TU WIEN Informatics

Cross-domain Selection Hyper-heuristics with Deep Reinforcement Learning

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Data Science

eingereicht von

Hannes Mayrhofer, BSc
Matrikelnummer 11922556

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Associate Prof. Dipl.-Ing. Dr.techn. Nysret Musliu
Mitwirkung: Univ.Ass. Dipl.-Ing. Dr.techn. Lucas Kletzander

Wien, 28. November 2024

Hannes Mayrhofer	Nysret Musliu

TU WIEN Informatics

Cross-domain Selection Hyper-heuristics with Deep Reinforcement Learning

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Data Science

by

Hannes Mayrhofer, BSc
Registration Number 11922556

to the Faculty of Informatics

at the TU Wien

Advisor: Associate Prof. Dipl.-Ing. Dr.techn. Nysret Musliu
Assistance: Univ.Ass. Dipl.-Ing. Dr.techn. Lucas Kletzander

Vienna, November 28, 2024

Hannes Mayrhofer | Nysret Musliu

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Hannes Mayrhofer, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, haben ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT- Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 28. November 2024

Hannes Mayrhofer

Acknowledgements

I want to thank my supervisor, Associate Prof. Dipl.-Ing. Dr.techn. Nysret Musliu, who supported me throughout my thesis with valuable feedback. I am grateful that he gave me the opportunity to get an impression of everyday life as a researcher at the Christian Doppler Laboratory, where I was able to experience a collegial atmosphere. I would like to thank my co-supervisor, Univ.Ass. Dipl.-Ing. Dr.techn. Lucas Kletzander, from whom I received immediate and precise assistance in all situations.

I also want to express my gratitude for my family and, in particular, my parents, who supported me and enabled me to go to university.

Special thanks go to my study group. With Daniela Böhm and Paul Czapka, I had an unforgettable time both during and outside of the studies. I can consider myself lucky that I have had the opportunity to go all the way together and to have found very good friends in Daniela and Paul.

The financial support by the Austrian Federal Ministry of Labour and Economy, the National Foundation for Research, Technology and Development and the Christian Doppler Research Association is gratefully acknowledged.

Kurzfassung

Während problemspezifische Heuristiken effektive Techniken zur Lösung kombinatorischer Optimierungsprobleme darstellen, bleibt die Entwicklung adaptiver Methoden, die über verschiedene Problemdomänen, einschließlich neuer und unbekannter Domänen, hinweg generalisieren, eine Herausforderung. Hyperheuristiken wählen untergeordnete Heuristiken mit nur sehr begrenzten Informationen über die aktuelle Lösung und die Problemdomäne iterativ aus. In diesem Kontext kann die Auswahlkomponente der Hyperheuristik als Reinforcement Learning Problem aufgefasst werden, für das bereits Methoden existieren. Es besteht jedoch ein Bedarf an weiterer Forschung, einschließlich der Integration von Deep Reinforcement Learning, das bereits Potenzial für ähnliche Aufgaben gezeigt hat.

Basierend auf der existierenden Hyperheuristik LAST-RL, die Reinforcement Learning verwendet, untersuchen wir die Auswirkungen neuer und bestehender Features des Zustands der heuristischen Suche und schlagen andere Erweiterungen vor, darunter einen erweiterten Aktionsraum und Anpassungen bei der Belohnungsfunktion. Wir entwickeln zwei Hyperheuristiken, die Deep Reinforcement Learning verwenden: eine verwendet Deep-Q-Learning, die andere Proximal Policy Optimization. Die Entwicklung liefert Einblicke in wesentliche Komponenten der Hyperheuristik, eine domänenübergreifende Strategie für die Hyperparameteroptimierung zeigt ihre Wirksamkeit, und die Auswertung auf einer Benchmarksammlung von Problemdomänen (HyFlex) hebt die Stärken und Schwächen der vorgeschlagenen Ansätze hervor. Zusätzlich werten wir die Fähigkeit unserer Ansätze, auf drei komplexen realen Personaleinsatzplanungsproblemen zu generalisieren, aus.

Die vorgeschlagenen Erweiterungen für LAST-RL ermöglichen Verbesserungen auf der Benchmarksammlung HyFlex. In den Problemdomänen, die Potenzial für weitere Verbesserungen zeigten, darunter MaxSAT und Bin Packing, zeigt sich, dass die Erweiterungen effektiv sind. Obwohl die beiden Methoden, die Deep Reinforcement Learning nutzen, auf der Benchmarksammlung von Problemdomänen nur teilweise konkurrenzfähig sind, zeigt die Hyperheuristik, die den Deep Q-Learning Algorithmus verwendet, ihre Fähigkeit, für die unbekannten realen Personaleinsatzplanungsprobleme zu generalisieren. Darüber hinaus integrieren wir eine Progressive Neural Network Architektur, die Transfer Learning ermöglicht, was durch eine bessere Leistung der Hyperheuristik nach dem Training dieser Architektur auf Trainingsinstanzen bestätigt wird. Weiters beobachten wir, dass mit mehr Trainingsepochen die Fähigkeit der Hyperheuristik für Transfer Learning auf unbekannten Probleminstanzen weiter verbessert wird.

Abstract

While domain-specific heuristics are effective techniques for solving combinatorial optimization problems, constructing adaptive methodologies that generalize across multiple problem domains, including new and unseen domains, remains a challenge. Hyper-heuristics are higher-level methodologies designed to address this challenge by iteratively selecting low-level heuristics with very limited information about the current solution and the problem domain. In this setting, the selection component of the hyper-heuristic can be seen as a reinforcement learning task, for which several approaches already exist. However, there is a need for further advancements, including the integration of deep reinforcement learning, which has already shown potential for similar tasks.

Based on the existing reinforcement learning hyper-heuristic LAST-RL, we investigate the impact of new and existing features of the heuristic search state and propose further enhancements, including an extended action space and modifications to the reward function. We design two deep reinforcement learning hyper-heuristics that both build upon the existing LAST-RL hyper-heuristic: the first one uses the deep Q-learning algorithm and the second one uses the Proximal Policy Optimization algorithm. The design process offers insights into the impact of crucial design choices, a custom hyperparameter tuning strategy across domains demonstrates its effectiveness, and the evaluation on the hyper-heuristics benchmark framework HyFlex highlights the strengths and weaknesses of the proposed approaches. Additionally, we test the ability of our approaches to generalize to three complex real-life personnel scheduling domains.

The proposed enhancements for LAST-RL enable improvements on the HyFlex benchmark suite. In the problem domains that showed potential for further improvements, including MaxSAT and Bin Packing, the enhanced version turns out to be effective. Although the two methods utilizing deep reinforcement learning are only partially competitive on the HyFlex benchmark suite, the hyper-heuristic that utilizes deep Q-learning shows its ability to generalize to unseen real-life personnel scheduling domains. Additionally, we integrate a progressive neural network architecture, which demonstrates its ability to transfer knowledge, as pre-training this architecture on training instances leads to better performance compared to an initial model with this architecture. We observe that increasing the number of pre-training epochs further enhances the hyper-heuristics' ability to transfer knowledge for the heuristic search on unseen problem instances.

Contents

CHAPTER 1

Introduction

Difficult real-world optimization problems require computationally efficient techniques to obtain good solutions in short time. These techniques, or algorithms, are called heuristics. They are designed to efficiently explore the solution space with the goal of obtaining a solution close to the optimum. In many use cases, domain knowledge is utilized to develop tailor-made heuristics for a specific problem. However, domain knowledge is often not available or too expensive. Therefore, significant work has been done to develop "cross-domain" selection hyper-heuristics [Drake et al., 2020].

Hyper-heuristics are algorithms that operate at a higher level than heuristics. They select which heuristic, or sequence of heuristics, should be employed without relying directly on domain knowledge [Burke et al., 2013]. These heuristics are then called low-level heuristics. There is ongoing research to enhance the generalizability, ensuring that the performance is reasonably good for all instances across all available domains and unseen domains, incorporating the "cross-domain" concept. Here, an instance refers to a specific configuration or example of an optimization problem, with its variables, constraints, and objective function, and a problem domain refers to a specific type of optimization problem.

The hyper-heuristics framework HyFlex [Ochoa et al., 2012], implemented at the University of Nottingham, offers several problem domains to compare new approaches with state-of-the-art algorithms. HyFlex was originally implemented for the Cross-Domain Heuristic Search Challenge 2011 (CHeSC 2011), which was an academic competition that intensified research in the field of selection hyper-heuristics.

The existing approach we will use as a starting point is Large-State Reinforcement Learning (LAST-RL) [Kletzander and Musliu, 2023], a reinforcement learning hyper-heuristic, with features that describe the most recent solution changes in a chain of selected low-level heuristics. It was the first approach that used an extensive set of 15 features for the heuristic search state. However, this search state remains under-explored.

Additionally, it uses the default set of low-level heuristics for its action space without varying available search parameters.

Therefore, we see potential for improvement in existing hyper-heuristics that use reinforcement learning methods, but also other reinforcement learning algorithms, including deep reinforcement learning, might be able to provide advancements in the field of selection hyper-heuristics. There exist approaches in the literature that use deep reinforcement learning for hyper-heuristics, often focusing on single problem domains, as seen in [Zhang et al., 2022], [Tu et al., 2023], and [Udomkasemsub et al., 2023]. Some of them also provide evaluation across multiple domains but to the best of our knowledge, only the approach by [Dantas and Pozo, 2021] reports results on the HyFlex domains, whereas the other approaches uitilize domain-specific knowledge which is not permitted in the setting of the HyFlex benchmark suite.

The LAST-RL hyper-heuristic is based on online-learning, i.e. it starts from an initial model and updates it using the feedback received from the environment. As outlined in [Burke et al., 2013], there are other techniques utilized for the hyper-heuristic search process. There are also offline-learning hyper-heuristics that are pre-trained on training instances to learn the strategy and deploy it on unseen instances. Mixed-learning is a similar concept which allows the offline-trained hyper-heuristic to adapt to unseen instances of the same or a new domain, with model updates during the search.

A mixed-learning concept for a deep reinforcement learning based hyper-heuristic might have the potential to transfer knowledge to the search on unseen instances and problem domains. Techniques in lifelong reinforcement learning could contribute to this promising approach.

1.1 Aim of the thesis

The main aims of this thesis are:

- To propose enhancements for the existing LAST-RL hyper-heuristic by
  - exploring the search state and the effect of feature groups,
  - exploring the extension of the action space for the hyper-heuristic.
- To develop hyper-heuristics that combine LAST-RL with Deep Reinforcement Learning by
  - determining the model design and configuration that yields the best results across domains,
  - exploring the potential of a pre-trained hyper-heuristic.

1.2 Contributions of the thesis

The main contributions of this thesis include:

- We propose new features for the existing LAST-RL hyper-heuristic. An ablation study for feature selection determines a set of features, including new proposed features, for the search state that improves the performance on the HyFlex benchmark suite.
- The extension of the action space by the variation of two search parameters shows partial improvement on the HyFlex benchmark suite.
- We design two deep reinforcement learning hyper-heuristics, DS-RL and maPPOs, both based on LAST-RL. The design process, which is confronted with key challenges of deep reinforcement learning, gives insights into how crucial design choices affect the performance.
- We integrated the progressive neural network architecture into the DS-RL hyper-heuristic which enables the pre-trained hyper-heuristic to transfer knowledge when being deployed for unseen problem domains or instances.
- We propose a new reward function design which results in improvement for the scores of DS-RL on the HyFlex benchmark.
- We evaluate the enhancements for LAST-RL and the two deep reinforcement learning hyper-heuristics on the HyFlex benchmark suite and compare with state of the art. Additionally, we compare our approaches with state of the art on three real-life personnel scheduling domains.

1.3 Structure of the thesis

The thesis is structured as follows:

After this Introduction, Chapter 2 gives the required background information covering selection hyper-heuristics, the HyFlex benchmark suite, and deep reinforcement learning. This is followed by a detailed review of the state-of-the-art cross-domain selection hyper-heuristics.
In Chapter 3, enhancements to the existing LAST-RL algorithm are motivated and proposed, after a discussion of its algorithmic components.
In Chapter 4, two deep reinforcement learning hyper-heuristics are designed, design choices and the ability to pre-train the model via techniques from lifelong reinforcement learning are described.
The experiments for the proposed enhancements and hyper-heuristics are conducted in Chapter 5, with final tests to compare against state of the art on the HyFlex benchmark suite and on three real-life domains.
Finally, the conclusion will summarize our contributions and outline future work.

CHAPTER 2

Background and state of the art

This chapter will give the required background information for reinforcement learning and how reinforcement learning can be utilized for hyper-heuristic search strategies (Section 2.1), followed by an introduction into the field of selection hyper-heuristics with important terminology (Section 2.2). Then the benchmark suite HyFlex, which will accompany the work for both development and evaluation purposes, is presented (Section 2.3), followed by the state-of-the-art approaches for selection hyper-heuristics (Section 2.4) with a focus on hyper-heuristics that are used across domains and utilize reinforcement learning techniques. The last section in this chapter displays the terminology and notation that is used throughout this thesis (Section 2.5).

2.1 Reinforcement Learning

This section that covers the background information for reinforcement learning and the terminology that we use is based on [Sutton and Barto, 2018].

Reinforcement learning is one of the three main pillars in machine learning, besides supervised and unsupervised learning. In contrast to supervised learning where the feedback is used to instruct giving correct actions, it evaluates the feedback that is generated in an environment by actions taken. This feedback is referred to as the reward and the main goal is to maximize the sum of future (discounted) rewards.

Reinforcement learning is based on modelling the interaction of an agent in an environment via Markov Decision Processes (MDP). A MDP consists of:

- environment's state s_t in the state space S at a discrete timestep t,
- action a_t to select by the agent in action space $\mathcal{A}$,
- transition probability $P(s_{t+1} = s' | s_t = s, a_t = a)$ for all $s', s \in S$ and $a \in \mathcal{A}$,

- scalar reward signal $r_t \in \mathbb{R}$,
- discount factor $\gamma \in [0, 1]$ to discount later rewards and force the agent to gain immediate rewards.

The sequential nature of MDPs result in a trajectory of the following form:

$$s_0, a_0, r_1, s_1, a_1, r_2, a_2, r_3, \dots$$

The dynamics of the MDP are then modelled by the transition probability $P(s_{t+1} \mid s_t, a_t)$. A key assumption in MDPs is the Markov property which says that future states do not depend on the prior history but only on the recently observed state s_t and action a_t.

Definition 2.1.1 (Markov property)**.** A state s_{t+1} fulfills the Markov property if and only if

$$P(s_{t+1} = s', r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, \dots, r_1, s_0, a_0) = P(s_{t+1} = s', r_{t+1} = r \mid s_t, a_t)$$

for all s', r and the prior dynamics $s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, \dots, r_1, s_0, a_0$.

The Markov property is violated in many scenarios where the state is only partially observable leading to the fact that the agent's state is not equal to the true state in the environment. Therefore the agent only indirectly observes the environment. This leads to partially observable MDPs where usually the Markov property is still assumed. Methods that do not update their value estimates on the basis of the value estimates of successor states do not suffer so much from this violation.

The agent's goal in reinforcement learning is to maximize the expected discounted return.

Definition 2.1.2 (Discounted return)**.** The discounted return G_t is the sum of the discounted future rewards. $\gamma \in [0, 1]$ is the discount rate and r_t is the reward received at time t.

$$G_t = r_{t+1} + \gamma r_{t+2} + ... + \gamma^{T-(t+1)} r_T = \sum_{k=t+1}^{T} \gamma^{k-(t+1)} r_k$$

Episodes terminate at terminal states at time T (for continuing tasks $T = \infty$).

The agent's decisions are based on a policy π which pursues the goal to maximize the expected discounted return, resulting in an optimal policy π^* .

The state-value function $v_\pi(s)$ and the action-value function $q_\pi(s, a)$ are used to evaluate the quality of actions and states.

$$v_\pi(s) = \mathbb{E}_\pi[G_t | s_t = s]$$
$$q_\pi(s, a) = \mathbb{E}_\pi[G_t | s_t = s, a_t = a]$$

They have both a fundamental relationship between the values of a state s and its subsequent state s', the Bellman optimality equation. For the state-value function, the Bellman optimality equation states:

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_\pi(s')]$$

This allows incremental updates with the feedback that is received while exploring the environment. The methods that are used and discussed in this work are dealing with the approximate value functions $V(s)$ and $Q(s, a)$, respectively, and parameterized forms of the value functions $Q_\theta(s, a)$. We will shortly introduce the reinforcement learning algorithms that are used throughout the work:

- SARSA(λ)

  The first method is the "action-value-extension" for temporal difference methods called SARSA(λ). It uses the TD error δ_t for action-values:

  $$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

  It is the error of the estimate at the time t but actually can be calculated only retrospectively because s_{t+1} is not yet available at time t.

  Furthermore it is an on-policy method, i.e. it learns and improves the policy that is also used at the same time. The method makes use of eligibility traces e which are effective in delayed reward settings. The trace decay rate parameter λ controls the importance of earlier transitions for the eligibility trace.

  The model updates happen right after each step which makes the method very sample efficient but it uses the recent state for updates and therefore suffers when the Markov property is violated. The value function is approximated with linear function approximation and an efficient algorithm for binary encoded state features is in [Sutton and Barto, 2018]. Later, we will go through the model updates and how SARSA(λ) is used for the hyper-heuristic LAST-RL.

  As already mentioned, it is an on-policy method where the updates depend on the action that is selected with the current policy, whereas Q-learning uses the maximum Q-value for the updates (regardless of the actual selected action) and is therefore an off-policy method. In the following, a deep reinforcement learning method which is based on the idea of Q-learning is introduced.

- Deep Q-learning algorithm

  Deep Q-networks use artificial neural networks to approximate the action value functions. In contrast to tabular Q-learning, the deep Q-learning algorithm can handle continuous states. The parameterized function that is learned ($Q_\theta(a, s) \approx Q(a, s)$) maps the current state s_t to a scalar value for each action.

Transitions $\tau_t = (s_t, a_t, r_t, s_{t+1})$ are stored and updates happen in an off-policy way with a quadratic loss function. More details and the enhancements for the classical deep Q-learning algorithm by [Mnih et al., 2013] are motivated and explained when the DS-RL hyper-heuristic is being introduced in Section 4.1.

- Proximal Policy Optimization (PPO)

  Proximal Policy Optimization, proposed in [Schulman et al., 2017], is a policy gradient method that directly learns the policy distribution $\pi(a \mid s, \theta)$. It makes use of the advantages of preceding policy gradient methods like Trust Region Policy Optimization (TRPO), see [Schulman et al., 2015a], and also claims to be more general and sample efficient.

  The method in [Schulman et al., 2017] is an actor-critic-style algorithm where we learn, besides the policy network π_θ, also a value network V_ψ. It can be useful to have shared parameters between actor and critic, and a shared loss function $J^{\text{clip}}(\theta, \psi)$ enables joint updates. For details, we refer again to later sections where we integrate PPO in our hyper-heuristic (Section 4.2).

Apart from the algorithms, there are other important components in reinforcement learning: the action selection policy with the trade-off between exploration and exploitation, and the reward function design. They will be covered later when we propose the methods that are used.

2.2 Selection hyper-heuristics

First, we give a concise quote that summarizes the idea of hyper-heuristics in general:

> *The term "hyper-heuristic" can be defined as a high-level automated search methodology which explores a search space of low-level heuristics (neighbourhood or move operators, or metaheuristics) or heuristic components, to solve computationally hard problems.* [Drake et al., 2020]

The hyper-heuristics literature distinguishes between two classes of hyper-heuristics: selection and generation hyper-heuristics. The goal of selection hyper-heuristics is to control a set of low-level heuristics by iteratively choosing and applying them on the current solution, and deciding if the modified solutions are accepted or rejected. So they consist of two main components, which are the selection and acceptance-rejection mechanisms [Drake et al., 2020].

Whereas generation hyper-heuristics explore and search in a different search space. They explore the components that are used to construct the heuristics, rather than selecting predefined low-level heuristics as it is done by selection hyper-heuristics. Both selection and generation hyper-heuristics output a solution at the end of their search but generation

hyper-heuristics also yield automatically constructed heuristics that can potentially be applied to new problems [Burke et al., 2013].

This work focuses on selection hyper-heuristics where [Burke et al., 2010] was responsible for an extended classification and terminology used in recent publications. In the remainder of the work, we will use the terms hyper-heuristics and selection hyper-heuristics synonymously, meaning both will refer to selection hyper-heuristics. There is another layer of how the methodologies discussed in this work can be narrowed down: all low-level heuristics that are available to be selected are perturbation operators and not construction operators, i.e. the starting point of the search is a constructed solution. Therefore, an even more precise classification is selection perturbative hyper-heuristics [Pillay and Qu, 2018].

As already motivated in the introduction (Chapter 1), heuristics usually rely on problem-specific knowledge and are designed for a specific type of problem and therefore often only work well in other problem domains with significant modifications. This is the main motivation for generalizable methods that work well independently from the problem domain. What can lead to improved performance across problem domains is, among others, the knowledge that at certain stages of the search some low-level operators tend to perform better than others. For example, in case of stagnation when a local optimum has been reached, the selection of low-level heuristics that diversify the search can be beneficial. In contrast, if the hyper-heuristic has little remaining time until timeout, the search could use the remaining time to intensify the current solution, i.e. enforcing late improvements with local search operators. Additionally, combinations of low-level heuristics have the potential to produce better solutions by applying a specific sequence of these operators, e.g. by switching between diversification and intensification stages. These are some intuitive examples and our aim is that self-adaptive methods that we will later use, can adapt to these phenomena, detect patterns that are less intuitive, and act accordingly with their selection policy.

As the goal is to have reliable hyper-heuristics that work well across domains and only rely on the search itself, the domain barrier has been introduced. This concept prevents that only the new observed objective function value is sent as feedback from the problem domain to the hyper-heuristic. Other information with respect to the solution space, unsatisfied constraints or problem specific information, like the number of variables, are not available.

In Figure 2.1, the feedback loop for selection hyper-heuristics is illustrated and it shows the domain barrier. Here, only the selection component is shown and the acceptance-rejection mechanism is not mentioned. The decision if a new solution is accepted or not is usually carried out after the new solution has been evaluated.

The domain barrier has been implemented in the HyFlex framework [Ochoa et al., 2012] which also comes with predefined sets of low-level heuristics for each of the provided problem domains. HyFlex enabled researchers to focus on the development of the (self-adaptive) search methodologies rather than having to implement the problem domains

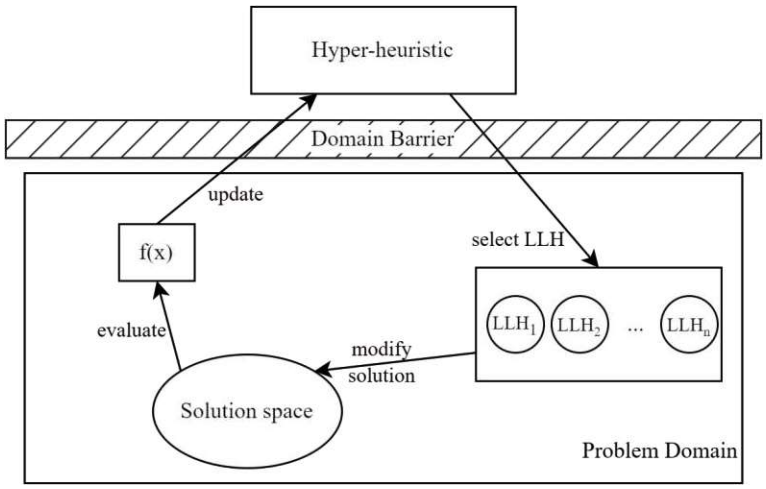

Figure 2.1: Feedback loop of selection hyper-heuristic.
Illustration of the two interacting layers: Hyper-heuristic and Problem Domain. This illustration is based on Figure 1 in [Burke et al., 2011].

and low-level heuristics themselves.
The following section will give more detailed insights into HyFlex and the available problem domains within this framework.

2.3 HyFlex

In [Ochoa et al., 2012], they proposed the Hyper-heuristics Flexible Framework (HyFlex) which is a Java class library that allows flexible implementation and evaluation of iterative heuristic search algorithms. HyFlex has been given attention as an academic benchmark for selection hyper-heuristics for Combinatorial Optimisation problems. Its modularity allows reliable comparison of the performance of a range of adaptive techniques.

The HyFlex Framwork was implemented by the Automated Scheduling, Optimisation and Planning (ASAP) research group of the University of Nottingham, UK and published in August 2010 for the Cross-Domain Heuristic Search Challenge (CHeSC 2011). The CHeSC 2011 was organised by ASAP and took place between August 2010 and June 2011 [Burke et al., 2011].

As outlined in the survey by Drake [Drake et al., 2020], a second version of the HyFlex framework was published with additional functionality. A batched mode treats multiple instances as a batch and allows to assign different execution times for each instance in order to put more focus on more difficult instances. Another novelty is the relaxation of the strict domain barrier, which enables to pass other information than only the fitness to the hyper-heuristic. These novelties were also announced in [Asta et al., 2013]. They also claim that a second Cross-Domain Heuristic Search Challenge (CHeSC 2014) was hosted. To our knowledge, there are no further sources about the second version of

HyFlex and the CHeSC 2014. In this work, we will only cover the setting of the first version of HyFlex and CHeSC 2011.

Although HyFlex is not maintained anymore, the framework is still available and used for teaching, training and research purposes. Relevant information was published by the researchers of ASAP [Ochoa et al., 2012], technical reports for the problem domains, i.e. MaxSAT [Hyde et al., 2010a], Bin Packing [Hyde et al., 2010b], permutation Flow Shop [Hyde et al., 2010c], Personnel Scheduling [Hyde et al., 2010d], and Vehicle Routing Problem [Walker et al., 2012] were released (to our knowledge, there is no report for the Traveling Salesman Problem). Surveys [Burke et al., 2013, Ross, 2014, Tauritz and Woodward, 2016, Drake et al., 2020] reflect the main ideas of the hyper-heuristic algorithms that were submitted in the CHeSC 2011 and of later approaches until state-of-the-art techniques.

As outlined in [Drake et al., 2020], the main feature of the HyFlex Framework is that it enables researches to focus on the selection algorithm for general purpose without the requirement of deep knowledge of the problem domains. The domain barrier in the HyFlex Framework strictly splits the hyper-heuristic algorithm from the problem domain. Consequently, problem specific configurations that are usually imposed by domain experts are implemented in HyFlex. These configurations are the solution representation which is not accessible for the hyper-heuristic, the initialisation routine which is generally a fast randomized constructive heuristic (see [Ochoa et al., 2012]), the objective function (or fitness) and available low-level heuristics.

[Ross, 2014] claims that more domain-specific information would be necessary for real-world applications. However the aim of HyFlex is to put focus on the high-level search methods that generalize well in "cross-domain" tasks, only having information about the search itself.

HyFlex covers six domains, where two of them were hidden at the CHeSC 2011 for evaluation to test the algorithms for generalizability to new problem domains. The four "open" domains for development and training are:

- Maximum Satisfiability (**MaxSAT**),
- one-dimensional **Bin Packing**,
- permutation **Flow Shop**,
- **Personnel Scheduling**.

The two hidden domains that were used for evaluation in the CHeSC 2011 are:

- Traveling Salesman Problem (**TSP**),
- capacitated Vehicle Routing Problem (**VRP**).

In the work of [Adriaensen et al., 2015], three new domains were added to the HyFlex benchmark set, which is the first known public extension since the release of HyFlex for the CHeSC 2011.

- 0-1 **Knap Sack** Problem,
- Quadratic Assignment Problem (**QAP**),
- **Max-Cut** Problem.

For each of these domains, a custom set of low-level heuristics were provided for hyper-heuristic selection. Each problem specific low-level heuristic can be classified into one of the following four groups:

- mutation

  Mutational operators are introduced to diversify solutions and to enable exploration of the whole search space. The mutational operators impose random perturbations or modify variables based on certain criteria, e.g. variables that break clauses or variables with highest age (i.e. variables that have been part of the solution for the longest time without having been altered). These operations usually lead to a worse solution but can be crucial in the search process.
  A prominent example - that is also used in HyFlex - is the bit-flip operator, which changes a binary variable (converts 0 to 1, or vice versa). For more details, see [Burke and Kendall, 2014].

- crossover

  Crossover methods (or recombination operators) choose two parent solutions and combine them to obtain a new solution. Components of the new solution may be selected randomly from the parents or based on characteristics of the variables to preserve desirable attributes of the parents.

- ruin-recreate

  The ruin and recreate principle, first introduced in [Schrimpf et al., 2000], removes part of the solution and recreates or rebuilds it, e.g. by random reinitialization. This simple principle is the key component in the Adaptive Large Neighborhood Search (ALNS) subsequently developed [Ropke and Pisinger, 2006].

- local search (hill climbing)

  Local search is an iterative procedure that produces a new solution with better or at least the same objective value as the original solution. In HyFlex for the MaxSAT domain, the local search operators iteratively apply bit-flip (see description of mutation above) to create neighbors of the original solution until a new solution has improved or equal fitness, or until a termination criterion is reached ([Hyde et al., 2010a]).

	Problem domain					
LLH	MaxSAT	Bin Packing	Flow Shop	Personnel	TSP	VRP
mutation	6 (α)	3 (α)	5 (α)	1 (α)	5 (α)	3 (α)
crossover	2	1	4	3 (α)	4	2
ruin-recreate	1 (α)	2 (α)	2 (α, β)	3 (α)	1 (α)	2 (α)
local search	2 (β)	2 (β)	4 (β)	5 (β)	3 (β)	3 (β)

Table 2.1: LLHs and hyperparameters of each problem domain.
Table shows the number of LLHs per problem domain and the available hyperparameters α or β for at least one of those LLHs. The first entry in the table refers to 6 mutational LLHs for the domain MaxSAT with at least one of them that uses the "depth of search" hyperparameter α.

Low-level heuristics are used to modify solutions and enable the hyper-heuristic to operate indirectly on the solution space. The first three low-level heuristic groups can be summarized as diversification strategies, aimed at exploring the search space widely to discover new promising solutions, while the heuristics of the last group can be characterized as an intensification strategy, which focuses on exploiting the current neighborhood and improving existing solutions.
Two hyperparameters that can be adapted by the selection hyper-heuristic at each step are the "intensity of mutation" $\alpha \in [0, 1]$ and "depth of search" $\beta \in [0, 1]$. α changes the behavior of mutation, crossover and ruin-recreate methods and β is responsible for the intensity of the local search. In the case of MaxSAT, α sets the proportion of the variables that are randomly reinitialised or modified for ruin-recreate and mutation, and β controls the number of iterations of the local search heuristic [Hyde et al., 2010a].

Some of the low-level heuristics used in HyFlex are designed specifically for the problem domains. For additional details, we refer to the technical reports.

2.3.1 Problem Domains

For each of the six problem domains that were used for the CHeSC 2011, we provide a description of the problem, specify the objective function, which is minimized in all cases, and give a short description of the solution initialisation method.

