Leveraging problem-independent hyper-heuristics for real-world test laboratory scheduling

Florian Mischek
Nysret Musliu
fmischek@dbai.tuwien.ac.at
musliu@dbai.tuwien.ac.at
Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling, DBAI, TU Wien
Vienna, Austria

ABSTRACT

The area of project scheduling problems has seen a tremendous amount of different problem variations. Traditionally, each problem variant requires custom solution approaches in order to produce high-quality solutions. Developing and tuning these methods is an expensive process that may have to be repeated as soon as the requirements or problem structures change. On the other hand, research into hyper-heuristics has produced general heuristic problem-solving techniques that were developed to achieve good results on multiple diverse problem domains. They work with a set of comparatively simple low-level heuristics and dynamically adapt themselves to each new problem variant. In this paper, we investigate hyper-heuristic approaches for a real-world industrial test laboratory scheduling problem and develop a new problem domain for the HyFlex hyper-heuristic framework. We propose a diverse portfolio of low-level heuristics that can be dynamically selected during the search process by hyper-heuristics to solve the problem. We evaluate and compare the performance of several problem-independent hyper-heuristics on this domain and show that they are able to match, and sometimes even exceed, the performance of state-of-the-art solution techniques that were developed and tuned specifically for this problem.

CCS CONCEPTS

• **Mathematics of computing → Combinatorial optimization**; • **Applied computing → Industry and manufacturing**; • **Computing methodologies →** *Planning and scheduling*; • **Theory of computation → Optimization with randomized search heuristics**.

KEYWORDS

Hyper-heuristics, HyFlex, Test Laboratory Scheduling

ACM Reference Format:
Florian Mischek and Nysret Musliu. 2023. Leveraging problem-independent hyper-heuristics for real-world test laboratory scheduling. In *Genetic and Evolutionary Computation Conference (GECCO '23), July 15–19, 2023, Lisbon, Portugal.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3583131.3590354

1 INTRODUCTION

Project scheduling problems appear in countless different variants across many areas. The quality of such schedules can have a tremendous impact on costs, time, service quality, and employee wellbeing. At the same time, the typically large number of activities to schedule and complex constraints makes manual scheduling time-consuming and error-prone even for experts, indicating the need for automated solutions. Here, the large variety in requirements between different settings or even across different instances of otherwise similar settings poses a big challenge, as solution methods are often specific to a single problem formulation. Adapting methods to new variants frequently requires expensive development work and tuning.

One such variant is a complex real-world project scheduling problem that arises in industrial test laboratories. The Test Laboratory Scheduling Problem (TLSP) was first introduced in [15]. It is an extension of the well-studied Resource-Constrained Project Scheduling Problem (RCPSP), where the solver has to group atomic tasks into larger units called jobs in addition to scheduling those jobs. Further aspects include several additional constraints, some of which are unique to the TLSP such as the requirement that some tasks have to be performed by the same employees, and a non-standard objective function which is a combination of multiple individual objectives.

Different solution methods have been proposed for the TLSP. Mischek and Musliu [14, 15] described metaheuristic approaches, using a combination of different neighborhood structures. Initially restricted to a subproblem with a fixed and predetermined task grouping, these methods were later extended to the full TLSP [17, 18]. For both problem versions, Simulated Annealing (SA) found the best results. An exact approach using Constraint Programming (CP) was proposed by Danzinger et al. [10, 11], based on an earlier model for the subproblem [12]. That work also includes a Very Large Neighborhood Search (VLNS), which repeatedly solves restricted subproblems consisting of one or a small number of projects, while the rest of the schedule remains fixed. Finally, Geibinger et al. [13] developed another exact solution method based on Constraint Answer-set Programming (CASP). While the CP and CASP solvers provide good solutions for small instances, their performance is not competitive on larger and practically sized instances compared to heuristic approaches like SA and VLNS. Both SA and VLNS are deployed in an industrial test laboratory, where they are successfully used to create and update the work schedules of the lab [10].

Something all previously proposed heuristic solution approaches have in common is that they underwent extensive parameter tuning based on the characteristics of the available benchmarking instances. On the one hand, this tuning enabled them to find high-quality solutions for these and similar instances. On the other hand, tuning is time-consuming and the resulting configuration may not be a good fit for future instances with different properties or potential newly arising variants of the problem.

In this paper, we follow a different approach: We investigate the use of problem-independent hyper-heuristics to solve the TLSP, which are a class of high-level heuristic problem solving techniques that are able to automatically adapt to new and unseen problem domains. Towards this end, we develop a new problem domain for the HyFlex hyper-heuristics framework, which is the de-facto standard framework for selection hyper-heuristics in the literature. Besides efficient implementations of several other components, such as instance and solution representation, we propose a diverse portfolio of modular low-level heuristics for the TLSP. This allows us to employ several state-of-the-art hyper-heuristics on the new TLSP domain. We perform a thorough evaluation of these hyper-heuristics and compare their results to those of the best available problem-specific approaches (SA and VLNS). Our experimental results show that even without any kind of problem-specific tuning, hyper-heuristics are able to compete with and sometimes even improve upon the best problem-specific methods.

The paper is structured as follows: In the next section, we provide an overview of hyper-heuristics in general and the HyFlex framework, followed by a description of the TLSP in Section 3. We introduce a new HyFlex problem domain based on the TLSP in Section 4, which also includes a description of the low-level heuristic portfolio. Our experimental evaluation of several state-of-the-art hyper-heuristics on this domain is described in Section 5, including a comparison with other problem-specific algorithms for the TLSP. Finally, we give concluding remarks in Section 6.

2 HYPER-HEURISTICS

Hyper-heuristics are a class of high-level problem solving techniques that operate over a portfolio of so-called low-level heuristics (LLHs). Instead of directly moving through the space of potential candidate solutions to a problem, hyper-heuristics instead select and apply these LLHs to one or a population of solutions. This indirection enables hyper-heuristics to have a high degree of adaptability and generality, as switching out the available LLHs can allow a hyper-heuristic to solve a completely different problem. Most hyper-heuristics also include adaptive components, which automatically adjust their behaviour towards LLHs that are beneficial for the current problem, instance, or even state of the search. For the purpose of this work, we consider only *selection* hyper-heuristics, for which the available LLHs are given as a discrete set of operators, in contrast to *generation* hyper-heuristics which try to generate new LLHs from basic algorithmic components [6].

