/jors.2013.71.
[2] E.K. Burke, M.R. Hyde, G. Kendall, G. Ochoa, E. Özcan, J.R. Woodward, A classification of hyper-heuristic approaches: revisited, in: Handbook of Metaheuristics, Springer, 2019, pp. 453–477.
[3] E.K. Burke, M. Gendreau, M. Hyde, G. Kendall, B. McCollum, G. Ochoa, A.J. Parkes, S. Petrovic, The cross-domain heuristic search challenge–an international research competition, in: International Conference on Learning and Intelligent Optimization, Springer, 2011, pp. 631–634.
[4] G. Ochoa, M. Hyde, T. Curtois, J.A. Vazquez-Rodriguez, J. Walker, M. Gendreau, G. Kendall, B. McCollum, A.J. Parkes, S. Petrovic, et al., HyFlex: a benchmark framework for cross-domain heuristic search, in: European Conference on Evolutionary Computation in Combinatorial Optimization, Springer, 2012, pp. 136–147.
[5] L. Kletzander, N. Musliu, Hyper-heuristics for personnel scheduling domains, in: A. Kumar, S. Thiébaux, P. Varakantham, W. Yeoh (Eds.), Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling, AAAI Press, 2022, pp. 462–470.
[6] E.K. Burke, P. De Causmaecker, G.V. Berghe, H. Van Landeghem, The state of the art of nurse rostering, J. Sched. 7 (6) (2004) 441–499.
[7] A. Ernst, H. Jiang, M. Krishnamoorthy, D. Sier, Staff scheduling and rostering: a review of applications, methods and models, Eur. J. Oper. Res. 153 (1) (2004) 3–27.
[8] J. Van den Bergh, J. Beliën, P. De Bruecker, E. Demeulemeester, L. De Boeck, Personnel scheduling: a literature review, Eur. J. Oper. Res. 226 (3) (2013) 367–385.
[9] P. De Bruecker, J. Van den Bergh, J. Beliën, E. Demeulemeester, Workforce planning incorporating skills: state of the art, Eur. J. Oper. Res. 243 (1) (2015) 1–16, https://doi.org/10.1016/j.ejor.2014.10.038.
[10] K.R. Baker, Workforce allocation in cyclical scheduling problems: a survey, J. Oper. Res. Soc. 27 (1) (1976) 155–167.
[11] N. Musliu, J. Gärtner, W. Slany, Efficient generation of rotating workforce schedules, Discrete Appl. Math. 118 (1–2) (2002) 85–98.
[12] N. Musliu, A. Schutt, P.J. Stuckey, Solver independent rotating workforce scheduling, in: International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research, Springer, 2018, pp. 429–445.
[13] L. Kletzander, N. Musliu, J. Gärtner, T. Krennwallner, W. Schafhauser, Exact methods for extended rotating workforce scheduling problems, in: Proceedings of the International Conference on Automated Planning and Scheduling, vol. 29, 2019, pp. 519–527.
[14] T. Becker, A decomposition heuristic for rotational workforce scheduling, J. Sched. 23 (5) (2020) 539–554.
[15] N. Musliu, A. Schaerf, W. Slany, Local search for shift design, Eur. J. Oper. Res. 153 (1) (2004) 51–64, https://doi.org/10.1016/S0377-2217(03)00098-5.
[16] L. Kletzander, N. Musliu, Solving large real-life bus driver scheduling problems with complex break constraints, in: Proceedings of the International Conference on Automated Planning and Scheduling, vol. 30, 2020, pp. 421–429.
[17] O. Ibarra-Rojas, F. Delgado, R. Giesen, J. Muñoz, Planning, operation, and control of bus transport systems: a literature review, Transp. Res., Part B, Methodol. 77 (2015) 38–75.
[18] A. Wren, J.-M. Rousseau, Bus driver scheduling — an overview, in: G. Fandel, W. Trockel, J.R. Daduna, I. Branco, J.M.P. Paixão (Eds.), Computer-Aided Transit Scheduling, vol. 430, Springer Berlin Heidelberg, Berlin, Heidelberg, 1995, pp. 173–187.