Maximum Satisfiability (MaxSAT)

The Maximum Satisfiability problem is closely related to the boolean satisfiability problem (SAT), where the goal is to assign values to boolean variables x to ensure a logical formula that contains these variables evaluates to true. For the MaxSAT problem, we consider the logical formula in conjunctive normal form and maximize the number of satisfied clauses c of the formula.
In formula 2.1 we give an example:

$$\underbrace{(x_1 \vee \neg x_2 \vee \neg x_3)}_{=c_1} \wedge \underbrace{(\neg x_1 \vee \neg x_3 \vee x_4)}_{=c_2} \wedge \quad \ldots \quad \wedge \underbrace{(x_2 \vee \neg x_3 \vee \neg x_4)}_{=c_n} \tag{2.1}$$

In HyFlex we only consider minimisation problems, so the objective is to minimise the number of clauses that evaluate to false:

$$\min\left\{\sum_{c\in C}(1-z_c)\right\}, \qquad \text{where } z_c = \begin{cases}1, & \text{if clause } c \text{ is satisfied}\\ 0, & \text{otherwise}\end{cases}$$

Initial solutions are produced by randomly assigning values to the boolean variables ([Hyde et al., 2010a]).

One-dimensional Bin Packing

The One-dimensional Bin Packing problem is dealing with the assignment x_{ij} of pieces j with weight w_j into bins i with capacity C. In HyFlex, a fitness function that aims to "*avoid large plateaus in the search space around the best solutions*" ([Hyde et al., 2010b]) is used.

$$\min\left\{1-\frac{1}{n}\sum_{i=1}^{n}\left(\frac{\sum_j w_j x_{ij}}{C}\right)^2\right\}$$

Here, n is the number of bins and the expression $\frac{\sum_j w_j x_{ij}}{C}$ indicates how full bin i is. Higher "fulness" is rewarded by applying the square function.

A constructive heuristic initializes the solutions by packing the pieces one after another into the first bin that they fit into, after shuffling the order of the pieces ([Hyde et al., 2010b]).

Permutation Flow Shop

The Permutation Flow Shop problem assigns n jobs j to be processed successively on m machines i with a processing time of p_{ij}. Each job is processed in the same order on the machines. The time when the job i is processed on the last machine m is denoted by $C_{i,m}$. The goal is to find a permutation π of the jobs so that the completion time of the last job $\pi(n)$ on the last machine m is minimal:

$$\min\left\{\max_i C_{i,m}\right\} = \min\left\{C_{\pi(n),m}\right\}$$

The solutions are initialised with the NEH algorithm. The technical report of the Permuation Flow Shop domain [Hyde et al., 2010c] refers to [Nawaz et al., 1983] for more details.

Personnel Scheduling

Personnel Scheduling problems can be very diverse, because they contain various constraints, objectives and weights (to prioritize objectives in multi-objective problems).

However, the HyFlex developers came up with a data file format that handles a wide range of variations of Personnel Scheduling instances.
Nurse rostering, a famous subproblem of Personnel Scheduling, is the problem of finding the assignment x_{ijk} of nurses i to shifts k on day j with two groups of objectives: shift coverage and employees' satisfaction. The overall objective is to minimize the weighted sum of the objectives $\bar{g}(x)$.

$$\min \left\{ \sum_i w_i \bar{g}_i(x) \right\}$$

Note that in the HyFlex framework, all constraints of Personnel Scheduling problems are modelled as objectives with very high weights. Solutions are initialised with one of the local search heuristics of this problem domain (*"new" shifts hill climber*, see [Hyde et al., 2010d]).

Traveling Salesman Problem (TSP)

The Traveling Salesman Problem describes the problem of a salesman who has to visit n cities with the objective to minimize the total tour length. The binary variable $x_{ij} = 1$ if the tour goes from city i to j, where cities i and j are at a distance of c_{ij} from each other.

$$\min \left\{ \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} c_{ij} x_{ij} \right\}$$

Initial solutions are produced with random permutation, see [Ochoa et al., 2012].

Capacitated Vehicle Routing Problem (VRP)

The Capacitated Vehicle Routing Problem is the problem of minimizing the number of required vehicles n_v to fulfill the demands of customers and satisfy additional constraints like vehicle capacity. Each vehicle has a predefined start and end point on its route and another goal is to minimize the length of all routes d. It is a generalization of the TSP.

$$\min \{cn_v + d\}$$

c is a constant that penalizes the number of vehicles that are required for a solution, in HyFlex c=1000. The solutions are initialised with a randomised constructive heuristic with a metric that gives the proximity of customers [Walker et al., 2012].

2.4 State of the art

In this section, state-of-the-art selection hyper-heuristics are covered and their main components are depicted. Table 2.2 gives an overview of the recent approaches that we will to take a closer look at. We distinguish between "strict" selection hyper-heuristics and others that also have a generation component (columns "Yes" and "No", respectively). Another distinction is that some approaches are evaluated on the HyFlex domains ("cross-domain") and others are only evaluated on selected domains, have no domain barrier or provide no results for the HyFlex benchmark suite (rows "Yes" and "No", respectively). As the title of the thesis suggests, we focus on the upper left part of the table where only one approach with deep reinforcement learning is listed [Dantas and Pozo, 2021].

We justify this selection of approaches, also containing "no strict" selection hyper-heuristics, with the focus of the thesis in reinforcement learning. These approaches are marked in bold in Table 2.2. However, we want to cover also a selection of three approaches whose main component does not utilize reinforcement learning algorithms and who performed well on the HyFlex benchmark suite.

They use two completely different ideas: SSHH [Kheiri and Keedwell, 2015] mainly focuses on learning effective sequences without restrictions for the available actions. Whereas FS-ILS [Adriaensen et al., 2014] uses Iterated Local Search to restrict the action space depending on the current stage. Also their acceptance-rejection mechanisms vary strongly. TS-ILS [Adubi et al., 2021] is an extension of FS-ILS and reports, to the best of our knowledge, one of the best results for the HyFlex benchmark.

We also included PHH [Udomkasemsub et al., 2023] and DR-ALNS [Reijnen et al., 2024] that are no strict selection hyper-heuristics, as they employ Proximal Policy Optimization for similar tasks.

		Selection Hyper-heuristics	
		Yes	No
Cross-domain (HyFlex)	Yes	SSHH [Kheiri and Keedwell, 2015] FS-ILS [Adriaensen et al., 2014] TS-ILS [Adubi et al., 2021] **QHH** [Choong et al., 2018] **RL** [Mischek and Musliu, 2022] **LAST-RL** [Kletzander and Musliu, 2023] **DQN** [1] [Dantas and Pozo, 2021]	
	No	**DRL-HH** [Zhang et al., 2022] **DRLHH** [Tu et al., 2023]	**PHH** [Udomkasemsub et al., 2023] **DR-ALNS** [Reijnen et al., 2024]

Table 2.2: State-of-the-art approaches for cross-domain selection hyper-heuristics. We distinguish between "strict" selection hyper-heuristics, i.e. it has no generation component, and others. We also differentiate between cross-domain selection hyper-heuristics that are evaluated on the HyFlex benchmark suite and other approaches that are not. The hyper-heuristics in bold font have reinforcement learning as their main component.

[1] has not been specifically named DQN

2.4.1 SSHH (2015)

SSHH by [Kheiri and Keedwell, 2015] utilizes a hidden Markov model to explore sequences of low-level heuristics. In contrast to an existing approach [Van Onsem et al., 2015], which models the solutions as hidden states and the objective function value as active states, here each low-level heuristic is modelled as a state in a hidden Markov model with a transition matrix that yields probabilities to apply the next low-level heuristic. It comes with two additional probability matrices that determine the search parameters and the acceptance-rejection strategy. The approach can be used to analyze and generate new sequences in an efficient way. In contrast to Iterated Local Search methods, that will be introduced later (Subsections 2.4.2, 2.4.3 and 2.4.6) the Markov model is not forced to switch between diversification and intensification phases because it is designed to determine effective sequences itself.
As the authors in [Kheiri and Keedwell, 2015] highlight, the strength of the hyper-heuristic is that it is very lightweight and can adapt easily to new problems, but also to changing stages within an instance.

The probabilities of the mentioned matrices for heuristic selection, acceptance-rejection, and search parameters (which have 11 levels equally split in [0,1]) are used to sample the next step. This step is stored for parameter updates. The likelihood of applying low-level heuristic i at time t is given as the product of the likelihood of last low-level heuristic j at time $t-1$, the transition probability from j to i and the corresponding entries in the search parameter matrix and the acceptance-rejection matrix, they are called emission matrices e^1 and e^2, respectively.

$$\ell_t(i) = \ell_{t-1}(j) \cdot a_{j,i} \cdot e_i^1(\alpha, \beta) \cdot e_i^2$$

In each step the acceptance-rejection mechanism either accepts the new solution regardless of the quality of the modified solution and proceeds without matrix updates. Or it calls the move acceptance procedure, which, in case of improvement to the overall best solution, accepts the new solution and updates the transition and emission matrices. All improvements lead to an increase of 1 of the parameters that are used to calculate the entries in the transition matrices. For more details and an exemplary explanation, we refer to [Kheiri and Keedwell, 2015]. We note that in their experiments, the crossover operators which require two solutions are ignored because their approach is a single point search method.

SSHH outperforms an existing method with Markov chains and it performs outstanding in the domains MaxSAT, Bin Packing, and TSP when compared to the results of CHesC 2011. In comparison to Iterated Local Search approaches like the CHeSC 2011 winner AdapHH ([Mısır et al., 2012]), that does not allow diversification operators until intensification does not lead to further improvement, SSHH decides when to switch between diversification and intensification.

2.4.2 FS-ILS (2015)

The Fair-Share Iterated Local Search (FS-ILS) hyper-heuristic ([Adriaensen et al., 2014]) combines the idea of Iterated Local Search with a conservative restart mechanism. It was designed with semi-automated algorithm selection approaches but the main objective was to get a simple and modular design.

The hyper-heuristic starts with random selection where the probability for each low-level heuristic is proportional to the average time spent. Later the selection probabilities are proportional to the number of applications that lead to new incumbent solutions weighted by the time spent. A solution is not a new incumbent if it is equal to the current one or if it is not accepted. After a perturbation operator, the local search procedure is called, which selects randomly. If an application of a local search heuristic does not lead to improvement, it is added to a tabu list and therefore not selected anymore. If all local search heuristic are in the tabu list, the local search procedure terminates, as it is assumed to be a local optimum for the available operators.

The acceptance mechanism has an important role because it is the only component that controls diversification by accepting also worsening solutions. FS-ILS uses the Metropolis acceptance condition where worse solutions are accepted with probability $e^{\frac{f(x_{current})-f(x_{new})}{temp \cdot \mu_{impr}}}$. The positive temperature parameter *temp* controls the likelihood of worsening solutions to be accepted and μ_{impr} is the moving average of improving changes in the objective function value which make the criterion in general and the choice for the parameter *temp* less domain dependent.

For the restart mechanism, a heuristic decides when the hyper-heuristic is stuck in a local optimum. This heuristic, adapted from [Hsiao et al., 2012], leads to a restart if for $a \cdot wait_{max}$ steps, the hyper-heuristic could not improve the current solution. $wait_{max}$ is the maximum number of steps until improvement. The choice of the parameter a is critical to avoid wasting time. As it is important in the beginning to let $wait_{max}$ grow to its true value and to avoid too early restarts, a should be chosen not too small. Therefore a is chosen to be decreasing over time t: $a(t) = \frac{T}{t}$. One additional criterion avoids "too late" restarts. If the remaining time is shorter than the minimum time it has taken to find a solution which is as good as the best candidate solution so far, the restart will not be executed.

For a detailed discussion of the design choices including a parameter sensitivity analysis, we refer to [Adriaensen et al., 2014]. In comparison to the CHeSC 2011 approaches, it shows particularly good performance in the domains MaxSAT, Flowshop and VRP, and it would have won the competition as a competitor.

2.4.3 TS-ILS (2021)

TS-ILS was proposed in [Adubi et al., 2021] and is a typical selection hyper-heuristic with selection and acceptance components. The selection mechanism is responsible for choosing LLHs to produce new solutions that are either accepted or rejected by the

acceptance mechanism. TS stands for Thompson Sampling, which is an online heuristic that is typically used for k-armed bandit problems. Here, this technique it not directly used for LLH selection but for controlling and learning the ILS perturbation operations. Perturbation operations include mutation, ruin-recreate and crossover.
The second important module of TS-ILS is the local search module in the intensification stage which forces the selection of local search operators.
TS-ILS combines the main ideas of two existing hyper-heuristics, the TS learning mechanism has been applied in TSHH [Alanazi, 2016] and selection and acceptance criteria are inspired by FS-ILS [Adriaensen et al., 2014].

In each step, the procedure selects one of the l predefined perturbation configurations based on their utility values $\Phi = \{\phi_1, ..., \phi_l\}$, i.e. select configuration $\text{argmax}_i \Phi$. The solution is perturbed by the selected perturbation operation. Available perturbation operations of TS-ILS contain perturbation with 1) mutation + ruin-recreate, 2) ruin-recreate + mutation, 3) mutation only, and 4) ruin-recreate only. Φ is based on vectors α and β that count the successes and failures of applications of each of the l perturbation configurations. "Success" means that the application of the perturbation operation led to a better objective function value after the local search operation.
The decision which LLH to select, is based on the speedNew mechanism which is a roulette wheel procedure whose parameters are based on a performance record of the last $2 \cdot 10^5$ iterations. The initial local search operator is selected based on the number of involvements in successful local search chains. Then a loop is entered which decides which next local search operators are selected based on a transition matrix $pScore$ that stores the number of transitions from heuristic i to heuristic j for each of the available local search operators i, j. This loop produces new solutions until no improvement is found. The matrix $pScore$ is initialized with 1 at each entry to avoid zero probabilities.
TS-ILS also uses the two search parameters $\alpha, \beta \in [0, 1]$ (discretized search space so that $\alpha, \beta \in \{0.1, 0.2, ..., 1.0\}$). For this purpose, a matrix stores a score for each LLH with all possible values of the search parameters and selection is done by a roulette wheel selection. For the updates of this matrix, the authors refer to [Kheiri and Keedwell, 2015].

The separation of ruin-recreate and mutation in the design of the configuration space led to superior generalization across the domains of HyFlex. Another important factor is that in the intensification phase, the transition matrix learns sequences of local search operators and therefore avoids random applications and also saves "lost" time by avoiding getting into cycles without improvement.

2.4.4 QHH (2018)

QHH is a Q-learning based selection hyper-heuristic whose action space consists of pairs of low-level heuristic selection and move acceptance methods. There are 6 LLH selection methods and 5 acceptance mechanisms implemented. The action space contains all combinations, thus 30 actions in total. The state space is a discrete one with only three distinct states, i.e. the normalized objective value of the solution in the last iteration being in $[0, \frac{2}{3})$, $[\frac{2}{3}, \frac{4}{3})$ or $[\frac{4}{3}, \infty)$.

The hyper-heuristic interacts with the environment episodically where within an episode the low-level heuristics are selected following an iterated local search approach. First, a perturbative action is selected, i.e. one of ruin-recreate or mutation, whereas crossover operators are ignored. The perturbative action is sampled from a categorical distribution (also referred to as roulette-wheel selection) where the probabilies are proportional to the values that are calculated with the LLH selection method. These 6 LLH evaluation scores refer to values that are extracted from characteristics of recent changes: the objective value change, if solutions were accepted by LLHs, duration of LLHs and if modified solution is new or stays identical. For more detailed definitions, see [Choong et al., 2018]. The application of the LLH creates a new, diversified solution. Then, the local search procedure is called to intensify the search. Here a random sequence of the available local search LLHs is applied until no further improvement is found. Finally, the move acceptance operator is called to decide whether the old solution is kept or overwritten by the modified solution. The 5 move acceptance methods consist of an Only Improvement (OI), an All Moves (AM), a Simulated Annealing (SA), a Late Acceptance (LA), and a Naive Acceptance (NA) method. More detailed explanations and related work of these acceptance methods are in [Choong et al., 2018].
After the acceptance mechanism, the evaluation scores are updated and the procedure is repeated until the maximum available time per episode is reached. This timeout for an episode is determined by a predefined number of episodes for the whole run. [Choong et al., 2018] tuned the number of episodes on the competition instances of HyFlex and found that 80 episodes are the best choice, so each episode has $\frac{T}{80}$ time.

Then, the episode terminates, the state is updated and the next episode starts if the timeout is not yet reached.

2.4.5 RL (2022)

In [Mischek and Musliu, 2022], a new hyper-heuristic approach using reinforcement learning was proposed. The hyper-heuristic corresponds to the reinforcement learning agent, the problem domain with the solution space is the environment, and the low-level heuristics are the actions to be selected. It is a single point method, so the crossover operators which require two solutions are excluded.

The RL hyper-heuristic addresses the problem of getting stuck in unpromising areas of the solution space by regular resets to the best solution found so far. It uses amplification which means that additional actions are added. It is used for intensification: the same operator is applied for a predefined duration and only improved solutions are accepted.

A major part of the paper [Mischek and Musliu, 2022] proposes multiple design choices and later performs experiments to come up with the best performing design: the search state is comprised of up to 4 features that use information about the search history, i.e. recently applied low-level heuristics, observed improvements and the remaining number of steps in the current solution chain. The action selection policies are ε-greedy and ε-softmax. For the reward function design, [Mischek and Musliu, 2022] evaluate four

different variants that are combinations of the improvement of the objective function value and the elapsed time of the last heuristic or the whole chain. The rewards are awarded after one episode which corresponds to one solution chain. The solution chain lengths are determined by the Luby sequence [Luby et al., 1993] which gives optimality guarantees and has shown good performance in practice by handling the trade-off between short chains (that may miss later improvement) and longer chains (that may waste time). For the state-action value function updates, Monte Carlo learning, SARSA, Q-learning and Expected-SARSA were tested.

The final hyper-heuristic uses the following configuration: two features which are the index of the last heuristic and a binary feature that indicates if in the last 10 steps, improvements were found. An ε-greedy policy with $\varepsilon = 0.1$ and a decreasing ε. The reward function is the improvement of the objective function value divided by the time that the whole chain has taken. The state-action values used the Monte Carlo learning update.

This new approach yielded insights in the design of hyper-heuristics and the potential and limitations of certain methods. The chosen final configuration would have achieved the second place in the CHeSC 2011.

2.4.6 LAST-RL (2023)

Large-State Reinforcement Learning hyper-heuristic [Kletzander and Musliu, 2023] learns the selection policy with a large search-state and an efficient version of the SARSA(λ) algorithm. It also comes with more elaborate reset and restart criteria, and an Iterated Local Search (ILS) component.

For this hyper-heuristic algorithm, that is the basis of the enhancements proposed in this thesis, we refer to Section 3.1. Here we will dive deep into the characteristics of the algorithm and explain enhancements to this algorithm in the subsequent Sections 3.2 and 3.3.

2.4.7 DRL-HH (2021)

[Zhang et al., 2022] developed a selection constructive hyper-heuristic that uses an offline trained double deep Q-network (DDQN) to handle uncertainties in combinatorial optimization problems. Applications that have uncertain variables revealed over time include container terminal truck routing. Training data were generated during interactions between agent and problem model which is an easy way to build data, also avoiding the danger of low-quality labels.

The proposed hyper-heuristic DRL-HH decides which action to select based on experience and two state vectors. As this is not a pure selection hyper-heuristic but has also a constructive component, it is not compatible with the "HyFlex-setting" where a strict domain barrier only allows selection of low-level heuristic operators without any domain knowledge or information about solutions. Here, the first of the two state vectors

describes the current partial solution ("explicit state vector") and the second one utilizes the deterministic model of the problem to estimate solution states for any future time point ("model derived state vector"). The states and LLHs are different to those in HyFlex, so we will focus solely on the model and its characteristics.

The agent learns the state-action value function $Q(s, a)$. The DDQN with experience replay [Lin, 1992] is chosen because it promises consistent performance and fast convergence behavior.

DDQN consist of two networks, the online and the target network with parameters θ and θ^-. The target network is fixed and updated every τ steps from the online network, so that $\theta_t = \theta_t^-$. This approach tends to reduce over-estimation. The learned target used by DDQN is:

$$Y_t^{\text{DDQN}} = R_{t+1} + \gamma Q(S_{t+1}, \text{argmax}_a Q(S_{t+1}, a; \theta_t); \theta_t^-)$$

Learning is done with gradient descent and squared error loss. The experience replay strategy is applied to sample mini-batches from the experience pool used for training.

The model has as input the state vectors that are first normalized to $[-50, 50]$ (depending on the model resulting in 16 or 24 input neurons), followed by two fully connected hidden layers with ReLU activation functions. The output layer produces the estimated state-action values for the 10 available LLHs. For training, a ε-greedy policy with $\varepsilon = 0.1$ was used and for testing, all actions were selected greedily ($\varepsilon = 0$). The experience replay buffer stored 3000 transitions (s_t, a_t, r_t, s_{t+1}) and for each update, 60 transitions were sampled. RMSprop algorithm with a learning rate of 0.001 was used to minimize the squared error loss.

2.4.8 DRLH (2023)

DRLH by [Kallestad et al., 2023] first generates the low-level heuristics and then the Deep Reinforcement Learning agent is responsible for selecting the low-level heuristics. It uses a stochstic policy π determined by a multi-layer perceptron that is updated via the Proximal Policy Optimization algorithm, see also [Schulman et al., 2017].

The crucial design choices are the state representation, the action space, and the reward function design:
They use a set of 12 problem independent features for which they provide an explanation on how they are intended to help the agent to improve the decisions at certain stages of the search. The action space is generated by the hyper-heuristic before the search and depends on the problem domain. The reward function is designed to guide the reinforcement learning agent to optimize the objective while not exploiting the reward function. They use the reward function R^{5310} that is derived from ALNS. The reward is 5 for an overall new best solution, 3 for improving the current solution, 1 if the new solution is accepted and 0 otherwise. This design also turns out to be stable and problem independent.

The acceptance-rejection mechanism uses simulated annealing, which always accepts improving solutions and in case of worsening, the new solution is accepted with probability $e^{-\frac{f(x_{new})-f(x_{current})}{temp}}$. For the inital temperature parameter *temp* and the cooling schedule, we refer to [Kallestad et al., 2023].

2.4.9 PHH (2023)

In [Udomkasemsub et al., 2023], they propose a hyper-heuristic using Proximal Policy Optimization. In contrast to selection hyper-heuristics in HyFlex, where perturbative low-level heuristics are selected, the underlying hyper-heuristic controls the selection of constructive low-level heuristics and therefore can not be evaluated on the HyFlex benchmark. Furthermore, the approach does not comply with the strict domain barrier of HyFlex. It uses features that describe the properties of the instance as well as current properties of the constraints, solutions and applied low-level heuristics. However, they still rely on limited domain knowledge for their design of generalized state and reward representations. The resulting Policy-Based Hyper-heuristic (PHH) outperforms existing approaches with its trained agents on the domains Bin Packing, TSP, and VRP.

2.4.10 DR-ALNS (2024)

The Adaptive Large Neighborhood Search (ALNS) algorithm by [Reijnen et al., 2024] uses Proximal Policy Optimization to control crucial selection and acceptance parameters. In contrast to existing methods which often come with computationally expensive operators, like DRLH in [Kallestad et al., 2023], this approach is more lightweight. It uses a search state with 7 features, including 4 binary features, and the reward function assigns a reward of 5 for operators that result in an improved solution and 0 otherwise. The presented algorithm demonstrates its ability to generalize across domains as it is not dependent on information from the underlying problem domain.

2.5 Terminology and Notation

The following Table 2.3 gives an overview of the terminology that is used throughout this thesis. In very few cases, we will not adhere to the given notation but in these cases, we will clearly point out deviations from our notation.

	Description	Notation
General	domain	$d \in \mathcal{D}$
	instance	$i \in \mathcal{I}$
	solution	$x \in X$
	objective/fitness function	$f : x \to f(x) \in \mathbb{R}$
	set of available low-level heuristics	$\mathcal{H}$
	timelimit	T
	"depth of search"-parameter	α
	"intensity of mutation"-parameter	β
	horizon	H
	chain length	c
	restart factor	rs
Reinforcement Learning	timestep/iteration	t
	state	$s \in \mathcal{S}$
	extracted state features after normalization	$\phi(s)$
	action	$a \in \mathcal{A} = \mathcal{H}$
	reward	r
	policy	π
	transition at time t	$\tau_t = (s_t, a_t, r_t, s_{t+1})$
	exploration rate (e.g. in ε-greedy)	ε
	discount rate	γ
	eligibility trace	e
	decay rate for eligibility traces (e.g. in SARSA(λ))	λ
	model parameters	θ, ψ
Deep Learning	batch size	b
	positive batch, negative batch	$(\tau_i^+)_{i=1}^b$, $(\tau_i^-)_{i=1}^b$
	(multi-agent) buffer	B
	activation function	σ
	activation of i-th hidden layer with size n_i	$h_i \in \mathbb{R}^{n_i}$
	number of hidden layers	L
	learning rate	l, l_π (actor), l_V (critic)
DQN	target model parameters	θ_-
PPO	actor network	$\pi_\theta : s \to \pi_\theta(s) \in \mathcal{R}^{\lvert\mathcal{A}\rvert}$
	policy distribution	$\pi(a \mid s, \theta)$
	critic network	$V_\psi : s \to V_\psi(s) \in \mathcal{R}$
	epochs	K
	epsilon clip (for clipped objective)	ε^{clip}
	clipped objective function	J^{clip}
	critic loss	$L^V(\psi)$
	weight for critic loss in joint surrogate objective	c_1
	weight for entropy of π in joint surrogate objective	c_2

Table 2.3: Terminology and Notation.

CHAPTER 3

Enhancements to the existing LAST-RL algorithm

In this chapter, we will dive deep into the LAST-RL (Large-State Reinforcement Learning) hyper-heuristic by [Kletzander and Musliu, 2023]. We will start by describing the components of the hyper-heuristic accompanied by pseudocode (Section 3.1), followed by the enhancements of LAST-RL. These include the introduction of new features for the search state by feature extraction (Section 3.2) and the extension of the action space (Section 3.3).

3.1 LAST-RL

The LAST-RL hyper-heuristic by [Kletzander and Musliu, 2023] has interesting components that we want to take a look at:

- ε-greedy policy with heuristic type selection based on Iterated Local Search (ILS) in the exploration case,
- learning with SARSA(λ) with eligibility traces,
- value function learned with linear approximation (using tile encoded search state),
- episode lengths determined by Luby sequence,
- extensive search state with a set of 15 features,
- comprehensive restart and termination criteria.

It outperformed existing hyper-heuristics based on reinforcement learning and also would have performed outstanding in the CHeSC 2011. Furthermore, the algorithm was evaluated on more complex real-life personnel scheduling domains (Bus Driver Scheduling, Rotating Workforce Scheduling and Minimum Shift Design) where it shows

strong potential. [Kletzander and Musliu, 2024] provides a detailed comparison of hyper-heuristics for the personnel scheduling domains, concluding that LAST-RL performed best among recent state-of-the-art hyper-heuristics.

The next two Subsections (3.1.1 and 3.1.2) provide explanations for the LAST-RL algorithm, closely aligned with the pseudocode in Algorithm 3.1:

3.1.1 Initialization

The hyper-heuristic starts its initialization for a given timeout T and instance $i \in I$ by loading the available action set $\mathcal{A}$. The search state s with an extensive set of features is initialised and the weights of the learning agent θ are set to zero. The hyper-heuristic modifies one solution at a time but several solutions are stored in x. One place is for the global best solution (x_{best}), one for the current best solution since last reset (x_{current}) and one slot is for the current solution (x_{new}). There are n_c additional slots for crossover operators, where one of these solutions is replaced every time a new best solution since last reset is found. Crossover takes the current solution and a random one of the crossover slots. The solution initialization procedure that is a domain specific one, is called 10 times before the main loop starts.

3.1.2 Main loop

In the main loop (see Algorithm 3.1, lines 5 to 24), an episodic reinforcement learning agent acts in the environment of our problem domain. Until the termination of an episode or until the timeout is reached, the RL agent selects low level heuristics and updates the solution in each step. Within each episode, every new solution is accepted, so there is no classic acceptance-rejection procedure. Each episode has a predefined length c that is sampled from the Luby sequence [Luby et al., 1993], however the actual length of the episode may differ, see the following.

Episode length

In [Luby et al., 1993] they introduced the Luby sequence for strategies that do not have prior knowledge which means in our case that the underlying distribution of the number of steps needed for improvement is not known. Using these sequence lengths for a strategy is proven to be close to the optimum with full knowledge. The Luby sequence l is iteratively defined as:

$$l_i = \begin{cases} 2^{k-1}, & \text{if } i = 2^k - 1 \\ l_{i-2^{k-1}+1}, & \text{if } 2^{k-1} \leq i < 2^k - 1 \end{cases}$$

$$l = (1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, 1, 1, ...)$$

As mentioned, this will determine the planned length of the next episode. But there are several cases, where the actual episode length deviates:

Algorithm 3.1: LAST-RL

Data: instance $i \in I$, domain $d \in D$, timeout T, crossoverslots n_c, α, β

1 $\mathcal{A} \leftarrow$ createActionSpace(d, α, β)
2 $s_0 \leftarrow$ initialState()
3 $\theta_0 \leftarrow$ initialModelParameters()
4 $x = (x_{\text{new}}, x_{\text{best}}, x_{\text{current}}, x_1^c, \ldots, x_{n_c}^c) \leftarrow$ initializeSolutions(n_c)
5 **while** $time < T$ **do**
6 $c \leftarrow$ chainLength() (see **Episode Length**)
7 **for** $t \leftarrow 1$ ***to*** c **do**
8 $a_t \leftarrow$ **selectLLH**(s_t, θ_t) (see **selectLLH** in Algorithm 3.2)
9 $x_{\text{new}} \leftarrow$ executeLLH(a_t, x_{new})
10 $r_t \leftarrow R(f(x_{\text{new}}), time)$ (see **Reward function**)
11 $s_{t+1} \leftarrow$ updateState$(s_t, a_t, f(x_{\text{new}}), time)$
12 $\theta_{t+1} \leftarrow$ **updateModel**$(s_t, a_t, r_t, s_{t+1}, \theta_t)$
 (see **updateModel** in Algorithm 3.3)
13 **if** $f(x_{new}) < f(x_{best})$ **then**
14 $x_{\text{best}} \leftarrow x_{\text{new}}$
15 $x_{\text{current}} \leftarrow x_{\text{new}}$
16 **break**
17 **end**
18 $t \leftarrow t + 1$
19 **end**
20 $x_{\text{new}} \leftarrow x_{\text{current}}$
21 **if** *restart*$(x, time)$ **then**
22 reset(s_t, x) (see **Reset behavior**)
23 **end**
24 **end**

Every time a new best solution since last reset is found, the current episode is terminated (see Algorithm 3.1, lines 13 to 17). In two cases, the episode continues after c steps but it will not exceed $2 \cdot c$: if the last heuristic was a local search operator or if the last step improved the objective value (but did not lead to an improvement of the current best since the last reset). The differences between domains led to another adaptation of the episode lengths: c is adapted with the average length $\bar{l}$ of previous successful chains, i.e. $c \leftarrow c \cdot \max\{\frac{\bar{l}}{2}, 1\}$.

Reset behavior

Besides the episode length, there is also a quite elaborate reset behavior: 1000 episodes without improvement or 100 episodes without improvement and time $2 \cdot rs \cdot T$ since last reset passed will lead to a reset of S where only the global best solution is kept. The parameter rs was set to 0.1 in the final version.

If $2 \cdot rs \cdot T$ is left until timeout, we move to the global best solution to enforce intensification, or if $3 \cdot rs \cdot T$ are left, a regular reset would also lead to an immediate move to the global best solution.

Value function approximation

In each step within an episode, the state-action value function is estimated with linear approximation. The search state features are the extracted features of the raw search state after normalization and tile coding. The extracted state features (see Section 3.2.1) are normalized with predefined bounds and then tile coded with n tiles and resolution of r tiles per dimension, resulting in a sparse state representation s_t of $n \cdot r$ binary features. The tile coded features s_t are then multiplied with the weight vector θ to obtain state-action value estimations. Note that only a small subset of these features are active (=nonzero).