Research on hyper-heuristics experienced a big boost in 2011 due to the Cross-Domain Heuristics Search Challenge 2011 (CHeSC 2011) [5], an international competition where participants had to develop hyper-heuristics that could perform well on multiple different domains. Specifically, the submitted hyper-heuristics were evaluated on the maximum satisfiability problem (MaxSAT), bin packing (BP), the flow-shop problem (FS), personnel scheduling (PS), the travelling salesman problem (TSP) and vehicle routing (VRP), six diverse and well-studied academic problem domains. Of these, the first four domains where announced in advance, while the TSP and VRP domains were hidden from the competitors until the final evaluation. Similarly, the hyper-heuristics were also evaluated at least in part on previously unknown instances. This required participants to develop hyper-heuristics that would work well on different problems, but also generalize to unseen instances and even completely new problems.

In total, there were 20 competing hyper-heuristics submitted for the challenge. They were scored based on their relative rank on each instance, inspired by the Formula 1 scoring scheme [5].

The competition was won by Mısır et al. [19]. Their algorithm AdapHH (sometimes also referenced as GIHH) combines several adaptive mechanisms: A subset of active LLHs with promising performance characteristics is managed and periodically updated using an adaptive dynamic heuristic set (ADHS) strategy. At each iteration, one of the active LLHs is selected and applied to the current solution with probabilities based on their previous performance. The search parameters of each LLH are maintained by a separate component employing a reward-penalty scheme. Alternatively, the algorithm may choose to apply a pair of LLHs in direct succession, if that pair has proved successful earlier in the run (relay hybridisation). The acceptance of the resulting solution is determined by an Adaptive iteration limited list-based threshold accepting (AILLA) mechanism, which can also decide to restart the search from from a new randomly generated initial solution if necessary.

All hyper-heuristics for the competition had to be implemented in the software framework HyFlex [20], which was developed for the competition. One important concept in HyFlex is the *domain barrier*, which guarantees the problem-independence of the hyper-heuristics. It is visualized in Figure 1. All problem-specific information of a domain, such as instance and solution representation, objective function, or LLHs, is hidden from the hyper-heuristic. The hyper-heuristic only knows the number and a high-level type of each available LLH, as well as the current objective value of each solution in memory. It interacts with the problem domain by instructing it to apply one of the LLHs (h_i) to a solution (s_j) and store the result in memory (s_k). The problem domain then returns the new objective value of the solution ($f(s_k)$)[1].

The LLHs are partitioned into four types:

- **Mutation (MU)** operators apply random changes to a candidate solution, without regard for objective values.
- **Ruin-and-recreate (RR)** operators destroy a part of a solution and attempt to rebuild or repair it afterwards.
- **Local search (LS)** heuristics attempt to improve the solution, typically via small and iterative changes. While finding such an improvement is not guaranteed, the resulting solution will never be worse than the original.
- **Crossover (CO)** operators combine two solutions to produce a new solution that contains assignments of both parents.

[1]Note that the solution s_k itself is not provided to the hyper-heuristic to ensure that it cannot gain any problem-specific information from the solution's structure.

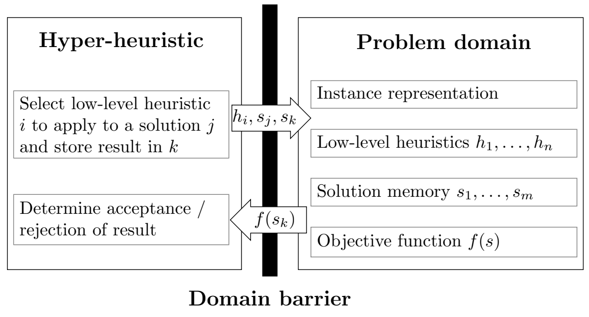
Hyper-heuristic
Select low-level heuristic i to apply to a solution j and store result in k
Determine acceptance / rejection of result
h_i, s_j, s_k
$f(s_k)$
Problem domain
Instance representation
Low-level heuristics $h_1, \ldots, h_n$
Solution memory $s_1, \ldots, s_m$
Objective function $f(s)$
Domain barrier

Figure 1: Structure of HyFlex. The domain barrier separates hyper-heuristics from the problem domain implementations and hides problem-specific information. Figures source: [16].

The hyper-heuristic has to select both parents, instead of only one solution as for the other operators.

In addition, LLHs can support either of two numerical parameters. The *depth of search* (DoS) parameter is intended to determine the number of steps performed by incremental improvement heuristics, while the *intensity of mutation* (IoM) parameter affects the strength of the perturbations applied by both mutation and ruin-and-recreate heuristics. Both parameters take values between 0.0 and 1.0, which can be set by the hyper-heuristic.

Since 2011, HyFlex has become the de-facto standard framework for selection hyper-heuristics. Several authors have developed new hyper-heuristics, such as FS-ILS [1], GEP-HH [21], MCTS-HH [22], QHH [7], FI [9], TS-ILS [3], and many more. Something most successful hyper-heuristics have in common, including all those mentioned above, is that they include some sort of adaptive component to adjust their behavior according to the performance of the LLHs on the current instance. Often, these components borrow concepts from online machine learning strategies, such as reinforcement learning [6, 7]. Many of the top hyper-heuristics also follow the general structure of iterated local search (ILS), alternating cycles of diversification and intensification, or perturbation and search [1, 3, 7].

Other authors have extended HyFlex with additional problem domains, including the knapsack problem (KP), quadratic assignment problem (QAP), and maximum-cut problem (MAC) [2].

3 PROBLEM STATEMENT

The TLSP is an extension of the Resource-Constrained Project Scheduling Problem (RCPSP). We give here a summary of the problem description, the full formal definition is provided as supplementary material and can also be found in [15].

In the TLSP, the goal is to find a schedule for multiple projects, each containing several tasks. Each task requires different kinds of resources (employees, workbenches, and several types of equipment) and must be executed in one of several modes, which affects both its duration and its resource requirements. Further, only a subset of all units of a resource is suitable for performing any particular task. Additional constraints include release dates and deadlines, precedence constraints between tasks of a project, fixed assignments, restrictions on which tasks can be grouped together into a job, and finally linked tasks, which are sets of tasks that must all be assigned the same employees.

