Reinforcement Learning for Cross-Domain Hyper-Heuristics

Florian Mischek , Nysret Musliu
Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling,
TU Wien
{florian.mischek, nysret.musliu}@tuwien.ac.at

Abstract

In this paper, we propose a new hyper-heuristic approach that uses reinforcement learning to automatically learn the selection of low-level heuristics across a wide range of problem domains. We provide a detailed analysis and evaluation of the algorithm components, including different ways to represent the hyper-heuristic state space and reset strategies to avoid unpromising areas of the solution space. Our methods have been evaluated using HyFlex, a well-known benchmarking framework for cross-domain hyper-heuristics, and compared with state-of-the-art approaches. The experimental evaluation shows that our reinforcement-learning based approach produces results that are competitive with the state-of-the-art, including the top participants of the Cross Domain Hyper-heuristic Search Competition 2011.

1 Introduction

Hyper-heuristics are an example of a high-level problem-independent approach that have been applied recently successfully to different problem domains (see recent surveys, e.g. [Burke *et al.*, 2019]). The aim of hyper-heuristics is to automate the design of heuristic methods based on a combination of low-level heuristics. Although various hyper-heuristics have been proposed over the last two decades, the Cross-Domain Heuristic Search Challenge (CHeSC) 2011 [Burke *et al.*, 2011] in hyper-heuristics intensified significantly the research in this area. This competition attracted considerable attention and various methods have been proposed including an approach that uses reinforcement learning. Although several other approaches have been proposed in recent years, the methods proposed in this competition still provide the best solutions for several problem domains. The reinforcement learning based hyper-heuristic approach [Di Gaspero and Urli, 2012] that was submitted to the CHeSC competition could not compete with the top ranked approaches and was outperformed by more than half of approaches that were submitted to the competition. Therefore, an interesting research question is if it is possible to develop reinforcement learning hyper-heuristic approaches that are competitive with the state-of the art approaches. Such an investigation is important, as reinforcement learning methods provide a general framework for hyper-heuristics and improving such solutions has an impact on several problem domains that include important NP-hard problems.

The main contribution of this paper are:

- We provide a new hyper-heuristic approach using reinforcement learning. Our approach includes several innovative ideas including different feature combinations to represent the state space and a reset mechanic combined with learning low-level heuristic weights to avoid getting stuck in unpromising areas of the search space.
- We deeply investigate various methods of reinforcement learning and give a systematic evaluation of these methods and their main components.
- Our methods are compared to the state of the art hyper-heuristics using the CHeSC competition rules. Our approach is placed in the top 3 hyper-heuristics and improves upon previous approaches that used reinforcement learning. This shows that reinforcement learning is a very well suited paradigm for hyper-heuristics.

2 Cross-Domain Optimization Hyper-Heuristics

For the CHeSC hyper-heuristic competition, researchers had to develop (selection) hyper-heuristics that perform well on a wide range of different problem domains. The problem domains were provided, including a set of low-level heuristics which are partitioned into four different types: *mutation*, *ruin and recreate*, *local search*, and *crossover* operators.

An important feature of the problem setting is the *domain barrier* (see Figure 1) that limits the information passed between hyper-heuristic and problem domain to ensure that the former cannot exploit domain-specific information. Hyper-heuristics can apply low-level heuristics to candidate solutions, both of which are referenced by indices only. In return, the problem domain provides the new objective function for that solution after the application.

The problem domains used for the CHeSC are the maximum satisfiability problem (MaxSAT), bin packing (BP), flow shop (FS), personnel scheduling (PS), the traveling salesman problem (TSP), and the vehicle routing problem (VRP). Of these, the latter two as well as some of the bench-

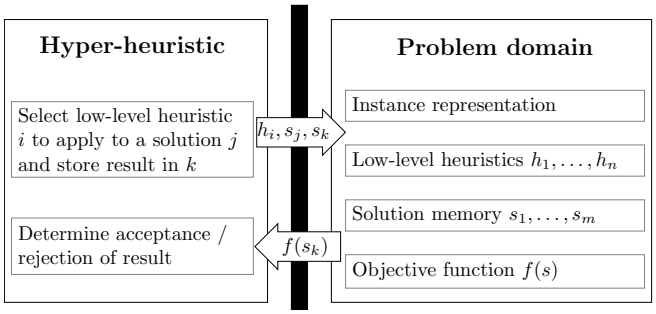

Figure 1: In HyFlex, the domain barrier limits the interchange of information between problem domains and hyper-heuristics to ensure domain-independence. Hyper-heuristics only know indices of low-level heuristics and candidate solutions and are provided with the objective function resulting from applying a low-level heuristic to a solution.