Policy π

An ε-greedy policy selects the actions based on the value function, i.e. in the greedy case, the action with the maximum state-action value estimation is selected. For the exploration case, [Kletzander and Musliu, 2023] propose a probability distribution based on Iterated Local Search, which switches between perturbation and intensification phases. The decision criteria for each heuristic type are covered in Algorithm 3.2 in the lines 4 to 15.

The type of the next action is sampled from this distribution. Within the chosen heuristic type, the next heuristic operator is then randomly selected from a uniform distribution, see line 16.

The initial action type is selected randomly. Otherwise the next action is based on the previous one. We stay in the same category depending on a_r, a_m, a_c, and a_l. These are the average numbers of ruin-recreate, mutation, crossover, and local-search operators applied in successful chains.

If the last heuristic was ruin-recreate, again ruin-recreate is applied with probability

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

or any other type with probability $\frac{1-p_r}{3}$. Crossover is followed by mutation or local search with probability $\frac{1-p_c}{2}$ and crossover is selected again with

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

After mutation, another mutation operator is selected with

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

Algorithm 3.2: LAST-RL: selectLLH (ILS based type-selection)

Data: Tile coded state features s_t, linear weights θ_t

if *greedy* **then**
  $a_t \leftarrow \underset{a \in \mathcal{A}}{\text{argmax}}\, Q_{\theta_t}(s_t, a)$ (select action greedily)
else
  if *lastType* = *ruin-recreate* **then**
    $$nextType \leftarrow \begin{cases} \text{ruin-recreate} & \text{with } p = p_r \\ \text{crossover} & \text{with } p = \frac{1-p_r}{3} \\ \text{mutation} & \text{with } p = \frac{1-p_r}{3} \\ \text{local-search} & \text{with } p = \frac{1-p_r}{3} \end{cases}$$
  end
  if *lastType* = *crossover* **then**
    $$nextType \leftarrow \begin{cases} \text{crossover} & \text{with } p = p_c \\ \text{mutation} & \text{with } p = \frac{1-p_c}{2} \\ \text{local-search} & \text{with } p = \frac{1-p_c}{2} \end{cases}$$
  end
  if *lastType* = *mutation* **then**
    $$nextType \leftarrow \begin{cases} \text{mutation} & \text{with } p = p_m \\ \text{local-search} & \text{with } p = 1 - p_m \end{cases}$$
  end
  if *lastType* = *local-search* **then**
    $nextType \leftarrow$ local-search
  end
  $a_t \leftarrow$ randomChoice(nextType, $\mathcal{A}$) (random action from selected type)
end
return a_t

or otherwise local search with probability $1 - p_m$.

The next low-level heuristic is selected according to this and the hyper-heuristic observes the new objective value and the current time. This information is used to calculate the reward, to update the search state, and the weights of the learning agent.

Reward function

The reward function was specifically designed to escape local optima.

$$R^{LAST-RL} = \begin{cases} \sqrt{maxFC} & \text{if } f(x_{\text{new}}) < f(x_{\text{current}}), \\ 0 & \text{otherwise} \end{cases}$$

With $maxFC$ being the maximum number of chains without improvement since the last reset. [Kletzander and Musliu, 2023] also added a penalty term for the runtime, also considering the timeout and chain length. This turned out to be an important adjustment for the application on their real-world instances. $R_{LAST-RL}$ is only awarded at the end of an episode because improvement of the objective since the last reset terminates an episode.

Update of learning agent

For the update (in Algorithm 3.3), the next action a_{t+1} is selected according to the ε-greedy policy π with the state-action values $\hat{q}$ that are linearly approximated with the current weight vector θ_t. Then the weights are updated with the SARSA(λ) update rule ([Sutton and Barto, 2018]):

$$\theta_{t+1} = \theta_t + \alpha\delta_t \mathbf{e}_t$$

The SARSA(λ) algorithm uses eligibility traces e which memorize changes in the recent transitions. The eligibility traces are updated with the iterative update rule ([Sutton and Barto, 2018]), see lines 2 to 6 in Algorithm 3.3:

Algorithm 3.3: LAST-RL: updateModel (using SARSA(λ))

Data: s_t, a_t, r_t, s_{t+1}, linear weights θ_t, discount rate γ, decay rate λ, step size α

1 $a_{t+1} \leftarrow \pi_{\theta_t}(s_{t+1})$ (see **selectLLH** in Algorithm 3.2)
2 **if** $t \geq 1$ **then**
3 $\quad \mathbf{e}_t \leftarrow \gamma\lambda\mathbf{e}_{t-1} + \nabla Q_{\theta_t}(s_t, a_t)$ (update eligibility trace)
4 **else**
5 $\quad \mathbf{e}_0 \leftarrow \nabla Q_{\theta_0}(s_0, a_0)$ (initialize eligibility trace)
6 **end**
7 $\delta_t \leftarrow r_t + \gamma Q_{\theta_t}(s_{t+1}, a_{t+1}) - Q_{\theta_t}(s_t, a_t)$ (TD error)
8 $\theta_{t+1} \leftarrow \theta_t + \alpha\delta_t \mathbf{e}_t$ (update weights)
9 **return** θ_{t+1}

This method is very efficient because only the active binary features (i.e. the corresponding indices) need to be stored.

These are the essential algorithmic components of LAST-RL, but the main novelty of LAST-RL was the large search state with 15 features. We will first give an overview about the used features for LAST-RL, then propose new features and use these new features to create feature groups. With these new features, we will do a feature ablation study in Chapter 5. These experiments will be guided by statistical tests.

3.2 Extraction of new state features

In [Kletzander and Musliu, 2023] a novel set of 15 state features was proposed. LAST-RL uses tile-encoding which explains why some of the following categorical features are not

binary encoded. We list these features with a short description but we go without mathematical definitions because it would require a lot of terminology.

3.2.1 Features of search state in LAST-RL

- lastHeur: index of last heuristic (ranging from -1, for first heuristic application or after reset, to $|\mathcal{A}| - 1$).
- lastType: type of last heuristic. -1 for first application and after reset, 0 for ruin-recreate, 1 for crossover, 2 for mutation, and 3 for local search.
- lastChangeSign: sign of recent change of objective value, $sgn(f(x_{new}) - f(x_{current}))$
- lastChangeMag: magnitude of recent change of objective value.
- chainProgress: number of steps that have been applied in current episode (relative to sequence length)
- lastImprMag: magnitude of steps since last improvement
- lastBestMag: magnitude of steps since best solution
- timeTotal: total runtime since previous reset
- relativeImprMag: magnitude of proportion between objective values of current and initial solution
- relativeBestMag: magnitude of proportion between objective values of current and best solution
- relImpr: relative number of steps that lead to improvement
- rel0: relative number of steps that do not change the objective
- avgChangeHMag: magnitude of average change of objective value of the last H steps
- relImprH: relative number of steps that lead to improvement in the last H steps
- rel0H: relative number of steps that do not change the objective in the last H steps

We propose new features that are mainly based on interactions between the raw features (time, value and heuristic) and characteristics of the hyper-heuristic (episode, horizon).

3.2.2 New proposed features to extend search state in LAST-RL

- avgChangeRuinrecreateHMag: magnitude of the average change in the last H evaluations of ruin-recreate applications
- avgChangeCrossoverHMag: magnitude of the average change in the last H evaluations of crossover applications

- avgChangeMutationHMag: magnitude of the average change in the last H evaluations of mutation applications
- avgChangeLocalsearchHMag: magnitude of the average change in the last H evaluations of local search applications
- timeDelta: time since last state update in ms (set to 0 for first search state and after reset)
- RuinrecreateInv: relative number of ruin-recreate operators involved in current episode
- CrossoverInv: relative number of crossover operators in current episode
- MutationInv: relative number of mutation operators in current episode
- LocalsearchInv: relative number of local search operators in current episode
- avgTimeRuinrecreate: mean of timeDelta of ruin-recreate operators
- avgTimeCrossover: mean of timeDelta of crossover operators
- avgTimeMutation: mean of timeDelta of mutation operators
- avgTimeLocalsearch: mean of timeDelta of local search operators

Another two features that can be added when we extend the action space with the search parameters α and β:

- lastAlpha: last value of search parameter α (intensity of mutation)
- lastBeta: last value of search parameter β (depth of search)

We can categorize all available features in 8 groups, 3 groups of the raw features (time, value, heuristic), one RL characteristic (episode), one group of features which are extracted from the value changes of the last H steps (horizon), and 3 interaction feature groups (time $\times$ heuristic, heuristic $\times$ episode, heuristic $\times$ horizon). Here, interaction features are features that are extracted from a combination of two or more other features. Table 3.1 lists all features and displays in which of the feature groups they are contained, indicated with ticks (✓). The new proposed features are indicated with *.

We will do an ablation feature selection study in Section 5.3 to determine the most effective set of features. We aim to gain insights into which groups of features can be linked with performance differences, identify the domains where the use of a different set of search state features affects performance, and determine whether the raw feature set alone is already sufficient. These feature sets (Ti, Va, He, Ep, Ho) will be either added to the *raw* feature set (in column raw) or removed from the *full* feature set (in column full). The features of the *raw* model are indicated also with the bold font.

		raw	full	LAST-RL	Ti	Va	He	Ep	Ho
time									
1	**timeTotal**	✓	✓	✓					
2	timeDelta*		✓		✓				
value									
3	lastChangeSign		✓	✓		✓			
4	**lastChangeMag**	✓	✓	✓					
5	**lastImprMag**	✓	✓	✓					
6	lastBestMag		✓	✓		✓			
7	relativeImprMag		✓	✓		✓			
8	relativeBestMag		✓	✓		✓			
9	relImpr		✓	✓		✓			
10	rel0		✓	✓		✓			
heuristic									
11	**lastHeur**	✓	✓	✓					
12	lastType		✓	✓			✓		
13	lastAlpha*								
14	lastBeta*								
episode									
15	**chainProgress**	✓	✓	✓					
horizon									
16	avgChangeHMag		✓	✓					✓
17	relImprH		✓	✓					✓
18	rel0H		✓	✓					✓
time × heuristic									
19	avgTimeRuinrecreate*		✓		✓		✓		
20	avgTimeCrossover*		✓		✓		✓		
21	avgTimeMutation*		✓		✓		✓		
22	avgTimeLocalsearch*		✓		✓		✓		
heuristic × episode									
23	RuinrecreateInv*		✓				✓	✓	
24	CrossoverInv*		✓				✓	✓	
25	MutationInv*		✓				✓	✓	
26	LocalsearchInv*		✓				✓	✓	
heuristic × horizon									
27	avgChangeRuinrecreateHMag*		✓				✓		✓
28	avgChangeCrossoverHMag*		✓				✓		✓
29	avgChangeMutationHMag*		✓				✓		✓
30	avgChangeLocalsearchHMag*		✓				✓		✓

Table 3.1: Features and their groups for ablation study for feature selection.

3.3 Extending the action space

In LAST-RL, the action space comprises the low level heuristics each with the search parameters α and β set to 0.2 which is their default value. We extend this search space by adding one more conservative and one more perturbative action, with 0.1 and 0.6 as search parameters, respectively.

What we hope to gain with this extension of the action space:

- The higher granularity could be advantageous in certain stages of the search. In the beginning, one could benefit from more perturbative actions and in late stages, more conservative actions might be more successful.

This design of the search space entails risks:

- The actual structure of this action space should be a hierarchical one where we would first select the LLH and in the second stage, we choose the sensitivity by adjusting α or β. But we design it as a flat action space where same LLHs with different search parameters are treated as separate actions. This is not a very natural, intuitive design and it might influence the behavior in the exploration case of the ε-greedy action selection, because it is then more likely to select LLHs that appear more often in the action space. Anyways, this mentioned bias could be corrected.
- The interpretability of the LLH's value is reduced because LLHs with a search parameter yield multiple state-action values.
- The flat design results in a large action space with 29 (MaxSAT), 22 (Bin Packing), 24 (Personnel Scheduling), 31 (Flowshop), 25 (TSP), and 24 (VRP) actions. This makes the prediction harder with higher risk of overfitting. Also the complexity and computational costs are higher.

We decided to use a flat action space design because only a fraction of the actions uses search parameters. A hierarchical action space would lead to a higher complexity of the model and would make it more difficult to control the learning behavior. Our enhancement should serve as an initial investigation of an extended action space for LAST-RL. In future work, also more complex action space designs, like a hierarchical one, could be taken into consideration.

[Drake et al., 2020] listed a selection of hyper-heuristics and what LLH set was used. The hyper-heuristic with the highest overall score [Mısır et al., 2012] uses the full action space, other approaches use a reduced set of LLHs ([Hsiao et al., 2012] and [Lehrbaum and Musliu, 2012]). To the best of our knowledge, there have not been explicit studies that analyze the impact of allowing variations of the search parameters and hence the effect of not only using the default value.

In [Drake et al., 2020], they mentioned also relay hybridisation which is the creation of new LLHs by combining two existing LLHs. This approach would be interesting because in the trajectories of LAST-RL, one can see that very regularly, the subsequent application of two LLHs are selected. This pair of LLHs often consists of a perturbative LLH, followed by a local search operator.

We test the extended action space against the default action space by comparing the scores to detect possible differences in the search. The experimental setup and results are in Section 5.4.

CHAPTER 4

Deep Reinforcement Learning Hyper-heuristics based on LAST-RL

In this chapter, two hyper-heuristics are introduced, both based on the existing LAST-RL hyper-heuristic by [Kletzander and Musliu, 2023]. These methods primarily differ from the existing method by integrating deep reinforcement learning algorithms (Deep Q-learning and Proximal Policy Optimization) and assess the potential of lifelong reinforcement learning techniques for pre-training across domains.

For both methods, we first explain the algorithms used and how they are integrated into the hyper-heuristic. After that, crucial design choices are outlined. Finally, the ability of pre-training DS-RL with a technique of lifelong reinforcement learning is described.

4.1 Deep-State Reinforcement Learning: A Deep Q-learning Hyper-heuristic

We replace the SARSA(λ) algorithm, which is an extension of the on-policy method SARSA with eligibility traces. Instead, we use the deep Q-learning algorithm, first introduced by [Mnih et al., 2013], which is an off-policy method. It uses experience replay to update the parameters. The experience replay mechanism was developed by [Lin, 1992], an effective way to reduce the number of action executions and to be more sample efficient. It was intended to achieve a speedup in robotic tasks, also allowing stable learning in non-Markovian environments.

The idea of deep Q-learning is to approximate the value of an action not based on discrete values, i.e. with a table like in Q-learning. Deep Q-learning takes continuous states

s and approximates Q-values ($Q_\theta(s, a)$) for each action a via a deep neural network [Sutton and Barto, 2018].

Although there already exist more advanced Deep Reinforcement Learning techniques, we want to find out, how LAST-RL is influenced by exchanging the SARSA(λ) learning mechanism with a Deep Learning architecture. What makes the DQN interesting also for the setting of hyper-heuristics is that it performed very well for the Atari 2600 games, which are video games that are used as a reinforcement learning benchmark suite. It learned a different policy for each of the 49 games but the input had the same structure and the used hyperparameters (network structure, learning rate, etc.) were the same. This corresponds to our goal: to find a hyper-heuristic that works well across all available domains without manually adapting the input or the hyperparameters.

We also remove the tile coding layer of LAST-RL because we do not need a discrete action space for deep Q-learning. Tile coding helps to stabilize learning and to boost efficiency with its sparse representation, see [Sutton and Barto, 2018]. This is particularly useful for algorithms whose update rules are designed for sparse input, e.g. Sarsa(λ) with binary features and linear function approximation (see [Sutton and Barto, 2018]) which is used by LAST-RL. Deep Q-learning does not profit from this sparse representation which would lead to a huge input layer (curse of dimensionality). So we use the (normalized) state features as direct inputs for the deep Q-network, expect it to generalize well for raw inputs, and take the higher risk of overfitting into account.

In the literature, there is already a cross-domain selection hyper-heuristic with a deep Q-network evaluated on the HyFlex instances, see [Dantas and Pozo, 2021]. One of the biggest differences to our approaches is the search state. In [Dantas and Pozo, 2021], they have one feature for each low-level heuristic which contains the normalized average reward for the last W steps obtained by this operator.

Preliminary experiments with our approach using deep Q-learning already showed superior performance to [Dantas and Pozo, 2021]. Anyways, their results were merely used to show advances compared to two simpler methods: an Upper Confidence Bound (UCB) Multi-Armed Bandit algorithm and Fitness-Rate-Rank Multi-Armed Bandit [Li et al., 2014]. The latter rates the application of low-level heuristic i at time t with the relative improvement of the objective function value to current solution, called Fitness Improvement Rate (FIR). The FIR was used as the reward for all three approaches. For the comparison of DQN with these two methods, they did not use additional hyper-heuristic components and also accepted all new solutions without any acceptance criteria or restart mechanisms for all three approaches ([Dantas and Pozo, 2021]). This is a drawback of their approach and another indication why it is not competitive with state of the art.

4.1.1 Deep Q-learning algorithm

The deep Q-learning algorithm is an off-policy reinforcement learning algorithm with a behavior and target policy based on Q-learning. The behavior policy is responsible for action selection. It is a non-linear function approximation (a multi-layer perceptron)

for the prediction of the state-value function $Q_\theta(a, s)$. Its parameters θ are regularly updated, e.g. after each episode or step. It is a temporal difference method that uses the the Bellman optimality equation for the model updates [Mnih et al., 2013]. The target policy is used for model updates.

This setup led to problems with divergence and instability because the sequential processing of observations caused correlations between the target value ($r_t + \gamma \max_{a'} Q_\theta(s_{t+1}, a')$) and the Q-value ($Q_\theta(s_t, a_t)$). Thinking about it as a supervised learning task, it would lead to both target and prediction being updated which is not reasonable in supervised learning, and here it also causes problems. Another problem is that the target is usually non-stationary, especially in our problem, the rewards are quite sparse, i.e. only non-zero for improving steps, which would cause difficulties [Mnih et al., 2013].

These issues were addressed with two adaptations:
First, a target network, which is a copy θ^- of the actual online Q-network, but is updated ($\theta^- = \theta$) only after C Q-network updates. Then the loss $L(\theta)$ has independent targets and predictions:

$$L(\theta) = [\underbrace{r_t + \gamma \max_{a'} Q_{\theta^-}(s_{t+1}, a')}_{\text{target}} - \underbrace{Q_\theta(s_t, a_t)}_{\text{prediction}}]^2$$

The Q-network is updated via mini-batch gradient descent with the loss $L(\theta)$. The delay of C in the target network stabilizes learning and reduces the risk caused by non-stationary targets.

Second, the experience replay, first introduced by [Lin, 1992], is an important component. It allows to revisit previous states in the model updates. Especially in phases of the search where improvements are rarely found, the experience replay mechanism allows to learn from old iterations τ_t^+ that were involved in successful chains , i.e. iterations τ_t whose episode has a positive cumulative reward. The experience replay buffer stores both "successful" (τ_t^+) and "unsuccessful" (τ_t^-) iterations in the experience replay buffer, which are called positive and negative buffer. In each model update step, a batch of size b is sampled from both positive and negative buffer which is then used for the update. This sampling from varying stages of the search stabilizes learning by removing correlation of subsequent observations. Depending on the size of the buffer, more or less experience from the past can be stored and used for learning. The experience buffer that we use here is a kind of prioritized replay buffer [Schaul et al., 2015], in fact it is a balanced prioritized replay buffer. There exists recent work that covers the use of probability estimates for sampling experiences in balanced prioritized replay buffers ([Sun et al., 2022], [Lou et al., 2024]).

In Figure 4.1, all mentioned components are illustrated:
In every step, the agent uses the estimated Q-values ($Q_\theta(s_t, a_t)$) and selects the next action a_t based on policy π, e.g. ε-greedy or softmax. The action is applied in the environment (in our case the selected low-level heuristic is applied on the current solution) and we get a reward r_t and a new state s_{t+1}. The full transition $\tau_t = (s_t, a_t, r_t, s_{t+1})$ is stored in the experience replay buffer. The updates after every step are indicated with

solid lines in Figure 4.1.
After each episode (indicated with dashed lines), the collected experiences are assigned to the positive or negative buffer which depends on whether the cumulative reward is positive or not. Each of the two buffers has a buffer size which is the maximum capacity for transitions. If the buffer size is exceeded, old experiences are replaced. Then the model parameters θ are updated as mentioned before, we sample a batch from both buffers and use them to update the model parameters θ with the update rule that was mentioned above.
Finally after C model updates (or episodes), the target network is updated with the parameters of the Q-network, i.e. $\theta^- = \theta$.

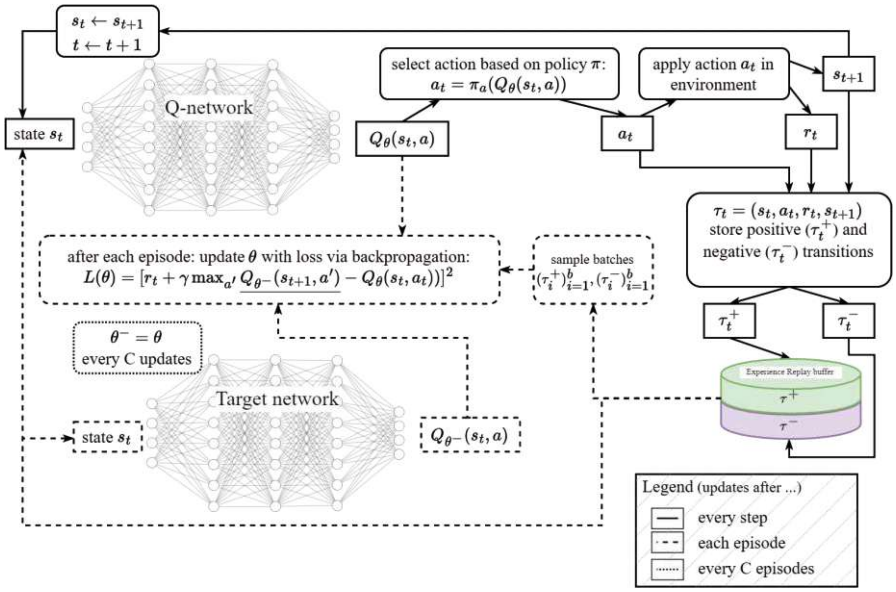

Figure 4.1: Deep Q-learning algorithm with experience replay and target network.

Deep Q-network architecture

The model architecture of the Q-network is shown in Figure 4.2. The same applies to the target network if θ is replaced by θ^-. The Q-network is a fully connected multi-layer perceptron with L hidden layers. The input layer gets the transformed features $\phi(s_t)$ which are the extracted state features after normalization. We apply normalization for all non-binary state features and have two available normalization functions: one that was also used in [Kletzander and Musliu, 2023] where a lower and upper bound for each feature has to be provided as hyperparameters (x_{min} and x_{max}). Then we normalize similar to min-max-normalization:

$$\phi_{\text{normalized}} = \frac{\phi - \phi_{min}}{\phi_{max} - \phi_{min}}$$

The alternative is z-score normalization.
After the forward pass through the L hidden layers, we get a Q-value estimation for each action in $\mathcal{A}$. Each hidden layer also has a bias term which is added and not connected with the previous layer. The bias terms are not shown in Figure 4.2 but are used throughout the model after each hidden layer.

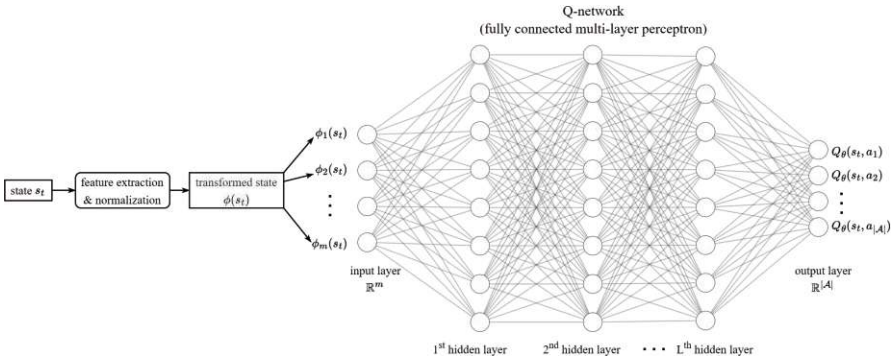

Figure 4.2: Deep Q-network architecture.
Illustration of how the Q-value $Q_\theta(s_t, a_t)$ is estimated from the raw state s_t. Note: After each hidden layer, an activation function σ is applied and bias terms are not part of this illustration but present in the model.

Double deep Q-learning

In [Hasselt et al., 2016], it is argued, that the deep Q-learning algorithm overestimates Q-values. The proposed double deep Q-network reduces this overestimation by "*evaluating the greedy policy according to the online network, but using the target network to estimate its value*" [Hasselt et al., 2016]. Minimal changes in the update rule led to superior performance for certain tasks in the Atari 2600 domains and successfully reduce overestimation.

The target estimation in the update rule changed from

$$r_t + \gamma \max_{a'} Q_{\theta^-}(s_{t+1}, a')$$

to

$$r_t + \gamma Q_{\theta^-}(s_{t+1}, \operatorname*{argmax}_{a'} Q_\theta(s_{t+1}, a')).$$

Apart from the update rule, all other components, including the architecture and action selection, stay as in the deep Q-network, see Figure 4.1.

4.1.2 Design choices for DS-RL

Apart from the hyperparameters that already play an important role in the LAST-RL hyper-heuristic (γ, ε, rs, H), we propose new design choices that derived during the development of DS-RL.

Reward function

The reward function is a crucial design choice in order to guarantee that the agent is learning the actual objective without exploiting the reward function.

- Multi-agent reward function (R^{MA})

  For multi-agent algorithms, we propose the following reward function:

$$R^{MA} = \begin{cases} \frac{f(x_{\text{agent best}}) - f(x_{\text{new}})}{f(x_{\text{best}})} & \text{if } f(x_{\text{new}}) < f(x_{\text{agent best}}), \\ 0 & \text{otherwise} \end{cases}$$

  This is similar to one of the reward functions in [Kallestad et al., 2023]:

$$R_t^{MC} = \frac{f(x_{\text{best}}) - f(x_{\text{new}})}{f(x_{\text{best}})}$$

  The authors claim that this reward function forces the agent to choose actions that directly improve the objective. Therefore it might be a better choice for searches with fewer iterations available. But the training of an agent is less stable with this reward function because good actions are often rewarded with a delay.

  Therefore, we modified this reward so that it does not punish actions that might result in worse fitness so strictly, thus emphasizing exploration. This is done by setting the reward to 0 if no improvement was found. Furthermore, we extended the approach for the multi-agent setting. The "knowledge" that an agent gains by improving solutions in a less promising area can also be valuable. This is achieved by replacing $f(x_{\text{best}})$ with $f(x_{\text{agent best}})$ in the numerator.

  The problem with reward functions that contain the fitness functions is that they are hard to compare across domains and therefore make learning of the actor-critic network across domains difficult.

  Although this reward function is actually designed for multi-agent algorithms like maPPOs, it can be also utilized for single-agent algorithms.

- LAST-RL reward function ($R^{LAST-RL}$)

  For LAST-RL [Kletzander and Musliu, 2023], the reward was specifically designed to escape local optima.

$$R^{LAST-RL} = \begin{cases} \sqrt{maxFC} & \text{if } f(x_{\text{new}}) < f(x_{\text{best}}), \\ 0 & \text{otherwise} \end{cases}$$

With $maxFC$ being the maximum number of chains/episodes without improvement since the last reset. They also added a penalty term for the runtime, also considering the total time and chain length. This turned out to be an important adjustment for the application on their real-world instances.

- R^{530}

  An alternative, adapted from [Kallestad et al., 2023]:

$$R^{530} = \begin{cases} 5 & \text{if } f(x_{\text{new}}) < f(x_{\text{best}}), \\ 3 & \text{if } f(x_{\text{new}}) < f(x_{\text{current}}), \\ 0 & \text{otherwise} \end{cases}$$

  This reward function is more robust by equally rewarding improvements not depending on how much better the new solution is. It rewards overall new best solutions but also the improvement of current best solutions. In [Kallestad et al., 2023], the reward function was called R^{5310} because they use an acceptance mechanism which results in a reward of 1 if the new solution is not an improvement.

- enhancements for LAST-RL reward (R^{DS-RL})

  Designing a meaningful reward function for our task is challenging. A promising approach from the literature is the Intrinsic Curiosity Module (ICM) by [Pathak et al., 2017], where an intrinsic reward signal is generated through self-supervised learning rather than relying on sparse rewards from the environment. It forces exploration by rewarding the agent based on a measure of curiosity, defined as the prediction error in forecasting a representation of the future state via the inverse dynamics model [Pathak et al., 2017].

  We mimic the idea of ICM and integrate a simplified measure of curiosity into the reward function. The function s_{ICM} uses this measure of curiosity to increase the reward. A potential risk is that the agent may be encouraged to increase prediction error when trying to maximize curiosity, which may lead to unexpected behavior.

$$s_{\text{ICM}}(r) = \begin{cases} r \cdot \left[1 + \max(0, \frac{r - Q_{\theta_t}(a_t, s_t)}{r})\right] & \text{if } f(x_{\text{new}}) < f(x_{\text{current}}), \\ r & \text{otherwise} \end{cases}$$

  We introduce an adjustment factor, $s_{\text{le}} \in [0, 1]$, which increases the reward when the last step yielding a new best solution is both a "large" improvement and requires "effort." Here, "large" refers to the improvement relative to the prior solution's difference from the overall best solution. "Required effort" means that the previous solution was worse than the overall best. The idea behind "effort" is that a worsening

step had to be made prior to the improvement. The intuition is that we do not want to reward frequent small steps that might result in a local optimum, anyway.

$$s_{\text{le}} = \begin{cases} \max\left(0, \frac{(f(x_{\text{best}})-f(x_{\text{new}}))-(f(x_{\text{current}})-f(x_{\text{best}}))}{f(x_{\text{best}})-f(x_{\text{new}})}\right) & \text{if } f(x_{\text{new}}) < f(x_{\text{best}}) < f(x_{\text{current}}) \\ 0 & \text{otherwise} \end{cases}$$

With these two adjustments $s_{\text{ICM}}(r)$ and s_{le}, we adapt the reward function from [Kletzander and Musliu, 2023] to get a new reward function:

$$R^{DS-RL} = \begin{cases} s_{\text{ICM}}(\sqrt{maxFC} \cdot (1 + s_{\text{le}})) & \text{if } f(x_{\text{new}}) < f(x_{\text{best}}), \\ 0 & \text{otherwise} \end{cases}$$

We will evaluate also variations of R^{DS-RL} to be able to analyze the individual components ($\sqrt{maxFC}$, s_{ICM}, s_{le}) in Subsection 5.5.2.

$$R^{DS-RL-1} = \begin{cases} s_{\text{ICM}}(\sqrt{maxFC}) & \text{if } f(x_{\text{new}}) < f(x_{\text{best}}), \\ 0 & \text{otherwise} \end{cases}$$

$$R^{DS-RL-2} = \begin{cases} s_{\text{ICM}}(1 + s_{\text{le}}) & \text{if } f(x_{\text{new}}) < f(x_{\text{best}}), \\ 0 & \text{otherwise} \end{cases}$$

$$R^{DS-RL-3} = \begin{cases} \sqrt{maxFC} \cdot (1 + s_{\text{le}}) & \text{if } f(x_{\text{new}}) < f(x_{\text{best}}), \\ 0 & \text{otherwise} \end{cases}$$

Normalization

[Andrychowicz et al., 2020] found that state feature normalization is crucial. We standardize each feature x with the z-score, i.e. subtract the empirical mean $\bar{x}$ and divide by the sample standard deviation $\widehat{SD}$ or a small constant to ensure numerical stability.

$$x' = \frac{x - \bar{x}}{max(\widehat{SD}, 10^{-6})}$$

They also recommend to clip the features in the interval $[-|x_{max}|, |x_{max}|]$ for robustness. x_{max} is a hyperparameter.