The solver has to determine a partition of the tasks into jobs, and then assign a mode, discrete time slots, and resources to each job. The properties and requirements of a job are determined by those of the tasks it contains. Within a job, tasks are executed sequentially, but their order is not defined[2]. As a result, the job must fulfill the requirements of all tasks for its whole duration, which is the sum of the durations of its contained tasks plus an additional setup time. For example, resource requirements of a job are the maximum required number among all its tasks for each resource type, and the set of available resource units that can be assigned is the intersection of the available units of its tasks. Similarly, an execution mode can be assigned to a job only if that mode is available for all its tasks.

The quality of a feasible schedule is determined as a linear combination of several objectives: In contrast to typical variants of RCPSP, we do not aim to minimize the makespan, but instead the total duration of each project, from the start of its first job to the end of its last scheduled job. Other objectives include minimizing the number of jobs, the number of different employees assigned to each project, and the assignment of non-preferred employees to jobs. Finally, the last objective aims to finish each job already by a certain target date, which is typically several time slots before the deadline. The relative weights of these objectives depend on the preferences and goals of individual laboratories in practice. In line with previous work, we have used uniform weights of 1 for the purpose of our evaluations. This makes our results comparable to those of previously described methods.

4 HYPER-HEURISTIC PROBLEM DOMAIN MODEL

To enable problem-independent hyper-heuristics to work on the TLSP, we needed to provide several problem-specific components. Here, the goal was to use flexible and modular components, such that they can easily be switched out and supplemented to deal with variations of the TLSP. Previously published solution approaches for the TLSP (e.g. [11, 17]) serve to provide inspirations and baseline implementations for useful building blocks towards this goal. A more detailed description of our problem domain implementation is provided as supplemental material.

For instance and solution representation, we reused the data structures developed for the metaheuristic approaches in [17]. These also provide us with methods to evaluate the objective function, including efficient delta evaluation after applying changes. Initial candidate solutions are created by first grouping tasks into as few jobs as possible, and then assigning random modes, time slots, and resources to the jobs.

Regarding the solution evaluation, we had to make a further adjustment to ensure compatibility with the HyFlex framework, which assumes that candidate solutions produced by construction heuristics and LLHs are always feasible. For the TLSP, finding any feasible solution is already an NP-hard problem, so we cannot guarantee this property. While both construction heuristics and all LLHs respect grouping constraints, time windows, precedence

[2]This allows tasks within a job to be freely reordered during the actual execution of the schedule, increasing the flexibility and robustness of the schedule

Table 1: Comparison of LLHs in different problem domains.

Domain	Source	MU	RR	LS	CO	Total
MaxSAT	[5]	4	1	2	2	9
BP	[5]	3	2	2	1	8
FS	[5]	5	2	4	3	14
PS	[5]	1	3	4	3	11
TSP	[5]	5	1	6	3	15
VRP	[5]	4	2	4	2	12
KP	[2]	5	2	6	3	16
QAP	[2]	2	3	2	2	9
MAC	[2]	2	3	3	2	10
TLSP		6	2	13	3	24

constraints, mode and resource availability as well as resource requirement constraints, the linked tasks and single assignment constraints can be violated. We allow such infeasible candidate solutions during search, but add a high penalty of 10000 for each remaining conflict. This provides a strong incentive for the hyper-heuristic to prioritize the search for feasible solutions at first and nearly guarantees that feasible solutions are evaluated as better than any infeasible ones[3].

4.1 Low-level heuristics

The portfolio of LLHs is the most important part of the problem domain. For most successful hyper-heuristics to work well, we need a selection of LLHs of each of the four types, with variety both in their complexity and the affected aspects of a solution. This enables hyper-heuristics to detect and apply the most useful LLHs for each instance and at any given moment in the search. In total, we developed 6 mutation operators, 13 local search heuristics, 2 ruin-and-recreate operators, and 3 crossovers (see Table 1 for a comparison with other HyFlex problem domains). The high number of operators compared to the other domains is explained by the TLSP's real-world complexity, with multiple aspects handled by different operators.

The first six domains are from the original competition [5], while the LLH portfolios for KP, QAP, and MAC were introduced in [2]. The *mutation* (MU) operators apply random changes to a given schedule:

Random mode move Randomly changes the assigned mode of a job to different one. Replacement modes can be selected only if this would not introduce conflicts regarding precedences or time windows. If this changes the number of required employees, randomly chosen employees are added or removed.

Random timeslot move Randomly moves a job to a different time slot.

Random resource move Randomly replaces a single workbench, employee, or device assigned to a job by a different one.

[3]This is particularly true for the instances in our benchmark set, where objective values of feasible solutions rarely reach above 7000

Random regrouping move Performs a single random move that alters the task grouping. This can be either a transfer of a task to a new job, merging two existing jobs into, or splitting off a subset of a job's tasks into a new job. If this move changes the resource requirements of any involved job, resources are added or removed randomly. Otherwise, existing time slot, mode, and resource assignments are kept if possible.

Randomize A larger mutation that selects a subset of all projects and randomizes the assignments of all jobs, leaving only the grouping intact. The IoM parameter determines the fraction of projects randomized.

Random walk Performs multiple moves of a random walk procedure, using a combination of all scheduling and regrouping neighborhoods employed by the first four LLHs. The number of moves is proportional to the DoS parameter, with a maximum of 100 moves at DoS = 1.0.

The *ruin-and-recreate* (RR) operators all follow the same principle: They delete assignments from a subset of all jobs and then greedily restore them one job at a time. The order in which the assignments will be restored is always the same: First, fixed assignments are added, then projects are selected in order of increasing release date of their earliest task. Within a project, jobs are selected in an arbitrary topological order.

Greedy reconstruct (multiple projects) Affects all jobs in a subset of all projects. Leaves the task grouping unaffected.

Greedy regrouping (single project) Affects all jobs of a single project, including their task grouping. A new grouping of tasks into jobs is also built greedily, assigning tasks to existing matching jobs wherever this is possible without conflict.