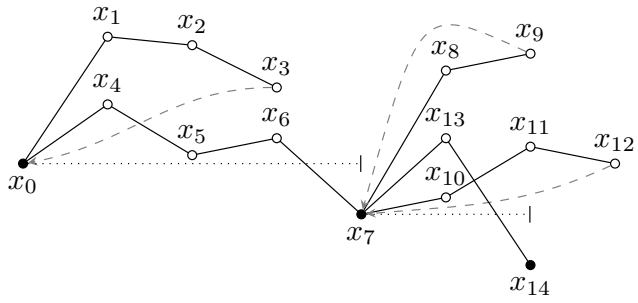

Figure 2: Example trajectory showing several solution chains. The vertical placement of a solution corresponds to its objective value, lower is better. Each chain ends either by finding a new best known solution (x_7, x_{14}) or a reset (after x_3, x_9, and x_{12}).

mark instances for the other four domains, were hidden from the participants before the competition.

The competition was won by [Mısır *et al.*, 2012], who combined several adaptive mechanisms to select low-level heuristics, determine acceptance of results and configure search parameters. The runner up [Hsiao *et al.*, 2012] used a self-adaptive Variable Neighborhood Search (VNS) with alternating shaking and local search phases. A similar cycle of diversification and intensification, plus an adaptive move acceptance criterion was also used by [Larose, 2011], based on an approach by [Meignan *et al.*, 2010]. Both Larose and Meignan et al. learn operator weights conditional on the previously applied operators, which are incremented each time a cycle finds an improved solution. In general, most approaches include one or several adaptive components, which change their behaviour in response to the progress of the search.

The competition entry by [Di Gaspero and Urli, 2012] containing a study of reinforcement learning approaches for this problem, achieved the 16th place. A revised and extended algorithm described in that paper and developed after the competition would have improved this result to the 13th place.

The hyper-heuristics developed for the CHeSC were implemented using the HyFlex framework [Ochoa *et al.*, 2012a]. Since then, HyFlex has become a popular framework for hyper-heuristics. Different authors have used it to develop and evaluate their own hyper-heuristics, such as a tensor-based hyper-heuristic [Asta and Özcan, 2015], a population-based Monte-Carlo tree search [Sabar and Kendall, 2015], and most recently an approach using solution chains similar to those used in this paper, though with no or only very limited learning [Chuang, 2020]. Other authors proposed extensions to the framework [Ochoa *et al.*, 2012b], or added additional problem domains [Adriaensen *et al.*, 2015].

3 Reinforcement Learning for Heuristic Selection

A typical view of the reinforcement learning (RL) setting contains several interlocking components: An *agent* interacts with an external *environment* by selecting and executing one of a set of available *actions*. It can then observe the new *state* of the environment and obtains a certain *reward*. The agent's goal is to maximize the total sum of rewards achieved. To achieve that goal, it can use the observations made to *update* its own beliefs about the environment. In our case, these beliefs are modeled as state-action values, estimates of the expected long-term reward achieved due to performing a certain action while the environment is in a certain state. These estimates are used by the agent's *policy* to select the next action.

This view translates naturally to the cross-domain optimization problem of the CHeSC, with the hyper-heuristic in place of the agent interacting with an environment in the form of a combination of a problem domain and a current candidate solution. The actions available to the agent are the low-level heuristics of the problem domain[1], which are applied to the current solution. For other components, we investigated several options, which are described later in this section.