The empirical mean $\bar{x}$ and unbiased sample standard deviation $\widehat{SD}$ are updated after each step with Welford's online algorithm [Knuth, 1997]:

$$\bar{x}_k = \begin{cases} x_1 & \text{if } k = 1, \\ \bar{x}_{k-1} + \frac{x_k - \bar{x}_{k-1}}{k} & \text{if } k > 1 \end{cases}$$

$$\widehat{SD}_k^2 = \begin{cases} 0 & \text{if } k = 1, \\ \frac{(x_k - \bar{x}_{k-1})(x_k - \bar{x}_k)}{k-1} & \text{if } k = 2, \\ \frac{\widehat{SD}_{k-1}^2}{k-2} + \frac{(x_k - \bar{x}_{k-1})(x_k - \bar{x}_k)}{k-1} & \text{if } k > 2 \end{cases}$$

Feature normalization can also be achieved by providing lower and upper bounds for the features (hyperparameters x_{min} and x_{max}), as it is done in [Kletzander and Musliu, 2023]. Note that the two hyperparameters are set in advance, thus the normalized values x' are not necessarily in the interval $[0, 1]$.

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Network architecture

We use multi-layer perceptrons to parameterize the action-value function $Q(a, s)$. The number of parameters depends on the following hyperparameters:

- number of hidden layers (L)
- size of hidden layers
- batch size (b)
- replay buffer size ($|B|$)
- activation function (σ)
  - ReLU: The Rectified Linear Unit activation function leaves positive values unchanged and maps all non-positive values to 0.

$$\text{ReLU}(x) = \max(0, x).$$

  - tanh: The hyperbolic tangent function maps the values on the interval $[-1, 1]$.

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Deep Q-network configuration

- **update rule**: The three variants that we use have differences in stability of learning and model complexity.
  - Online Q-Network (Online DQN): Only the online network is used to update the model and select actions.
  - Target Q-Network (Target DQN): This variant utilizes a copy of the online network (target network) to stabilize training.

  - Double Deep Q-Network (DDQN): Like Target Q-network, but a modified update rule that reduces overestimation is used.

- **C**: this hyperparameter controls the target network update frequency, i.e. every C steps/episodes, the target network is updated.

- **learning starts**: Learning and model updates start after this waiting period (in milliseconds). Before that, only random action selection is performed. The idea is to reduce bias and exploitative behavior through the first few action selections and to fill the replay buffer before the first updates.

- **greedy selection**: In the greedy case of ε-greedy, we allow the agent to choose between the argmax function (which is common for ε-greedy) or, alternatively, the softmax function can be used to sample the next action, which is a smooth approximation of the argmax function.

Action space

Besides the default action space with $\alpha = \beta = 0.2$, we allow actions to be selected from the extended action space with $\alpha, \beta \in \{0.1, 0.2, 0.6\}$. We call the hyperparameter for this decision **action space**.

4.1.3 Pre-training of DS-RL

We will test the ability of our deep reinforcement learning hyper-heuristic DS-RL to memorize information of the search from training instances of the same or of other problem domains. There is a research field that covers this task, known as lifelong reinforcement learning, first proposed by [Thrun and Mitchell, 1995], who showed that memorizing knowledge from similar training tasks will help the agent to learn new problems with less effort.

We will pre-train the hyper-heuristic in an online fashion. While solving the training instances, the hyper-heuristic is trained with a pre-training strategy known from lifelong reinforcement learning. We face some common issues of lifelong and deep reinforcement learning. These include catastrophic forgetting, which is the inability of the model to memorize knowledge from previous domains, instances or information about the search from a few steps ago. This is related to the choice of the learning rate, to the replay buffer size, the mini-batch size, and the number of neurons and layers, i.e. a sub-optimal learning setup has great influence on the generalizability of the hyper-heuristic.

Pre-training a double deep Q-network for hyper-heuristics has been successfully done in [Zhang et al., 2022], but they do not have a domain barrier in their hyper-heuristic. This means that the hyper-heuristic also uses domain-specific features for updating their policy, whereas strict cross-domain selection hyper-heuristics only have information about the search itself, i.e. objective function value, time and information about the selection policy.

We will use a concept known from lifelong reinforcement learning: progressive neural networks.

Progressive neural network

We design a progressive neural network that is suited for the problem domains in HyFlex but can be extended with new domains or used for inference on new domains. The architecture is mainly based on [Rusu et al., 2016], who demonstrated the superior performance of the proposed progressive neural network in comparison to baselines for a variety of reinforcement learning benchmarks. The basic idea is to evade catastrophic forgetting by adding new blocks for the neural network for each new task (in our case problem domain). The Progressive network is then extended with a new column of blocks that has lateral connections from previous tasks that are in columns that were learned before.

The neural network has L hidden layers. At layers $i = 1, \ldots, L$, the parameters $\theta_i^{(1)} \in \mathbb{R}^{n_i \times n_{i-1}}$ are learned for task 1. When we switch to task 2, the next column of layers with parameters $\theta_i^{(2)} \in \mathbb{R}^{n_i \times n_{i-1}}$ are trained and the previously trained parameters are "frozen", i.e. not updated anymore until we train this task again. Lateral layers are used to transfer the information from task 1 to task 2, which are also learned with the layer of task 2. The parameters $U_i^{(k:j)}$ are responsible for the lateral relationship between previous tasks j and the new task k with $j < k$.

For task k, the output of each layer i after applying the activation function σ can be denoted as:

$$h_i^{(k)} = \sigma\left(\theta_i^{(k)} h_{i-1}^{(k)} + \sum_{j<k} U_i^{(k:j)} h_{i-1}^{(j)}\right)$$

For the non-linear activation function σ, either ReLU or tanh is used. The network receives h_0 as input, which are in our case the transformed search states, i.e. raw states after feature extraction and normalization.

Like in [Rusu et al., 2016], we will account for overlap or adversarial relationships between tasks by introducing non-linear lateral connections. We use a slightly simplified version of [Rusu et al., 2016], where for each task j, a factor $\alpha^{(j)}$ is weighting the resulting activation:

$$h_i^{(k)} = \sigma\left(\theta_i^{(k)} h_{i-1}^{(k)} + \sum_{j<k} \sigma\left(U_i^{(k:j)} h_{i-1}^{(j)} \alpha^{(j)}\right)\right)$$

All explanations leave out the bias term for better readability, but each linear layer has a bias term included.

This approach enables the network to memorize information of the search from previous domains without the danger that learned features are modified if layers are fine-tuned across different tasks. Also the weighting factor $\alpha^{(j)}$ can serve as an indicator if the learned features from domain j can lead to better performance in different domains.

In Figure 4.3, a progressive network architecture with 2 hidden layers and $|D|$ tasks is shown. The linestyle of the arrows between the layers indicates if the parameters are frozen (dashed lines) or learned (solid lines). In the illustration, inference with the column "final" is shown, where the parameters from its own column, the lateral connections from all other columns to the "final" column $U_i^{(final:1)}, \ldots, U_i^{(final:|D|)}$, and the weighting factors $\alpha = \alpha^{(1)}, \ldots, \alpha^{(|D|)}$ are updated.

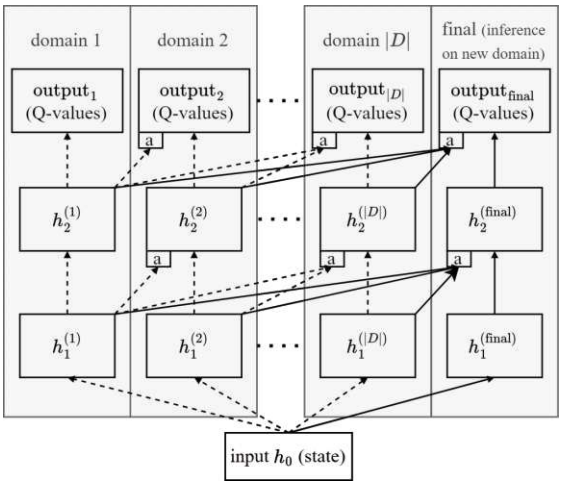

Figure 4.3: Progressive neural network architecture.
Based on Fig. 1 in [Rusu et al., 2016].

As the "final" column in this setting requires a lateral connection for each of the source tasks, we decided to modify the final layer to be more compact and to have less parameters to train when the model runs on a new problem. So for the "final" column, we sum the activations of all columns from the previous layer weighted by the weighting factors $\alpha = \alpha^{(1)}, \ldots, \alpha^{(|D|)}$, and the activation of the "final" column $h_{i-1}^{(\text{final})}$ with weight 1.

$$h_i^{(\text{final})} = \sigma\left(\theta_i^{(\text{final})}\left(h_{i-1}^{(\text{final})} + \sum_{j<k} h_{i-1}^{(j)}\alpha^{(j)}\right)\right)$$

In [Rusu et al., 2016], transfer analysis is conducted with two metrics that introduce random perturbation into layers and measure the impact on the performance. One metric, the Average Perturbation Sensitivity (APS), measures the global effects of this noise and the other one, the Average Fisher Sensitivity (AFS) captures local changes. We will focus on the learning effect and how the scores are affected throughout learning. A full

transfer analysis is out of the scope of this work, therefore, we lie the focus on training and validation curves.

For HyFlex, we know the available domains, so we create the model with 6 domain columns (in the order: MaxSAT, Bin Packing, Personnel, Flowshop, TSP, VRP) and one "final" column already in advance.

The training and evaluation procedure looks like this: We start with the initial model and apply one training run, which updates the model's first column on one training instance of the first domain (MaxSAT), then we iteratively go through all other columns with one training instance for all other domains (Bin Packing, Personnel, Flowshop, TSP, VRP). After this, an evaluation run is carried out, which - similary to the training run - evaluates one instance per domain using the "final" column which is always reinitialized for inference, takes the pre-trained layers from the other domains and the weights of the "final" column are only online learned. With this setup, we can also follow the training procedure and how the learning and validation curves look like. We have 5 splits for both training and validation, so after these 5 rounds, the model will train on already seen instances again. So we can also draw conclusions about how the model remembers instance specific information.

An initial experiment for DS-RL with a progressive neural network architecture is conducted in Subsection 5.5.3.

4.2 Multi-agent PPO State Learning: A Proximal Policy Optimization Hyper-heuristic

In this chapter, we propose another hyper-heuristic algorithm based on LAST-RL [Kletzander and Musliu, 2023] and Proximal Policy Optimization (PPO). We call this hyper-heuristic **maPPOs** (**m**ulti-**a**gent **P**roximal **P**olicy **O**ptimization **s**tate Hyper-heuristic).
As the name already suggests, we also allow multiple agents for this hyper-heuristic, where each of these agents acts on another solution but with the same policy. They are updated jointly with the Proximal Policy Optimization algorithm in actor-critic-style (see [Schulman et al., 2017]).

4.2.1 Proximal Policy Optimization algorithm

Learning is done with Proximal Policy Optimization. This is a policy gradient method that is based on trust region policy optimization (TRPO) [Schulman et al., 2015a] which is known for its data efficiency and reliable performance. The advantage of PPO is that it uses only first-order optimization which results in better sample efficiency and robustness. The main novelty of PPO was the objective with clipped probability ratios which delivers a more pessimistic estimate of the policy.

Existing policy gradient methods estimate policy gradient and plug it into stochastic gradient descent (SGD). [Schulman et al., 2017] argue that multiple steps of optimization are not justified because they often lead to destructively large policy updates. Therefore, Trust Region Methods were introduced. They minimize the surrogate objective with a constraint on the size of the policy update, i.e. the Kullback Leibler divergence between π_{old} and π_{new}. Trust Region Policy Optimization (TRPO) is a reformulation that results in an unconstrained optimization problem with a parameter β that controls the penalty on the KL divergence.

In the model update, we first have to estimate the advantage function A_t^π, which is defined as:

$$A_t^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$$

For the estimation $\widehat{A_t}$, there are two options:

- either using the parameterized value function $V_\psi^{\pi_\theta}$ and Q^{π_θ} which is the expected return under policy π_θ:

$$\widehat{A_t}(s_t, a_t) = Q^{\pi_\theta}(s_t, a_t) - V_\psi^{\pi_\theta}(s_t)$$

- or Generalized advantage estimation (GAE) [Schulman et al., 2015b], which reduces the variance with some cost of bias. The estimation is similar to TD(λ) estimation in [Sutton and Barto, 2018]:

$$\widehat{A_t}^\gamma = \sum_{i=0}^{\infty} (\lambda\gamma)^i \delta_{t+i}^V$$

  with

$$\delta_{t+i}^V = r_{t+i} + \gamma V(s_{t+i+1}) - V(s_{t+i})$$

  For this estimator, in [Schulman et al., 2015b], theoretic results to limit the introduced bias are provided. For $V(s_t)$, again V^{π_θ} is employed.

Then, with slight adaptation to the objective of TRPO

$$J^{\text{TRPO}} = \mathbb{E}\left[\rho(\theta)\widehat{A}_{\theta_{\text{old}}}(s, a)\right]$$

with

$$\rho(\theta) = \frac{\pi_\theta(a \mid s)}{\pi_{\theta_{\text{old}}}(a \mid s)} \quad \text{(probability ratio between the current and old policy)}$$

we arrive at the clipped objective function:

$$J^{\text{clip}}(\theta) = \mathbb{E}\left[\min\left(\rho(\theta)\widehat{A}_{\theta_{\text{old}}}, \text{clip}(\rho(\theta), 1 - \varepsilon^{\text{clip}}, 1 + \varepsilon^{\text{clip}})\widehat{A}_{\theta_{\text{old}}}\right)\right]$$

The clip() function sets $\rho(\theta)$ to $1 - \varepsilon^{\text{clip}}$ if the value is smaller, or to $1 + \varepsilon^{\text{clip}}$ if it is larger. This should make learning more stable and give the ability to control the influence of very large rewards on the policy update. If parameters of actor (θ) and critic (ψ) are not updated jointly, then in each update, θ is updated via $J^{\text{clip}}(\theta)$ and ψ via $L^V(\psi) = (V_\psi(s_t) - V^{\pi_\theta}(s_t))^2$, which is the squared error loss for the critic network.

In case of a shared parameter setting between actor and critic networks, the loss for the critic is integrated into J^{clip} and additionally, the entropy for the policy distribution can be added which gives us the ability to control how evenly distributed the policy distribution is and thus how explorative the agent will act:

$$J^{\text{clip}}(\theta, \psi) = \mathbb{E}\left[J^{\text{clip}}(\theta) + c_1 L^V(\psi) - c_2 H_\pi(\mathcal{A})\right]$$

where

- $H_\pi(\mathcal{A}) = -\sum_{a \in \mathcal{A}} \pi(a \mid s_t, \theta) \log(\pi(a \mid s_t, \theta))$ is the entropy of the policy distribution,
- c_1 and c_2 are the corresponding weights that are configured as hyperparameters.

The objective function is optimized with stochastic gradient descent (we use Adam optimizer). A description for actor-critic-style PPO updates is in [Schulman et al., 2017]. We will give more details about the model update in Algorithm 4.3 which is adjusted to our task and terminology.

4.2.2 Components of maPPOs

The hyper-heuristic consists of multiple blocks which we cover in the following algorithms (Algorithm 4.1 to Algorithm 4.3). The terminology that was introduced in Section 2.5 will be utilized.

Besides the usual input for hyper-heuristics, the domain and instance to solve, we have a broad range of hyperparameters that need to be set in advance. These hyperparameters and design choices are described in Subsection 4.2.3. In Section 5.6, we conduct hyperparameter tuning and experiments with the design choices.

Initialization

If we have all parameters set, the hyper-heuristic starts with the initial steps. The initialization procedure of maPPOs is shown in Algorithm 4.1 in lines 1 to 10. First, we need to initialize the action space $\mathcal{A}$ which is the set of LLHs that can be selected. If we do not use the search parameters α (intensity of mutation) and β (depth of search), we leave them at their default value. Otherwise we extend the search space with variations of actions that use at least one of the search parameters, i.e. in addition to the default value 0.2, we also choose 0.1 and 0.6 for α and β, as it was done earlier in this work.

Algorithm 4.1: maPPOs hyper-heuristic

Data: domain d, instance i, timelimit T, hyperparameter configuration Λ

$\mathcal{A} \leftarrow$ createActionSpace(d, α, β)
$\theta, \psi \leftarrow$ initialModelParameters() or loadPretrainedModel()
for $j = 1, ..., n_{agents}$ **do**
  $s_0^j \leftarrow$ initialState()
  $x^j = (x^j_{\text{new}}, x^j_{\text{current}}) \leftarrow$ initializeSolutions()
  if $f(x^j_{current}) < f(x_{best})$ **then**
    $x_{\text{best}} \leftarrow x^j_{\text{current}}$ (initialise overall best solution)
  end
end
$x = (x_{\text{best}}, \underbrace{x^1_{\text{new}}, x^1_{\text{current}}}_{\text{solutions agent 1}}, \ldots, \underbrace{x^{n_{agents}}_{\text{new}}, x^{n_{agents}}_{\text{current}}}_{\text{solutions agent } n_{agents}}, \underbrace{x^c_1, \ldots, x^c_{n_c}}_{\text{crossover slots}})$
while $time < T$ **do**
  $c \leftarrow$ chainLength() (see **Episode Length**)
  for $j \leftarrow 1$ ***to*** n_{agents} **do**
    for $t \leftarrow 0$ ***to*** c **do**
      $a_t \leftarrow$ **selectLLH**(s^j_t, π_θ) (see Algorithm 4.2)
      $x^j_{\text{new}} \leftarrow$ executeLLH(a_t, x^j_{new})
      $r^j_t \leftarrow R(f(x^j_{\text{new}}), time)$ (see **Reward function** of DS-RL)
      $s^j_{t+1} \leftarrow$ updateState$(s^j_t, a_t, f(x^j_{\text{new}}), time)$
      $\tau^j_t \leftarrow (s^j_t, a^j_t, r^j_t, s^j_{t+1})$
      $B^j.add(\tau^j_t)$ (store transition in buffer)
      if $f(x^j_{new}) < f(x^j_{current})$ **then**
        $x^j_{\text{current}} \leftarrow x^j_{\text{new}}$
        if $f(x^j_{new}) < f(x_{best})$ **then**
          $x_{\text{best}} \leftarrow x^j_{\text{new}}$
        end
        break (improvement ends the episode)
      end
      $x^j_{\text{current}} \leftarrow x^j_{\text{new}}$
      $t \leftarrow t + 1$
    end
  end
  $\theta, \psi \leftarrow$ **updateModel**(B, θ, ψ) (see Algorithm 4.3)
  if *restart*$(x, time)$ **then**
    reset(s_t, x) (see **Reset behavior** of LAST-RL)
  end
end
return x_{best}

Algorithm 4.2: maPPOs: selectLLH

Data: Normalized state features $\phi(s_t)$, actor network θ

1 $a_t \leftarrow$ sample from $\pi(a \mid s_t, \theta)$
(use categorical policy distribution $\pi(a \mid s, \theta)$, see **Policy** (4.2.3))

Then we initialize the model which consists of a policy network π with parameters θ and a value network V with parameters ψ, in the literature also known as actor and critic. These two networks are multi-layer perceptrons with L hidden layers and an activation function after each hidden layer (ReLU or tanh). The model parameters are initialized or pre-trained blocks are loaded. θ corresponds to the actor and ψ to the critic network. In our setting, the two models influence each other in the weight updates but they have two separate loss functions. There are also other variants of Proximal Policy Optimization that use a joint loss, see [Schulman et al., 2017], which is basically a weighted sum of the two objectives of the actor and the critic. We will later explore both variants.

Every agent gets an initial solution assigned by calling the solution initialization procedure which is customized for every domain. For each domain, we provide a short description of the initialization procedure and references to more details in the technical reports in Subsection 2.3.1. For every agent, the initialization procedure is called 3 times and the best solution is chosen. Here we also initialize the 5 crossover slots which are filled up with the agents' solutions. In x, we store the current solutions: one slot is for the overall best x_{best}, then for each agent j we store its new solution x^j_{new}, the best solution found so far x^j_{best} and the current best solution x^j_{current}, which is the solution where the episode started and to which the agent is also reset after an episode without improvement.

Main loop

The main loop (see Algorithm 4.1, lines 11 to 36) iterates until the timelimit T is reached. However, it is essential in HyFlex to check regularly for the timelimit because a solution is only valid if it is confirmed that it was found before the timelimit was reached. So it is desired to include multiple checkpoints, which are not relevant for the pseudocode.

The agents sample actions in the environment with the same policy π_θ. The action selection is shown in Algorithm 4.2, which is basically the sampling of the next action from the policy distribution. How this distribution looks like and the different options for action selection policies (including mechanisms that include Iterated Local Search) are given in 4.2.3. For each action a_t that is chosen, we store the current state s_t and the resulting reward r_t. This is done until the first improvement or until the sequence length is reached. The sequence length is determined with the Luby sequence which is also used in the hyper-heuristics RL ([Mischek and Musliu, 2022]), LAST-RL ([Kletzander and Musliu, 2023]), and DS-RL (4.1). The iterative definition of the Luby sequence is provided in 3.1.2.

Within one sequence of actions, any worsening is accepted. In case of improvement, the

episode is terminated, we update the agent's best solution x^j_{best}. We also check if it improved the overall best solution x_{best}. At the end of a sequence (without having found improvement), the agent is taken back to its current best solution x^j_{current}. We add the latest transition τ_t to the buffer. A parameter (number of updates) determines how often in the total runtime, the model parameters are updated. This determines, how long each agent has time to sample actions. After each episode, it is checked if the time for this agent is up. Note that we limit the Luby sequence to 512 to avoid possible out-of-memory issues. Another approach would be to sample mini-batches from B and replace the oldest transitions with new ones, as it is done in the equal selection buffer of DS-RL.

We do this for every agent and after that, a learning step updates our model parameters θ and ψ.

Model update

The update of the actor-critic model is based on [Schulman et al., 2017], except that we do not sample mini-batches but we use the whole previous episode (of each agent).

In on-policy algorithms, model updates are typically derived from samples of the current policy. Whereas off-policy methods also allow the use of trajectories that were sampled under different policies, i.e. they can make use of experience replay. Using older trajectories in on-policy learning could introduce bias and reduce the effectiveness of updates, as they differ in the policy that they were sampled with.

For the model update, we first estimate the advantage function (optionally with generalized advantage estimation) and then update for K epochs either with a joint objective or with individual objectives for actor and loss. The pseudocode in Algorithm 4.3 illustrates the procedure.

4.2.3 Design choices for maPPOs

Advantage function

The implemented Proximal Policy Optimization algorithm estimates the advantage function at each step of the weight update and serves as the main component in the actor loss. The easiest way to estimate the advantage is by subtracting the observed discounted rewards by the estimation of the critic network. Another opportunity is to use generalized advantage estimation (GAE) which reduces the variance with some cost of bias. The two variants are shown in 4.2.1 and the main difference is the bias-variance trade-off that can be controlled for GAE with the additional parameter λ. For all later experiments, this parameter is set to 0.95.

The advantage is estimated once before each model update and then remains unchanged. Before the advantage estimation is passed to the loss function, it can be beneficial to standardize by subtracting its mean and scaling by the standard deviation[1].

[1] a small constant is added to the standard deviation for stability

Algorithm 4.3: maPPOs: updateModel (using actor-critic-style PPO)

Data: multi-agent buffer B (after one episode for each agent), actor network θ, critic network ψ, actual chain length for j-th agent c^j

for $j \leftarrow 1$ ***to*** n_{agents} **do**
  for $t \leftarrow 0$ ***to*** c^j **do**
    $\widehat{A_t^j} \leftarrow$ estimate advantage function (using buffer of j-th agent B^j)
  end
end
for *epoch* $1 \rightarrow K$ **do**
  if *use joint loss* **then**
    $J^{\text{clip}}(\theta, \psi) \leftarrow$ estimate joint loss with current θ and ψ
    $\theta, \psi \leftarrow$ update actor and critic with one step of SGD with $J^{\text{clip}}(\theta, \psi)$
  else
    $L^V(\psi) \leftarrow$ estimate critic loss with current ψ
    $\psi \leftarrow$ update critic with one step of SGD with $L^V(\psi)$
    $J^{\text{clip}}(\theta) \leftarrow$ estimate surrogate actor loss with current θ and θ_{old}
    $\theta \leftarrow$ update actor with one step of SGD with $J^{\text{clip}}(\theta)$
  end
end
return θ, ψ

Number of agents

The number of agents (or in [Andrychowicz et al., 2020] referred to as number of environments) determines the number of parallel agents with the same policy interacting on different solutions. The use of multiple agents could stabilize learning as the updates are from potentially diverse stages of the search. In case of a single agent, one could face the issue of being stuck in a local optimum and not producing any rewards until the next model update. The next model updates are then solely learning from transitions without any improvement which destabilizes training.

PPO loss function

As described in 4.2.1, there are two options for the loss function that is used for the model updates:

- separate updates for actor and critic with the clipped objective for the actor

$$J^{\text{clip}}(\theta) = \mathbb{E}\left[\min\left(\rho(\theta)\widehat{A}_{\theta_{\text{old}}}, \text{clip}(\rho(\theta), 1-\varepsilon^{\text{clip}}, 1+\varepsilon^{\text{clip}})\widehat{A}_{\theta_{\text{old}}}\right)\right] - c_2 H_\pi(\mathcal{A})$$

  and the critic loss $L^V(\psi) = (V_\psi(s_t) - V^{\pi_\theta}(s_t))^2$,

- the joint surrogate loss for actor and critic:

$$J^{\text{clip}}(\theta, \psi) = \mathbb{E}\left[J^{\text{clip}}(\theta) + c_1 L^V(\psi)\right].$$

Note that we shifted the entropy of the policy distribution to the actor loss so that we can control the exploration-exploitation trade-off also for the separate update strategy.

Acceptance mechanism

All new solutions are accepted within one episode, so actually the acceptance mechansim is to accept new solutions in all cases. But we return to the current best solution after a number of steps determined by the Luby sequence or alternatively, the number of steps per episode can be fixed.

Episode length

The actual episode length depends on if a termination criterion ends the current chain or if the chain length is extended (see Episode length of LAST-RL in 3.1.2). However, usually the episode length is sampled from the Luby sequence or is fixed beforehand.

Network architecture

A large scale empirical study [Andrychowicz et al., 2020] summarizes important findings and gives recommendations for on-policy methods in deep reinforcement learning, with a major focus on actor-critic networks with Proximal Policy Optimization. The study only covers problems with a continuous action space, but the findings are also relevant for the discrete case.

The main finding of the paper was that the initialisation of the actor network has a high impact on the performance. Before the first model updates, we wish to have action distribution centered around $\frac{1}{|\mathcal{A}|}$ with low variance. This issue can be solved by focusing on the initialisation of the last layer of the actor network. Instead of the default initialisation, i.e. parameters sampled from a normal distribution $\mathcal{N}(\mu = 0, \sigma^2 = 0.01)$, we divide all values by 10, resulting in the weights of the last actor layer sampled from $\mathcal{N}(\mu = 0, \sigma^2 = 0.0001)$.

Besides the initialization, they also recommend in [Andrychowicz et al., 2020] to use tanh activation and discourage to use ReLU. But we keep in mind that tanh suffers more from the vanishing/exploding gradient problem than ReLU which better controls the magnitude of the gradients.

Another finding in [Andrychowicz et al., 2020] is that the critic network should be wider, i.e. have more units in the hidden layers, than the actor network. It is recommended to tune the width of the actor network and to avoid shared layers between actor and critic networks.

Number of model updates (n_{updates})

This hyperparameter determines the time that each agent samples actions before the next model update. It is important to recognize that some problem domains require fewer model updates, benefit from a lot of search steps and will lose search time due to too frequent model updates, e.g. Bin Packing. Whereas other domains, like Personnel Scheduling, require smaller batches and more model updates because the execution of actions takes longer. This is due to the characteristics of the problem domain, as the design of the hyper-heuristic can not influence the duration of the search operators, except with the search parameters α and β. Consequently, the model updates take short time in comparison to the actions. The reason for this is the higher complexity of the problems and their heuristic search operators. Other domains might profit from more frequent model updates because they require knowledge gained by the search in the recent steps.

Additionally, we account for memory limitations, thus limiting the trajectories to a length of $c = 512$, and consider how a different number of model updates could possibly influence other hyperparameters and require adjustments (e.g. learning rates, number of agents or termination criteria).

The available time that each agent has before the next model update is then $\frac{T}{n_{\text{updates}} \cdot n_{\text{agents}}}$. This timelimit is checked after each episode, thus only serves as a lower bound. This simple compromise addresses the two issues previously mentioned. In problem domains where low-level heuristics are faster, updates occur after more steps than in domains with more complex and time-consuming operators. Additionally, with a reasonable choice for this parameter, the buffer size will remain manageable and memory issues are prevented.

Action space

We enriched the action space regarding the two search parameters α (intensity of mutation) and β (depth of search). In addition to the default value of 0.2, we also have a more conservative option 0.1 and one more perturbative option 0.6. In Table 2.1, you can see for each domain, how many low-level heuristics use search parameters.

The selection policy, that will be described in the following, also has an impact on the available actions in certain stages: Similar as two approaches ([Adubi et al., 2021] and [Kletzander and Musliu, 2023]), we add an Iterated Local Search (ILS) component. In [Adubi et al., 2021], they created a set of configurations which consist of a pair of two operators from the set of mutation and ruin-recreate heuristics. This pair is then called subsequently in the perturbation phase, followed by the intensification phase, where local search heuristics are called until no further improvement is found. In [Kletzander and Musliu, 2023], a probability distribution decides which type of heuristic should be called in the next step, based on the type of the last heuristic and on the average number of successes of the heuristic types.

We first try a similar approach to [Adubi et al., 2021] by using the same perturbation and intensification steps, but without modifying the action space per se. We modify

the output of the actor network so that there are only non-zero probabilities for actions of the type that we wish to select based on ILS. In the following, we will discuss four selection strategies, including the two ILS components and how we integrated them into our exploration-exploitation strategies.

Policy

- **ε-greedy**

  The ε-greedy action selection strategy selects the next action with the highest state-action value, our in our case with the highest output of the actor network. With probability ε, the next action is selected randomly [Sutton and Barto, 2018]

$$\pi(a \mid s, \theta) = \begin{cases} 1 - \varepsilon & \text{if } a = \text{argmax}_{b \in \mathcal{A}} h(s, b, \theta), \\ \varepsilon \frac{1}{|\mathcal{A}|} & \text{otherwise,} \end{cases}$$

  or with the softmax action selection strategy:

$$\pi(a \mid s, \theta) = \begin{cases} 1 - \varepsilon & \text{if } a = \text{argmax}_{b \in \mathcal{A}} h(s, b, \theta), \\ \varepsilon \frac{e^{h(s,a,\theta)}}{\sum_{b \in \mathcal{A}} e^{h(s,b,\theta)}} & \text{otherwise.} \end{cases}$$

- **Softmax action selection**

  The output layer of our actor network is a softmax layer that returns the policy distribution π and the next action is sampled from this distribution. In [Sutton and Barto, 2018], they refer to parameterized numerical preferences for each state–action pair, i.e. $h(s, a, \theta) \in \mathbb{R}$. The exponential softmax distribution (also known as Gibbs and Boltzmann distribution) is:

$$\pi(a \mid s, \theta) = \frac{e^{h(s,a,\theta)}}{\sum_{b \in \mathcal{A}} e^{h(s,b,\theta)}}$$

- **Softmax action selection with heuristic type masking based on ILs strategy of [Adubi et al., 2021]**

  This strategy is basically the same as softmax action selection but in each step, the action space is masked with the ILS strategy of [Adubi et al., 2021].