The *local search* (LS) heuristics mostly perform one or several moves of a local search procedure with different neighborhoods. Those search heuristics that allow worsening moves return the best solution found at any point during the search, not necessarily the last solution. Three different search heuristics are considered: Hill climbing selects the locally best move across all jobs, which is expensive but also guarantees to find local improvements if they exist. MinConflict selects a random job and applies the best possible move for that job. This scales well with increasing numbers of jobs, but may miss potential improvements in unselected jobs. Finally, stochastic hill climbing performs random moves and uses the Metropolis criterion to decide whether to accept them: Improvements or solutions of equal quality are always accepted, while worsening moves may still be accepted with probability $e^{-\Delta/T}$, where Δ is the difference in objective value and T is a fixed parameter called temperature. At higher temperatures, worse solutions are accepted with higher probability.

Hill climbing (mode, timeslot) Selects the best change in either mode or time slot for any job at each move.

Hill climbing (resources) Selects the best change of a single assigned resource unit for any job at each move.

Hill climbing (JobOpt) Selects the best (re-)assignment of mode, time slot and all resources for any job at each move. This corresponds to the changes available in the *JobOpt* neighborhood of [17].

Hill climbing (regrouping) Selects the best regrouping option for any job at each move, with the same options as for the **Random regrouping move** LLH.

MinConflict (mode, timeslot) Selects the best change in either mode or time slot for a randomly chosen job at each move.

MinConflict (resources) Selects the best change of a single assigned resource unit for a randomly chosen job at each move.

MinConflict (JobOpt) Selects the best (re-)assignment of mode, time slot and resources for any job at each move.

MinConflict (regrouping) Selects the best regrouping option for a randomly chosen job at each move.

Stochastic hill climbing (high temperature) Performs a large number of random moves, using the Metropolis acceptance criterion at a temperature of 100. Possible moves include all options for reassignments of mode, time slot and resources, as well as the regrouping options described earlier. The weight of hard constraint violations is set to 50 for the purpose of the Metropolis criterion. At this high temperature, moves that add no more than one or two hard constraint violations will likely be accepted.

Stochastic hill climbing (low temperature) Performs random moves with the Metropolis acceptance criterion as the previous operator, but at a temperature of 5. At this low temperature, only moves that do not result in additional constraint violations have a realistic chance to be accepted.

Stochastic hill climbing (minimal temperature) Performs random moves with the Metropolis acceptance criterion as the previous two operators, but at a temperature of 1. At this minimal temperature, moves are unlikely to be accepted if they increase the penalty at all, although there is a chance for very small increases.

All the above LLHs use the DoS parameter to determine the number of moves performed in a single LLH application. Those using hill climbing perform at most 100 moves at DoS = 1.0, those following the MinConflict-based search perform at most 500 moves and those based on stochastic hill climbing perform up to 50000 moves (as selecting random moves can be done very fast).

The remaining two local search heuristics follow a different structure:

Single project CP Uses a Constraint Programming solver to find the optimal solution for a single, randomly chosen project, while the rest of the schedule is kept fixed. The CP model used is the one described in [11]. If the optimum cannot be found within a given time, the best solution found is returned instead. The available time is determined by the DoS parameter, with a maximum of 30s at DoS = 1.0. As a CP solver, we used Chuffed [8].

Job-wise greedy Iterates over all jobs and replaces each job's assignments with the locally best ones, relative to the current assignment of all other jobs. Jobs are ordered according to their project's earliest release date, and within a project in an arbitrary topological order.

Incorporating the *Single project CP* required extending the HyFlex framework itself due to the call to the external CP solver, which the timekeeping module was not equipped to handle correctly. Our extension allows problem domains to keep track of time spent waiting on external processes in addition to their own processing time.

Finally, the crossover operators take two parent schedules as input and produce as offspring a new schedule that contains assignments of both parents.

Random project Randomly selects for each project whether the offspring should take the assignments of the first or the second parent schedule.

Single point Randomly selects a time slot. Assignments for all projects starting before that point are taken from the first parent, the remaining assignments are taken from the second parent.

Two point Randomly selects two time slots. Assignments for all projects starting before the first or after the second slot are taken from the first parent, the remaining assignments are taken from the second parent.

The LLHs described above were newly developed for the TLSP problem domain, although some reuse components of previously published methods. In particular, the MU operators (except *Randomize*) as well as the *Hill climbing*, *MinConflict*, and *Stochastic hill climbing* LS operators use the neighborhood relations described in [17], while the *Single project CP* operator uses the CP model described in [11], as mentioned.

5 EVALUATION

We evaluated our implementation on a set of state-of-the-art hyper-heuristics available for HyFlex: We compare TS-ILS [3][4] and FS-ILS [1][5], the first and third place hyper-heuristics[6] of a recent comparison [16], plus RL, a hyper-heuristic based on reinforcement learning described in that paper[7]. The authors of TS-ILS also published a new hyper-heuristic just this year, EA-ILS [4][8], which was not yet available for the previous comparison. Finally, we also included AdapHH [19][9], the competition winner of 2011.

The selected hyper-heuristics were run on a benchmark server with 224GB RAM and two AMD Opteron 6272 Processors each with a frequency of 2.1GHz and 16 logical cores (the same machine used for the evaluation of SA and VLNS [10, 17]). Each hyper-heuristic was executed 15 times on each of the 33 benchmark instances used in [17], with a timeout of 10 minutes per run. This instance set consists of 30 randomly generated instances of various sizes and properties, plus 3 real-world instances taken from our industrial partner. Notably, we did not perform any problem-specific tuning or adaptation for any of the evaluated hyper-heuristics.

The results of these evaluations are listed in Table 2. For comparison we also included results of the current state-of-the-art solvers for the TLSP, SA [17] and VLNS [10] [10].

[4]Source code: https://github.com/dubystev/Synergy-HH
[5]Source code: https://github.com/Steven-Adriaensen/FS-ILS
[6]Source code of the second place hyper-heuristic, GEP-HH [22], was not available, so we could not run it on the new domain.
[7]Source code: https://gitlab.tuwien.ac.at/florian.mischek/hyper-heuristics-public
[8]Source code: https://github.com/dubystev/Synergy-HH
[9]Source code obtained via personal communication
[10]Results for VLNS with a timeout of 10 minutes were also taken from [17]. That evaluation included only 5 runs of VLNS.

Table 2: Results for the five evaluated hyper-heuristics on the TLSP problem domain, plus results for SA [17] and VLNS [10], under a time limit of 10 minutes. For each hyper-heuristic, the table lists the number of feasible solutions found (Feas), the average objective value among feasible solutions (Avg), and the best solution found (Best). The best objective value found by any method within this time limit for each instance is marked in bold.

