Large-State Reinforcement Learning for Hyper-Heuristics

Lucas Kletzander, Nysret Musliu

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

Abstract

Hyper-heuristics are a domain-independent problem solving approach where the main task is to select effective chains of problem-specific low-level heuristics on the fly for an unseen instance. This task can be seen as a reinforcement learning problem, however, the information available to the hyper-heuristic is very limited, usually leading to very limited state representations. In this work, for the first time we use the trajectory of solution changes for a larger set of features for reinforcement learning in the novel hyper-heuristic LAST-RL (Large-State Reinforcement Learning). Further, we introduce a probability distribution for the exploration case in our epsilon-greedy policy that is based on the idea of Iterated Local Search to increase the chance to sample good chains of low-level heuristics. The benefit of the collaboration of our novel components is shown on the academic benchmark of the Cross Domain Heuristic Challenge 2011 consisting of six different problem domains. Our approach can provide state-of-the-art results on this benchmark where it outperforms recent hyper-heuristics based on reinforcement learning, and also demonstrates high performance on a benchmark of complex real-life personnel scheduling domains.

Introduction

Instead of providing problem-specific solution methods for different domains, hyper-heuristics aim to stay fully domain-independent by enforcing a strict domain barrier and only having access to a set of low-level heuristics (LLHs) to apply on a given instance. Often hyper-heuristics try to find certain combinations of LLHs that are beneficial to apply, e.g., a combination of a perturbation followed by a local search. However, a combination that is beneficial in the early part of the search might not work well in a later part of the search. On the other hand, the solution value and the execution time are the only information the hyper-heuristic gets back from the LLH.

Still, the trajectory of solution values can help to draw conclusions about the current state of the search. Therefore, we propose to use this information to learn efficient applications of low-level heuristics and introduce a new set of 15 features to characterize the current state of the search. Then we use these features for the novel hyper-heuristic LAST-RL (**La**rge **St**ate **R**einforcement **L**earning) based on reinforcement learning using the linear state-action value function approximation method tile coding, the learning method SARSA-lambda, and an epsilon-greedy policy that uses probability distributions based on Iterated Local Search for increased performance. It further uses solution chains based on the Luby sequence, where we introduce some novel improvements as well, and a restart strategy.

The method is then evaluated on instances from six different domains from the Cross Domain Heuristic Search Challenge 2011, where the contributions of the individual components are evaluated, and where LAST-RL can outperform previous approaches based on reinforcement learning, as well as on a benchmark of three complex real-life personnel scheduling domains, providing several improved results compared to both previous hyper-heuristics and domain-specific solution methods on two of these domains.

Related Work

The term hyper-heuristics was introduced in 2000 (Cowling, Kendall, and Soubeiga 2000), even though the general ideas were used even earlier. In the most common classification of hyper-heuristics (Burke et al. 2010, 2019), this paper introduces a perturbative selection hyper-heuristic, a class that perturbs existing solutions by selecting from a set of existing low-level heuristics.

A review was published in 2013 (Burke et al. 2013) and 2020 (Drake et al. 2020). Much work in the area is related to the Cross Domain Heuristic Search Challenge 2011 (Burke et al. 2011), where the framework HyFlex (Ochoa et al. 2012a) was introduced for a uniform implementation of different hyper-heuristics, featuring six different domains and low-level heuristics in four different categories, which are local search, mutation, ruin-and-recreate, and crossover. The challenge was won by Mısır et al. (2012) with a combination of adaptive mechanisms to manage a set of active low-level heuristics. A streamlined version of this algorithm was published by Adriaensen and Nowé (2016). The following places in the competition were taken by a self-adaptive variable neighbourhood search (Hsiao, Chiang, and Fu 2012), and an algorithm using cycles of diversification and intensification (Larose 2011). Extensions to the framework have been provided later (Ochoa et al. 2012b), including several additional domains (Adriaensen, Ochoa, and Nowé 2015).

Reinforcement learning (RL) is based on the notion of executing actions depending on the current state, making the system transition to a different state and returning a reward (Sutton and Barto 2018). Many hyper-heuristics employ some kind of learning. Reinforcement learning in the classical sense is used in the original competition by Larose (2011) based on work on multiple agents with individual learning (Meignan, Koukam, and Créput 2010), and by Di Gaspero and Urli (2012), where an action is to select the low-level heuristic type. The state-of-the-art approach by Choong, Wong, and Lim (2018) uses Q-learning to select a combination of an LLH selection method and a move acceptance method, together with a simple state aggregation. They report very good results (only beaten by the winner of the original competition), closely followed by the new approach (Mischek and Musliu 2022) using RL with a discrete state representation based on four features to learn the application of individual LLHs. Apart from HyFlex, deep RL has also recently been used for hyper-heuristics (Zhang et al. 2021). In contrast to the existing work, our approach uses a rich state representation not previously used in literature, and an exploration policy with adaptive probabilities based on Iterated Local Search.