One disadvantage of indiscriminately following this strategy is that most low-level heuristics are stochastic and will often result in worse solutions (in particular for mutations). While this might be beneficial and even necessary for escaping local optima and reaching new areas of the solution space, it can also lead towards unpromising areas from which it may be difficult to return. To avoid getting stuck in those areas, we decided to keep track of the best solution found so far during the search and periodically reset the current solution back to this best known solution. In effect, this splits the trajectory of the search into several chains of solutions, which either end with a new best known solution or a reset back to the starting point (see Figure 2).

A further modification we made was to use *amplification of low-level heuristics*, as they had a large impact on the solution quality in [Chuang, 2020] and also our preliminary experiments. Amplification is a technique that adds additional actions available to the hyper-heuristic. Each of these actions corresponds to repeated applications of a specific low-level heuristic for 10ms, where worse solutions are immediately rejected.

[1]For the purpose of this paper, we do not use the crossover operators as we only work on a single solution.

3.1 State Representation

An agent's internal representation of the environment's state should capture all relevant information necessary to make an informed decision about the next action. However, the domain barrier of HyFlex hides most of this information, so we necessarily have to use features of the search history, such as applied low-level heuristics and observed rewards, to represent our states. For our analysis, we chose to use a modular approach that allows us to study the impact of different feature combination. A state $s = (f_1, f_2, \ldots, f_n)$ is a tuple of n feature values $f_i \in F_i$, where F_i is the set of possible values for that feature. The whole state space is then $S = F_1 \times F_2 \times \cdots \times F_n$.

We have investigated the following features:

Last heuristic (LH). The index of the last heuristic applied (or -1 for the first heuristic in a chain).

Last heuristic type (LHT). As above, but using the type of the last heuristic.

Tail length (TLx). The number of heuristics remaining in the current chain before a reset. An intuitive example why this may be useful is that we probably do not want to apply mutations as the last heuristic in a chain, as these operators are unlikely to result in an immediate improvement and the result will likely be thrown out in the reset afterwards. To keep the cardinality of this feature low, all tail lengths equal to or greater than the parameter x are treated as equivalent.

Recent improvement (RIw). This binary feature captures long term performance by keeping track of whether an improvement was found within the last w chains. A long time without improvement may signal that a shift to more exploratory operators would be useful.

3.2 Action Selection Policy

The policy used to select the next action based on the current state-action values estimates has to serve a dual purpose of both maximizing rewards by selecting actions that have high estimates and improving the current estimates by setting up new observations for less frequently used heuristics.

A necessary condition for the state-action value estimates to converge to the true values is that all actions are eventually chosen infinitely often (assuming infinite run time). A popular class of policies that satisfy this condition are ε-policies, which select uniformly among all action with probability ε and follow another policy otherwise.

As the base policy we have investigated two different options:

ε-greedy. Selects the action with the highest current value with probability $(1 - \varepsilon)$.

ε-softmax. Performs roulette-wheel selection, where the weight of each action is equal to its state-action value.

A variant of these policies have ε decreasing over the course of the run, as the state-action value estimates get closer to their true values. This helps in maximizing the total rewards if the problem is mostly stationary, i.e. the true values do not change (a lot) over time.

3.3 Rewards

Given that we want to find a solution of minimum objective value, it would be natural to use the change in objective function after applying a low-level heuristic as the reward, as was done by [Di Gaspero and Urli, 2012]. However, this does not consider the fact that bad solutions will be rejected at the end of each chain, resulting in an objective value change of 0. Also, it would heavily penalize low-level heuristics like mutations which often lead to much worse solutions that however enable subsequent heuristics to find further improvements. For these reasons we decided to give out rewards only at the end of each solution chain, and a reward of 0 if the chain is reset without improvement. For the case that the the chain is successful (i.e. a new best known solution was found), we studied the following variants:

Constant reward (1)**.** We award a constant reward of 1 each time a chain is successful.

Total objective delta ($\sum \Delta o$). The reward is the difference in objective function between the previous and the new best known solution.

Delta per total time ($\frac{\sum \Delta o}{\sum \Delta t}$). This reward also accounts for the time taken to find the improvement by scaling the objective delta by the total time taken by all heuristics in the chain.