#	RL			FS-ILS			TS-ILS			EA-ILS			AdapHH			SA			VLNS		
	Feas	Avg	Best	Feas	Avg	Best	Feas	Avg	Best	Feas	Avg	Best	Feas	Avg	Best	Feas	Avg	Best	Feas	Avg	Best
1	15/15	57.0	**57**	15/15	57.0	**57**	15/15	57.0	**57**	15/15	57.0	**57**	15/15	57.0	**57**	15/15	58.0	58	5/5	57.0	**57**
2	15/15	71.0	**71**	15/15	71.0	**71**	15/15	71.0	**71**	15/15	71.0	**71**	15/15	71.0	**71**	15/15	72.4	72	5/5	71.0	**71**
3	15/15	141.7	**141**	15/15	141.1	**141**	15/15	141.3	**141**	15/15	144.3	**141**	15/15	141.0	**141**	15/15	156.1	147	5/5	141.0	**141**
4	15/15	102.1	**101**	15/15	103.0	**101**	15/15	103.3	102	15/15	104.0	**101**	15/15	101.2	**101**	15/15	114.5	103	5/5	101.0	**101**
5	15/15	285.1	281	15/15	284.5	281	15/15	292.9	287	15/15	296.5	286	15/15	282.0	280	15/15	303.3	296	5/5	242.2	**240**
6	15/15	148.3	141	15/15	147.9	143	15/15	158.8	155	15/15	158.0	152	15/15	144.5	**140**	15/15	165.1	157	5/5	140.0	**140**
7	15/15	304.1	292	15/15	302.2	298	15/15	314.9	304	15/15	309.9	305	15/15	306.1	297	15/15	300.9	296	5/5	287.0	**283**
8	15/15	298.9	290	15/15	298.1	296	15/15	309.3	301	15/15	309.4	300	15/15	295.7	291	15/15	302.4	292	5/5	284.6	**283**
9	15/15	478.5	454	15/15	452.1	437	15/15	476.0	452	15/15	464.7	442	15/15	457.7	434	15/15	470.6	456	5/5	442.2	**429**
10	15/15	634.5	556	15/15	584.0	556	15/15	633.5	584	15/15	624.3	581	15/15	589.9	549	15/15	566.5	551	5/5	590.2	**547**
11	15/15	914.5	868	15/15	913.2	872	15/15	966.1	909	15/15	966.6	898	15/15	878.7	843	15/15	938.4	912	5/5	860.8	**840**
12	15/15	689.3	656	15/15	689.5	670	15/15	719.1	701	15/15	728.5	706	15/15	666.5	655	15/15	675.8	666	5/5	660.6	**653**
13	15/15	330.5	324	15/15	328.0	320	15/15	334.7	330	15/15	334.6	325	15/15	331.3	318	15/15	338.2	327	5/5	311.8	**309**
14	15/15	425.4	415	15/15	423.5	415	15/15	438.9	432	15/15	437.9	426	15/15	417.3	414	14/15	420.4	418	5/5	415.0	**412**
15	15/15	1366.2	1115	14/15	1330.6	1235	15/15	1341.1	1259	15/15	1431.3	1253	15/15	1183.3	1082	15/15	1063.5	**1014**	5/5	1106.2	1025
16	10/15	1305.4	1251	14/15	1266.9	1227	15/15	1336.0	1286	15/15	1339.6	1275	15/15	1220.5	**1172**	13/15	1236.7	1216	5/5	1184.4	1175
17	15/15	1289.9	1177	15/15	1216.7	1141	15/15	1248.5	1207	15/15	1265.9	1220	15/15	1198.3	1145	15/15	1185.3	**1140**	5/5	1166.6	1150
18	15/15	1560.3	1412	14/15	1540.9	1469	15/15	1585.5	1505	14/15	1630.4	1495	15/15	1441.9	**1389**	15/15	1527.4	1500	5/5	1482.2	1436
19	9/15	2487.1	2131	11/15	2418.5	2224	13/15	2482.3	2277	10/15	2787.1	2391	14/15	2155.6	**2075**	14/15	2239.9	2133	5/5	2419.2	2350
20	6/15	2741.8	2512	7/15	2651.6	2502	4/15	2754.8	2473	4/15	3014.3	2786	13/15	2359.4	**2265**	15/15	2489.7	2391	5/5	2986.6	2955
21	7/15	727.7	627	13/15	713.0	656	15/15	769.9	677	15/15	733.0	668	14/15	658.7	612	15/15	673.9	632	5/5	596.8	**570**
22	15/15	832.1	766	15/15	821.4	786	15/15	838.3	811	15/15	856.9	800	15/15	760.3	**736**	15/15	784.6	755	5/5	775.0	763
23	9/15	2599.9	2178	10/15	2436.1	2200	11/15	2808.9	2557	8/15	2705.6	2528	13/15	2201.3	**2025**	12/15	2167.7	2070	0/5	-	-
24	11/15	2116.4	2016	12/15	2119.4	1981	15/15	2279.5	2084	8/15	2300.4	2071	14/15	1996.9	1921	15/15	1869.3	**1807**	0/5	-	-
25	8/15	3716.4	2765	5/15	3410.8	3215	9/15	3650.0	3180	5/15	3560.6	3491	14/15	2703.9	**2583**	10/15	2897.6	2601	0/5	-	-
26	6/15	3278.8	2945	3/15	3189.0	3014	7/15	3288.3	3127	3/15	3593.0	3198	9/15	2999.2	2861	13/15	2971.3	**2809**	5/5	3345.4	3109
27	12/15	2662.1	2195	13/15	2546.8	2387	13/15	2811.9	2499	12/15	3005.1	2451	13/15	2451.3	2281	15/15	2124.3	**2017**	5/5	2750.6	2473
28	12/15	2667.4	2477	14/15	2572.4	2530	13/15	2745.5	2608	13/15	2787.4	2548	15/15	2416.8	**2310**	15/15	2516.5	2468	5/5	2407.2	2372
29	0/15	-	-	0/15	-	-	2/15	4494.0	4361	1/15	5767.0	5767	5/15	4402.0	4010	10/15	4358.5	**3965**	0/5	-	-
30	1/15	6746.0	6746	1/15	5655.0	5655	0/15	-	-	1/15	6124.0	6124	2/15	5432.0	5219	14/15	5104.1	**4989**	0/5	-	-
Lab1	0/15	-	-	0/15	-	-	1/15	4298.0	4298	0/15	-	-	0/15	-	-	4/15	3558.0	**3511**	5/5	4080.8	3923
Lab2	0/15	-	-	0/15	-	-	0/15	-	-	0/15	-	-	0/15	-	-	1/15	2811.0	2811	5/5	2896.8	**2779**
Lab3	2/15	2753.0	2738	0/15	-	-	1/15	2974.0	2974	0/15	-	-	1/15	2653.0	2653	3/15	2616.7	**2606**	5/5	2687.4	2646

Table 3: Percentage of feasible solutions found by each hyper-heuristic within 10 minutes, as well as SA and VLNS.