A recent survey (Drake et al. 2020) provides an overview of current work, updated classification, and different application areas for hyper-heuristics. Newer approaches based on other concepts beside RL include a population-based approach using Gene Expression Programming (Sabar et al. 2014), a hyper-heuristic based on Monte-Carlo tree search (Sabar and Kendall 2015), a hyper-heuristic based on a hidden Markov model (Kheiri and Keedwell 2015), an iterated multi-stage hyper-heuristic (Kheiri and Özcan 2016), work using solution chains (Chuang 2020) based on the Luby sequence (Luby, Sinclair, and Zuckerman 1993), and Iterated Local Search with a learning component based on Adaptive Thomson Sampling (Adubi, Oladipupo, and Olugbara 2021).

Introducing LAST-RL

Reinforcement Learning is an iterative process based on the concept of states, actions, and rewards. The system is in a current state $\mathbf{s}$, in our case the search state. The learning agent chooses an action $a \in \mathbf{A}$ based on a notion of value for the current state and the expected state after executing this action. In our case, each low-level heuristic corresponds to an action. Then the learning agent L receives a reward r for the action a it chose and the system transfers to the new state $\mathbf{s}'$. The agent wants to maximize the rewards it collects and updates its value estimations based on the reward it got.

While the set of actions is clear, many decisions need to be made regarding the other aspects of a reinforcement learning system. This paper introduces the novel hyper-heuristic LAST-RL (**La**rge **St**ate **R**einforcement **L**earning), with the core learning loop as shown in Algorithm 1:

First, the problem domain is initialized for the given instance. The initial state is set up as well as the feature weights $\mathbf{w}$, which are initially set to 0. Hyper-heuristics can work on a single solution or on multiple solutions. In our approach, we mainly work on a single solution, but still need to keep several solutions in $\mathbf{S}$:

- The global best solution found so far for this instance
- The best solution found so far since the previous reset
- The current working solution
- A set of 5 solutions for use with the crossover LLHs is kept similar to GIHH (Misir et al. 2011). A random one is replaced each time a new best solution since the previous reset is found. Each crossover is then applied on the current working solution and a random solution from this crossover pool.

At the start the construction heuristic provided by the domain is executed 10 times, and the best result is set for all solutions in $\mathbf{S}$.

Algorithm 1: LAST-RL learning loop

```
Data: Instance I, action set A, and timeout T
initialize(I);
s ← initialState;
w ← 0⃗;
S ← initializeSolutions();
while time < T do
    c ← chain();
    for i ← 1 to c do
        x' ← x(s, A);
        a ← π(x', A, w);
        Δ, t ← executeLLH(a, S);
        r ← R(Δ, t);
        s' ← updateState(s, a, Δ, t);
        w ← L(s, a, r, s', w);
        s ← s';
        if newBest(S) then
            updateBest(S);
            break;
        end
    end
    gotoBest();
    if restart(S, time) then
        reset(s, S);
    end
end
```

Solution Chains

The main loop is executed until the timeout T is reached. We use episodes which end when either a new best solution since the last reset is found, or a certain number of heuristic applications have passed. After each episode, the working solution is set to the best solution found since the last reset. The decision how to choose the lengths of these chains, represented by the function $chain$, is a very critical one.

Chuang (2020) showed that the Luby sequence (Luby, Sinclair, and Zuckerman 1993) is optimal regarding the expected number of operations until an improvement is found, given that the probability $q(\ell)$ of finding an improvement

within at most ℓ heuristic applications is unknown.

$$\mathcal{L}(i) = \begin{cases} 2^{k-1} & \text{if } i = 2^k - 1 \\ \mathcal{L}(i - 2^{k-1} + 1) & \text{if } 2^{k-1} \leq i < 2^k - 1 \end{cases} \quad (1)$$

$$\mathcal{L} = 1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, 1, 1, 2, \ldots \quad (2)$$

Equation (1) shows the recursive definition of the Luby sequence, Equation (2) shows the first few values of the sequence. Most of the resulting numbers are very small, but with exponentially increasing distance to the start exponentially larger numbers occur in the sequences.

The good results (Chuang 2020) were further confirmed recently by Mischek and Musliu (2022), and indeed the sequence also showed very good results in our approach. However, there are two adaptations used in our work based on the fact that we sometimes do have some insight regarding the probability distribution $q(\ell)$.

The first novel adaptation is based on the fact that predominantly the hyper-heuristic will use cycles of diversification and intensification, as explained in the policy section later. Therefore, the solution value is expected to first increase, followed by a decrease using local search heuristics. This lead to the adaptation of line 7, continuing the inner loop beyond c if the previous heuristic was not a local search heuristic (still in the process of diversification), or the previous heuristic lead to an objective improvement (still in a promising phase of intensification). In any case, however, the chain is aborted if the length reaches $2 \cdot c$.

