Reinforcement Learning Hyper-heuristics for Optimisation

A THESIS SUBMITTED TO THE UNIVERSITY OF NOTTINGHAM FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

AUTHOR:

FAWAZ ALANAZI

UNITED KINGDOM, 2017

THE UNIVERSITY OF NOTTINGHAM

Abstract

Hyper-heuristics are search algorithms which operate on a set of heuristics with the goal of solving a wide range of optimisation problems. It has been observed that different heuristics perform differently between different optimisation problems. A hyper-heuristic combines a set of predefined heuristics, and applies a machine learning technique to predict which heuristic is the most suitable to apply at a given point in time while solving a given problem. A variety of machine learning techniques have been proposed in the literature. Most of the existing machine learning techniques are reinforcement learning mechanisms interacting with the search environment with the goal of adapting the selection of heuristics during the search process. The literature on the theoretical foundation of reinforcement learning hyper-heuristics is almost non-existing. This work provides theoretical analyses of reinforcement learning hyper-heuristics. The goal is to shed light on the learning capabilities and limitations of reinforcement learning hyper-heuristics. This improves our understanding of these hyper-heuristics, and aid the design of better reinforcement learning hyper-heuristics. It is revealed that the commonly used additive reinforcement learning mechanism, under a mild assumption, chooses asymptotically heuristics uniformly at random. This thesis also proposes the problem of identifying the most suitable heuristic with a given error probability. We show a general lower bound on the time that "every" reinforcement learning hyper-heuristic needs to identify the most suitable heuristic with a given error probability. The results reveal a general limitation to learning achieved by this computational approach. Following our theoretical analysis, different reusable and easy-to-implement reinforcement learning hyper-heuristics are proposed in this thesis. The proposed hyper-heuristics are evaluated on well-known combinatorial optimisation problems. One of the proposed reinforcement learning hyper-heuristics outperformed a state-of-the-art algorithm on several benchmark problems of the well- known CHeSC 2011.

Acknowledgement

I would like to express my profound gratitude to Almighty God for all the physical and mental strength He bestowed on me during all the activities of the project and finally during the preparation of this thesis.

Need less to say, I am deeply grateful to Dr. Per Kristian Lehre , my supervisor who generously rendered help and encouragement during the process of this thesis. Whatever I have accomplished in pursuing this undertaking is due to his guidance and thoughtful advice. To him my thanks.

I would like also to thank all the automated scheduling, optimisation, and planning (ASAP) research group members.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Nowadays, many decisions are made automatically by decision-support systems ranging from personnel scheduling to self-driving cars. The concept of *optimisation* is central in the context of decision-support systems and often interpreted as the action of finding the best possible solution among various alternatives under given constraints (see, for example, (Steiglitz and Papadimitriou, 1982); and (Chong and Zak, 2013)). Accordingly, the goal of optimisation algorithms is to find efficiently an optimal or a near-optimal solution from a set of solutions. Optimisation algorithms may broadly be classified into two main classes: exact algorithms and heuristic algorithms. The process of designing exact algorithms for a particular problem often consists of extensively learning about and exploiting structural properties of the problem. Many real-world optimisation problems of practical interest are NP-hard, and known exact algorithms fail to solve them efficiently. In contrast, heuristics do not require deep insights about the problem at hand. They can be considered as "rules of thumb", aiming to return a solution of acceptable quality within a reasonable time. Therefore, practitioners often resort to heuristics in an attempt to achieve desirable results in a reasonable time when no exact algorithm is known or too expensive to design.

Randomised search heuristics are multi-purpose optimisation algorithms that are applicable

to a wide range of optimisation problems. Some randomised search heuristics gather information about the problem only through evaluating the candidate solutions using the objective function. This property allows the heuristic algorithm to be applied to many optimisation problems rather than a single one, and these problems are referred to as black-box optimisation problems (Droste et al., 2002). An important result in this context is the No Free Lunch (NFL) theorem (Wolpert and Macready, 1997) which states that all search algorithms have the same performance when averaged over all possible discrete optimisation problems (Auger and Doerr, 2011). The NFL theorem holds for any class of problems that is closed under permutation of the search space (Jansen, 2013). However, it should be noted that the NFL theorem does imply any restriction on the generality of black-box algorithms when the considered set of problems is not closed under permutation.