HH	% feasible
RL	0.73
FS-ILS	0.75
TS-ILS	0.79
EA-ILS	0.74
AdapHH	0.83
SA	0.88
VLNS	0.85

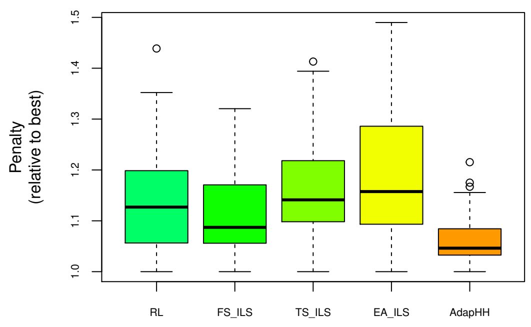

Figure 2: Comparison between results of different hyper-heuristics on the TLSP problem domain. Results for each instance were scaled by the best result achieved on that instance.

The five hyper-heuristics managed to find feasible solutions for between 73% (RL) and 83% (AdapHH) of all runs, as shown in Table 3. Despite being proposed already in 2011, AdapHH clearly managed to find solutions of better quality in general than the other more recent hyper-heuristics (see Figure 2) in addition to finding the most feasible solutions.

The three real-world instances seem to be particularly hard to solve across all methods except VLNS, and particularly so for the evaluated hyper-heuristics. According to an analysis of these instances in [17], an important factor in making them more challenging to solve than the randomly generated instances of comparable size seems to be inclusion of employee vacations and other absences. Modeled as additional blocker tasks with a fixed employee and time slot assignment (and no other resources required), these absences split the available periods of each employee into several smaller fragments, making it more difficult to fit the remaining tasks around the blocker tasks. Given that VLNS uses a CP solver to produce the initial (already feasible) solution, it is not impacted as much by these more tightly constrained instances.

The comparison with SA and VLNS is shown in Figure 3. Even without any problem-specific tuning, AdapHH was able to find solutions that match the quality of those produced by the specialised algorithms, although it did find slightly fewer feasible solutions. AdapHH managed to find the best known solutions under that time limit for 13 instances, 8 of which are better than the best results produced by SA and VLNS. After scaling the objective value of each feasible solution by the best objective value found for that instance, median results for AdapHH, SA, and VLNS are 1.043, 1.068, and 1.022, respectively. Overall, AdapHH produced better results than SA (one-sided Wilcoxon rank-sum test, $U = 70159$, $p < 10^{-8}$), though it is beaten by VLNS ($U = 24151.5$, $p = 0.002$). Looking at small (≤ 20 projects) instances separately, one can see that SA struggles with finding the best solution on small instances, strengthening the advantage of AdapHH on those instances ($U = 16825$, $p < 10^{-14}$). Conversely, VLNS falls behind on larger instances to the point where it is overtaken by AdapHH ($U = 4440$, $p = 0.048$).

5.1 Usage of low-level heuristics

Finally, we also investigated the use of the different LLHs we introduced for the TLSP. For this purpose, we logged the number of calls to each LLH made by AdapHH during a single run over all instances. The results are displayed in Figure 4. While this distribution of course depends on the behavior of the hyper-heuristic, particularly with respect to difference between different LLH types, we can still gain some interesting insights from it. Overall, AdapHH clearly favored fast LLHs over slower and more expensive ones. This affects mostly LS heuristics, except for LS:OneByOneGreedy and to a lesser extent MinConflict with the two smaller neighborhoods, but can also be seen in the MU heuristics where those that change only a single assignment are used more often.

An interesting difference can also be seen between large and small instances: The weaker MU operators (MU:Random[Mode/Timeslot/Resource]Move) were chosen far more often on small instances. A potential reason for this behavior is that smaller perturbations are already sufficient to reach new areas of the solution space on these instances, while stronger perturbations destroy too large parts of a solution. On larger instances on the other hand, AdapHH placed more emphasis on LS heuristics, particularly those that can be applied quickly, and the RR:Regroup operator. We observed that in general, runtimes of each operator increase with instance size, although some scale faster than others, particularly those that have to examine all jobs in each iteration. Accordingly, fewer operators can be applied within the time limit for large instances, while the number of potential sites for local improvements increases. This allows a larger number of initially successful LS applications and by the time the solution quality reaches local optima that would require more exploration to allow further improvements, the time limit is already reached.

This trend can also be observed when looking at the percentage of successful applications of each LLH (Figure 5), i.e. those that resulted in a new best solution for the run. As expected, most improvements were found by LS and RR heuristics, particularly those searching more complex neighborhoods. Here it can be seen that success probabilities are consistently higher on larger instances, indicating that the search on smaller instances spends most of its time in regions of the search space that are already close to optimal, such that further improvements are difficult to find.

6 CONCLUSIONS

In this paper we have investigated the use of problem-independent hyper-heuristics to solve a real-world industrial project scheduling problem. To be able to apply general purpose hyper-heuristics to this problem, we proposed a new problem domain for the TLSP as an extension to the well-known hyper-heuristic framework HyFlex. In doing so, we developed a diverse portfolio of LLHs of all four types supported by HyFlex, which have different characteristics and affect different aspects of the problem. They can be selected and applied by hyper-heuristics during the search in order to find high-quality solutions.