  Their strategy has predefined perturbation configurations which refer to the application of two subsequent perturbation heuristics ("double shaking"), including ruin-recreate, mutation and crossover operators. These configurations are then selected based on utility values. The perturbation stage is followed by an intensification stage, where local search operators are applied until no further improvement can be found.

If we would adapt our action space to have these configurations instead of the heuristic operators, we would have a too large action space. For example for the domain MaxSAT, we have $6+2+1=9$ perturbation operators, resulting in $9^2=81$ configurations. Also the separation of perturbation and intensification stage is difficult to realize.

We propose a straightforward solution to this problem: we make use of our actor network and add a mask before the softmax layer that sets the probability for local search operators in the perturbation stage to zero, and the same in reverse, sets the probability of perturbation operators in the intensification stage to zero. Hence, only action types that are allowed by the current ILS-stage (perturbation or intensification) are allowed.

$$h'(s,a,\theta) = \begin{cases} h(s,a,\theta) & \text{if (perturb. stage} \wedge \text{type}(a) \in \{\text{ruin-r., mutation, crossover}\}) \\ & \quad \vee (\text{intensification stage} \wedge \text{type}(a) \in \{\text{local search}\}) \\ 0 & \text{otherwise} \end{cases}$$

After the ILS-mask, we apply the softmax function to obtain the distribution:

$$\pi(a \mid s,\theta) = \frac{e^{h'(s,a,\theta)}}{\sum_{b\in\mathcal{A}} e^{h'(s,b,\theta)}}$$

Note that this is a naive approach that also has the risk of deterministic behavior, but we also have to mention that this mask h' will not influence the learning directly. The mask is only used for the action selection and will prevent the selection of certain heuristic types without modifying the outputs of the actor network that are used for model updates.

- **ε-greedy with ILS based on [Kletzander and Musliu, 2023]**

  In [Kletzander and Musliu, 2023], a probability distribution based on ILS is proposed. The type of the next action is sampled from this distribution in the exploration case of their ε-greedy policy. A detailed description of this distribution has been given in Subsection 3.1.2.

  We adapt this strategy as we do not select the operator within the chosen heuristic type randomly but we sample from the policy distribution after applying the resulting ILS-mask.
  In the greedy case, we still sample from the policy distribution with softmax action selection.

Reward function

The used reward functions were already described in the design choices of DS-RL (see 4.1.2).

Normalization

Again, the used normalization techniques were already described in the design choices of DS-RL (see 4.1.2).

CHAPTER 5

Experiments and discussion

The previous sections were introducing the enhancements that we propose for the existing LAST-RL hyper-heuristic and two hyper-heuristics that also build on LAST-RL, mainly with its reset, restart and sequence length criteria and the extensive search state. We described a lot of design choices, addressed the issues that they target, and motivated their possible influence on the performance.
In this section, first, we give an overview of the technical setup of all experiments and the requirements for the implementation in Section 5.1. Then we describe the experiments for LAST-RL for feature selection (Section 5.3) and for the extension of the action space (Section 5.4). Then in Section 5.5, we have the experiments of DS-RL that include hyperparameter tuning, the pre-training of a progressive neural network to test the ability of transfer learning, and further experiments for the design choices. Similar to that, experiments for maPPOs in Section 5.6 first determine the best model configuration via hyperparameter tuning and then the previously described design choices will be evaluated.

5.1 Technical setup and implementation details

The hyper-heuristics were implemented in the HyFlex framework in Java using OpenJDK 21.0.2.0. The handling of arrays for Deep Learning and the training of parameters uses Deep Java Library (version 0.22.1), a Java API.

All experiments that are used for evaluation were performed single-threaded on a computing cluster running Ubuntu 22.04.2 LTS on a GNU/Linux 6.8.0-40-generic x86_64 kernel. The cluster employs 13 benchmarking nodes where each node has 2x Intel Xeon CPU E5-2650 v4 (12 cores @ 2.20GHz, without hyperthreading and 256GB RAM). HyFlex provides a script that determines the timelimit for our setup that is allowed to be comparable with the original evaluation of CHeSC 2011. For our setup, 392 seconds is the timelimit for a single run.

The setup, calls and post-processing of hyperparameter tuning (SMAC3, version 2.2.0 [Lindauer et al., 2022]) and pre-training runs, were handled with python scripts (Python 3.10.12 for post-processing on local machine and Python 3.12.3 on cluster). For visualizations, we used the R package ggplot2 (3.5.1).Statistical tests were conducted in R using the packages stats (4.4.3) and scmamp (0.3.2) [Calvo and Santafé, 2016].

5.2 Evaluation methods

The following experiments are evaluated using parallel boxplots. This type of visualization allows immediate comparison of the group medians, both across domains and within domains. The boxes and whiskers provide additional information about the distributions. We also show notches which give roughly 95% confidence intervals ($\pm 1.58\frac{IQR}{\sqrt{n}}$), see also [McGill et al., 1978]. Not overlapping notches suggest significantly different medians.

The experimental setup for the ablation study for feature selection (Section 5.3) is outlined in Subsection 5.3.1.

For the remaining experiments, we will use the CHeSC 2011 scoring mechanism which rewards 10 points for the best hyper-heuristic, 8 for the second best, and 6, 5, 4, 3, 2 and 1 for the remaining ranked hyper-heuristics per instance. With this scoring mechanism, the top 8 approaches receive points. Sometimes, we compare less than 8 approaches, so we also use a modification of this scoring mechanism: we only award the "winner", i.e. the approach with the lowest objective function value per instance, 1 point. An alternative to compare the results is to replace the objective function values with ranks on instance level. For each set of algorithms being compared, we rank all runs including 31 runs from each algorithm. We assign rank 1 for the best run, 2 for the second-best, and so on.

After ranking the results on instance level for the groups that we compare, we apply Kruskal-Wallis rank sum tests [Kruskal and Wallis, 1952] which are appropriate for unpaired samples. It is a non-parametric statistical test, which suits our task because distributional assumptions are difficult to be fulfilled with the objective function values of different domains. The test statistic is:

$$H = \frac{\frac{12}{N(N+1)} \sum_{i=1}^{C} \frac{R_i^2}{n_i} - 3(N+1)}{1 - \sum_{i=1}^{C} \frac{(t_i^3 - t_i)}{(N^3 - N)}} \sim \chi^2_{(C-1)}$$

where:
C is the number of groups,
N total number of samples,
n_i number of samples of i-th group,
R_i sum of the ranks of i-th group,
t_i number of ties in i-th group.

The test is usually applied for multiple groups but is also suited for post-hoc tests on pairs of groups. In case of post-hoc tests, we account for multiple comparisons with Bonferroni correction, which corrects the significance level α by the number of tests m, i.e. $\alpha \leftarrow \frac{\alpha}{m}$.

5.3 Ablation study for feature selection

We described the features of the existing search state of LAST-RL and proposed new features in Section 3.2. All features were assigned to 8 groups, see Table 3.1. Testing all possibilities with these 8 groups would require $2^8 - 1 = 255$ tests. Therefore, we decide to do an ablation study for feature selection, i.e. testing the effect of adding or removing components of the model. In our case, these components are sets of features that are grouped into categories (Ti, Va, He, Ep, Ho). As in [Kletzander and Musliu, 2023], the parameter H for the horizon is set to 10 for all involved features. Note that the features lastAlpha and lastBeta are only used for extended action spaces, and therefore not tested here as we use the default action space.

In the following, the experimental design is explained, then we explain each experiment and report its results.

5.3.1 Experimental design for ablation feature selection study

1. Run experiments

   For each experiment, we have M algorithms that we run on the $N = 37$ instances that were not used for the CHeSC competition because the competition instances will be used for the final evaluation and comparison with state of the art. We run each instance 31 times with a timeout of 392 seconds and calculate the median to get a robust estimate of the algorithms' performance for each instance.

2. Comparison based on CHeSC 2011 ranking scheme

   Before we test for statistical significance, we rank the tested algorithms based on the CHeSC 2011 score. Besides overall performance, we take a closer look at the domains where LAST-RL would not have achieved a top 5 rank in the CHeSC 2011 (MaxSAT and Bin Packing).

3. Omnibus test

   Then, we conduct a Friedman test with Iman and Davenport correction, see [Iman and Davenport, 1980]. This non-parametric method is suited for our task because our data are the objectives from different domains. Therefore, it is difficult to verify distributional assumptions for parametric methods.

   The Friedman test overcomes this issue by assigning ranks to the M algorithms for the N instances. Rank 1 for the best algorithm, rank M for the worst, and for ties, the mean of the corresponding ranks is assigned. Then the null-hypothesis H_0 is

that there is no difference between all algorithms, and the alternative H_1 is that there is a significant difference for at least one pair of algorithms.

The test assumes independent data of two or more groups and paired observations.

The test statistic for the Friedman test is the following ([Demšar, 2006]):

$$\chi_F^2 = \frac{12N}{M(M+1)} \left(\sum_{j=1}^{M} R_j^2 - \frac{M(M+1)^2}{4} \right) \sim \chi_{(M-1)}^2$$

where R_j is the average rank of algorithm j over all instances.

We use Iman and Davenport correction [Iman and Davenport, 1980], which has lower risk of type II error, i.e. it does not tend to be too conservative. Friedman test with Iman and Davenport correction has this test statistic:

$$T_F = \frac{(N-1)\chi_F^2}{N(M-1) - \chi_F^2} \sim \mathcal{F}_{(M-1),(M-1)(N-1)}$$

If we can reject H_0 at the significance level $\alpha = 0.05$, i.e.

$$p = \mathcal{P}\left[\mathcal{F}_{(M-1),(M-1)(N-1)} \geq T_F \mid H_0\right] \leq 0.05$$

, we will proceed with a post-hoc test to identify which pair of algorithms has a significant difference.

4. Post-hoc test

   For post-hoc testing, we use the Nemenyi test to compare all pairs [Nemenyi, 1963, Demšar, 2006]. Two algorithms are significantly different if the difference of their average ranks reaches the critical distance

$$CD = q_\alpha \sqrt{\frac{M(M+1)}{6N}}$$

   where critical values q_α are calculated as the student range statistic $\frac{R_{max} - R_{min}}{s/\sqrt{n}}$ divided by $\sqrt{2}$, with R_{max} and R_{min} being the largest and smallest average ranks and s the pooled sample standard deviation.

5.3.2 Experiment 1.a: Adding feature groups to raw model

We will test for the 5 groups time, value, heuristic, episode and horizon. We compare the *raw* feature set (with only 4 features timeTotal, lastChangeMag, lastHeur and chainProgress) against feature sets that contain the remaining features of the 5 groups with interaction features, and additionally with the *LAST-RL* features. These tests should indicate if the introduction of a group of features is beneficial for the search.

1. Run experiments

We run the experiments with the following 7 feature sets for comparison: *raw*, *LAST-RL*, *raw+Ti*, *raw+Va*, *raw+He*, *raw+Ep*, and *raw+Ho*. The results are first summarized with the formula 1 based ranking scheme of CHesC 2011.

2. Comparison based on CHeSC 2011 ranking scheme

	Total	SAT	BP	PS	FS	TSP	VRP
LAST-RL	242.68	**41.85**	46.25	44.0	**58.0**	29.0	27.0
raw	170.15	31.75	30.0	33.0	24.5	18.0	31.0
raw+Ti	187.96	30.35	29.25	41.5	33.0	**41.0**	12.0
raw+Va	197.11	41.1	33.0	24.0	39.0	25.0	32.0
raw+He	184.41	26.6	31.0	37.5	37.0	25.0	27.0
raw+Ep	179.53	25.75	42.25	39.0	29.5	20.0	25.0
raw+Ho	**244.16**	30.6	**54.25**	**47.0**	45.0	32.0	**36.0**

Table 5.1: Evaluation of Experiment 1.a

The feature set *raw+Ho* performed best among the new feature sets when we consider the total score. The feature set *raw+Ho* also has the highest score in three domains (Bin Packing, Personnel Scheduling and VRP) and - based on the ranking scheme - additionally outperforms the feature set in [Kletzander and Musliu, 2023] in TSP.

3. Omnibus test

We conduct a Friedman test with Iman and Davenport correction to determine whether there is a significant difference in the median score across the HyFlex non-competition instances of the LAST-RL hyper-heuristic with seven different feature sets.

The Friedman test revealed a statistically significant difference, with $T_{F_{6,216}} = 3.6794, p = 0.0017$. Thus, we will proceed with a post-hoc test to identify which pairwise differences are significant.

4. Post-hoc test

The Nemenyi test returns a critical difference of 1.4928 at the significance level $\alpha = 0.05$, with 252 degrees of freedom. This difference was exceeded by two pairs of feature sets. The *raw* feature set was significantly outperformed by *raw+Ho* (CD = 1.5946) and the *LAST-RL* version in [Kletzander and Musliu, 2023] (CD = 1.5676).

In Figure 5.1, the critical difference plot that was proposed in [Demšar, 2006] is shown. In this type of plot, the bold horizontal lines connect feature sets with no statistical difference. A pair of feature sets performs significantly different if they are not connected by at least one horizontal line. The different feature sets are mapped to their average ranking and also the critical distance is shown above.

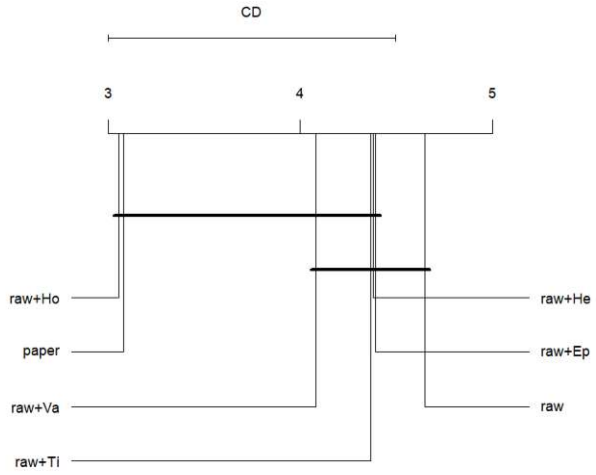

Figure 5.1: Post-hoc pairwise comparison with Nemenyi test for Experiment 1.a

We can conclude that adding features to the raw model is beneficial and for two feature sets even significant differences can be achieved. It is also interesting that these two feature sets (*LAST-RL* and *raw+Ho*) have a rather low critical distance of 0.0270, although these two feature sets differ a lot with all features extracted directly from the objective value not being in *raw+Ho*. The set *raw+Ho* is superior in the domains Bin Packing, Personnel Scheduling and VRP, and *LAST-RL* achieves most points in MaxSAT and Flowshop.

5.3.3 Experiment 1.b: Adding feature groups to raw + Ho model

We will proceed with *raw+Ho*, omit the other combinations with the raw features, and test again by extending this feature set with the remaining groups (time, value, heuristic, episode).

1. Run experiments

Again, 7 feature sets are used for comparison: *raw*, *LAST-RL*, *raw+Ho*, *raw+Ho+Ti*, *raw+Ho+Va*, *raw+Ho+He*, and *raw+Ho+Ep*.

2. Comparison based on CHeSC 2011 ranking scheme

Here, the additional features lead to improved performance on the total score. The feature set *raw+Ho* still has the second highest overall score of the new tested feature sets. The new best feature set is *raw+Ho+Va* which is ranked first in the domains MaxSAT, Bin Packing, and VRP, the remaining domains are dominated by *LAST-RL* (Flowshop) and *raw+Ho+Ep* (Personnel Scheduling and TSP). The *LAST-RL* version is dominated in each domain except Flowshop.

	Total	SAT	BP	PS	FS	TSP	VRP
LAST-RL	225.93	35.58	39.25	41.5	**60.0**	30.0	23.0
raw	154.94	24.5	27.5	31.5	22.5	17.0	30.0
raw+Ho	215.94	24.75	46.25	44.0	42.0	29.0	31.0
raw+Ho+Va	**238.36**	**46.0**	**53.5**	42.0	36.0	25.0	**36.0**
raw+Ho+Ti	181.19	35.08	43.25	29.0	36.5	18.0	22.0
raw+Ho+He	180.9	34.75	27.0	26.5	38.5	31.0	21.0
raw+Ho+Ep	208.73	27.33	29.25	**51.5**	30.5	**40.0**	27.0

Table 5.2: Evaluation of Experiment 1.b

Here, we can conclude that *raw+Ho+Va* provides a strong improvement, especially in the domains MaxSAT and Bin Packing where we strive for improvements. Adding the feature sets *Ti*, *He* and *Ep* to *raw+Ho* do not improve the total score.

3. Omnibus test

We now have the same setting for the omnibus test as in Experiment 1.a, but with extended feature sets. The Friedman test revealed a statistically significant difference, with $T_{F_{6,216}} = 3.0584, p = 0.0068$. Thus, we will proceed with a post-hoc test to identify which pairwise differences are significant.

4. Post-hoc test

The Nemenyi test returns a critical difference of 1.4928 at the significance level $\alpha = 0.05$, with 252 degrees of freedom. The raw feature set was significantly outperformed by raw+He. Apart from that, no significant differences can be obtained, see Figure 5.2.

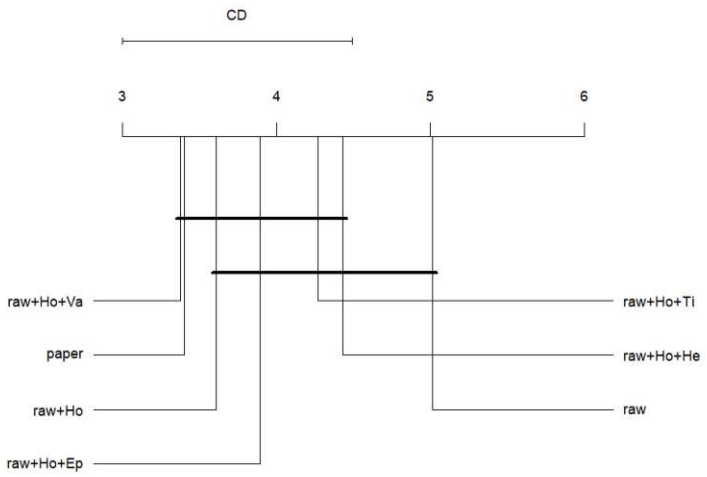

Figure 5.2: Post-hoc pairwise comparison with Nemenyi test for Experiment 1.b

Here, we can conclude that adding another feature group still yields better results than the *raw* model in terms of competition scores. The feature set *raw+Ho* from Experiment 1.a is still performs well in this experiment and also better than *raw* with statistical significance. *raw+Ho+Va* is now the best performing feature set.

5.3.4 Experiment 2.a: Removing feature groups from full model

In addition to the two forward selection steps (Experiments 1.a and 1.b), we perform similar tests where we first run the *full* model and then remove each of the 5 feature groups once (backward selection). With these tests, the intuition is to identify noisy features. Anyways, the results of these tests highly depend on the hyperparameters of LAST-RL (e.g. number of tiles for the tile encoding) and on the increased computational effort by introducing more features.

1. Run experiments

We will compare the following 7 feature sets: *full*, *LAST-RL*, *full-Ti*, *full-Va*, *full-He*, *full-Ep*, and *full-Ho*.

2. Comparison based on CHeSC 2011 ranking scheme

	Total	SAT	BP	PS	FS	TSP	VRP
LAST-RL	220.08	32.93	42.03	36.5	**51.68**	29.0	29.0
full	188.11	25.1	34.43	33.75	30.68	**31.0**	30.0
full-Ti	**225.0**	33.43	**47.03**	**56.0**	39.18	28.0	25.0
full-Va	200.2	**41.1**	34.03	28.75	40.93	27.0	27.0
full-He	188.56	27.93	36.43	39.25	47.68	23.5	18.0
full-Ep	200.62	29.6	39.03	42.25	24.43	24.5	**38.0**
full-Ho	183.42	37.93	33.03	29.5	31.43	27.0	23.0

Table 5.3: Evaluation of Experiment 2.a

The *full* feature set without the time features (*full-Ti*) is superior to the other variants. For Bin Packing and Personnel Scheduling, improvements can be seen in comparison to the version in the *LAST-RL*. The *full* model without removing a feature group is outperformed by the *LAST-RL* version.

3. Omnibus test

The Friedman test does not indicate significant differences, with $T_{F_{6,216}} = 0.8577, p = 0.5269$. Therefore, we do not need to perform a post-hoc test for pairwise comparison but we show the critical difference plot, see Figure 5.3.

The next step is to reduce the reduced full model (*full-Ti*) from experiment 2.a with the remaining sets of features.

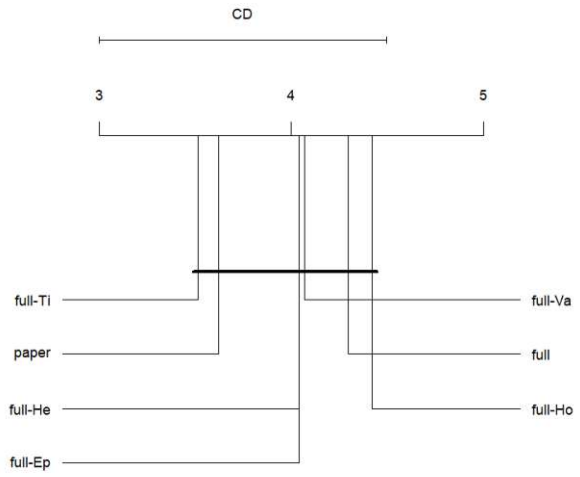

Figure 5.3: Post-hoc pairwise comparison with Nemenyi test for Experiment 2.a

5.3.5 Experiment 2.b: Removing feature groups from *full-Ti* model

We chose the best performing model (*full-Ti*) greedily and now remove the remaining feature groups from this model.

1. Run experiments

The following 7 feature group configurations are teste in experiment 2.b: *full*, *LAST-RL*, *full-Ti*, *full-Ti-Va*, *full-Ti-He*, *full-Ti-Ep*, and *full-Ti-Ho*.

2. Comparison based on CHeSC 2011 ranking scheme

	Total	SAT	BP	PS	FS	TSP	VRP
LAST-RL	206.59	33.93	39.0	37.0	**47.53**	28.0	23.0
full	181.03	26.6	31.0	31.83	29.03	31.0	28.0
full-Ti	208.87.71	32.35	42.0	**48.33**	42.53	24.5	23.0
full-Ti-Va	192.04	35.51	34.0	24.33	37.03	26.0	32.0
full-Ti-He	**230.03**	33.85	**50.0**	42.0	41.43	29.0	**34.0**
full-Ti-Ep	206.29	29.35	40.0	43.0	38.53	**32.0**	24.0
full-Ti-Ho	181.15	**36.43**	30.0	39.5	29.93	19.5	26.0

Table 5.4: Evaluation of Experiment 2.b

Removing the feature group heuristic yields the best results in total and in two domains (Bin Packing and VRP). The *full* model is now outperformed in most domains.

3. Omnibus test

The Friedman test does not indicate significant differences, with $T_{F_{6,216}} = 1.0641, p = 0.3851$. Again, we do not need to perform a post-hoc test for pairwise comparison but we show the CD plot, see Figure 5.4.

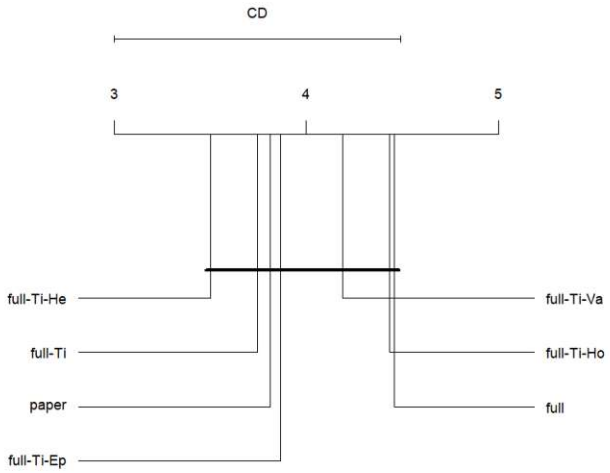

Figure 5.4: Post-hoc pairwise comparison with Nemenyi test for Experiment 2.b

full-Ti-He does not significantly improve *full-Ti*, but the scores show that it is advantageous to remove the groups time and heuristic. Again, the *LAST-RL* version is superior in the domain Flowshop.

5.3.6 Findings of ablation study for feature selection

We conclude the ablation study for feature selection experiments with a table with the CHeSC scoring mechanism (Table 5.5) and one table where we count the number of wins (or break the point for a win in case of a tie) in Table 5.6. We take the best new proposed feature set of each experiment.

	Total	SAT	BP	PS	FS	TSP	VRP
LAST-RL	240.52	38.85	40.5	44.0	**53.6**	34.0	30.0
raw+Ho	237.27	32.6	47.5	46.0	42.6	33.0	35.0
raw+Ho+Va	**254.4**	**49.35**	**54.0**	42.0	38.6	30.0	**39.0**
full-Ti	239.02	39.1	43.5	**53.0**	46.6	33.0	26.0
full-Ti-He	249.79	38.1	45.5	46.0	49.6	**35.0**	35.0

Table 5.5: CHeSC score of best performing feature sets in Experiments 1 & 2.

Feature set *raw+Ho+Va* is ranked first with both ranking methods, especially for the domains MaxSAT and Bin Packing, where we were striving for superior performance

	Total	SAT	BP	PS	FS	TSP	VRP
LAST-RL	7.75	1.2	0.5	0.5	**2.2**	**2.0**	1.0
raw+Ho	3.38	0.2	0.5	1.0	0.7	0.0	1.0
raw+Ho+Va	**12.63**	**3.03**	**4.0**	1.0	0.2	1.0	**3.0**
full-Ti	6.44	1.03	0.5	**2.5**	1.7	1.0	0.0
full-Ti-He	6.8	0.53	1.5	2.0	**2.2**	1.0	0.0

Table 5.6: Count of best scores in Experiments 1 & 2.
The point for a win is split up in case of a tie.

due to the competition results of LAST-RL, see [Kletzander and Musliu, 2023]. It also achieves most points in the domain VRP.

If we look at the table of the feature groups, we find that most features overlap between the feature groups *LAST-RL* and *raw+Ho+Va*. *raw+Ho+Va* does not contain the lastType but it has the 4 features with the average changes of each heuristic type in the horizon H.

The results of the ablation feature selection study answers the first research question: Which additional features of the hyper-heuristic search state result in improved performance for the LAST-RL hyper-heuristic, which features can be omitted, and which subset of features is particularly useful for 'cross-domain' selection?

We summarize the main findings in the following:

- Good performance of *raw+Ho* in experiment 1.a highlighted that the features that are extracted from the changes in the horizon (last H steps) are crucial. For some domains, adding only *Ho* is even sufficient, if we compare the scores to *LAST-RL*. These are the features that are a combination of heuristic × horizon (see features 27-30 in Table 3.1). This supports the statement that the search state profits from short term information about the search of the types of low-level heuristics.

- The only difference between *LAST-RL* and *full-Ti-He* feature sets is that *full-Ti-He* does not use the feature lastType. The feature lastType could be redundant due to two reasons: this information is encoded in lastHeuristic already and the ILS-based action selection in the greedy case selects the low-level heuristic based on the type. Anyways, the performance between *LAST-RL* and *full-Ti*-He is rather similar but with slightly more stable results with *full-Ti*-He across the domains, see Table 5.5. The feature lastType can therefore be omitted if the heuristic type is still part of other features. There is one feature set (*raw+Va*) that has no information about the heuristic type involved, which performs poorly in experiment 1.a, especially for the Personnel Scheduling domain.

- For the Personnel Scheduling domain, the feature set with the most features, *full-Ti*, outperforms the other feature sets. This means that for Personnel Scheduling

instances, it is beneficial to add the new proposed features (except the time-dependent features). *Full-Ti* has already been ahead of the other tested feature sets in Experiment 2.a and 2.b for the Personnel Scheduling instances.

- The time dependent features perform poorly across the HyFlex domains. Especially *full-Ti* shows that removing these features is beneficial and also experiment 1.a with *raw+Ti* which yields a low score. This applies especially for the domains Bin Packing and Personnel Scheduling.

- *raw+Ho+Va* is the best performing feature set of the ablation study. For MaxSAT and Bin Packing, the results are way better (see both Table 5.5 and Table 5.6). In comparison to *LAST-RL*, *raw+Ho+Va* has the same features but excludes the lastType and includes the heuristic × horizon features that were mentioned before.

5.4 Experiments for action space extension

In Figure 5.5, the runs of 2 different configurations of LAST-RL are shown, we test the LAST-RL version with both the default action space and the extended action space. We see that for these 2 runs, there are some significant differences in the median score. Based on the median rank, the default action space is only better for Flowshop and outperformed in all other domains. With the Kruskal-Wallis rank sum test, significant differences can be found in VRP and Bin Packing where the extended action space has lower ranks, and for Flowshop, where the default action space is better.

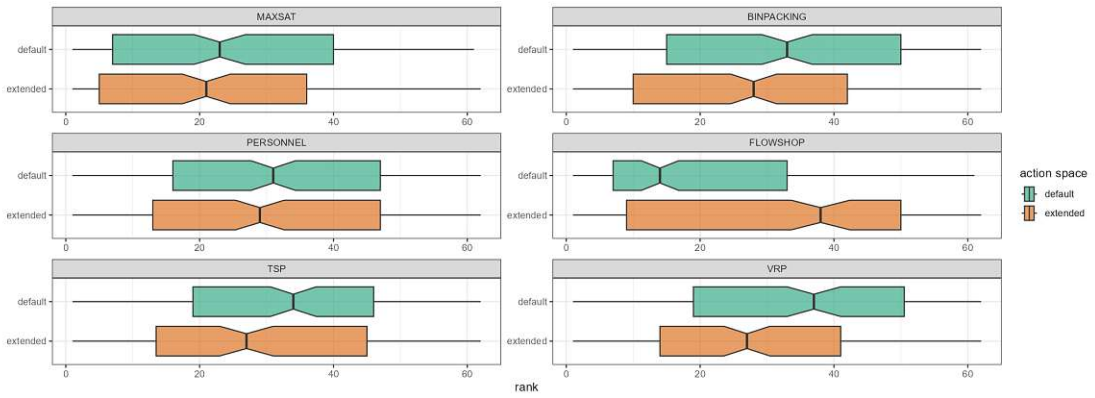

Figure 5.5: Boxplots to compare default and extended action spaces.
The vertical line indicates the median and the notches give roughly 95% confidence intervals ($\pm 1.58 \frac{IQR}{\sqrt{n}}$), see also [McGill et al., 1978]. Not overlapping notches suggest significantly different medians.

We also performed a Welch T-test over the normalized scores of all runs (T = -0.48895, df = 1425.3, p = 0.625) which does not show significant better performance for the extended action space. The required assumptions for the test can be assumed to be fulfilled

after checking the samples visually (approximately normal distributed and independent samples).