The second novel adaptation is to use information collected from previous chains, more precisely to use the average length $\bar{\ell}$ of successful chains (chains leading to a new best solution since the last reset). Therefore, c is actually obtained by multiplying the value returned from the Luby sequence with $\max\{\bar{\ell}/2;\ 1\}$ (the division prevents excessive growth), unless it is already larger than the longest successful chain. This approach allows to use longer chains for problems where longer chains are successful more often.

Reset Behaviour

At the end of each solution chain, the hyper-heuristic sets the active solution to the best solution found since the last reset. However, sometimes the search might not find any improvements any more for a longer period of time. Therefore, in line 21 a reset of all solutions except the global best is performed after either 1000 consecutive chains without improvement, or at least 100 consecutive chains without improvement and at least time of $rs \cdot T$ since the last reset, where rs is a parameter. The reason there are two options is the vast difference in runtime of the individual low-level heuristics between different domains. While the solutions are reset to a new initial solution, the weights for reinforcement learning are kept, potentially allowing much faster improvements after the reset.

In any case, for the last $2rs \cdot T$, the global best solution is set as the active solution in order to promote further intensification. If there would be a regular reset within the last $3rs \cdot T$, this change to the global best is started immediately.

Reinforcement Learning Components

In each inner loop in Algorithm 1, first the function approximation is calculated from the state $\mathbf{s}$ and the action set $\mathbf{A}$ by the function x. The resulting vector $\mathbf{x}'$ is then linearly weighted by the weight vector $\mathbf{w}$ and used to compute an approximate value for each combination of the current state $\mathbf{s}$ and an action $a \in \mathbf{A}$. The policy π uses these values to chose an action a to execute.

This execution results in a change Δ in the solution objective, and a used runtime t. These values are used to calculate a reward r by using the function R. Further the new state $\mathbf{s}'$ is calculated as the result of the action execution.

Finally the learning agent L uses the data obtained from the previous state $\mathbf{s}$, the action a, the reward r, and the new state $\mathbf{s}'$ to update the weight vector $\mathbf{w}$. The system moves to the new state, updates the solutions in case a new best solution since the last reset was obtained, otherwise the next loop continues.

Value Function Approximation. For the value function approximation we focused on linear value function approximations to try to balance learning and runtime considerations. It starts from the feature vector $\mathbf{s}$ of dimension 15, normalizing it according to the bounds for each feature, resulting in $\hat{\mathbf{s}}$. Then $\mathbf{x}$ of dimension d is calculated from $\hat{\mathbf{s}}$ based on the chosen approximation method. $\mathbf{x}'$ of dimension $d \cdot |\mathbf{A}|$ is calculated as a cross-product of the state features $\mathbf{x}$ and the action set $\mathbf{A}$.

Different linear approximations were considered for the transformation from $\hat{\mathbf{s}}$ to $\mathbf{x}$, resulting in the choice for tile coding. Here, the state space is partitioned into portions of equal size, called tiles. n_t such tilings, each with r_t tiles per dimension (resolution), and a different offset created with a random jitter, are used to create a representation of the state space. Tile coding produces one feature for each tile in each of the tilings. However, for each tiling, only one feature is active for any state (features are binary), and many tiles are never active during the whole learning process. This allows a sparse representation that only stores active features, and therefore to use many features while still staying very efficient.

Rewards. Once a certain action a has been chosen, the corresponding low-level heuristic is executed in a certain runtime t and results in a change in objective Δ. Since we use an episodic formulation, a reward is only created at the end of each solution chain.

The reward for a new best solution is set to $\sqrt{maxFC}$, where $maxFC$ is the maximum length of consecutive unsuccessful chains since the last reset. This value serves as a measure of difficulty to escape the current local optimum.

Since the total runtime for the hyper-heuristic is limited, wasting runtime is a negative aspect and should be penalized. However, putting too much weight on this aspect can easily lead to problems since every heuristic execution takes time, generating a negative reward, while even good combinations of heuristics only provide a positive reward with some probability.

For the academic benchmark we use, most LLHs on each domain show similar behaviour regarding runtime, therefore

no advantage could be found using runtime as part of the reward function. However, on the real-life domains differences are much larger, therefore adding runtime to the reward function was the only adaptation that was done to fit the hyper-heuristic better for the application on our real-life domains.

Based on experiments leading to the conclusions presented above, the following reward r was chosen to be applied at the end of each solution chain of length c' with total execution time t':

$$isNewBest = \begin{cases} 1 & \text{if new best solution found} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

$$r = isNewBest \cdot \sqrt{maxFC} - \frac{100 \cdot t'}{T \cdot c'} \tag{4}$$

Equation (3) distinguishes whether the chain ended due to a new best solution or reverting to the previous best, Equation (4) combines the positive reward for new solutions with the penalty for runtime, which is relative to the total runtime T and the chain length c'.