We experimentally evaluated our approach on five state-of-the-art hyper-heuristics. In particular the hyper-heuristic AdapHH, the winner of the CheSC 2011 hyper-heuristics competition, proved to be very successful on this new problem domain. Also compared to the currently best problem-specific algorithms for the TLSP, SA and VLNS, AdapHH could provide very good solutions, including 13 best known solutions for the given time limit. Moreover, it achieved this level of performance on instances of all available sizes, while both SA and VLNS excel only on particularly large, respectively small instances.

Our results clearly show that general and problem-independent hyper-heuristics, using an appropriate set of modular and complementary LLHs, can compete with specialised methods even without any problem-specific tuning.

Of particular note is that due to the constraints of the HyFlex framework, each run of a hyper-heuristic started out without any knowledge of problem domain or LLH performance characteristics, a major drawback compared to previous approaches which include problem-specific knowledge already in their design and tuning. Storing the information gained during earlier runs and using it to boost hyper-heuristic performance on repeated runs on the same problem domain could lead to even better results.

In the future, we also intend to build upon these results by analyzing the impact of individual LLHs on the performance of different hyper-heuristics, as well as their interaction with each other and the parameter values chosen by hyper-heuristics for those LLHs. This includes an investigation of the suitability of the LLH portfolio for similar project scheduling problems, including potentially additional operators to deal with new aspects of those problems.

We also plan to integrate our problem domain implementation into the existing real-world scheduling system, so that hyper-heuristics can be used in practice next to the existing solution methods.

ACKNOWLEDGMENTS

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

REFERENCES

[1] Steven Adriaensen, Tim Brys, and Ann Nowé. 2014. Fair-Share ILS: A Simple State-of-the-Art Iterated Local Search Hyperheuristic. In *Proceedings of the 2014*

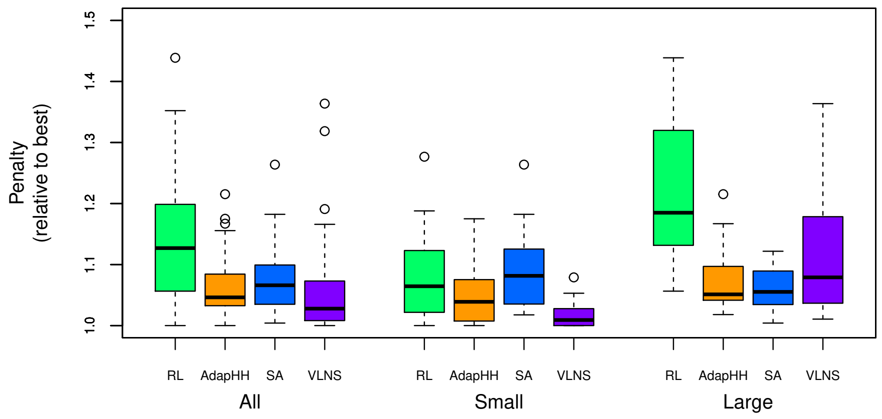

Figure 3: Comparison of results for the RL and AdapHH hyper-heuristics with those achieved by SA [17] and VLNS [10]. Results for each instance were scaled by the best result achieved on that instance. Results are shown separately for all instances, for small (≤ 20 projects) instances only, and for large (> 20 projects) instances only.

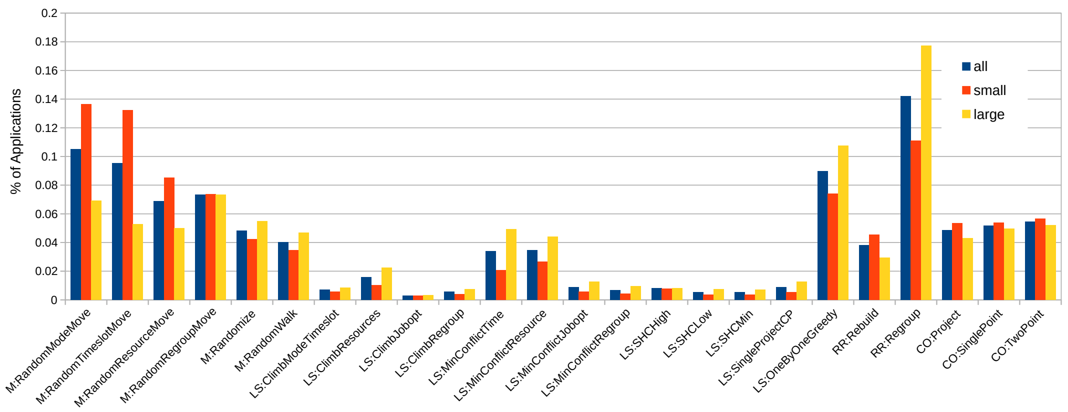

Figure 4: Relative usage of each LLH by AdapHH. Shown are average statistics over all instances, only small instances, or only large instances.

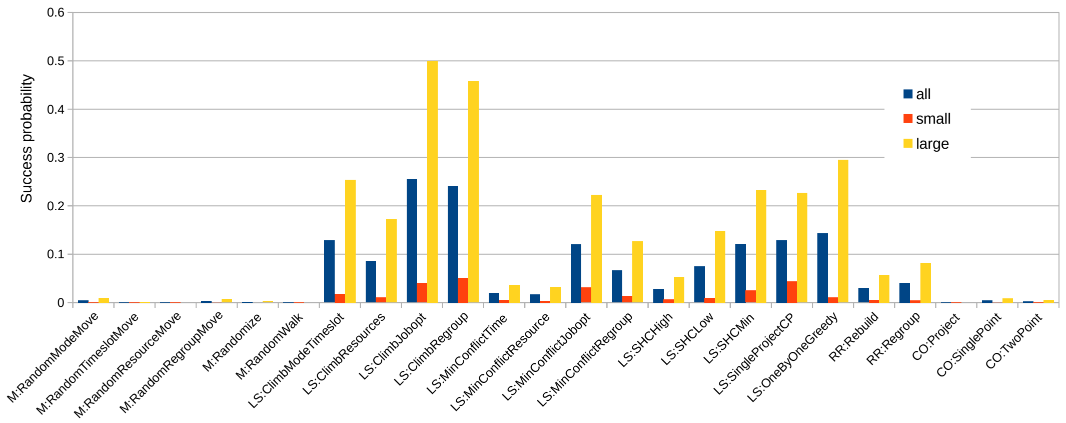

Figure 5: Success probability of each LLH by AdapHH. A successful application is defined as one that directly results in a new best solution. Shown are average statistics over all instances, only small instances, or only large instances.