Delta per self time ($\frac{\sum \Delta o}{\Delta t_n}$). Alternatively, we can also consider only the time taken by the last heuristic in the chain (which is the one that found the improved solution), assuming that the setup by the other heuristics before was necessary and shorter setups would not necessarily have resulted in an improvement.

3.4 State-Action Value Update Rule

The estimates for the state-action values have to include not only rewards observed directly as a result of applying an action, but also future rewards that follow further down the line. There are several techniques how we can calculate these estimates and update them after making observations, to bring them closer towards their true values. For this paper, we have evaluated four different popular update rules, following definitions by [Sutton and Barto, 2018].

Given a set of observed rewards, their mean value is the maximum-likelihood estimate for the expected value of the actual reward distribution. **Monte-Carlo (MC) learning** accordingly calculates the state-action values $q(s, a)$ as the mean of all observed rewards from performing action a in state s, which are stored in a list called $Returns(s, a)$. Let the current episode with $T + 1$ steps be represented as a list of 3-tuples (s_t, a_t, r_t) denoting the state at step t, $0 \leq t \leq T$, the action taken, and the reward observed, respectively. For each step $0 \leq t \leq T$, we append $\gamma^{T-t} r_T$ to $Returns(s_t, a_t)$ and define $q(s_t, a_t)$ as $average(Returns(s_t, a_t))$.

$\gamma \in [0, 1]$ is a *discount factor*, to reflect the fact that more recent heuristics are likely to have a stronger influence on the result than those far earlier in the chain. In this paper, we use $\gamma = 0.8$, based on the results of preliminary experiments.

Instead of learning only at the end of each episode, we can already update our state-action values after each individual

action, by using our current estimates about the expected future rewards, which we already have represented by the state-action value of the successor state and action. Accordingly, this technique is called **SARSA** ("state-action-reward-state-action") and the update of $q(s_t, a_t)$ is performed after choosing $q(s_{t+1}, a_{t+1})$:

$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \alpha(r_t + \gamma q(s_{t+1}, a_{t+1}) - q(s_t, a_t)) \tag{1}$$

This makes use of the learning rate parameter α, which determines the magnitude of the update. α is applied to the difference (or estimation error) between the current estimate $q(s_t, a_t)$ and the sum of the observed reward r_t and the discounted expected future reward ($\gamma q(s_{t+1}, a_{t+1})$). In our evaluation, we have considered both constant learning rates and rates that decrease over time, either linearly or exponentially, to account for the fact that the more observations we have gathered, the less we want single recent observations to derail our presumably good estimates.

A close variant of SARSA is **Q-learning**. While SARSA uses the state-action value of the actually chosen subsequent action, Q-learning assumes that at least eventually the policy will converge towards greedy selection given that this policy will maximize rewards if the true state-action values are known. Following this assumption, Q-learning uses the state-action value of the (estimated) *best* action for the successor state to update the estimate of the current action. This leads to the following update function, applied after each action once the reward and successor state are observed:

$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \alpha(r_t + \gamma \max_a q(s_{t+1}, a) - q(s_t, a_t)) \tag{2}$$

Of course, in practice we do not use the greedy policy, as we need to continuously update our estimates for all state-action values in the limited time available. A hybrid approach between SARSA and Q-learning is **Expected-SARSA** (E-SARSA), which takes into account the probabilities $\pi(a|s)$ of choosing each action a under the current policy π when in state s. The expectation of the state-action values with respect to π is then used in the update function as follows:

$$E(t) = \sum_a \pi(a|s_{t+1}) q(s_{t+1}, a) \tag{3}$$

$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \alpha(r_t + \gamma E(t) - q(s_t, a_t)) \tag{4}$$

3.5 Solution Chain Length

The length of each solution chain, i.e. the number of heuristic applications without improvement before the chain is reset, is an important parameter. Determining the solution chain length involves a trade-off between sampling many short chains, which may miss improvements further down the chain, and few longer chains, which may waste large amounts of time if they do not result in an improvement.

In this paper, we assume that the length of each chain is known in advance and fixed at the start of the chain.