Learning Agent. To update the weight vector $\mathbf{w}$, SARSA-Lambda uses the policy to obtain the action to use for calculating the estimated value for the next state, and an eligibility vector $\mathbf{e}$ that accumulates the information of the previous steps of the learning agent for improved learning of longer sequences of actions that lead to a reward. The updates are done as follows:

$$a' = \pi(x(\mathbf{s}', \mathbf{A}), \mathbf{A}, \mathbf{w}) \tag{5}$$

$$\delta = r + \gamma \cdot \hat{q}(\mathbf{s}', a', \mathbf{w}) - \hat{q}(\mathbf{s}, a, \mathbf{w}) \tag{6}$$

$$\mathbf{e}' = \lambda \cdot \mathbf{e} + \nabla \hat{q}(\mathbf{s}, a, \mathbf{w}) \tag{7}$$

$$\mathbf{w}' = \mathbf{w} + \alpha\delta \cdot \mathbf{e}' \tag{8}$$

Equation (6) calculates δ using the next action obtained by the current policy in Equation (5) and the discount factor γ. Equation (7) updates the eligibility vector $\mathbf{e}$ using the decay factor λ. The update to $\mathbf{w}$ is done via $\mathbf{e}$ in Equation (8) with the learning rate α. This approach showed to be superior to Q-learning in preliminary experiments.

Search-state Features

The main issue of finding a suitable representation of the search state for hyper-heuristics is that not much information is available in the first place. The hyper-heuristic knows the number and type of low-level heuristics and the time limit. From each execution it gets the new solution value and the time the LLH took to execute.

However, the limited information that is available still allows the hyper-heuristic to know where in the search progress it is by the time and the trajectory of the objective values, and it allows to build a picture of the current state. Therefore, in total we introduce the following novel set of 15 features to characterize the search state. We present the definition as well as lower and upper bounds for normalization. Some of the following expressions use the following definition of the magnitude of a real value x based on the natural logarithm that works both for positive and negative values and transitions gracefully at $x = 0$:

$$\mathrm{mag}(x) = \mathrm{sgn}(x) \cdot \ln(1 + |x|) \tag{9}$$

- `lastHeur`: The index of the last heuristic that was applied. Range: Integer from -1 (first application or reset to best known solution) to $numHeur - 1$ (where $numHeur$ is the total number of available heuristics).
- `lastType`: The type of the last heuristic. Range: Integer from -1 (first application or reset to best known solution) to 3, as the four categories ruin-recreate, crossover, mutation, and local search are available.
- `lastChangeSign`: The sign of the last objective change. Range: Integer in $\{-1, 0, 1\}$, for a coarse distinction of the current direction of the objective.
- `lastChangeMag`: The magnitude of the change in solution value from the last application $\mathrm{mag}(\Delta)$. Range: $-\mathrm{mag}(initial)$ to $\mathrm{mag}(initial)$ (where $initial$ is the value of the initial solution). Ranges for solution values differ by orders of magnitude between different domains, resulting in the focus on the magnitude of the change and the normalization based on the initial solution value.
- `chainProgress`: The percentage of the current solution chain that has been processed so far. Range: $[0;\ 1]$.
- `lastImprMag`: The magnitude $\mathrm{mag}(hL)$ of the number of heuristic applications since the last improvement hL. Range: 0 to 5. Used as a measure of whether the current state is in a local optimum that is difficult to escape, can vary greatly.
- `stepsMag`: The magnitude $\mathrm{mag}(hT)$ of the total number of LLH applications since the last reset. Range: 0 to 5. Like for the previous feature, the actual values can also vary greatly between different domains.
- `time`: The total runtime since the last reset. Range: 0 to T. This feature measures time progress intended to help differentiate between early and later parts of the search.
- `relativeImprMag`: The magnitude of the relation between the objective of the current solution and the initial solution $\mathrm{mag}(cur/initial)$. Range: 0 to 2.
- `relativeBestMag`: The magnitude of the relation between the objectives of the current and best solution $\mathrm{mag}(cur/best)$ where $best$ is the objective value of the best solution since the last reset. Range: 0 to 2.
- `relImpr`: The relative number of improving heuristic applications hI/hT. Range: 0 to 1.
- `rel0`: The relative number of heuristic applications that did not change the objective $h0/hT$. Range: 0 to 1.

The previous features always capture either a current change from the last heuristic application or a collective value for the whole search since the last reset. In contrast, the last three features capture values only for a recent part of the search, allowing to put more focus on the recent history. These features each use the horizon $H = 10$ for the number of heuristic applications that are taken into account.

- `avgChangeHMag`: The magnitude of the average change in the last H heuristic applications $\mathrm{mag}(\bar{\Delta})$. Range: $-\mathrm{mag}(initial)$ to $\mathrm{mag}(initial)$.

- `relImprH`: The relative number of improvements in the last H heuristic applications. Range: 0 to 1.
- `rel0H`: The relative number of heuristic applications that did not change the objective in the last H heuristic applications. Range: 0 to 1.