Annual Conference on Genetic and Evolutionary Computation (Vancouver, BC, Canada) *(GECCO '14)*. Association for Computing Machinery, New York, NY, USA, 1303–1310. https://doi.org/10.1145/2576768.2598285

[2] Steven Adriaensen, Gabriela Ochoa, and Ann Nowé. 2015. A benchmark set extension and comparative study for the HyFlex framework. In *2015 IEEE Congress on Evolutionary Computation (CEC)*. 784–791. https://doi.org/10.1109/CEC.2015.7256971

[3] Stephen A. Adubi, Olufunke O. Oladipupo, and Oludayo O. Olugbara. 2021. 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)*. 1372–1379. https://doi.org/10.1109/CEC45853.2021.9504841

[4] Stephen A. Adubi, Olufunke O. Oladipupo, and Oludayo O. Olugbara. 2022. Evolutionary Algorithm-Based Iterated Local Search Hyper-Heuristic for Combinatorial Optimization Problems. *Algorithms* 15, 11 (2022). https://www.mdpi.com/1999-4893/15/11/405

[5] Edmund K. Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Barry McCollum, Gabriela Ochoa, Andrew J. Parkes, and Sanja Petrovic. 2011. The Cross-Domain Heuristic Search Challenge – An International Research Competition. In *Learning and Intelligent Optimization*, Carlos A. Coello Coello (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 631–634.

[6] Edmund K Burke, Matthew R Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R Woodward. 2019. A classification of hyper-heuristic approaches: revisited. In *Gendreau, M., Potvin, JY. (eds) Handbook of Metaheuristics.and International Series in Operations Research Management Science, vol 272*. Springer, Cham., 453–477. https://doi.org/10.1007/978-3-319-91086-4_14

[7] Shin Siang Choong, Li-Pei Wong, and Chee Peng Lim. 2018. Automatic design of hyper-heuristic based on reinforcement learning. *Information Sciences* 436-437 (2018), 89–107. https://doi.org/10.1016/j.ins.2018.01.005

[8] Geoffrey Chu. 2011. *Improving combinatorial optimization.* Ph. D. Dissertation. University of Melbourne, Australia. http://hdl.handle.net/11343/36679

[9] Chung-Yao Chuang. 2020. *Combining Multiple Heuristics: Studies on Neighborhood-base Heuristics and Sampling-based Heuristics.* Ph. D. Dissertation. Carnegie Mellon University.

[10] Philipp Danzinger, Tobias Geibinger, David Janneau, Florian Mischek, Nysret Musliu, and Christian Poschalko. 2022. A System for Automated Industrial Test Laboratory Scheduling. *ACM Transactions on Intelligent Systems and Technology* (2022). https://doi.org/10.1145/3546871

[11] Philipp Danzinger, Tobias Geibinger, Florian Mischek, and Nysret Musliu. 2020. Solving the Test Laboratory Scheduling Problem with Variable Task Grouping. In *International Conference on Planning and Scheduling (ICAPS) 2020.*

[12] Tobias Geibinger, Florian Mischek, and Nysret Musliu. 2019. Investigating Constraint Programming for Real World Industrial Test Laboratory Scheduling. In *Proceedings of the Sixteenth International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR 2019).*

[13] Tobias Geibinger, Florian Mischek, and Nysret Musliu. 2021. Constraint Logic Programming for Real-World Test Laboratory Scheduling. *Proceedings of the AAAI Conference on Artificial Intelligence* 35, 7 (May 2021), 6358–6366. https://ojs.aaai.org/index.php/AAAI/article/view/16789

[14] Florian Mischek and Nysret Musliu. 2018. A Local Search Framework for Industrial Test Laboratory Scheduling. In *Proceedings of the 12th International Conference on the Practice and Theory of Automated Timetabling (PATAT-2018), Vienna, Austria, August 28–31, 2018*. 465–467.

[15] Florian Mischek and Nysret Musliu. 2021. A local search framework for industrial test laboratory scheduling. *Annals of Operations Research* 302 (2021), 533–562. https://doi.org/10.1007/s10479-021-04007-1

[16] Florian Mischek and Nysret Musliu. 2022. Reinforcement Learning for Cross-Domain Hyper-Heuristics. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, Luc De Raedt (Ed.). ijcai.org, 4793–4799. https://doi.org/10.24963/ijcai.2022/664

[17] Florian Mischek, Nysret Musliu, and Andrea Schaerf. 2021. Local Search Approaches for the Test Laboratory Scheduling Problem with Variable Task Grouping. *Journal of Scheduling* (2021). https://doi.org/10.1007/s10951-021-00699-2

[18] Florian Mischek, Nysret Musliu, and Andrea Schaerf. 2022. Local Search Neighborhoods for Industrial Test Laboratory Scheduling with Flexible Grouping. In *Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling (PATAT-2022).*

[19] Mustafa Mısır, Katja Verbeeck, Patrick De Causmaecker, and Greet Vanden Berghe. 2012. An Intelligent Hyper-Heuristic Framework for CHeSC 2011. In *Learning and Intelligent Optimization*, Youssef Hamadi and Marc Schoenauer (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 461–466.

[20] Gabriela Ochoa, Matthew Hyde, Tim Curtois, Jose A. Vazquez-Rodriguez, James Walker, Michel Gendreau, Graham Kendall, Barry McCollum, Andrew J. Parkes, Sanja Petrovic, and Edmund K. Burke. 2012. HyFlex: A Benchmark Framework for Cross-Domain Heuristic Search. In *Evolutionary Computation in Combinatorial Optimization*, Jin-Kao Hao and Martin Middendorf (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 136–147.

[21] Nasser R. Sabar, Masri Ayob, Graham Kendall, and Rong Qu. 2015. Automatic Design of a Hyper-Heuristic Framework With Gene Expression Programming for Combinatorial Optimization Problems. *IEEE Transactions on Evolutionary Computation* 19, 3 (2015), 309–325. https://doi.org/10.1109/TEVC.2014.2319051

[22] Nasser R. Sabar and Graham Kendall. 2015. Population based Monte Carlo tree search hyper-heuristic for combinatorial optimization problems. *Information Sciences* 314 (2015), 225–239. https://doi.org/10.1016/j.ins.2014.10.045