Additionally, we tested for all domains individually. Again Welch T-tests are used, but we also account for multiple comparison with Bonferroni correction, which corrects the significance level α by the number of tests m, i.e. $\alpha \leftarrow \frac{\alpha}{m}$. For Bin Packing ($T = 5.0366$, $df = 310.35$, $p < 0.0001$) and TSP ($T = 4.8219$, $df = 207.47$, $p < 0.0001$), the differences of the means in normalized scores were significantly different and in both cases, the mean for the extended action space was lower. For Flowshop, the mean of the default action space was lower and the test was significant ($T = -7.1516$, $df = 257.72$, $p < 0.0001$). For the other domains, no significance could be observed.

We conclude that it is worth to extend the action space to achieve superior performance across the domains. However, the domain Flowshop indicates that this enhancement might be disadvantageous for single domains.

5.5 Experiments for DS-RL

First, we want to obtain a version of DS-RL with the best hyperparameter configuration across all 6 domains. In the second step, we determine the best model for each individual domain. After comparison of the resulting models, we proceed with the best-performing one to examine the impact of the design choices that were introduced earlier.

5.5.1 Hyperparameter optimization for DS-RL

We run hyperparameter optimization with SMAC3 [Lindauer et al., 2022] to obtain the best-performing configuration Λ and to gain insights into the involved components. First, we do this for all domains with a tuning strategy that enables partial parallelization of the tuning process with stratified splits of the training instances and a custom aggregate score. This is described in more detail in the following. After that, we perform tuning within each of the six domains of HyFlex, which provides insights into domain-specific aspects, but also determines whether the tuning strategy across domains leads to superior generalizability.

Hyperparameter tuning across domains

First, we do hyperparameter tuning for all domains to obtain a configuration that performs reasonable across domains with the desired ability to generalize well.

SMAC3 calculates a score for each run and bases its optimization process on a score that we have to provide for each update. We have to make sure that the score is on the same scale for all domains, in order to force the optimization process to equally weight all domains. Obviously, if we would simply take the objective function value, we would not get the desired result. For example, VRP has the length of all routes as the objective which is usually a large integer and Bin Packing has a value between 0 and

1. Improvements for instances of VRP would then be much bigger, thus resulting in an imbalance and introducing a bias for the hyperparameter optimization process. To overcome this issue, we first run all instances on LAST-RL to get a baseline objective value $f(x_b)$. We use a relative score for each instance i:

$$score_{rel}(\Lambda)_i = \frac{median(f(x \mid \Lambda, i)) - f(x_b)}{median(f(x \mid \Lambda, i))}$$

Here, $f(x \mid \Lambda, i)$ contains the objective value for every run $i \in \{1, 2, ..., n_{\text{runs}}\}$, so we take the median of $n_{\text{runs}} = 31$ runs, as it is done in the CHeSC 2011 for the competition score. The baseline value of LAST-RL, $f(x_b)$, is also the median of 31 runs.

Now we split the training instances by a stratified 5-fold split, as there are at least 5 instances per domain in the training set of HyFlex. Each of the stratified 5-fold splits contains one instance per domain. Table 5.7 shows the splits $k_1, ..., k_5$ with their instances.

Domain	**Training splits**					**Competition instances**
	k_1	k_2	k_3	k_4	k_5	
MaxSAT	8	1	6	9	2	3,5,4,10,11
Bin Packing	5	3	4	0	6	7,1,9,10,11
Personnel	2	1	7	3	6	5,9,8,10,11
Flowshop	2	4	5	6	7	1,8,3,10,11
TSP	1	5	3	4	9	0,8,2,7,6
VRP	4	8	3	7	0	6,2,5,1,9

Table 5.7: Training and competition instances of HyFlex framework.
Stratified 5-fold splits of the training instances and competition instances.

In each run of SMAC, we take one split, test the configuration for each instance in the split to obtain a score and finally aggregate the scores. We take the mean over all instances:

$$\frac{1}{n} \sum_{i=1}^{n} score_{rel}(\Lambda)_i,$$

The splits, each containing one instance per domain, allow partial parallelization of the hyperparameter search. The aggregated scores for each split provide reasonable scores that guide the search.

We use multi-fidelity optimization of SMAC3 [Lindauer et al., 2022] with successive halving. The idea of multi-fidelity optimization with successive halving is the following: We have a maximum number of trials that we allow for the whole process (alternatively or additionally we can provide a timeout). Based on that, an initial number of design configurations are created. All configurations are then evaluated with a small budget, then the best performing configurations are iteratively evaluated with increased budgets and the remaining configurations are discarded. A parameter η controls the amount

of configurations that "survive" a stage based on their score, e.g. $\eta = 3$ means that the best $\frac{1}{3}$ of the configurations are kept for further stages. We use the default multi-fidelity optimization components, i.e. a random initial design with a random forest as the surrogate model, the log expected improvement as the acquisition function and a hyperband intensifier.
For more details we refer to [Lindauer et al., 2022] and the online documentation of SMAC3 [1].

We call a SMAC3 multi-fidelity run with 500 trials. SMAC3 then creates 100 design configurations, η is set to 3 and each configuration runs on one up to five splits of the 5-fold split of training instances, depending on the stage of the search.

In the following, we report the results of the hyperparameter tuning in Table 5.8. The column "configuration space" contains the possible values for each hyperparameter. Note that the domains are either continuous intervals, e.g. $[0.02, 0.1]$ for restart, or discrete sets, e.g. $\{R^{LAST-RL}, R^{MA}, R^{530}\}$ for reward function. In the column "tuned configuration", we report the hyperparameter configuration Λ_{best} with the best performance based on the aggregated score.

	DS-RL (Hyperparameter tuning for all domains)	
parameter	configuration space	tuned configuration (Λ_{best})
restart	$[0.02, 0.1]$	0.0879
ε	$[0.001, 0.2]$	0.1229
γ	$[0.7, 0.99]$	0.9697
reward function	$\{R^{LAST-RL}, R^{MA}, R^{530}\}$	$R^{LAST-RL}$
num. of hidden layers	$\{1, 2, 4\}$	4
size of hidden layers	$\{32, 64, 128, 256\}$	64
batch size	{8, 16, 32, 64, 128, 256}	16
replay buffer size	{64, 512, 5000, 10000}	64
activation function	{ReLU, tanh}	tanh
fixed parameters		
action space	default	
update rule	Online DQN	
C	-	
learning starts	0	
greedy selection	argmax	

Table 5.8: Hyperparameter tuning for DS-RL.
Table shows the hyperparameters for DS-RL with the configuration space and tuned configuration Λ_{best}. The lower half of the table ("fixed parameters") shows the hyperparameters that were not tuned here.

[1] github.com/automl/SMAC3)

Note that for these hyperparameter tuning runs, we did not use all possible configurations for DS-RL. The remaining configurations were fixed and are shown in the lower part of the table where only the default value is given. These design choices were made for this hyperparameter tuning run to reduce some degrees of freedom. But later, we will also focus on these parameters:

- search parameters: The default action space with $\alpha = \beta = 0.2$ was used.
- update rule: We use the online deep Q-network update rule. For this update rule, the hyperparameter C is not used.
- learning starts: Learning and model updates start after 0 ms, so we do not start with random selection.
- greedy selection: In the greedy case of our ε-greedy policy, the argmax function selects the next action.

Hyperparameter tuning within domains

We want to identify which components can be specifically tuned for each domain to achieve potentially better results than the previously optimized model that was tuned across all domains.

The same splits as before are used (see Table 5.7) but we run only the instances of the domain for which we tune the hyperparameters. The remaining setup is the same as before but the number of trials is reduced to 200 per domain.

After hyperparameter tuning, which resulted in the hyperparameter configurations in Table 5.9, we evaluate the models on the competition instances and visualize the results using parallel boxplots in Figure 5.6 for each domain. These plots compare the six tuned models for each domain against the model tuned across all domains (DS-RL (cross-domain)) and against LAST-RL. LAST-RL is shown in orange, DS-RL (cross-domain) in blue, and the tuned models in grey, except for the domain-specific tuned model, which is highlighted in green. Along the x-axes, the objective function value is shown.

For the Bin Packing and Personnel competition instances, the domain-specific tuned models outperform the "cross-domain" model. However, for the other domains, the "cross-domain" model shows a lower median in the majority of the instances, suggesting that the domain-tuned models do not generalize well, even to unseen instances from the same domain. This issue is particularly observable in VRP, where the DS-RL (vrp) model performs worse than the others with significant difference. These visualizations indicate that LAST-RL and DS-RL (cross-domain) perform best across the domains. This is confirmed by further comparisons that are provided by the CHeSC 2011 scoring mechanism (see Table 5.10).

If we look in Table 5.9 at the hyperparameter configuration for the two models of Bin Packing and Personnel, we see that they disagree in most hyperparameters. For Bin Packing, the hyperparameter tuning determines "learning starts" to be 100000,

DS-RL (Hyperparameter tuning within domains)						
parameter	MaxSAT	Bin Packing	Personnel	Flowshop	TSP	VRP
action space	default	**extended**	extended	**default**	**default**	extended
C	**0**	100 steps	**1000 ep.**	**100 ep.**	10 ep.	1000 steps
update rule	online	target	target	**DDQN**	**DDQN**	target
learning starts	**0**	**100000**	0	5000	**0**	**50000**
greedy selection	**argmax**	**softmax**	argmax	**softmax**	argmax	softmax
num. of hidden layers	**2**	4	2	2	**4**	2
size of hidden layers	32	256	**128**	256	128	32
batch size	**32**	**32**	128	32	8	32
replay buffer size	64	**10000**	64	64	64	512
fixed parameters						
reward function	$R^{LAST-RL}$					
restart	0.08					
ε	0.12					
γ	0.96					
activation function	tanh					

Table 5.9: Hyperparameter tuning for DS-RL within domains.
Table shows the hyperparameters for DS-RL with the tuned configurations Λ_{best}. The statistically significant hyperparameters are highlighted in bold. These results come from Kruskal-Wallis tests, which test the hypothesis that different hyperparameter values do not lead to statistically significant performance differences. A hyperparameter is marked in bold if there is, for at least one of the possible values, a significant difference in the score for a given domain. This does not imply that the best performing value is the one that causes the significance, but it indicates that the parameter plays a crucial role in the design of the hyper-heuristic for the provided values.

which is approximately 25% of the total runtime and indicates that random selection is advantageous at the beginning. In contrast to that, Personnel selects "learning starts" to be 0, so random selection is not advantageous here. This shows how the hyperparameter choices between the two domains vary. They do agree on the update rule to be the target Q-network.

	Total	SAT	BP	PS	FS	TSP	VRP
DS-RL (maxsat)	192.5	30.0	29.0	31.5	**40.0**	**33.0**	29.0
DS-RL (binpacking)	133.0	18.5	**39.0**	16.5	15.0	18.0	26.0
DS-RL (personnel)	71.0	15.0	5.0	**36.0**	5.0	5.0	5.0
DS-RL (flowshop)	99.5	10.0	10.0	16.5	21.0	26.0	16.0
DS-RL (tsp)	132.5	27.0	15.0	19.5	25.0	26.0	20.0
DS-RL (vrp)	79.5	6.5	28.0	7.0	10.0	12.0	16.0
DS-RL (cross-domain)	**200.0**	**39.0**	25.0	31.0	39.0	**33.0**	**33.0**
LAST-RL	**262.0**	**49.0**	**44.0**	**37.0**	**40.0**	**42.0**	**50.0**

Table 5.10: DS-RL tuned within domains in comparison with LAST-RL (CHeSC 2011 score).

We have seen in most experiments that DS-RL takes longer for the model updates than LAST-RL, which uses a very efficient update rule and less model parameters. We want to test if competitive scores can be achieved, if we allow approximately the same number

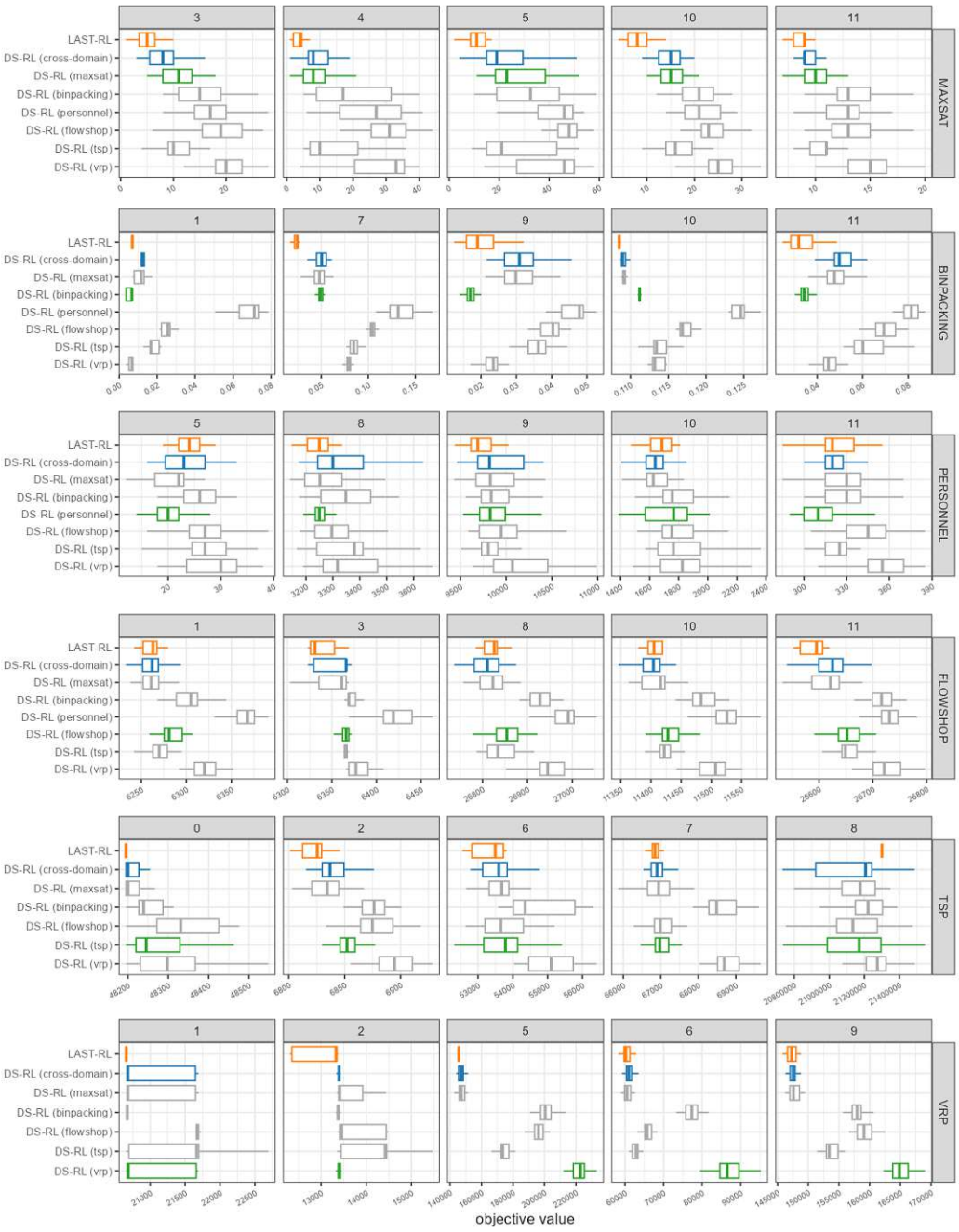

Figure 5.6: Boxplots comparing LAST-RL vs. tuned DS-RL configurations.
For each domain, scores on competition instances are displayed in parallel boxplots, comparing the tuned DS-RL versions (both cross-domain and domain-specific) against LAST-RL.

of steps by adapting the timeout with a correction factor. Table 5.11 shows the count of best scores on the competition instances. This shows that DS-RL now has more points in total and the performance on some domains is dominated by one of the hyper-heuristics. MaxSAT is dominated by DS-RL, whereas Bin Packing has better scores produced by LAST-RL. The other domains are more or less balanced.

	Total	SAT	BP	PS	FS	TSP	VRP
DS-RL (cross-domain) *	**15.5**	**4.5**	0.0	**3.0**	**3.0**	**3.0**	2.0
LAST-RL	14.5	0.5	**5.0**	2.0	2.0	2.0	**3.0**

Table 5.11: DS-RL (cross-domain) in comparison with LAST-RL (count of best scores). * For DS-RL, the timeout was adapted so that approximately the same number of steps as LAST-RL can be taken, i.e. a correction factor was determined for each instance by taking the ratio of the median number of steps of five runs of each instance of LAST-RL and DS-RL.

A comparison of the ranks in Figure 5.7 confirms what we have just seen. Allowing approximately the same number of steps as LAST-RL improves the results in every domain

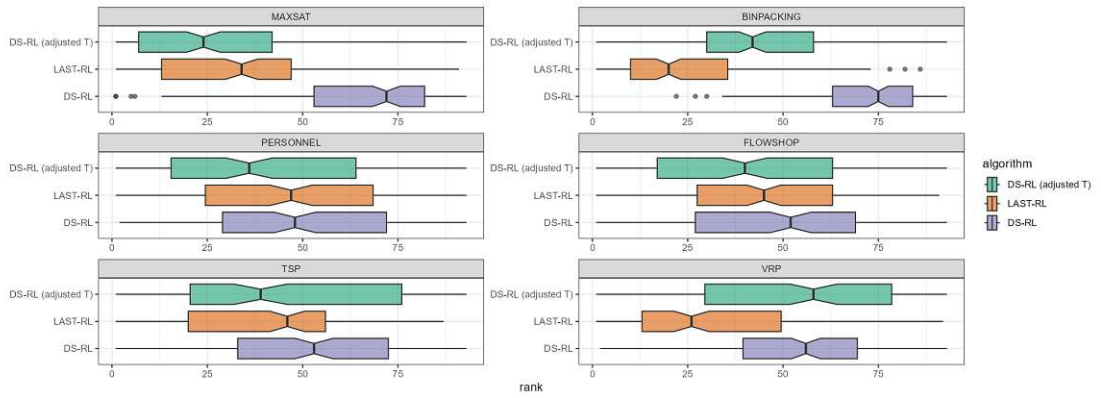

Figure 5.7: Parallel boxplots of ranks in the domains for LAST-RL and DS-RL. DS-RL is shown once with same timeout T as LAST-RL (in purple) and once with adjusted timeout (in dark green).

We conclude that we found a version of DS-RL that, after tuning its hyperparameters across all domains, outperformed those versions whose hyperparameters were tuned solely on individual domains. We conclude that the version DS-RL (cross-domain) generalizes well across domains to unseen instances. This confirms that our tuning strategy with stratified splits and a custom aggregate score is effective.
We also conclude that the fact that the Deep Reinforcement learning algorithm used by DS-RL (deep Q-learning) is less efficient than the one of LAST-RL (SARSA(λ)). If the same number of steps is allowed for both, we get comparable scores, where DS-RL can outperform LAST-RL in MaxSAT, but is still not competitive in the Bin Packing domain.

5.5.2 Experiments with design choices of DS-RL

After determining the best model in the hyperparameter tuning runs, we will continue with analysis of the design choices. For these experiments we start from DS-RL (cross-domain) which turned out to perform best in the last section and vary the corresponding design choices. Note that in some cases, a change in one hyperparameter also requires other modifications which include setting parameters that are not used in the DS-RL (cross-domain) model or changing parameters that obviously have huge influence on the design choice of interest.

In the following, we use ranks on instance level instead of normalized objective function values. For each set of algorithms being compared, we rank all runs including 31 runs from each algorithm. We assign rank 1 for the best run, 2 for the second-best, and so on.

We adapt our evaluation method because the distributional assumptions are generally hard to fulfill with scores from different domains. In the ablation study (Section 5.3) and the extended action space experiment (Section 5.4), we normalized the results using LAST-RL and measured the effects of marginal changes to it. However, in the following experiments, there is no clear baseline for normalization, and the changes to the algorithms are not only marginal. Therefore, we use the ranks of the objective function values for the compared algorithm variants, combined with non-parametric statistical tests.

Experiment 1: Deep Q-network update rule

We test the online DQN, the target DQN and double DQN (DDQN) against each other. Note that some design decisions highly depend also on other hyperparameters that are possibly not considered by the tuned version. The tuned version uses online DQN, which does not use the hyperparameter that determines the number of steps between target model updates. We took the hyperparameters of the best model that uses the corresponding design choice of those models that were tuned for each domain.

For the target DQN, we took this crucial parameter from DS-RL (binpacking), so we update every 100 steps. For DDQN, updates occur every 10 episodes like in DS-RL (tsp).

	Total	SAT	BP	PS	FS	TSP	VRP
online DQN	**14.0**	2.0	0.0	**3.0**	**4.0**	1.0	**4.0**
target DQN	3.0	0.0	1.0	1.0	0.0	1.0	0.0
DDQN	13.0	**3.0**	**4.0**	1.0	1.0	**3.0**	1.0

Table 5.12: DS-RL (cross-domain): update rule (count of best scores).

We see in Table 5.12 and Figure 5.8 that online DQN has a very similar overall performance as DDQN. Nevertheless, within the domains, we see differences. Figure 5.9 and a statistical test show that DDQN has significantly lower ranks in the Bin Packing domain, in contrast to online DQN, which is superior in the Flowshop and VRP instances. The option target DQN is inferior across the domains.

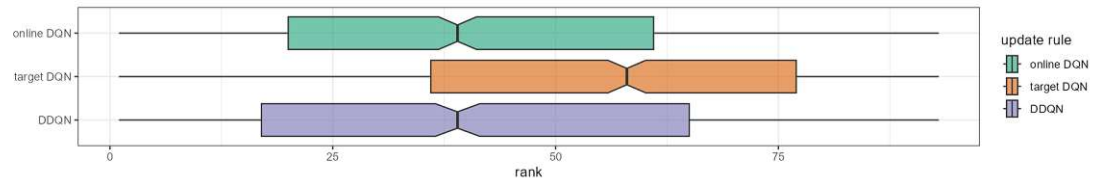

Figure 5.8: DS-RL: Boxplots of ranks to compare update rule.

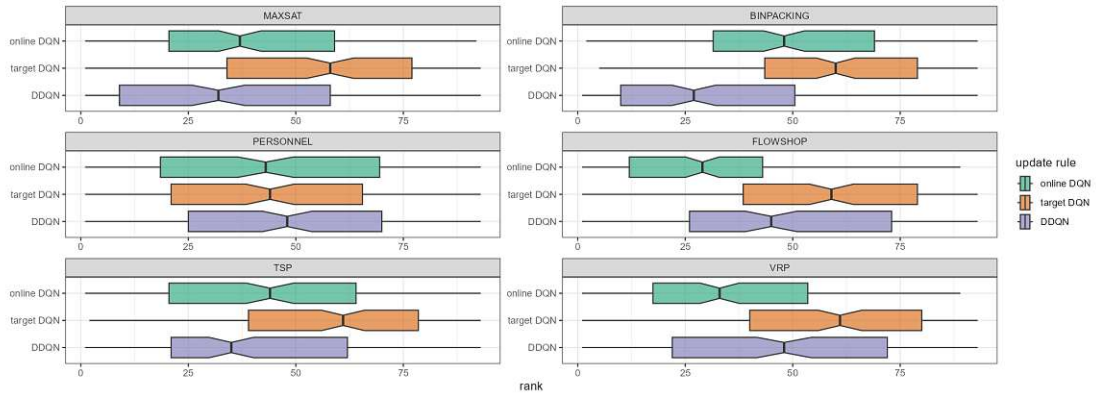

Figure 5.9: DS-RL: Boxplots of ranks to compare update rule across domains.

We conclude that this design choice has no clear impact on the cross-domain selection ability, but within domains it shows interesting differences. Also the minor adjustments in the update rule between target DQN and DDQN result in huge differences for the overall performance. We note that the update frequency that differs between the three tested variants might be a confounder here.

Experiment 2: Feature normalization

We proposed two variants of feature normalization. For this experiment, we assume that no further adjustments of the configuration are needed as there is no intuitive relationship to other parameters.

	Total	SAT	BP	PS	FS	TSP	VRP
norm	**17.0**	**5.0**	2.0	**3.5**	**4.5**	1.0	1.0
zscore	13.0	0.0	**3.0**	1.5	0.5	**4.0**	**4.0**

Table 5.13: DS-RL (cross-domain): feature normalization (count of best scores).

The only non-significant difference is in the domain VRP. The remaining domains show significant differences where "norm" seems to provide better ranks and those domains where "zscore" is better have minor significance levels. We conclude that "norm" is the preferred choice for the activation function, which is confirmed by Table 5.13 and Figure 5.10.

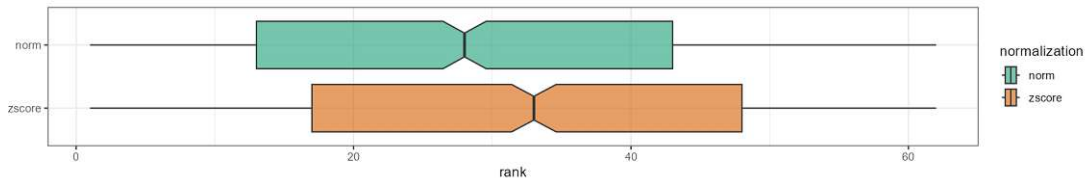

Figure 5.10: DS-RL: Boxplots of ranks to compare feature normalization.

Experiment 3: Activation function

We compare two non-linear activation functions that are applied after each hidden layer: tanh and ReLU.

	Total	SAT	BP	PS	FS	TSP	VRP
tanh	**17.5**	1.5	0.0	**4.0**	**4.0**	**4.0**	**4.0**
relu	12.5	**3.5**	**5.0**	1.0	1.0	1.0	1.0

Table 5.14: DS-RL (cross-domain): activation function (count of best scores).

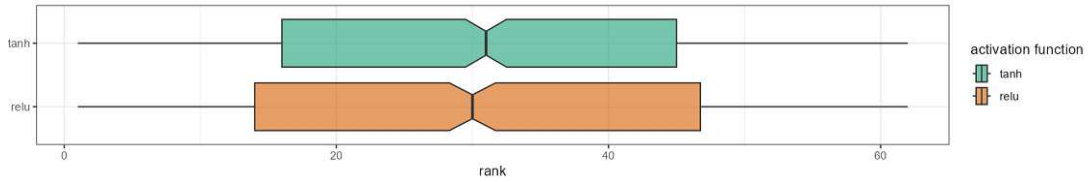

Figure 5.11: DS-RL: Boxplots of ranks to compare activation function.

In total, ReLU has marginally better ranks based on the ranks of all runs, captured by the boxplots in Figure 5.11. Dominance of ReLU is mainly found in domains Bin Packing and MaxSAT, with significance in domain Bin Packing but with no significance overall. Better scores are achieved by tanh in all other domains, which explains also why the count of best scores over all instances in Table 5.14 disagrees with the boxplots in Figure 5.11. Therefore we conclude that there is no clear indication of superior performance of one of the activation functions.

Experiment 4: Reward function

We test the performance of five different reward function designs that were introduced in Subsection 4.1.2.

In Figure 5.13 we immediately observe poor performance of $R^{DS-RL-2}$ in the domain VRP. Figure 5.12 shows that $R^{DS-RL-2}$ is inferior in comparison to the other reward function designs. It is the only tested reward function design that does not weigh improvements by the factor $\sqrt{maxFC}$. $R^{DS-RL-2}$ is also inferior in TSP, Flowshop and MaxSAT. But for two domains (Bin Packing and Personnel), it yields the best ranks,

	Total	SAT	BP	PS	FS	TSP	VRP
R^{DS-RL}	**9.75**	**2.08**	0.0	0.0	**2.67**	1.0	**4.0**
$R^{DS-RL-1}$	6.25	1.58	**2.33**	0.33	0.0	**2.0**	0.0
$R^{DS-RL-2}$	7.0	0.0	**2.33**	**4.33**	0.33	0.0	0.0
$R^{DS-RL-3}$	9.58	1.58	1.0	1.33	**2.67**	**2.0**	1.0
$R^{LAST-RL}$	4.42	0.75	1.33	1.0	1.33	0.0	0.0

Table 5.15: DS-RL (cross-domain): reward function (count of best scores).

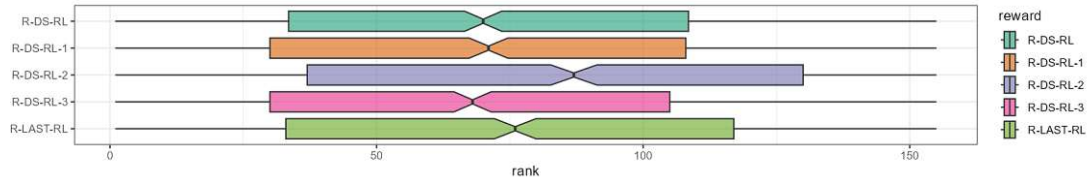

Figure 5.12: DS-RL: Boxplots of ranks to compare reward function.

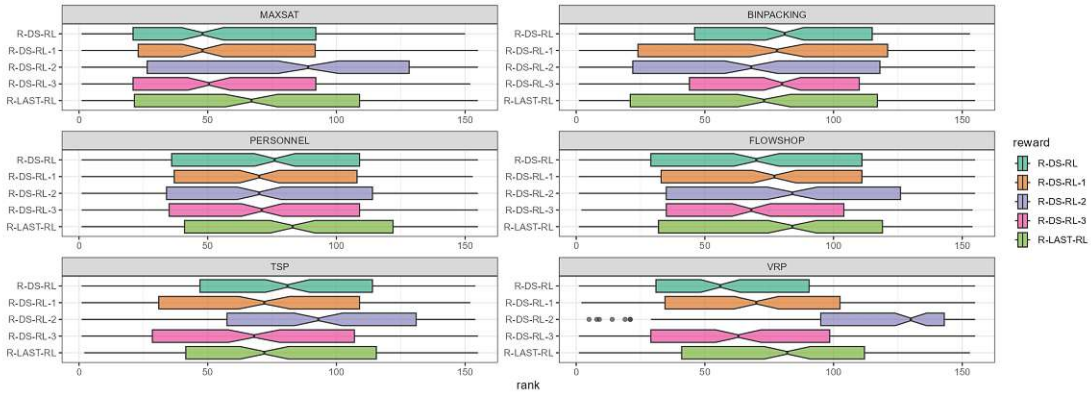

Figure 5.13: DS-RL: Boxplots of ranks to compare reward function across domains.

however with minor difference and no significance. Also the scores in Table 5.15 show that while $R^{DS-RL-2}$ on the one hand is top in two domains, it is the only approach with zero points in three domains.

Apart from this finding, $R^{DS-RL-3}$ is the best performing with marginal difference, based on Figure 5.12. $R^{DS-RL-3}$, in comparison to $R^{LAST-RL}$, weighs $\sqrt{maxFC}$ by the improvement relation factor s_{le} that we motivated in Section 4.1.2. This enhancement seems to be beneficial in the domains Flowshop, MaxSAT, and VRP.

These are very interesting insights. We see that the factor $\sqrt{maxFC}$ is crucial in all domains except Bin Packing and Personnel. In [Kletzander and Musliu, 2023], the factor $\sqrt{maxFC}$ was proposed with the intuition to give higher reward if it is difficult to escape local optima. The results show the effectiveness of this factor. Additionally, we have found that integrating our new proposed factor s_{le} is effective. The intuition that rewarding larger improvements in comparison to previous worsening has a measurable

positive impact.