In contrast to the other features, the first three features are integer. Therefore, we do not apply tile coding to these dimensions of the feature vector, but keep the discrete values. All other features are continuous and are normalized to the interval $[0;\ 1]$ according to their lower and upper bounds.

Exploration with Iterated Local Search

The policy π should favour actions with higher value for exploitation, but should also return all possible actions with a non-zero probability to allow exploration. A classical policy frequently used in reinforcement learning is epsilon-greedy. The action with the highest current state-action value is chosen with probability $1-\varepsilon$, while a random action is chosen with probability ε for a selected value of ε. However, this did not show to be sufficient to reliably find good chains of heuristics, as there are too many possibilities how to build chains of heuristics in relation to the available runtime. Given the set $\mathbf{A}$ and a chain of length c, there are $|\mathbf{A}|^c$ possibilities for heuristic chains. The domains from CHeSC 2011 have 8 to 15 LLHs, the real-life domains up to 22.

On the other hand, many of the best performing hyper-heuristics from literature are based on the ideas of Iterated Local Search (Lourenço, Martin, and Stützle 2003), where perturbation and intensification phases are repeated. This lead to the following novel improvement for reinforcement learning for hyper-heuristics: Instead of using uniform random selection with probability ε, in the exploration case the category of the next LLH is chosen according to probabilities based on ILS. Only within the chosen category the next LLH is chosen based on a uniform random distribution.

The next heuristic category is therefore chosen according to the following procedure: The initial category for the very first heuristic of a chain is chosen according to uniform random probabilities (if crossover would have been chosen, but is not available, ruin-recreate is selected instead). If the previous heuristic was of type ruin-and-recreate, the next one is either of the same category with probability p_r, or in any of the others with equal probability $\frac{1-p_r}{3}$. After crossover it is in the same category with probability p_c, or in the categories mutation or local search with equal probability $\frac{1-p_c}{2}$. After mutation it is in same category with probability p_m, or in the category local search with probability $1-p_m$, and after local search, the next one will be local search as well.

This way the categories have a fixed order going through large-scale perturbations first (ruin-and-recreate, then crossover), followed by mutations and finally local search until the end of the chain, or until a new best solution is found. While the order is fixed, every category except local search can be skipped.

The probabilities to stay in the same category p_r, p_c, and p_m are defined based on previous experience collected during the execution of the hyper-heuristic so far, and the fact that if a sequence is stopped at each element with probability p, the expected length of the sequence is $\frac{1}{p}$:

Param	Min	Max	Manual	Tuned	Mode
α	10^{-5}	0.1	0.001	$1.2 \cdot 10^{-5}$	log
γ	0.1	1	0.8	0.25	
λ	0	1	0.8	0.99	
ϵ	0.01	0.9	0.1	0.07	
rs	0.01	0.25	0.1	0.09	
n_t	1	20	10	4	int
r_t	1	10	3	6	int

Table 1: Parameter tuning

$$p_r = 1 - \frac{1}{1 + \frac{a_r \cdot c}{a_r + a_c + a_m + a_l}} \tag{10}$$

$$p_c = 1 - \frac{1}{1 + \frac{a_c \cdot c}{a_c + a_m + a_l}} \tag{11}$$

$$p_m = 1 - \frac{1}{1 + \frac{a_m \cdot c}{a_m + a_l}} \tag{12}$$

Equations (10), (11), and (12) show the definition based on the average number of heuristics of type ruin-and-recreate, crossover, mutation, and local search in successful chains represented by a_r, a_c, a_m, and a_l respectively. The reason for this computation is that different domains might need vastly different ratios of heuristic types.

Using this novel ILS-based selection for epsilon-greedy allows to greatly focus the hyper-heuristic on sequences of LLHs that are expected to provide good performance, since it greatly reduces the chance that a large amount of time is wasted on very unlikely combinations of LLHs.

Evaluation on the CHeSC Domains

The first evaluation was performed on the six domains from CHeSC 2011, the Cross Domain Heuristic Search Challenge 2011 (Burke et al. 2011) implemented in the HyFlex framework (Ochoa et al. 2012a), which are MaxSAT (SAT), Bin Packing (BP), Personnel Scheduling (PS), Flowshop (FS), Travelling Salesperson Problem (TSP), and Vehicle Routing Problem (VRP). The implementation of reinforcement learning is done using the BURLAP library version 3.0.1 (MacGlashan 2016).