For a simple policy that selects uniformly among all low-level heuristics, [Chuang, 2020] prove that selecting the solution chain lengths according to *Luby's sequence* [Luby *et al.*, 1993] provides both some theoretical optimality guarantees and results in good performance in practice. This sequence is defined as follows:

$$\mathcal{L}[i] = \begin{cases} 2^{k-1} & \text{if } i = 2^k - 1 \\ \mathcal{L}[i - 2^{k-1} + 1] & \text{if } 2^{k-1} \leq i < 2^k - 1 \end{cases} \tag{5}$$

$$\mathcal{L} = (1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, 1, 1, 2, \dots) \tag{6}$$

Each time an improvement is found, the sequence should be restarted from its beginning. To avoid overly long chains that take up a considerable part of the overall available time, the sequence is also restarted after it reaches a value of 1024.

4 Evaluation

We implemented our approaches as hyper-heuristics in the HyFlex framework and evaluated them on the benchmark instances provided for the six original HyFlex problem domains. The source code for our implementation is available for download at https://gitlab.tuwien.ac.at/florian.mischek/hyper-heuristics-public.

The experiments were performed each using a single thread of a computing cluster with 10 nodes of 24 cores, an Intel(R) Xeon(R) CPU E5–2650 v4 @ 2.20GHz and 252 GB RAM. On this hardware, the benchmarking tool provided for the CHeSC determined that a time limit of 392s corresponds to the 10 minutes available on the original competition machine.

4.1 Component Analysis

To analyze the effects of each algorithm component on the performance, we compared the different variants to a baseline configuration that performed well in preliminary experiments. This baseline configuration uses a state representation consisting of the last heuristic and the tail length up to a maximum length of 5 (LH+TL5), a 0.2-greedy action selection policy, constant rewards for improved solutions, a learning rate of 0.1 where required, and Luby's sequence to determine chain lengths. We repeated all experiments with the four update rules (MC, Q-learning, SARSA, E-SARSA) and performed 10 runs per instance, using the 10 early instances for the four problem domains known in advance of the CHeSC (MaxSAT, BP, FS, and PS). To make objective values of different problem domains comparable, we normalized them to the range $[0 - 1]$ as follows [Di Gaspero and Urli, 2012]:

$$o_{norm} = \frac{o - o_{min}}{o_{max} - o_{min}} \tag{7}$$

o_{max} and o_{min} are the worst and the best objective value achieved for each instance, respectively.

First we compared the performance of the baseline configuration under the four update rules (Figure 3). Both MC learning and E-SARSA found results of comparable quality, slightly better than SARSA and Q-learning. Although the differences look small, due to this being normalized objective values they can actually translate to significant improvements for some problem domains - particularly under the CHeSC scoring system, which awards points for relative rank compared to the other competitors instead of absolute values.

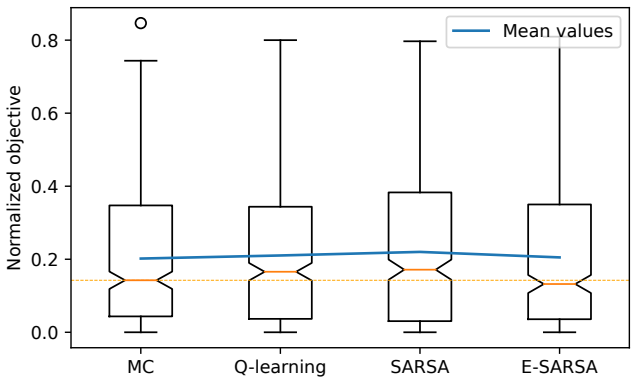

Figure 3: Normalized objective values for different state-action value update rules. The dotted line shows the median value of MC learning.