[19] L. Kletzander, N. Musliu, P. Van Hentenryck, Branch and price for bus driver scheduling with complex break constraints, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, 2021, pp. 11853–11861.
[20] T.M. Mazzoli, L. Kletzander, P. Van Hentenryck, N. Musliu, Investigating large neighbourhood search for bus driver scheduling, in: 34th International Conference on Automated Planning and Scheduling, 2024.
[21] L. Kletzander, T.M. Mazzoli, N. Musliu, Metaheuristic algorithms for the bus driver scheduling problem with complex break constraints, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2022, pp. 232–240.

[22] R.M. Rosati, L. Kletzander, C. Blum, N. Musliu, A. Schaerf, Construct, merge, solve and adapt applied to a bus driver scheduling problem with complex break constraints, in: AIxIA 2022–Advances in Artificial Intelligence: XXIst International Conference of the Italian Association for Artificial Intelligence, Springer, 2023, pp. 254–267.
[23] N. Musliu, Heuristic methods for automatic rotating workforce scheduling, Int. J. Comput. Intell. Res. 2 (4) (2006) 309–326.
[24] J.H. Drake, A. Kheiri, E. Özcan, E.K. Burke, Recent advances in selection hyper-heuristics, Eur. J. Oper. Res. 285 (2) (2020) 405–428.
[25] P. Laborie, D. Godard, Self-adapting large neighborhood search: application to single-mode scheduling problems, in: Proceedings MISTA-07, vol. 8, 2007.
[26] C. Thomas, P. Schaus, Revisiting the self-adaptive large neighborhood search, in: International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research, Springer, 2018, pp. 557–566.
[27] C.-Y. Chuang, Combining multiple heuristics: Studies on neighborhood-base heuristics and sampling-based heuristics, Ph.D. thesis, Carnegie Mellon University, 2020.
[28] M. Luby, A. Sinclair, D. Zuckerman, Optimal speedup of Las Vegas algorithms, Inf. Process. Lett. 47 (4) (1993) 173–180.
[29] M. Mısır, K. Verbeeck, P. De Causmaecker, G. Vanden Berghe, An intelligent hyper-heuristic framework for CHeSC 2011, in: International Conference on Learning and Intelligent Optimization, Springer, 2012, pp. 461–466.
[30] S. Adriaensen, A. Nowé, 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), IEEE, 2016, pp. 1485–1492.
[31] A. Lehrbaum, N. Musliu, A new hyperheuristic algorithm for cross-domain search problems, in: International Conference on Learning and Intelligent Optimization, Springer, 2012, pp. 437–442.
[32] A. Kheiri, E. Keedwell, A sequence-based selection hyper-heuristic utilising a hidden Markov model, in: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, 2015, pp. 417–424.
[33] A. Kheiri, E. Özcan, An iterated multi-stage selection hyper-heuristic, Eur. J. Oper. Res. 250 (1) (2016) 77–90.
[34] S. Adriaensen, T. Brys, A. Nowé, Fair-Share ILS: a simple state-of-the-art iterated local search hyperheuristic, in: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, Association for Computing Machinery, 2014, pp. 1303–1310.
[35] S.A. Adubi, O.O. Oladipupo, O.O. Olugbara, Configuring the perturbation operations of an iterated local search algorithm for cross-domain search: a probabilistic learning approach, in: 2021 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2021, pp. 1372–1379.
[36] S.A. Adubi, O.O. Oladipupo, O.O. Olugbara, Evolutionary algorithm-based iterated local search hyper-heuristic for combinatorial optimization problems, Algorithms 15 (11) (2022) 405.
[37] F. Mischek, N. Musliu, Reinforcement learning for cross-domain hyper-heuristics, in: International Joint Conference on Artificial Intelligence (IJCAI), 2022, pp. 4793–4799.
[38] L. Kletzander, N. Musliu, Large-state reinforcement learning for hyper-heuristics, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 37, 2023, pp. 12444–12452.
[39] L. Kletzander, N. Musliu, Modelling and solving the minimum shift design problem, in: International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research, Springer, 2019, pp. 391–408.