The hyper-heuristic[1] was implemented in the Java HyFlex framework and run using OpenJDK 14.0.1. The experiments were run on a Windows system with an Intel 12th Gen i9-12900K with 3.19 GHz and 32 GB of RAM, each individual run was performed single-threaded. The competition benchmark script allowed 240 seconds for each run on our setup. For each of the domains, five instances are run 31 times each. Then, the median result for each instance is compared against the other hyper-heuristics using a formula-one-based system for distributing points: 10 points to the best hyper-heuristic, followed by 8, 6, 5, 4, 3, 2, 1 points for the following ranks. If different approaches provide the same result,

[1] Available at: https://gitlab.tuwien.ac.at/lucas.kletzander/last-rl

Version	Total		SAT		BP		PS		FS		TSP		VRP	
ILS-only	129.25	2	27.85	4	0	14	38.5	1	15.0	6	11.9	8	36	1
RL-only	137.40	2	0.00	15	17	5	36.0	1	30.5	3	17.9	4	36	1
Simple-state	142.10	2	28.20	4	14	5	26.0	3	20.0	5	17.9	4	36	1
LAST-RL	166.50	1	24.10	4	19	4	36.0	1	28.5	4	24.9	3	34	1

Table 2: Evaluation of LAST-RL versions in the original CHeSC 2011

Method	Total		SAT		BP		PS		FS		TSP		VRP	
LAST-RL	141.97	1	15.93	6	14	6	36.0	1	24.5	4	19.55	4	32	1
GIHH	141.72	2	25.18	3	40	1	7.0	10	30.0	1	31.55	1	8	10
QHH	123.97	3	37.43	1	10	7	0.0	16	28.0	3	27.55	3	21	3
RL	107.68	4	24.18	4	37	2	1.0	13	21.5	6	14.00	6	10	7
VNS-TW	94.22	5	27.18	2	0	16	33.0	2	23.5	5	6.55	11	4	13
ML	92.88	6	4.33	10	5	11	28.5	3	28.5	2	9.55	9	17	5

Table 3: Comparison of LAST-RL against other methods in the original CHeSC 2011

the points for the affected ranks are summed up and shared between the equal competitors.

Based on thorough initial experimentation, we validated our modifications for the solution chains and came up with a manually tuned set of parameters. Next we also used parameter tuning with SMAC 3 version 1.1.1 (Lindauer et al. 2021) allowing 1 million seconds using the additional CHeSC instances not used for the competition for training. Both sets of parameters are shown in Table 1. The tuned configuration performed slightly, but not significantly better than the manually tuned one on the CHeSC benchmark, but generalized worse to the real-life domains. First, the similar performance despite considerable difference in parameters shows that our method is robust in this regard and not dependent on a very narrow parameter setting. Second, the degrading performance on the real-life instances indicates overfitting on the training domains, therefore we continued using the manual configuration for maximum generality, using the same configuration for all domains. For maximum performance, domain-based tuning could be considered in the future.

Comparison of Different Versions

To highlight the contributions of the most important aspects of LAST-RL, which are rich state reinforcement learning and the ILS-based epsilon-greedy policy, we compare the following versions:

- ILS-only: Setting $\varepsilon = 1$, this version only uses the ILS heuristic type selection, ignoring reinforcement learning.
- RL-only: Using the default epsilon-greedy procedure, this version lacks the influence of the ILS-based heuristic type selection.
- Simple-state: The state representation is reduced to just the previous heuristic in order to investigate the effect of the rich state representation. The previous heuristic still allows to learn good chains of heuristics, but without taking into account the current position in the search space.
- LAST-RL: The full final version of the hyper-heuristic.

The following comparison is done evaluating each version individually against the 20 participants in the original CHeSC 2011. Table 2 shows the points achieved in total and in the individual domains, and the rank among the 21 hyper-heuristics. The results clearly show that the full version performs significantly better than the other versions.

Reinforcement learning is especially important for BP where a score of 0 is reached using just the ILS selection, but also for FS and TSP, while top results can be achieved for SAT, PS, and VRP without reinforcement learning. On the other hand, the results for RL-only show that without the ILS component, SAT ends up with a score of 0, and TSP is also still at a low score. Replacing the rich state representation with a simple state had a significant negative effect on four domains. Only SAT and VRP stay at the same levels as LAST-RL, while especially PS, FS, and TSP are affected.

Regarding the individual domains, all variants perform very well on the VRP domain, constantly achieving the first place in the competition. For most of the other domains, at least one of the versions lacking a core component performed at a similar level as LAST-RL. For TSP, however, only the combination of all core components was able to get good results, showing the benefit in the combination.

Comparison Against Other Methods

To compare against other methods, we add the two state-of-the-art hyper-heuristics based on reinforcement learning to the competition, which are QHH (Choong, Wong, and Lim 2018) and RL (Mischek and Musliu 2022), and present the top six results on now 23 competitors in Table 3. Both for the total and the individual domains the table shows points and rank. From the original competition, these include the winner GIHH (Misir et al. 2011), VNS-TW (Hsiao, Chiang, and Fu 2012), and an original entry based on reinforcement learning (Larose 2011).