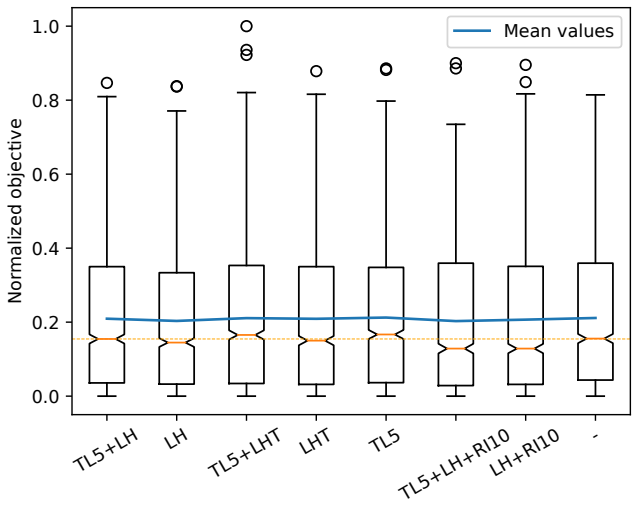

Figure 4: Normalized objective scores for state representations with different feature combinations: Tail length (TL5), last heuristic (LH), type of last heuristic (LHT), recent improvements (RI10). The dotted line shows the median of the baseline configuration (TL5+LH). The rightmost entry is a configuration that does not distinguish separate states at all.

For the remaining components, the effect of different configurations was mostly independent of the chosen update rule. For this reason, the figures and analysis presented below show combined results for all four update rules.

Figure 4 shows a comparison of different state representations. It can be seen that the choice of features does not result in drastic changes to the overall results. Surprisingly, even a configuration that does not distinguish between different states at all and learns a single value estimate for each low-level heuristic globally does not perform worse than the baseline configuration. However, the configurations using only the last heuristic applied (LH) or only the type of the last heuristic (LHT) perform slightly better than this.

Differentiating whether the hyper-heuristic found recent improvements with a window size of 10 (RI10) further improves the results by a small margin, both with and without tail length. This window size seems particularly well suited to model medium-term memory, as we did not find any similar improvement and sometimes even worse results with both longer and shorter window sizes. We also did an experiment comparing different cutoff-values for the maximum differentiated tail length, but found no significant effect on the results.

The effect of different action selection policies is shown in Figure 5. Both ε-greedy and ε-softmax produced similar results to each other, but a difference can be seen regarding the choice of ε. While a minimum value of ε is required to achieve good results, lower values tend to perform slightly better if ε remains constant over the course of the runtime. This is not the case for variants with decreasing ε, which show consistent performance over the whole value range.

Compared to other components, the choice of the solution chain length turned out to be crucial to achieve good results (see Figure 6). Both chains of constant length 1 and chains of infinite length (until an improvement is found or time runs out, no resets) result in particularly bad performance. While chains of longer constant lengths provide better results, following Luby's sequence turned out to be clearly better than any of the evaluated alternatives.

A small improvement can also be achieved by assigning rewards equal to the total objective delta divided by the time taken by the last heuristic in the chain (Figure 7). This follows the intuition that heuristics which find good solutions fast should be prioritized.

Finally, we also compared different learning rates for Q-learning, SARSA, and E-SARSA but found no significant differences between different values for α and no general trend towards higher or lower values.

4.2 Final Evaluation

The analysis of the previous section indicates that the following configuration is likely to perform for a hyper-heuristic: A state representation of LH+H10, with a 0.1-softmax policy with decreasing ε and rewards of $\frac{\sum \Delta o}{\sum \Delta t}$. The solution chain length follows Luby's sequence and state-action values are updated via MC learning.

In the following, we denote this configuration as RL. We evaluated our approach on all six domains contained in HyFlex, using the same set of benchmark instances that were used for the CHeSC. To compare the results with others, we used the same scoring scheme as for the competition, awarding points per instance according to the ranking following the Formula 1 system [Burke *et al.*, 2011]. If RL had been participated in the competition, it would have achieved second place (see Table 1), ahead of the approaches by [Hsiao *et al.*, 2012] and by [Larose, 2011].