The performance of search heuristics is usually measured based on their runtime to find an optimal solution or a solution of satisfactory quality (Auger and Doerr, 2011). The *runtime* (also called optimisation or hitting time) of a search heuristic is defined as the number of objective function evaluations required to obtain an optimal solution. Since randomised search heuristics make arbitrary decisions during their runs, they may perform differently in every run, even on the same problem instance. This implies that the runtime of the search heuristic is a random variable. The expected runtime of the search heuristic is therefore used as a performance measure. *Runtime analysis* is the activity of estimating the *runtime* and the *success probability* of a search heuristic via rigorous mathematical proofs (see, for example, (Auger and Doerr, 2011); (Neumann and Witt, 2012); (Jansen, 2013)). The definition of the *success probability* of search heuristics is given in Section 2.6.

There is a growing interest in search heuristics that combine a set of heuristics and adaptively learn how to choose and apply the most suitable heuristic at any given time during the search process, such as hyper-heuristics (Burke et al., 2013). Hyper-heuristics are multi-purpose

search techniques for either selecting heuristics from a set of predefined heuristics or generating heuristics based on a set of predefined heuristic components with the goal of solving a given optimisation problem (Chakhlevitch and Cowling, 2008) (Burke et al., 2013) (Ross, 2014). The former is termed *selection hyper-heuristics*, and the latter *generation hyper-heuristics* (Burke et al., 2010b). The term "hyper-heuristic" was recently introduced by Cowling et al. (2001); however, the idea of combining multiple heuristics for solving a given problem originated in the 1960*s* (Fisher and Thompson, 1963) (Crowston et al., 1963).

Most of the previously proposed hyper-heuristics in the literature are single-point based selection hyper-heuristics (Ross, 2005) (Özcan et al., 2008) (Burke et al., 2013). A typical single-point selection hyper-heuristic evolves an initially generated solution by iteratively (i) selecting a heuristic from a set of predefined low-level heuristics (variation operators); (ii) applying the selected heuristic to the candidate solution, thus generating a new solution; (iii) evaluating the newly generated solution and deciding whether it should replace the existing solution or reject it. This process is repeated until the termination conditions are satisfied. A selection hyper-heuristic employs a *heuristic selection* method to choose a heuristic from a pool of predefined low-level heuristics, and a *move acceptance* strategy to decide whether to accept or discard the newly generated solution (Özcan et al., 2008). On a conceptual level, the current *de facto* conceptions of selection hyper-heuristics are based on separating the hyper-heuristic from the problem domain by a *domain barrier*. The *domain barrier* is meant to prevent problem-dependent information from being passed to the hyper-heuristic level such that hyper-heuristics interact with the problem domain only through problem-independent information such as the change in the objective function value after applying a selected low-level heuristic. This design decision increases the generality of the hyper-heuristics and allows the reusability of their main components on unseen problem instances or problem domains (Chakhlevitch and Cowling, 2008) (Burke et al., 2013). Selection hyper-heuristics have been favourably applied to several optimi-

Table 1.1: Examples of selection hyper-heuristic application domains.

Problem Domain	Reference
University timetabling	(Burke et al., 2005a)
Vehicle routing problem	(Pisinger and Ropke, 2007)
Packing problems	(Marín-Blázquez and Schulenburg, 2007)
Personnel scheduling	(Cowling and Chakhlevitch, 2003)
Quadratic assignment problem	(Dokeroglu and Cosar, 2016)
Production scheduling	(Marín-Blázquez and Schulenburg, 2007)
Space allocation	(Bai and Kendall, 2005)
Nurse rostering	(Asta et al., 2016)
Water distribution network	(Kheiri et al., 2015)
Workforce scheduling	(Remde et al., 2007)

sation problems. Some examples of selection hyper-heuristic application domains are given in Table 1.1.

A primary goal of a class of selection hyper-heuristics is to provide solutions of acceptable qualities on various problems by exploiting the strengths of multiple heuristics (Chakhlevitch and Cowling, 2008) (Özcan et al., 2008). A selection hyper-heuristic operates on multiple low-level heuristics, and applies a machine learning technique to predict which heuristic is the most suitable to apply at a given point in time. Reinforcement learning is a general machine learning technique inspired by the natural way that human beings learn about their environments (Kaelbling et al., 1996) (Sutton and Barto, 1998). A reinforcement learning mechanism in hyper-heuristics is a heuristic selection method interacting with the search environment with the goal of identifying the most appropriate low-level heuristic to apply at any given time while solving a given problem. A challenge for reinforcement learning hyper-heuristics is that typical low-level heuristics perform differently during the search process. Therefore, reinforcement learning hyper-heuristics have to balance between exploring the effectiveness of low-level heuristics with respect to the current search point, and exploiting the most suitable low-level heuristics based on previous observations. This is termed the "exploration versus exploitation" trade-off. The focus of this thesis is on this aspect of hyper-heuristics, how reinforcement learning hyper-heuristics

adapt themselves for solving a given problem.