The evaluation shows that our approach can reach the first place slightly before GIHH, and clearly ahead of the recent reinforcement learning methods QHH and RL. LAST-

Inst	SA	CH-PR	GIHH	L-GIHH	LAST-RL
10	**14717**	14806	14787	14774	14779.8
20	30861	30671	30732	30694	**30669.4**
30	50947	50904	**50766**	50854	50890.0
40	69120	68848	68640	68645	**68478.2**
50	87013	87034	86762	86730	**86681.8**
60	103968	103465	103139	103150	**102935.8**
70	122754	122026	121672	**121661**	121916.2
80	140482	139209	139123	**139042**	139250.2
90	156385	154972	155094	155113	**154915.0**
100	173524	**171182**	171278	171325	171589.4

Table 4: Best results for BSD with different solution methods

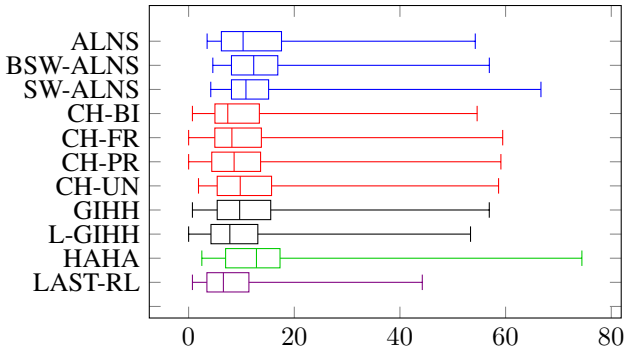

Figure 1: Best result deviations for full MSD

RL also achieves the best overall results in two of the domains. While the ranking for individual domains varies, our approach is never worse than rank 6, and never achieves less than 14 points in any domain, while no other competitor has a value of at least 10 in each domain. This highlights that LAST-RL is indeed very general and can adapt to many very different domains without problems.

Evaluation on Real-Life Scheduling Domains

Beside the academic benchmark CHeSC 2011, we also evaluate LAST-RL on several real-life personnel scheduling domains where recently results with hyper-heuristics were presented (Kletzander and Musliu 2022), and investigate how well the reinforcement learning approach transfers to these domains, which have more constraints, longer runtime, and more low-level heuristics with more differences in their runtimes up to a gap from milliseconds to more than three minutes for some calls to best improvement heuristics. The problem domains are implemented in Python and run using PyPy 7.3.5, the hyper-heuristics are run using OpenJDK 8u292. All evaluations were performed with one hour of runtime (same as the previous results) on a computing cluster running Ubuntu 16.04.1 LTS with Intel Xeon CPUs E5-2650 v4 (max. 2.90GHz, 12 physical cores, no hyperthreading), but each individual run was performed single-threaded. Each method was run on each instance 5 times.

The first domain in this comparison is Bus Driver Scheduling according to laws of an Austrian collective agreement (Kletzander and Musliu 2020). The benchmark data set includes 5 instances each for 10 size categories (10 to 100). Table 4 shows the average of the best results per instance category for Simulated Annealing, all hyper-heuristics that could achieve a best result in any size category from previous work, and the new hyper-heuristic LAST-RL. This comparison shows very strong results of LAST-RL on this difficult real-life domain, with a clear majority of 5 out of the 10 size categories being won by LAST-RL. It can further provide the best average objective for 16 of the instances, compared to 14 for L-GIHH (Adriaensen and Nowé 2016) and less then 10 for every other hyper-heuristic in comparison on this problem.

The second domain deals with optimization variants of Rotating Workforce Scheduling (Kletzander et al. 2019), where the goal is the assignment of shifts to employees according to rules for consecutive assignments. In case of this domain, LAST-RL could not provide improved results compared to GIHH and L-GIHH.

The third domain is Minimum Shift Design (Musliu, Schaerf, and Slany 2004), where new shifts need to be designed such that a given demand is covered as well as possible, the number of different shifts is kept to a minimum, and the average shift length is within given boundaries. LAST-RL was applied to both the reduced problem (objectives 1 to 3) and the full problem (objectives 1 to 4) on the realistic data sets 3 and 4 (33 instances). On the reduced problem LAST-RL can reach the second place after L-GIHH.

Figure 1 shows the deviations from a given lower bound for Minimum Shift Design with all 4 objectives in comparison to the other hyper-heuristics from previous work. In this comparison, LAST-RL can outperform all other hyper-heuristics, reaching the best values for lower quartile, median, upper quartile, and maximum.

Conclusion

This paper presented the new hyper-heuristic LAST-RL based on reinforcement learning using a novel large state representation. It combines this approach with an new exploration strategy using probabilities based on Iterated Local Search and expands the ideas using solution chains based on the Luby sequence.

The evaluation of the individual components shows that their strengths on different domains combine favourably to provide very good performance and generality of the overall method. The comparison against competitors in CHeSC 2011 as well as recent hyper-heuristics based on reinforcement learning shows that LAST-RL can outperform state-of-the-art RL-based hyper-heuristics. Further the evaluation on a recent real-life set of benchmark domains shows high performance on these more complex domains, and therefore high potential for further applications of LAST-RL to real-world problems.