We also compare with the more recent approach by [Chuang, 2020], who also used repeated solution chains following Luby's sequence. The thesis describes four different algorithms, which differ in the heuristic selection policy: *Uniform* selects uniformly among all low-level heuristics. The other three do the same in a warm-up period. Afterwards, *Pruning* removes low-level heuristics which were not successful at all, *Frequency* subsequently selects low-level-

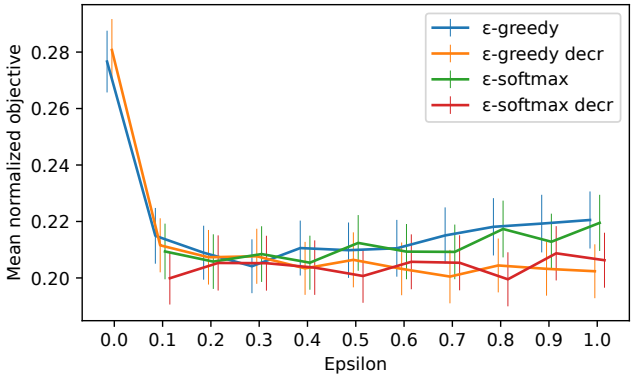

Figure 5: Mean normalized objective value for different policies at different values of ε. Softmax with $\varepsilon = 0$ led to numerical stability issues on some instances and is thus not included. The thin vertical lines denote the 95% confidence interval of the mean.

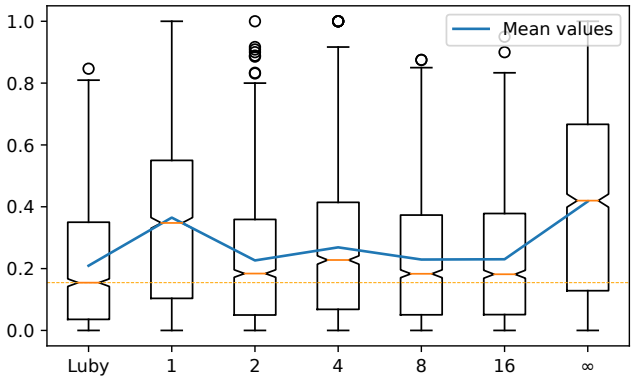

Figure 6: Normalized objective scores for different solution chain lengths. The dotted line shows the median of the baseline configuration (Luby).

heuristics according to their frequency of success, and *Bigram* does the same also considering the previously applied heuristic. Note that the three latter policies perform only a single learning step at the end of the warm-up period and do not change afterwards. Unfortunately, the source code of the algorithms was not available, so we re-implemented them according to the description in [Chuang, 2020] and evaluated them on our own machine. The results can be seen in Table 2. Clearly, while solution chains with resets in cases of no improvement are an important factor for the success of the approach (see previous section), RL is able to improve upon

Rank	Method	Reference	Score
1	AdapHH	[Mısır *et al.*, 2012]	170.10
2	**RL**		126.60
3	VNS-TW	[Hsiao *et al.*, 2012]	125.10
4	ML	[Larose, 2011]	118.50

Table 1: Top 4 approaches if RL had been a CHeSC participant.

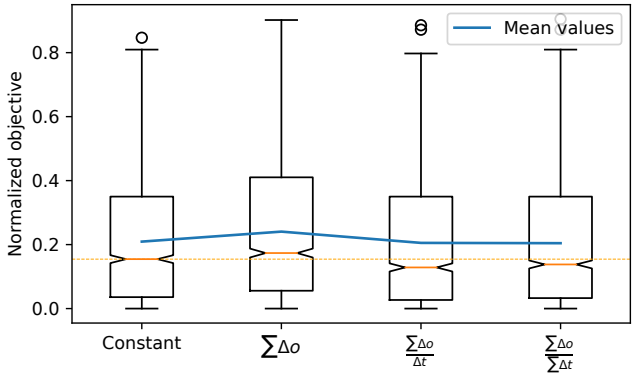

Figure 7: Normalized objective scores for different reward schemes. The dotted line shows the median of the baseline configuration (Constant).

Method	Rank	Score
RL	2	126.60
Uniform	6	80.10
Pruning	4	96.00
Frequency	6	81.20
Bigram	5	84.95

Table 2: Comparison of results for RL with the re-implementation of algorithms from [Chuang, 2020]. Shown are the rank and total score the respective algorithm would have achieved if it had participated in the CHeSC.

simpler action selection policies with no or limited learning.