1.1 Motivation and Research Questions

Runtime analysis of reinforcement learning hyper-heuristics is almost non-existing. Such analyses often show how the runtime of a search heuristic depends on both the design of the search heuristic and the characteristics of the problem. Runtime analysis can also provide insights into the limitations and capabilities of search heuristics and sheds light on the reasons for the success or failure. This can, in turn, aid the design of better search heuristics. This thesis provides runtime analyses of reinforcement learning hyper-heuristics. Commonly used heuristic selection methods in hyper-heuristics are simple reinforcement learning mechanisms such as *random gradient*, *greedy*, and the *additive reinforcement learning mechanism* (Özcan et al., 2008) (Cowling et al., 2001) (Özcan et al., 2010). We present the first theoretical study evaluating the performance of these reinforcement learning mechanisms and compare them to a uniform at random selection hyper-heuristic on a benchmark problem. Several empirical studies have reported inconclusive results about the performance of the additive reinforcement learning mechanism (Nareyek, 2004) (Remde et al., 2007) (Özcan et al., 2010). A question arises: under which conditions the additive reinforcement learning mechanism does not improve the performance when compared to a uniform at random selection hyper-heuristic?

Furthermore, each reinforcement learning hyper-heuristic needs to gather information about the performance of low-level heuristics to predict which low-level heuristic is the most suitable to apply at a given point in time during the search process. An important question in this context is: how much runtime does each reinforcement learning hyper-heuristic need to explore the performance of the low-level heuristics and identify the most suitable heuristic with a given error probability? This research question has not been investigated in the field of hyper-heuristics

and contributes to a better understanding of the learning capabilities of reinforcement learning hyper-heuristics. This thesis formulates the problem of identifying the most suitable low-level heuristic with a given error probability and theoretically investigates the above question.

The so-called *multi-armed bandit* problem provides general models that allow evaluating the capabilities of machine learning mechanisms for handling the exploration versus exploitation trade-off. Thompson (1933) proposed a reinforcement learning mechanism, called Thompson Sampling. Thompson Sampling is an optimal reinforcement learning mechanism for a so-called *stationary* multi-armed bandit (Agrawal and Goyal, 2013) (Granmo, 2010). This thesis presents the first study investigating Thompson Sampling in the field of hyper-heuristics and investigates the following question. Can Thompson Sampling be used as an *adaptive* learning mechanism in a hyper-heuristic?

1.2 Contributions

The contributions of this thesis are summarised as follows.

- We propose an artificial benchmark problem called Uniform Bernoulli (see Chapter 4). The proposed problem allows performance evaluation of hyper-heuristics with respect to different scenarios in a controlled environment. This contributes to both theoretical and empirical evaluations of hyper-heuristics.

- We present the first study investigating theoretically the behaviour of the commonly used additive reinforcement learning mechanism, and show the conditions under which it chooses asymptotically low-level heuristics uniformly at random (see Chapter 4). This contribution not only shows limitations to learning achieved by the additive reinforcement learning mechanism but also participates in building the theoretical foundation of reinforcement learning hyper-heuristics.

- Following our theoretical analysis, a modified *additive* reinforcement learning mechanism has been proposed in this thesis (see Chapter 5). The proposed algorithm is tested and compared to a large number of hyper-heuristics, including a state-of-the-art algorithm, within a well-known competition in the field of hyper-heuristics called CHeSC 2011, on the bin packing problem. The results show that the proposed algorithm produces competitive, often better, results compared to the CHeSC 2011 hyper-heuristics.

- We present the first study investigating Thompson Sampling in the field of hyper-heuristics and contribute the following algorithms (see Chapter 5):

  - We propose an adaptive Thompson Sample hyper-heuristic for a set of well-known combinatorial optimisation problems: Boolean satisfiability, bin packing, personnel scheduling, permutation flow-shop, travelling salesman problem, and the vehicle routing problem. To show the effectiveness of the proposed approach, we compare the proposed hyper-heuristic to a large number of hyper-heuristics, including a state-of-the-art algorithm, within the CHeSC 2011 competition on the problems noted above. The results show the proposed hyper-heuristic outperforms the CHeSC 2011 hyper-heuristics on three problem domains: personnel scheduling, permutation flow-shop, and the travelling salesman problem.

  - We introduce the *Hierarchical Thompson Sampling* hyper-heuristic for tackling given optimisation problems with a large number of low-level heuristics. We evaluate the proposed hyper-heuristic with a large number of low-level heuristics and compare it to the so-called Hyper-random algorithm on two well-known optimisation problems: permutation flow-shop and travelling salesman problem. The results show that the proposed algorithm achieves competitive, if not better, results compared to the Hyper-random algorithm.