Experiment 5: Action space

The default action space is better across all domains with significant differences, as seen in Table 5.16 and Figure 5.14. The barplots in Figure 5.15 allow deeper analysis of how the ranks are distributed. The barplots for Bin Packing show that the extended action space is beneficial to achieve top ranks, as the first four barplots have higher proportions of the extended action space (in dark green), an interesting pattern that can not be observed with boxplots.

	Total	SAT	BP	PS	FS	TSP	VRP
default	**28.0**	**5.0**	**3.0**	**5.0**	**5.0**	**5.0**	**5.0**
extended	2.0	0.0	2.0	0.0	0.0	0.0	0.0

Table 5.16: DS-RL (cross-domain): action space (count of best scores).

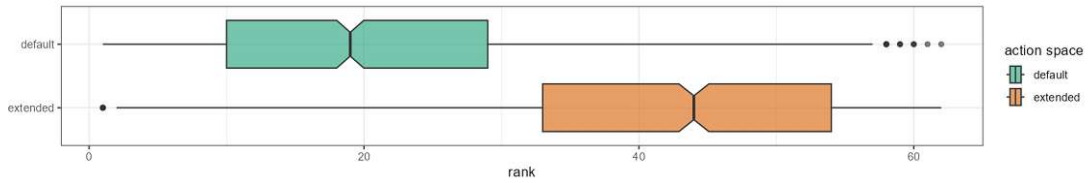

Figure 5.14: DS-RL: Boxplots of ranks to compare action space.

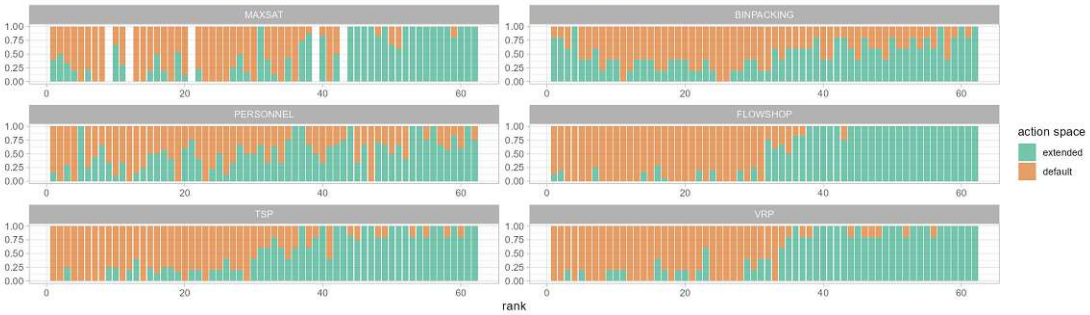

Figure 5.15: DS-RL: Barplots of ranks to compare action space.

Experiment 6: Search state

Here we test the performance of the set of features that was determined in the ablation study in Section 5.3 to be the best performing for the HyFlex domains for the LAST-RL hyper-heuristic, here indicated as "tuned" search state, against the default set of features for LAST-RL. The "tuned" search state is better in the domain Bin Packing with significant difference in the ranks. Otherwise, superior performance is achieved if we

use the default search state of LAST-RL which also reflects on the count of best scores in Table 5.17 and the large gap in the medians in Figure 5.16.

	Total	SAT	BP	PS	FS	TSP	VRP
default	**24.0**	**5.0**	2.0	**3.0**	**5.0**	**4.0**	**5.0**
tuned	6.0	0.0	**3.0**	2.0	0.0	1.0	0.0

Table 5.17: DS-RL (cross-domain): search state (count of best scores).

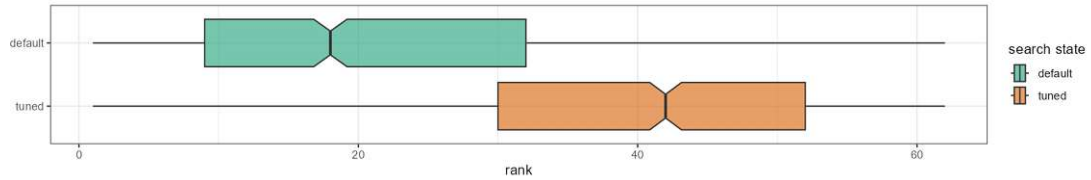

Figure 5.16: DS-RL: Boxplots of ranks to compare search state.

5.5.3 Experiments for DS-RL with progressive neural network

A full transfer analysis is out-of-scope of this thesis. However, we provide initial experiments for DS-RL where the multi-layer perceptron of the deep Q-network architecture is replaced by a progressive neural network architecture (see Subsection 4.1.3). We pre-train the progressive network on stratified splits of the non-competition instances of HyFlex, see Table 5.7. We will evaluate the hyper-heuristic with the trained progressive network after 0, 5, 10 and 15 epochs. One epoch refers to one run over 30 training instances where we alternate between the domains in the same order as the columns of the progressive neural network (MaxSAT, Bin Packing, Personnel, Flowshop, TSP, VRP). The evaluation on the competition instances uses a final column that is solely online trained while the domain columns are freezed.

We visualize the results as boxplots in Figure 5.17 across domains and Figure 5.18 for each domain, and see that there are improvements in the ranks in all domains except Personnel Scheduling. In particular for MaxSAT, TSP and VRP, we see that more epochs lead to better scores. This approach seems to be ineffective for Personnel Scheduling in this initial experiments.

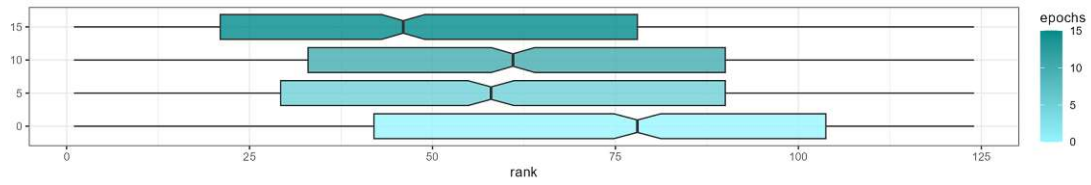

Figure 5.17: DS-RL with progressive neural network: Rank of scores after 0, 5, 10 and 15 epochs.

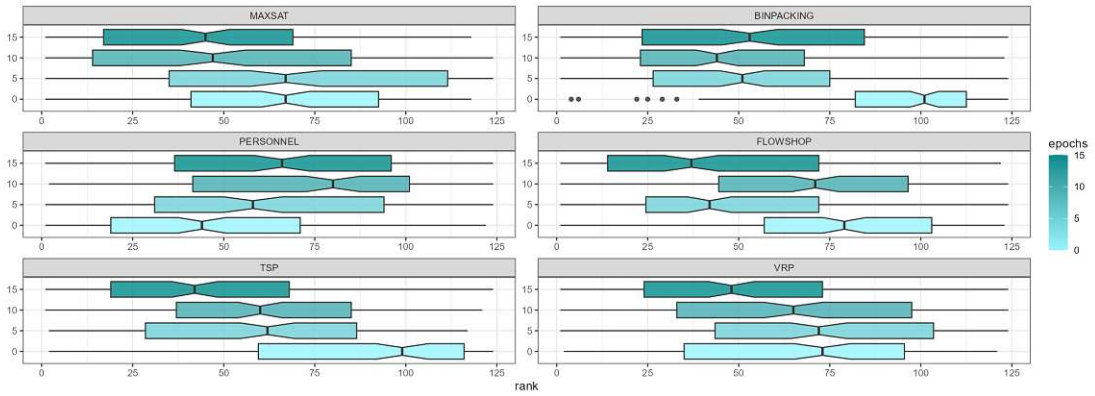

Figure 5.18: DS-RL with progressive neural network: Rank of scores after 0, 5, 10 and 15 epochs within domains.

We conclude these initial experiments by highlighting a potential direction for future work: pre-training seems to be effective for specific domains and a full transfer analysis with the metrics discussed in Subsection 4.1.3 would be appropriate.

5.6 Experiments for maPPOs

We will follow the same procedure as for DS-RL, starting with hyperparameter tuning across domains, then performing hyperparameter tuning within each domain, and finally comparing the resulting hyper-heuristics. Afterwards, we will test different design choices to determine their impact on the scores, both at domain level and across domains. We apply the same procedure as for DS-RL, therefore we refer to the DS-RL experiments in Section 5.5 for detailed descriptions of the experimental setup.

5.6.1 Hyperparameter optimization for maPPOs

We use the same setup as in 5.5.1 where relative scores for a hyperparameter configuration Λ were introduced to guide the optimization process in a meaningful way and with the same splits of training instances. Some parameters are shared between DS-RL and maPPOs but the majority is new.

Hyperparameter tuning across domains

The results of the hyperparameter tuning are presented in Table 5.18. The column "configuration space" lists the possible values for each hyperparameter, consisting of continuous intervals for seven hyperparameters and discrete sets for 13 hyperparameters or design choices. The "tuned config" column displays the hyperparameter configuration Λ_{best} that achieved the best performance based on the aggregated score.

What we observe is that, compared to the tuning of DS-RL, both methods tend to perform better with tanh activation function and with the $R^{LAST-RL}$ reward function. Also the size of the layers are in the same order of magnitude (both actor of maPPOs and deep Q-network have 64 units per hidden layer). The hyperparameter ε that controls the exploration-exploitation trade-off is for DS-RL ($\varepsilon = 0.1229$) considerably larger than for maPPOs ($\varepsilon = 0.0481$).

The tuning seems to prefer more the more complex variant in several design choices, including the extended action space, the model update with separate PPO loss functions and generalized advantages estimation.

	maPPOs (Hyperparameter tuning for all domains)	
parameter	configuration space	tuned config (Λ_{best})
restart	$[0.02, 0.2]$	0.1337
ε	$[0.001, 0.2]$	0.0481
γ	$[0.7, 0.99]$	0.8681
action space	{default, extended}	extended
activation function	{ReLU, tanh}	tanh
feature normalization	{zscore, norm}	norm
n_{updates}	{50,100,200}	50
n_{agents}	{1,2,4,8}	1
standardize advantages	{yes,no}	yes
epochs K	{1,5,10,20,30}	5
$\varepsilon^{\text{clip}}$	$[0.1, 1.0]$	0.2574
PPO loss	{joint loss, separate}	separate
actor lr	$[10^{-5}, 0.1]$	0.0141
critic lr	$[10^{-5}, 0.1]$	0.0046
advantage estimation	{simple, GAE}	GAE
policy mask	{LAST-RL mask, TSILS mask}	LAST-RL mask
reward function	$\{R^{LAST-RL}, R^{MA}, R^{530}, R^{DS-RL-3}\}$	$R^{LAST-RL}$
size of actor layers	$\{32, 64, 128, 256\}$	64
size of critic layers	$\{32, 64, 128, 256\}$	128
c_2	$[0.0, 1.0]$	0.1616

Table 5.18: Hyperparameter tuning for maPPOs.
Table shows the hyperparameters for maPPOs with the configuration space and tuned configuration Λ_{best}.

Hyperparameter tuning within domains

As before in Subsection 5.5.1, we want to identify components that can be specifically tuned for each domain to achieve better results than the previously optimized model tuned across all domains. Again, we provide a budget of 200 trials per domain. The resulting best configurations are shown in Table 5.19.

What has been observed in the experimental study of [Andrychowicz et al., 2020] and can be also confirmed by our results is that the size of the actor layers tends to be smaller than the size of the critic layers. This can be seen in 4 out of 6 domains and additionally in the tuning across all domains in the previous experiment.

The tuned variants of maPPOs agree on the activation function (tanh for 5 out of 6 domains), feature normalization (norm for 5 out of 6 domains), and action space (extended for 4 out of 6 domains). The other design choices do not show a clear trend.

maPPOs (Hyperparameter tuning within domains)						
parameter	MaxSAT	Bin Packing	Personnel	Flowshop	TSP	VRP
restart	0.0473	0.0848	0.1130	0.1052	0.1427	0.0794
ε	0.0037	0.1164	0.1145	-	0.15913	0.0394
γ	0.7210	0.8548	0.8202	0.9837	0.7685	0.7
action space	**default**	**extended**	**default**	**extended**	**extended**	**extended**
activation function	**tanh**	**tanh**	tanh	tanh	tanh	**relu**
feature normalization	norm	norm	norm	zscore	norm	norm
n_{updates}	50	**100**	**50**	200	**50**	**100**
n_{agents}	**1**	**2**	1	**2**	**2**	**2**
standardize advantages	**yes**	yes	**no**	**no**	no	**yes**
epochs K	**1**	**1**	**30**	**5**	**5**	**1**
$\varepsilon^{\text{clip}}$	0.8122	0.5425	0.1318	0.4651	0.3022	0.2930
PPO loss	**joint**	**separate**	**separate**	joint	**joint**	separate
actor lr	-	0.0538	0.0088	-	-	0.001
critic lr	-	0.0956	0.0321	-	-	0.001
joint lr	0.0774	-	-	0.0790	0.0483	-
advantage estimation	**simple**	**simple**	**GAE**	**simple**	**GAE**	**GAE**
reward function	$\mathbf{R^{LAST-RL}}$	$\mathbf{R^{DS-RL-3}}$	$R^{LAST-RL}$	$\mathbf{R^{MA}}$	$\mathbf{R^{MA}}$	$\mathbf{R^{530}}$
size of actor layers	**64**	**32**	**64**	**32**	**32**	**128**
size of critic layers	**256**	**128**	32	**256**	**128**	64
c_1	0.7071	-	-	0.0256	0.2784	-
c_2	0.5992	0.2911	0.1936	0.5525	0.1340	0.6973

Table 5.19: Hyperparameter tuning for maPPOs within domains.
Table shows the hyperparameters for maPPOs with the tuned configurations Λ_{best}. The statistically significant hyperparameters are highlighted in bold (only for discrete hyperparameters). Just as for for the hyperparameter tuning for DS-RL, these results come from Kruskal-Wallis tests.

The Table 5.20 displays the scores of the tuned versions of maPPOs with the CHeSC 2011 scoring mechanism. As for DS-RL, again the hyper-heuristic that was tuned across all domains shows the best scores and only the tuning on VRP shows similar performance. Additionally, a comparison to the scores of LAST-RL is provided and it turns out that maPPOs can not compete with the scores of LAST-RL.

As for DS-RL, we want to compare with LAST-RL after adapting the timeout to allow approximately the same number of steps. Table 5.21 summarizes this experiment with the count of best scores. In contrast to DS-RL, this does not lead to competitive scores.

We will take a closer look at important design choices in the following Subsection 5.6.2.

Figure 5.19: Boxplots comparing LAST-RL vs. tuned maPPOs configurations.
For each domain, scores on competition instances are displayed in parallel boxplots, comparing the tuned maPPOs versions (both cross-domain and domain-specific) against LAST-RL.

	Total	SAT	BP	PS	FS	TSP	VRP
maPPOs (maxsat)	64.0	**28.0**	5.0	10.0	10.0	6.0	5.0
maPPOs (binpacking)	142.5	25.5	26.0	20.0	22.0	26.0	23.0
maPPOs (personnel)	100.0	5.0	11.0	24.0	23.0	15.0	22.0
maPPOs (flowshop)	104.5	27.5	14.0	17.0	17.0	19.0	10.0
maPPOs (tsp)	140.0	14.0	**41.0**	16.0	22.0	25.0	22.0
maPPOs (vrp)	168.5	21.5	27.0	**32.0**	**26.0**	**32.0**	30.0
maPPOs (cross-domain)	**170.5**	23.5	33.0	26.0	25.0	30.0	**33.0**
LAST-RL	**280.0**	**50.0**	38.0	**50.0**	**50.0**	**42.0**	**50.0**

Table 5.20: maPPOs tuned within domains in comparison with LAST-RL (CHeSC 2011 score).

	Total	SAT	BP	PS	FS	TSP	VRP
maPPOs (cross-domain)*	5.0	0.0	2.0	1.0	0.0	2.0	0.0
LAST-RL	**25.0**	**5.0**	**3.0**	**4.0**	**5.0**	**3.0**	**5.0**

Table 5.21: maPPOs (cross-domain) in comparison with LAST-RL (count of best scores). * For maPPOs, the timeout was adapted so that approximately the same number of steps as LAST-RL can be taken, i.e. a correction factor was determined for each instance by taking the ratio of the median number of steps of five runs of each instance of LAST-RL and maPPOs.

5.6.2 Experiments with design choices of maPPOs

The previous experiments have already given insights into important components of the hyperparameter configuration. Just like for DS-RL, the version of maPPOs that was tuned across the problem domains managed to generalize better than the domain specific versions. We utilize maPPOs (cross-domain) for the following experiments where we want to gain insights into interesting design choices.

In some cases, other hyperparameters that highly depend on the tested design choice are also adjusted. These modifications are described at the respective place.

Experiment 1: Advantage estimation

First, we want to determine whether it is worth to use generalized advantage estimation which comes with higher computational effort than the alternative. However, advantages are estimated only once per model update, so this higher computational effort is minimal.

Generalized advantage estimation yields better ranks than the alternative across all domains and with significance in the domains Bin Packing, MaxSAT, TSP, and VRP. Both the boxplots in Figure 5.20 and the score counts in Table 5.22 agree on that.

Experiment 2: Advantage standardization

Both Table 5.23 and Figure 5.21 show that standardization before the model updates help to stabilize training in all domains except Personnel Scheduling. If we think about

	Total	SAT	BP	PS	FS	TSP	VRP
GAE	**23.5**	**5.0**	**4.0**	**2.5**	**3.0**	**5.0**	**4.0**
simple	6.5	0.0	1.0	**2.5**	2.0	0.0	1.0

Table 5.22: maPPOs (cross-domain): advantage estimation (count of best scores).

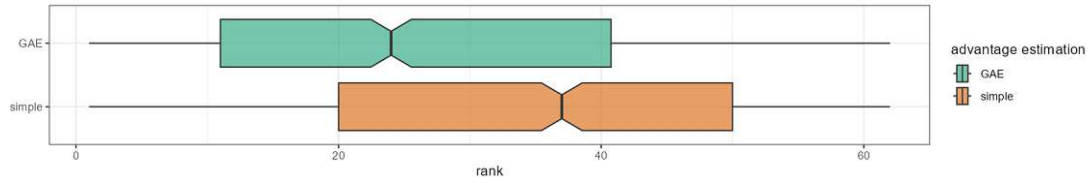

Figure 5.20: maPPOs: Boxplots of ranks to compare advantage estimation.

differences to the other domains, then this might be related to unreliable estimates caused by the small sample size that the batch in Personnel Scheduling is expected to have. In Personnel Scheduling, the low-level heuristics usually take longer and therefore result in smaller batches than in other domains, as the number of updates for maPPOs is equal across domains.

	Total	SAT	BP	PS	FS	TSP	VRP
standardize	**19.0**	**4.0**	**4.0**	1.0	**3.0**	**4.0**	**3.0**
not standardize	11.0	1.0	1.0	**4.0**	2.0	1.0	2.0

Table 5.23: maPPOs (cross-domain): advantage standardization (count of best scores).

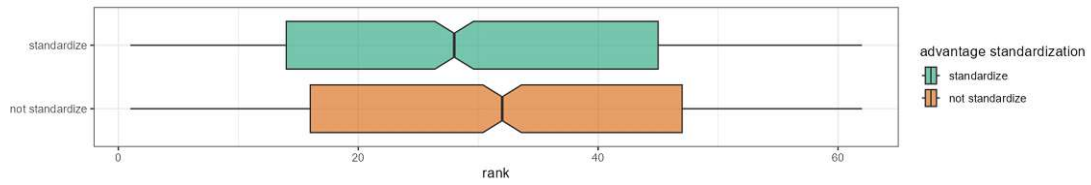

Figure 5.21: maPPOs: Boxplots of ranks to compare advantage standardization.

Experiment 3: PPO loss function

The tuned version used separate loss functions and updates for actor and critic. Therefore, also the learning rates for actor and critic were tuned. For the alternative, the joined update, we take the learning rate of DS-RL (tsp). The hyperparameter c_1 that weighs the critic loss in the joint loss was set to 0.05.

The results in Table 5.24 clearly demonstrate that the separate loss functions and updates of actor and critic are beneficial. The boxplots in Figure 5.22 show a huge gap between the median of the ranks. A closer look reveals the domain MaxSAT as the only exception, where the joint update is better. However, it is the domain with the lowest evidence.

The barplots in Figure 5.23 support our argument. With exception of MaxSAT, all domains show clear advantages for the separate loss.

	Total	SAT	BP	PS	FS	TSP	VRP
separate	**25.0**	1.5	**5.0**	**4.5**	**5.0**	**5.0**	**4.0**
joint	5.0	**3.5**	0.0	0.5	0.0	0.0	1.0

Table 5.24: maPPOs (cross-domain): PPO loss (count of best scores).

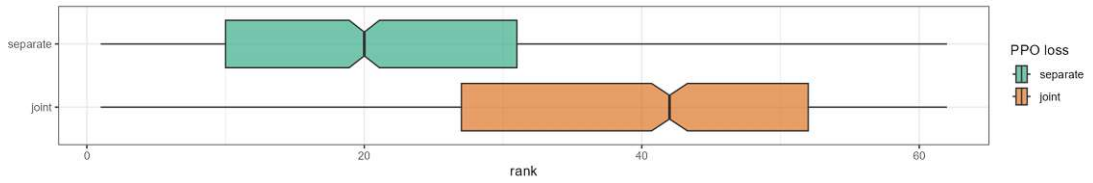

Figure 5.22: maPPOs: Boxplots of ranks to compare PPO loss.

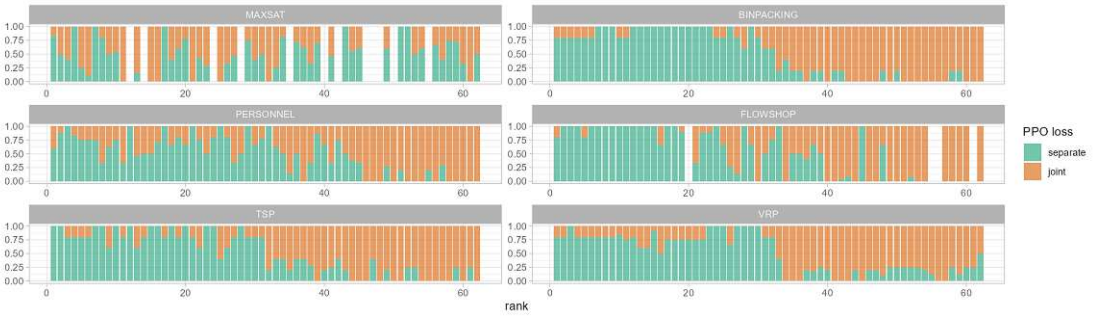

Figure 5.23: maPPOs: Barplots of ranks to compare PPO loss.

Experiment 4: Policy

We use the softmax policy to sample actions but also integrate simple Iterated Local Search components: we test the ILS-masks by LAST-RL [Kletzander and Musliu, 2023] and by TSILS [Adubi et al., 2021]. We have described them in detail in Subsection 4.2.3.

The comparison of the action masks that extend our softmax policy indicates that the ILS-mask of LAST-RL is better across the domains with significant differences in the ranks, see Table 5.25 and Figure 5.24.

	Total	SAT	BP	PS	FS	TSP	VRP
softmax + LAST-RL mask	**19.5**	**5.0**	**5.0**	2.0	1.5	1.0	**5.0**
softmax + TSILS mask	10.5	0.0	0.0	**3.0**	**3.5**	**4.0**	0.0

Table 5.25: maPPOs (cross-domain): Policy (count of best scores).

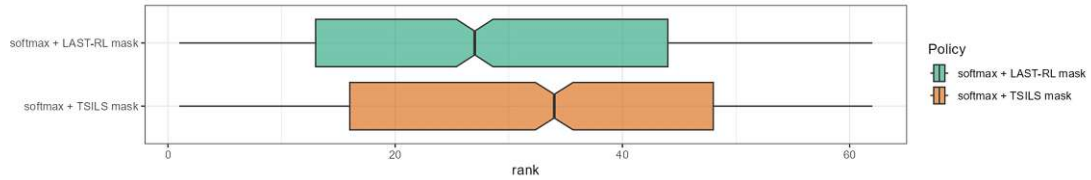

Figure 5.24: maPPOs: Boxplots of ranks to compare Policies.

An inspection of the single domains shows significant differences in each domain. Actually both tested options outperform the alternative in three domains. The large gaps in Bin Packing and VRP, see boxplots in Figure 5.25, lead to significant better ranks of the "LAST-RL mask" overall.

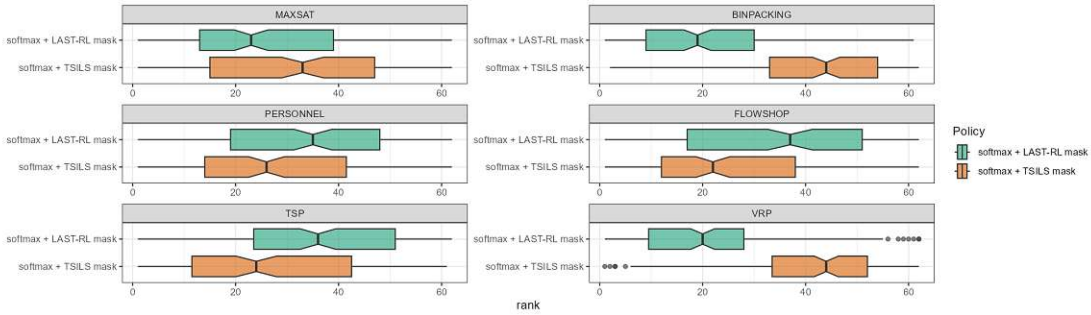

Figure 5.25: maPPOs: Boxplots of ranks to compare policies across domains.

Experiment 5: Feature normalization

For feature normalization, we again see the pattern that the large differences for Bin Packing and VRP lead to the overall result that "norm" is better with significant difference, see Figure 5.26. However, the counts of best scores in Table 5.26 suggest the opposite. In the Personnel Scheduling domain, "zscore" provides significantly more top ranks, but in the remaining domains there are minor differences.

We conclude that there is no clear sign which of the two choices should be preferred across the domains.

	Total	SAT	BP	PS	FS	TSP	VRP
norm	13.5	**3.0**	**3.0**	1.0	0.5	**3.0**	**3.0**
zscore	**16.5**	2.0	2.0	**4.0**	**4.5**	2.0	2.0

Table 5.26: maPPOs (cross-domain): Feature normalization (count of best scores).

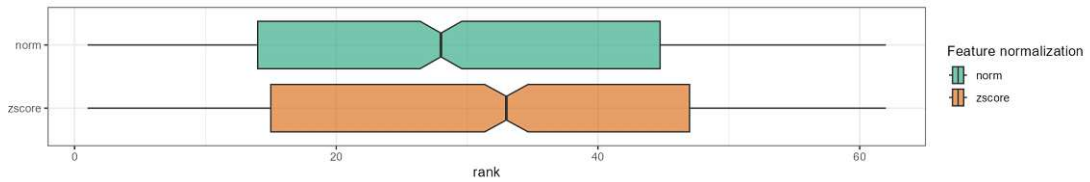

Figure 5.26: maPPOs: Boxplots of ranks to compare feature normalization.

Experiment 6: Action space

The extended action space yields improvement, with strong significance for MaxSAT, Bin Packing, and VRP, see Table 5.27 and barplots in Figure 5.28. For Bin Packing, almost every run with the default action space is outperformed by all runs with the extended action space. Only Flowshop suggests the inverse but with weaker evidence.

We conclude that the extension of the action space is a meaningful choice for maPPOs.

	Total	SAT	BP	PS	FS	TSP	VRP
extended	**22.5**	**5.0**	**5.0**	**2.5**	1.0	**4.0**	**5.0**
default	7.5	0.0	0.0	**2.5**	**4.0**	1.0	0.0

Table 5.27: maPPOs (cross-domain): Action space (count of best scores).

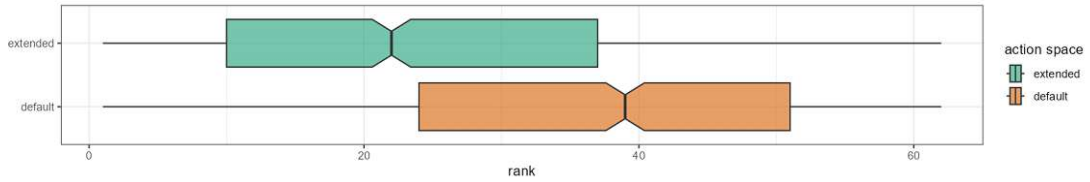

Figure 5.27: maPPOs: Boxplots of ranks to compare action space.

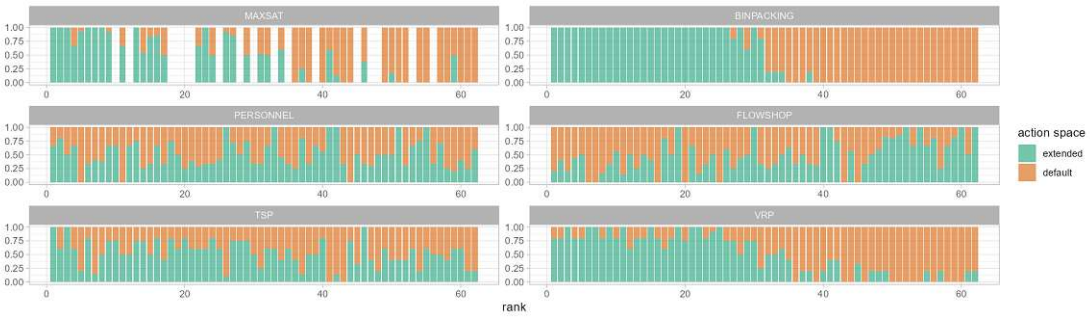

Figure 5.28: maPPOs: Barplots of ranks to compare action space.

Experiment 7: Number of agents

We test the single agent version against the tuned multi-agent version with 2 agents, as all tuned multi-agent models within the domains figured out that 2 agents are the best choice.

Both the count of best scores in Table 5.28 and the boxplots with ranks over all domains in Figure 5.29 show significant advantages for the 2-agent version.

	Total	SAT	BP	PS	FS	TSP	VRP
single agent	12.5	2.0	1.0	1.5	**3.0**	**4.0**	1.0
2 agents	**17.5**	**3.0**	**4.0**	**3.5**	2.0	1.0	**4.0**

Table 5.28: maPPOs (cross-domain): Number of agents (count of best scores).

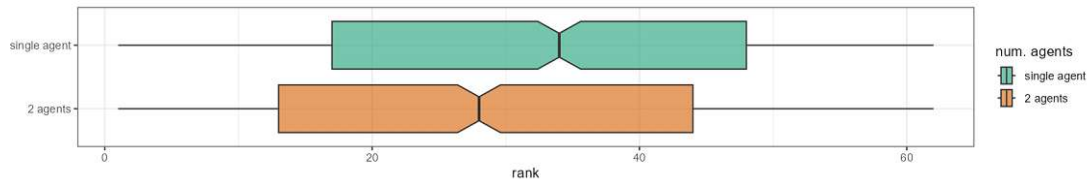

Figure 5.29: maPPOs: Boxplots of ranks to compare the number of agents.

This finding does not align with the results of the cross-domain hyperparameter tuning. The tuning approach based on the aggregate score of normalized objective function values does not align with the evaluation of the experiments for the design choices on the ranks on instance level. The normalized objective function value is less robust than the ranking method applied here. Although the tuning across domains showed its ability to generalize with a single agent, it may have overfitted to some instances.

Experiment 8: Search state

The set of features *raw+Ho+Va* (tuned) is tested against the default search state from [Kletzander and Musliu, 2023].