5 Conclusions

In this paper, we have described a new hyper-heuristic approach for cross-domain optimization that models the solver as a reinforcement learning agent. The hyper-heuristic learns selection weights for the low-level heuristics and features strategic resets to avoid getting stuck in regions of the search space that do not yield good solutions. We have compared several different variants of the algorithm components and empirically shown that good choices for the representation of the state space, the reward model, the decision policy and most importantly the length of solution chains between resets allow reinforcement learning to produce high-quality solutions. The experimental evaluation shows that our approach would be placed among the top participants of the CHeSC hyper-heuristic competition and improves upon both earlier work using reinforcement learning and an approach using a similar reset mechanic but with no or limited learning.

For the future, it would be interesting to investigate the function approximation approach for reinforcement learning. Additionally, it would be conceivable to define a similarity metric for different states and distribute the observed rewards also to similar states for more robust results.

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 *et al.*, 2015] Steven Adriaensen, Gabriela Ochoa, and Ann Nowé. A benchmark set extension and comparative study for the hyflex framework. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 784–791, 2015.

[Asta and Özcan, 2015] Shahriar Asta and Ender Özcan. A tensor-based selection hyper-heuristic for cross-domain heuristic search. *Information Sciences*, 299:412–432, 2015.

[Burke *et al.*, 2011] Edmund K. Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Barry McCollum, Gabriela Ochoa, Andrew J. Parkes, and Sanja Petrovic. The cross-domain heuristic search challenge – an international research competition. In Carlos A. Coello Coello, editor, *Learning and Intelligent Optimization*, pages 631–634, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[Burke *et al.*, 2019] Edmund K. Burke, Matthew R. Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R. Woodward. *A Classification of Hyper-Heuristic Approaches: Revisited*, pages 453–477. Springer International Publishing, Cham, 2019.

[Chuang, 2020] Chung-Yao Chuang. *Combining Multiple Heuristics: Studies on Neighborhood-base Heuristics and Sampling-based Heuristics*. PhD thesis, Carnegie Mellon University, 2020.

[Di Gaspero and Urli, 2012] Luca Di Gaspero and Tommaso Urli. Evaluation of a family of reinforcement learning cross-domain optimization heuristics. In Youssef Hamadi and Marc Schoenauer, editors, *Learning and Intelligent Optimization*, pages 384–389, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[Hsiao *et al.*, 2012] Ping-Che Hsiao, Tsung-Che Chiang, and Li-Chen Fu. A vns-based hyper-heuristic with adaptive computational budget of local search. In *2012 IEEE Congress on Evolutionary Computation*, pages 1–8, 2012.

[Larose, 2011] Mathieu Larose. A hyper-heuristic for the chesc 2011. Report submitted for the CHeSC competition, 2011.

[Luby *et al.*, 1993] Michael Luby, Alistair Sinclair, and David Zuckerman. Optimal speedup of las vegas algorithms. *Information Processing Letters*, 47(4):173–180, 1993.

[Meignan *et al.*, 2010] David Meignan, Abderrafiaa Koukam, and Jean-Charles Créput. Coalition-based metaheuristic: a self-adaptive metaheuristic using reinforcement learning and mimetism. *Journal of Heuristics*, 16(6):859–879, 2010.

[Mısır *et al.*, 2012] Mustafa Mısır, Katja Verbeeck, Patrick De Causmaecker, and Greet Vanden Berghe. An intelligent hyper-heuristic framework for chesc 2011. In Youssef Hamadi and Marc Schoenauer, editors, *Learning and Intelligent Optimization*, pages 461–466, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[Ochoa *et al.*, 2012a] 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. Hyflex: A benchmark framework for cross-domain heuristic search. In Jin-Kao Hao and Martin Middendorf, editors, *Evolutionary Computation in Combinatorial Optimization*, pages 136–147, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[Ochoa *et al.*, 2012b] Gabriela Ochoa, James Walker, Matthew Hyde, and Tim Curtois. Adaptive evolutionary algorithms and extensions to the hyflex hyper-heuristic framework. In *International Conference on Parallel Problem Solving from Nature*, pages 418–427. Springer, 2012.

[Sabar and Kendall, 2015] Nasser R. Sabar and Graham Kendall. Population based monte carlo tree search hyper-heuristic for combinatorial optimization problems. *Information Sciences*, 314:225–239, 2015.

[Sutton and Barto, 2018] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.