In future work we want to further investigate components of LAST-RL, including a detailed analysis of the contributions of individual features and the exploration of different function approximation methods including deep reinforcement learning.

Acknowledgments

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

References

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

Adriaensen, S.; Ochoa, G.; and Nowé, A. 2015. A benchmark set extension and comparative study for the hyflex framework. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, 784–791. IEEE.

Adubi, S. A.; Oladipupo, O. O.; and Olugbara, O. O. 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. IEEE.

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

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

Burke, E. K.; Hyde, M.; Kendall, G.; Ochoa, G.; Özcan, E.; and Woodward, J. R. 2010. A classification of hyper-heuristic approaches. In *Handbook of metaheuristics*, 449–468. Springer.

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

Choong, S. S.; Wong, L.-P.; and Lim, C. P. 2018. Automatic design of hyper-heuristic based on reinforcement learning. *Information Sciences*, 436: 89–107.

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

Cowling, P.; Kendall, G.; and Soubeiga, E. 2000. A hyper-heuristic approach to scheduling a sales summit. In *International conference on the practice and theory of automated timetabling*, 176–190. Springer.

Di Gaspero, L.; and Urli, T. 2012. Evaluation of a family of reinforcement learning cross-domain optimization heuristics. In *International Conference on Learning and Intelligent Optimization*, 384–389. Springer.

Drake, J. H.; Kheiri, A.; Özcan, E.; and Burke, E. K. 2020. Recent advances in selection hyper-heuristics. *European Journal of Operational Research*, 285(2): 405–428.

Hsiao, P.-C.; Chiang, T.-C.; and Fu, L.-C. 2012. A vns-based hyper-heuristic with adaptive computational budget of local search. In *2012 IEEE congress on evolutionary computation*, 1–8. IEEE.

Kheiri, A.; and Keedwell, E. 2015. A sequence-based selection hyper-heuristic utilising a hidden Markov model. In *Proceedings of the 2015 annual conference on genetic and evolutionary computation*, 417–424.

Kheiri, A.; and Özcan, E. 2016. An iterated multi-stage selection hyper-heuristic. *European Journal of Operational Research*, 250(1): 77–90.

Kletzander, L.; and Musliu, N. 2020. Solving large real-life bus driver scheduling problems with complex break constraints. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, 421–429.

Kletzander, L.; and Musliu, N. 2022. Hyper-Heuristics for Personnel Scheduling Domains. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 32, 462–470.

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

Larose, M. 2011. A hyper-heuristic for the chesc 2011. *CHeSC2011 Competition*.

Lindauer, M.; Eggensperger, K.; Feurer, M.; Biedenkapp, A.; Deng, D.; Benjamins, C.; Sass, R.; and Hutter, F. 2021. SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization. arXiv:2109.09831.

Lourenço, H. R.; Martin, O. C.; and Stützle, T. 2003. Iterated local search. In *Handbook of metaheuristics*, 320–353. Springer.

Luby, M.; Sinclair, A.; and Zuckerman, D. 1993. Optimal speedup of Las Vegas algorithms. *Information Processing Letters*, 47(4): 173–180.

MacGlashan, J. 2016. Burlap: Brown-UMBC reinforcement learning and planning. http://burlap.cs.brown.edu/index.html. Accessed: 2022-01-26.

Meignan, D.; Koukam, A.; and Créput, J.-C. 2010. Coalition-based metaheuristic: a self-adaptive metaheuristic using reinforcement learning and mimetism. *Journal of Heuristics*, 16(6): 859–879.

Mischek, F.; and Musliu, N. 2022. Reinforcement Learning for Cross-Domain Hyper-Heuristics. In Raedt, L. D., ed., *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, 4793–4799. International Joint Conferences on Artificial Intelligence Organization. Main Track.

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

Mısır, M.; Verbeeck, K.; De Causmaecker, P.; and Berghe, G. V. 2012. An intelligent hyper-heuristic framework for chesc 2011. In *International Conference on Learning and Intelligent Optimization*, 461–466. Springer.

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

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

Ochoa, G.; Walker, J.; Hyde, M.; and Curtois, T. 2012b. Adaptive evolutionary algorithms and extensions to the hyflex hyper-heuristic framework. In *International Conference on Parallel Problem Solving from Nature*, 418–427. Springer.

Sabar, N. R.; Ayob, M.; Kendall, G.; and Qu, R. 2014. Automatic design of a hyper-heuristic framework with gene expression programming for combinatorial optimization problems. *IEEE Transactions on Evolutionary Computation*, 19(3): 309–325.

Sabar, N. R.; and Kendall, G. 2015. Population based Monte Carlo tree search hyper-heuristic for combinatorial optimization problems. *Information Sciences*, 314: 225–239.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Zhang, Y.; Bai, R.; Qu, R.; Tu, C.; and Jin, J. 2021. A deep reinforcement learning based hyper-heuristic for combinatorial optimisation with uncertainties. *European Journal of Operational Research*.