No clear differences are determined by the comparison of the tuned and default search state, see Figure 5.30. Anyway, Table 5.29 reveals minor improvement with the tuned search state across the domains. In contrast to DS-RL, the tuned search state seems to at least a meaningful choice.

	Total	SAT	BP	PS	FS	TSP	VRP
default state	12.5	1.0	2.0	**2.5**	1.0	**4.0**	2.0
tuned	**17.5**	**4.0**	**3.0**	**2.5**	**4.0**	1.0	**3.0**

Table 5.29: maPPOs (cross-domain): Search state (count of best scores).

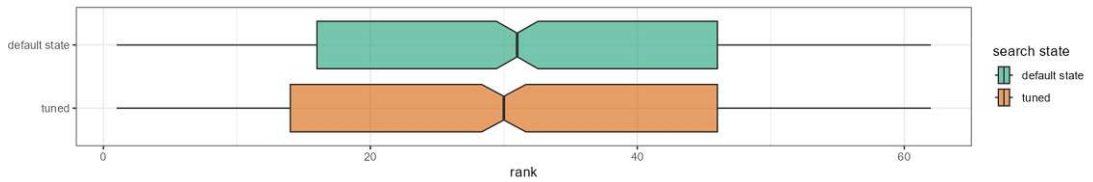

Figure 5.30: maPPOs: Boxplots of ranks to compare search state.

5.7 Final comparison

We want to summarize our findings with final evaluations of the hyper-heuristics involved in the thesis. On the one hand, we proposed enhancements for LAST-RL considering the search state and the action space. Both extensions showed potential for improvement. For this final enhanced version of LAST-RL, we also use, instead of $R^{LAST-RL}$, the new introduced reward function $R^{DS-RL-3}$ which yielded superior results for DS-RL. On the other hand, we developed two hyper-heuristics where we focused on proposing components that address issues that we have faced in preliminary experiments. First, we provide an analysis on the HyFlex benchmark suite, followed by a comparison on three real-life personnel scheduling domains.

5.7.1 Final comparison on HyFlex benchmark suite

Figure 5.31 shows better performance of the enhanced version (in dark orange) overall in comparison with the default LAST-RL (in dark green) with significant difference. A more detailed view gives Figure 5.32 where we see advantages for the enhanced version in the domains MaxSAT and Bin Packing. Also in Flowshop and VRP, the enhancements provide better scores after ranking them, but without significance. The version of LAST-RL in [Kletzander and Musliu, 2023] is still superior in the TSP domain.

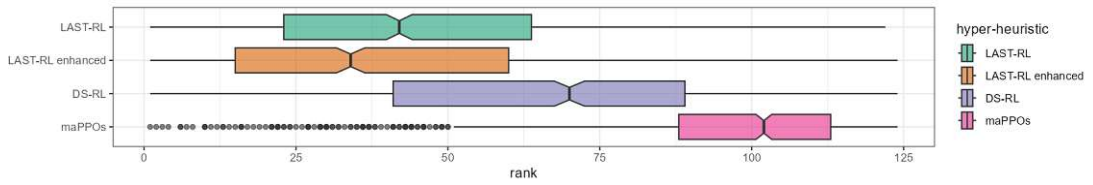

Figure 5.31: LAST-RL (enhanced), DS-RL, and maPPOs: Boxplots to compare enhancements and new approaches.

Now we also include DS-RL and maPPOs in the evaluation, which are both already contained in the boxplots in Figure 5.31 and Figure 5.32. Both DS-RL (in purple) and maPPOs (in pink) are the tuned versions across all domains: These are DS-RL (cross-domain) from Subsection 5.5.1 and maPPOs (cross-domain) from Subsection 5.6.1.

We see that DS-RL is significantly inferior to the two LAST-RL versions. Whereas maPPOs barely achieves top ranks, see outliers in the boxplot of maPPOs in Figure 5.31.

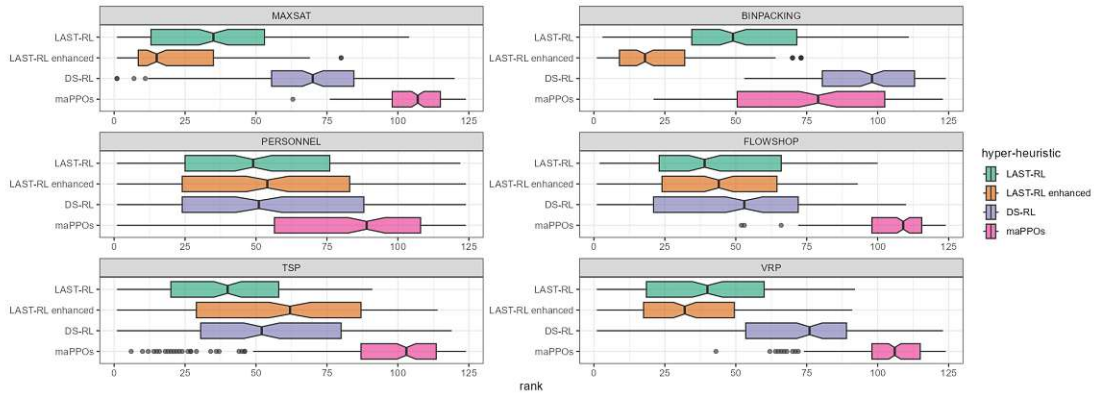

Figure 5.32: LAST-RL (enhanced), DS-RL, and maPPOs: Boxplots to compare enhancements and new approaches within domains.

In Figure 5.33, we have adjusted the timouts of DS-RL and maPPOs so that approximately the same number of steps can be taken by these two more computationally intensive hyper-heuristics. The outcome is similar to Figure 5.31 where maPPOs (in pink) could not manage to improve a lot. In contrast, DS-RL with adjusted timeout (in purple) does not have significantly different ranks than LAST-RL. Closer inspection within the domains reveals that DS-RL can not manage to be competitive in Bin Packing and VRP. However, it now achieves the best median rank in three domains (Personnel, Flowshop, TSP). This shows that the efficiency of the method matters a lot and that, with the same number of steps, the agent of DS-RL is competitive with LAST-RL and in three domains can achieve better scores. For Bin Packing and VRP, it is still not on the same level as LAST-RL.

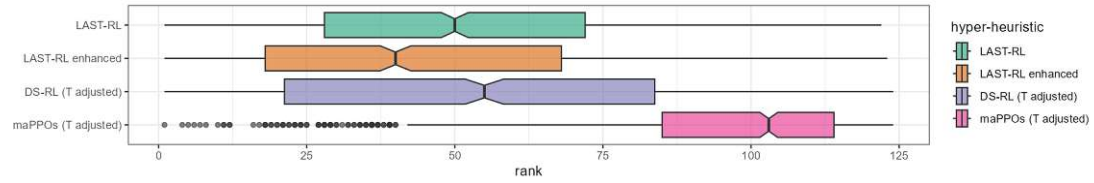

Figure 5.33: LAST-RL (enhanced), DS-RL, and maPPOs: Boxplots to compare enhancements and new approaches with adjusted timeout.

In Table 5.30, we provide a final comparison of the enhanced version of LAST-RL, DS-RL and maPPOs. For DS-RL and maPPOs, we additionally add the results with the adjusted timeout which were already mentioned in the experiments for DS-RL (Subsection 5.6.1) and maPPOs (Subsection 5.5.1). The table shows the points of each of the approaches as if they had participated in the CHeSC 2011.

DS-RL and maPPOs show inferior performance in the domains MaxSAT and Binpacking. maPPOs only achieves points in the domains Personnel and TSP. If we allow approximately the same number of steps as LAST-RL, maPPOs also gets points in the domain VRP

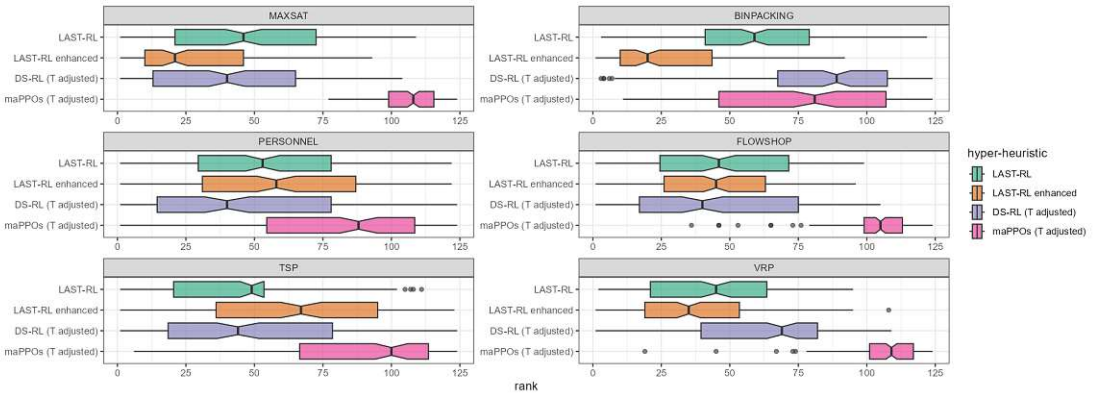

Figure 5.34: LAST-RL (enhanced), DS-RL, and maPPOs: Boxplots to compare enhancements and new approaches within domains and with adjusted timeout.

but it is still not competitive in total. DS-RL achieves points in all domains, except Bin Packing, and most points in the domain VRP. Again, if approximately the same number of steps as LAST-RL are allowed, also points for Bin Packing instances are gathered. The enhancements for LAST-RL improve the results for LAST-RL mainly in the domains MaxSAT and Bin Packing. Only slight changes are seen in the domains Personnel, Flowshop and TSP. In the domain VRP, the improvements lead to the best score of the compared approaches. Note that for LAST-RL, we use the results that we have produced with our setup and not the results that are provided in [Kletzander and Musliu, 2023]. This also applies the previous experiments with LAST-RL.

	Total	SAT	BP	PS	FS	TSP	VRP
TS-ILS	**239.5**	**43.0**	41.0	**35.5**	**46.0**	**43.0**	31.0
AdapHH	181.0	34.75	**45.0**	9.0	37.0	40.25	15.0
FS-ILS	177.0	**43.0**	18.0	7.0	44.0	34.0	31.0
QHH	150.6	39.6	16.0	0.0	35.0	34.0	26.0
ML	131.5	14.5	12.0	31.0	39.0	13.0	22.0
RL	126.6	30.35	37.0	2.0	27.25	18.0	12.0
LAST-RL (enhanced)	124.1	24.1	26.0	9.0	18.0	11.5	**35.5**
DS-RL*	119.0	17.0	9.0	23.0	23.0	19.0	28.0
LAST-RL	92.1	8.1	14.0	8.0	18.5	14.5	29.0
FRAMAB	71.0	4.0	9.0	20.0	0.0	3.0	35.0
DS-RL	44.6	0.6	0.0	6.0	13.0	8.0	17.0
AVEG-Nep	21.0	12.0	0.0	0.0	0.0	0.0	9.0
maPPOs*	11.5	0.0	0.0	3.5	0.0	4.0	4.0
maPPOs	5.0	0.0	0.0	3.0	0.0	2.0	0.0
MCHH-S	4.75	4.75	0.0	0.0	0.0	0.0	0.0
Ant-Q	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 5.30: Comparison of approaches if they had participated in the CHeSC 2011. Our approaches, state of the art, and all reinforcement learning hyper-heuristics which provide results are shown. With * indicated approaches are the versions of DS-RL and maPPOs with adjusted timeout, which allows them to take approximately same number of steps as LAST-RL.

We can conclude that the two approaches using deep Q-learning and Proximal Policy

Optimization were not able to achieve top ranks, whereas the enhancements for LAST-RL have a positive effect on the scores.

To the best of our knowledge, the only deep reinforcement learning approach evaluated on the HyFlex benchmark is [Dantas and Pozo, 2021], using DQN. However, they report mean scores, instead of medians, and do not provide results for all instances. Therefore, it can not be included in this evaluation. A comparison with their reported mean scores shows that their DQN can not compete with our results.

5.7.2 Final comparison on real-life domains

In addition to the final evaluation on the HyFlex benchmark suite, we also provide results on three real-life domains, which are personnel scheduling domains: Rotating Workforce Scheduling (RSW), Bus Driver Scheduling (BDS), and Minimum Shift Design (MSD). They have different characteristics but for all three, there exist hyper-heuristics that can compete with specialized algorithms. For more details about the three domains and the existing hyper-heuristics, refer to [Kletzander and Musliu, 2024].

The evaluation can be seen in Table 5.31. We use the ranking scheme of CHeSC 2011, for the mean scores of feasible solutions of 5 runs with a timeout of one hour. The number of instances per domain is not balanced so we weigh the total scores by the relative number of instances for each domain. Also for the real-life domains, maPPOs does not show the ability to compete with state of the art. Whereas DS-RL achieves more than 25 points in each of the three domains and demonstrates its ability to generalize to unseen domains, even if the performance is not on the level of state-of-the-art approaches. One approach that uses Q-learning (QLEARN in Table 5.31) is outperformed in all domains. Also SSHH, that was described in Subsection 2.4.1, and FS-ILS (Subsection 2.4.2), which was ranked above DS-RL, are both outperformed on all three real-life domains.

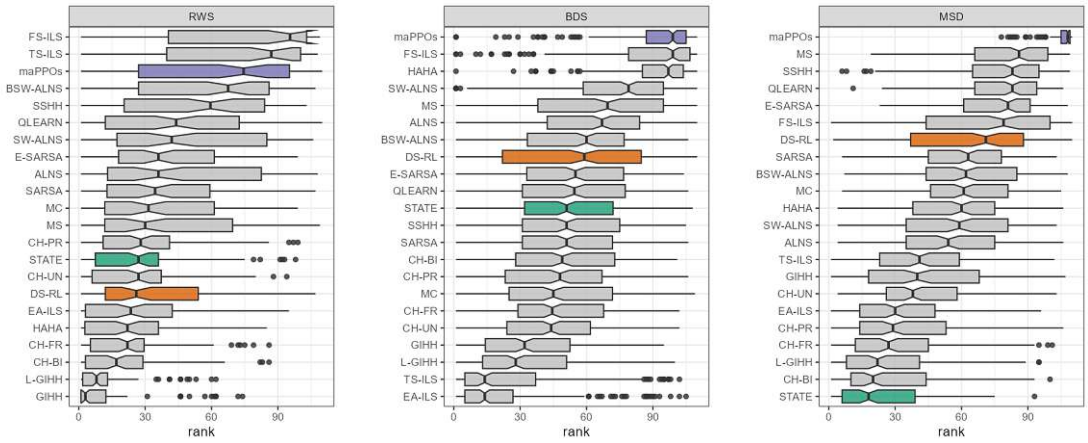

Figure 5.35: Comparison on real-life domains with boxplots of instance level ranks.

	Total	RSW	BDS	MSD
L-GIHH	**508.55**	**106.16**	214.6	172.0
EA-ILS	491.27	64.93	**351.6**	133.0
LAST-RL	439.46	38.93	161.6	**251.5**
GIHH	387.89	105.66	187.1	75.0
TS-ILS	311.81	2.63	346.0	67.0
CH-BI	294.81	56.83	70.6	143.0
CH-FR	239.26	36.43	81.6	116.0
CH-PR	219.01	29.86	74.6	112.0
CH-UN	159.63	26.93	85.6	52.5
HAHA	140.34	67.51	0.0	23.5
DS-RL	124.88	27.68	56.6	37.0
MS	121.53	58.38	28.0	2.0
MC	119.25	22.01	89.1	19.5
SARSA	76.43	18.43	51.6	9.0
QLEARN	73.59	32.83	25.1	0.0
E-SARSA	53.22	22.76	20.6	0.0
SSHH	53.18	11.13	48.1	1.0
BSW-ALNS	52.1	8.7	42.0	8.0
ALNS	46.7	13.92	12.0	14.0
FS-ILS	42.29	3.63	1.0	34.0
SW-ALNS	41.98	13.11	2.6	17.0
maPPOs	19.86	11.57	0.0	0.0

Table 5.31: Comparison on real-life domains with CHeSC 2011 scoring mechanism. Total score is the sum of the scores in the domains weighted by the number of instances per domain.

Figure 5.35 shows the ranks on instance level in parallel boxplots. DS-RL can compete with the state of the art in the RWS domain and in the domains BDS and MSD it shows that it can generalize but is not among the best approaches. For BDS, the large inter-quartile range of the boxplot for DS-RL indicates larger variance. maPPOs is not competitive in all three real-life domains.

In comparison to HyFlex, where a timeout of 392 seconds is used, a longer timeout of one hour is allowed here. The instances, and consequently also the low-level heuristics, are more complex and time-consuming in the real-life domains. Additionally, the runtimes of different low-level heuristics vary more significantly. While our hyper-heuristics would benefit from being tuned for this changed setup, we decided to test their generalizability to new problem domains using the tuned hyperparameter configuration that was determined on the HyFlex benchmark. This underlines the ability of DS-RL to generalize to unseen domains, even with a possibly suboptimal configuration.

CHAPTER 6

Conclusion

In this thesis, we proposed enhancements to the existing reinforcement learning based LAST-RL hyper-heuristic and designed two deep reinforcement learning based hyper-heuristics. Using the HyFlex benchmark suite, which includes instances for six problem domains, we conducted experiments for these enhancements and evaluated crucial design choices.

First, the existing search state for LAST-RL was extended with new features, whose influence on the scores was measured in a feature selection study. The experiments on the search state showed that a reduced set of five raw features is insufficient, as extracted features, including new features that have been proposed, improved the performance on the benchmark instances. Further experiments revealed advantages when excluding the time-dependent features and including the new proposed features that capture the improvement in the last H steps for each heuristic type. A welcome side effect is that these adjustments to the search state resulted in improved scores, particularly for two domains, MaxSAT and Bin Packing, where LAST-RL still showed potential for improvement on the benchmark instances. The second enhancement was an extended action space that outperforms the default action space in five (out of six) domains, where statistical significance is provided for two of them.

The major part of the work was the design of two hyper-heuristics that use the (double) deep Q-learning and the Proximal Policy Optimization algorithms. Both variants build upon the existing LAST-RL hyper-heuristic. In this design process, a broad range of design choices were introduced to address issues observed during preliminary experiments. The mentioned issues include, among others, the sparse rewards, catastrophic forgetting, the exploration-exploitation trade-off, non-stationarity, and numerical instabilities. The design choices and the available hyperparameters of the respective reinforcement learning algorithm were tuned with Bayesian Optimization to get best-performing models across the domains and, additionally, tuned models within each domain. Results showed that domain-specific models tend to overfit, while the models tuned across all domains

demonstrated superior ability to generalize to unseen instances. This underlines that our tuning strategy, with stratified splits of instances and a custom aggregate score, was effective. In further experiments with the model tuned across all domains, the impact and importance of higher and lower level design choices were evaluated. These experiments provided valuable insights: some design choices improved the performance across domains, while others showed interesting patterns within certain domains. Among the significant design choices for DS-RL are the feature normalization, the action space, and reward function design. For maPPOs, the choice of the advantage estimation, the number of agents, and the action space are among the most important design choices that have a measurable impact on the performance. Another objective of this work was to enable pre-training of the hyper-heuristic and the transfer of knowledge to new domains. We designed a progressive neural network architecture that can be pre-trained on various domains and used for inference on unseen domains. This approach was tested on the HyFlex benchmark and showed potential in initial experiments.

The final comparison with state of the art on the HyFlex benchmark suite showed that the enhancements for LAST-RL were effective and improved the scores for LAST-RL. The two proposed methods using deep Q-learning can only partially compete with the best approaches, showing significant drawbacks in the domains MaxSAT and Bin Packing. Allowing approximately the same number of steps as the more efficient LAST-RL hyper-heuristic led to more competitive results overall, also enabling better scores in the domains MaxSAT and Bin Packing. The approach with Proximal Policy Optimization turned out to be not competitive, even if we allow approximately the same number of steps as LAST-RL. We additionally provided evaluation on three real-life personnel scheduling domains where DS-RL demonstrated its ability to generalize to unseen domains but could not compete with the best approaches in these domains.

In future work, a full transfer analysis for the proposed progressive neural network could be conducted, and other techniques in lifelong reinforcement learning could be explored. Additionally, the use of distributional reinforcement learning could be promising, as the distribution of Q-values in this sparse reward setting is worth taking a closer look. However, increased computational costs of distributional reinforcement learning methods have to be taken into account. The evaluation of the proposed hyper-heuristics could be extended to other real-life domains.
Furthermore, hyperparameter tuning on real-life domains could be considered, as both the runtimes of the low-level heuristics and the timeout are longer than in the HyFlex domains, which might reduce the negative effect of the more computationally intensive methods.

Overview of Generative AI Tools Used

Generative AI-tools were only used as assistance in the thesis. ChatGPT helped to increase the expressiveness and readability without changing my own arguments and for proofreading. DeepL was employed for translating individual words or phrases. ChatGPT supported debugging tasks and reduced the effort for simpler tasks (e.g. reading log files) by providing code snippets. The outputs generated by these tools were treated carefully and only served as starting points or as checks for my own formulations and code.

List of Figures

List of Tables

List of Algorithms

Bibliography

[Adriaensen et al., 2014] Adriaensen, S., Brys, T., and Nowé, A. (2014). Fair-share ils: a simple state-of-the-art iterated local search hyperheuristic. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, page 1303–1310. Association for Computing Machinery.

[Adriaensen et al., 2015] 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)*, pages 784–791.

[Adubi et al., 2021] 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)*, pages 1372–1379.

[Alanazi, 2016] Alanazi, F. (2016). Adaptive thompson sampling for hyper-heuristics. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8. IEEE.

[Andrychowicz et al., 2020] Andrychowicz, M., Raichuk, A., Stanczyk, P., Orsini, M., Girgin, S., Marinier, R., Hussenot, L., Geist, M., Pietquin, O., Michalski, M., Gelly, S., and Bachem, O. (2020). What matters in on-policy reinforcement learning? a large-scale empirical study. *ArXiv*, abs/2006.05990.

[Asta et al., 2013] Asta, S., Özcan, E., and Parkes, A. J. (2013). Batched mode hyper-heuristics. In *Learning and Intelligent Optimization*, pages 404–409. Springer.

[Burke and Kendall, 2014] Burke, E. and Kendall, G. (2014). *Search methodologies: Introductory tutorials in optimization and decision support techniques, second edition.* Springer.

[Burke et al., 2011] 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 *Learning and Intelligent Optimization*, pages 631–634. Springer.

[Burke et al., 2013] 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 et al., 2010] Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Woodward, J. R. (2010). *A Classification of Hyper-heuristic Approaches*, pages 449–468. Springer.

[Calvo and Santafé, 2016] Calvo, B. and Santafé, G. (2016). scmamp: Statistical comparison of multiple algorithms in multiple problems. *The R Journal*, 8(1):248–256.

[Choong et al., 2018] Choong, S. S., Wong, L.-P., and Lim, C. P. (2018). Automatic design of hyper-heuristic based on reinforcement learning. *Information Sciences*, 436-437:89–107.

[Dantas and Pozo, 2021] Dantas, A. and Pozo, A. (2021). Online selection of heuristic operators with deep q-network: A study on the hyflex framework. In *Intelligent Systems*, pages 280–294. Springer.

[Demšar, 2006] Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research (JMLR)*, 7:1–30.

[Drake et al., 2020] 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.

[Hasselt et al., 2016] Hasselt, H. v., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, page 2094–2100. AAAI.

[Hsiao et al., 2012] 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*, pages 1–8. IEEE.

[Hyde et al., 2010a] Hyde, M., Ochoa, G., Vázquez-Rodríguez, J. A., and Curtois, T. (2010a). A hyflex module for the max-sat problem. Technical report, Automated Scheduling, Optimisation and Planning (ASAP) Group, School of Computer Science, University of Nottingham.

[Hyde et al., 2010b] Hyde, M., Ochoa, G., Vázquez-Rodríguez, J. A., and Curtois, T. (2010b). A hyflex module for the one dimensional bin packing problem. Technical report, Automated Scheduling, Optimisation and Planning (ASAP) Group, School of Computer Science, University of Nottingham.

[Hyde et al., 2010c] Hyde, M., Ochoa, G., Vázquez-Rodríguez, J. A., and Curtois, T. (2010c). A hyflex module for the permutation flow shop problem. Technical report, Automated Scheduling, Optimisation and Planning (ASAP) Group, School of Computer Science, University of Nottingham.

[Hyde et al., 2010d] Hyde, M., Ochoa, G., Vázquez-Rodríguez, J. A., and Curtois, T. (2010d). A hyflex module for the personnel scheduling problem. Technical report,

Automated Scheduling, Optimisation and Planning (ASAP) Group, School of Computer Science, University of Nottingham.

[Iman and Davenport, 1980] Iman, R. L. and Davenport, J. M. (1980). Approximations of the critical region of the friedman statistic. *Communications in Statistics - Theory and Methods*, 9(6):571–595.

[Kallestad et al., 2023] Kallestad, J., Hasibi, R., Hemmati, A., and Sörensen, K. (2023). A general deep reinforcement learning hyperheuristic framework for solving combinatorial optimization problems. *European Journal of Operational Research*, 309(1):446–468.

[Kheiri and Keedwell, 2015] 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*, page 417–424. Association for Computing Machinery.

[Kletzander and Musliu, 2023] Kletzander, L. and Musliu, N. (2023). Large-state reinforcement learning for hyper-heuristics. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(10):12444–12452.

[Kletzander and Musliu, 2024] Kletzander, L. and Musliu, N. (2024). Hyper-heuristics for personnel scheduling domains. *Artificial Intelligence*, 334:104172.

[Knuth, 1997] Knuth, D. E. (1997). *The art of computer programming, volume 2 (3rd ed.): seminumerical algorithms*. Addison-Wesley Longman Publishing Co., Inc.

[Kruskal and Wallis, 1952] Kruskal, W. H. and Wallis, W. A. (1952). Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association*, 47(260):583–621.

[Lehrbaum and Musliu, 2012] Lehrbaum, A. and Musliu, N. (2012). A new hyperheuristic algorithm for cross-domain search problems. In *Learning and Intelligent Optimization*, pages 437–442. Springer.

[Li et al., 2014] Li, K., Fialho, A., Kwong, S., and Zhang, Q. (2014). Adaptive operator selection with bandits for a multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 18(1):114–130.

[Lin, 1992] Lin, L.-J. (1992). *Reinforcement learning for robots using neural networks*. PhD thesis, Carnegie Mellon University, USA.

[Lindauer et al., 2022] Lindauer, M., Eggensperger, K., Feurer, M., Biedenkapp, A., Deng, D., Benjamins, C., Ruhkopf, T., Sass, R., and Hutter, F. (2022). Smac3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54):1–9.

[Lou et al., 2024] Lou, Z., Wang, Y., Shan, S., Zhang, K., and Wei, H. (2024). Balanced prioritized experience replay in off-policy reinforcement learning. *Neural Computing and Applications*, 36(25):15721–15737.

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

[McGill et al., 1978] McGill, R., Tukey, J. W., and Larsen, W. A. (1978). Variations of box plots. *The American Statistical Association*, 32(1):12–16.

[Mischek and Musliu, 2022] Mischek, F. and Musliu, N. (2022). Reinforcement learning for cross-domain hyper-heuristics. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*, pages 4793–4799. IJCAI.

[Mısır et al., 2012] Mısır, M., Verbeeck, K., De Causmaecker, P., and Vanden Berghe, G. (2012). An intelligent hyper-heuristic framework for chesc 2011. In *Learning and Intelligent Optimization*, pages 461–466. Springer.

[Mnih et al., 2013] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. (2013). Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602.

[Nawaz et al., 1983] Nawaz, M., Enscore, E. E., and Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95.

[Nemenyi, 1963] Nemenyi, P. (1963). *Distribution-free Multiple Comparisons*. PhD thesis, Princeton University.

[Ochoa et al., 2012] Ochoa, G., Hyde, M., Curtois, T., Vazquez-Rodriguez, J. A., Walker, J., Gendreau, M., Kendall, G., McCollum, B., Parkes, A. J., Petrovic, S., and Burke, E. K. (2012). Hyflex: A benchmark framework for cross-domain heuristic search. In *Evolutionary Computation in Combinatorial Optimization*, pages 136–147. Springer.

[Pathak et al., 2017] Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the 34th International Conference on Machine Learning*, page 2778–2787. JMLR.

[Pillay and Qu, 2018] Pillay, N. and Qu, R. (2018). *Hyper-Heuristics: Theory and Applications*. Springer.

[Reijnen et al., 2024] Reijnen, R., Zhang, Y., Lau, H. C., and Bukhsh, Z. (2024). Online control of adaptive large neighborhood search using deep reinforcement learning. *Proceedings of the International Conference on Automated Planning and Scheduling*, 34(1):475–483.

[Ropke and Pisinger, 2006] Ropke, S. and Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472.

[Ross, 2014] Ross, P. (2014). *Hyper-heuristics*, pages 611–638. Springer.

[Rusu et al., 2016] Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. (2016). Progressive neural networks. *ArXiv*, abs/1606.04671.

[Schaul et al., 2015] Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay. *CoRR*, abs/1511.05952.

[Schrimpf et al., 2000] Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H., and Dueck, G. (2000). Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, 159(2):139–171.

[Schulman et al., 2015a] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015a). Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pages 1889–1897. PMLR.

[Schulman et al., 2015b] Schulman, J., Moritz, P., Levine, S., Jordan, M. I., and Abbeel, P. (2015b). High-dimensional continuous control using generalized advantage estimation. *CoRR*, abs/1506.02438.

[Schulman et al., 2017] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347.

[Sun et al., 2022] Sun, H., Li, R., Yang, H., and Zhu, W. (2022). Balanced prioritized experience replay. In *3rd International Conference on Electronic Communication and Artificial Intelligence (IWECAI)*, pages 200–203.

[Sutton and Barto, 2018] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction (2nd ed.)*. MIT Press.

[Tauritz and Woodward, 2016] Tauritz, D. R. and Woodward, J. (2016). Hyper-heuristics. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, page 273–304. Association for Computing Machinery.

[Thrun and Mitchell, 1995] Thrun, S. and Mitchell, T. M. (1995). Lifelong robot learning. In *The Biology and Technology of Intelligent Autonomous Agents*, pages 165–196. Springer.

[Tu et al., 2023] Tu, C., Bai, R., Aickelin, U., Zhang, Y., and Du, H. (2023). A deep reinforcement learning hyper-heuristic with feature fusion for online packing problems. *Expert Systems with Applications*, 230:120568.

[Udomkasemsub et al., 2023] Udomkasemsub, O., Sirinaovakul, B., and Achalakul, T. (2023). Phh: Policy-based hyper-heuristic with reinforcement learning. *IEEE Access*, 11:52026–52049.

[Van Onsem et al., 2015] Van Onsem, W., Demoen, B., and De Causmaecker, P. (2015). Learning a hidden markov model-based hyper-heuristic. In *Learning and Intelligent Optimization*, pages 74–88. Springer.

[Walker et al., 2012] Walker, J. D., Ochoa, G., Gendreau, M., and Burke, E. K. (2012). Vehicle routing and adaptive iterated local search within the hyflex hyper-heuristic framework. In *Learning and Intelligent Optimization*, pages 265–276. Springer.

[Zhang et al., 2022] Zhang, Y., Bai, R., Qu, R., Tu, C., and Jin, J. (2022). A deep reinforcement learning based hyper-heuristic for combinatorial optimisation with uncertainties. *European Journal of Operational Research*, 300(2):418–427.