- How much learning do reinforcement learning hyper-heuristics need to identify the most suitable low-level heuristic? We formulate this problem, for the first time in the field of hyper-heuristic, and contribute a lower bound on the runtime that each reinforcement learning hyper-heuristic must perform in order to identify the most suitable low-level heuristic with a given error probability on a general model (see Chapter 6).

1.2.1 Publications

The following scientific papers and posters have been published.

- Alanazi, F. and Lehre, P.K. (2016). Limits to Learning in Reinforcement Learning Hyper-heuristics. In Evolutionary Computation in Combinatorial Optimization, pages 170-185. Springer International Publishing.

- Alanazi F, Lehre P.K. (2014). Runtime analysis of selection hyper-heuristics with classical learning mechanisms. In IEEE World Congress on Evolutionary Computation (CEC), pages 2515-2523. IEEE.

- Alanazi, F. (2016). Adaptive Thompson Sampling for hyper-heuristics. In Symposium Series on Computational Intelligence (SSCI), pages (1-8). IEEE.

- Alanazi, F. and Lehre, P.K. (2013). A theoretical investigation of learning mechanisms within a hyper-heuristic framework. In the sixth Saudi students conference. Edinburgh University, Scotland.

- Alanazi, F. and Lehre, P.K. (2015). Reinforcement learning mechanisms for hyper-heuristics. In the eighth Saudi students conference. Imperial College London, United Kingdom.

Additionally, the author presented an abstract about the runtime analysis of hyper-heuristics in the 7th workshop on Theory of Randomised Search Heuristics (ThRaSH 2013) took place

in September (2013) in Aberystwyth, Wales. A poster about theoretical foundations of hyper-heuristics was presented in the East Midlands Universities (EMU) conference took place in September (2013) in Derby, England.

1.3 Thesis Structure

This thesis manuscript is structured as follows:

- Chapter 1: provides a brief introduction to the research described in this thesis. Additionally, it highlights the contributions of this thesis.
- Chapter 2: gives an overview of the main concepts and the relevant backgrounds of the research studies carried out in this thesis. It also summaries the research methods.
- Chapter 3: presents the literature review discussing the existing solutions of the research problems.
- Chapter 4: presents the theoretical benchmark problems and shows our runtime analyses of a generic hyper-heuristic with a set of reinforcement learning mechanisms and no-learning methods.
- Chapter 5: introduces three reinforcement learning hyper-heuristics: the adaptive Thompson Sampling, Hierarchical Thompson Sampling, and the multiplicative weight update hyper-heuristics. It also provides detailed evaluations of their performances on a wide range of optimisation problems.
- Chapter 6: formulates the problem of identifying the most suitable heuristic with a given error probability. We present general theorems on the runtime that each reinforcement learning hyper-heuristic needs to identify the most suitable low-level heuristic with a given error probability.

- Chapter 7: draws the conclusions from the research studies carried out in this thesis along with directions for future studies.

Chapter 2

Preliminaries

This chapter provides preliminaries for the research described in this thesis, as well as definitions of the research tools and methods.

2.1 Single-objective Optimisation

Optimisation problems are usually investigated where the value of one or more *objective* functions one need to be maximised or minimised. Objective functions (also called fitness, cost, or evaluation functions) assign to each solution a value that represents its quality. Optimisation problems are often grouped into two main categories: *single-objective* and *multi-objective* optimisation problems. In the former category, one single objective function has to be optimised; whereas in the latter category, multiple objective functions have to be optimised at the same time. Single-objective optimisation involves the process of finding a solution(s) that maximises or minimises the function value of the given single objective function. A single-objective optimisation problem can either be a problem with a discrete search space or a problem with a continuous search space. The former is often referred to as a *combinatorial optimisation problem*, while the latter is known as a *continuous optimisation problem*. This thesis focuses on

single-objective combinatorial optimisation problems. We consider such optimisation problems for two main reasons: (1) most of the proposed hyper-heuristics in the literature are designed for single-objective optimisation (Burke et al., 2013); (2) research into the runtime analysis of Evolutionary Algorithms reveals that it is important first to analyse the algorithms on simple combinatorial optimisation problems before more complicated problems are considered. For example, almost all the work on the runtime analysis of Evolutionary Algorithms was on a simple Evolutionary Algorithm, called $(1+1)$ Evolutionary Algorithm, on simple combinatorial optimisation problems until the mid of the last decade (see, for example, (Oliveto et al., 2007)).

A single-objective combinatorial optimisation problem is defined as follows (Cormen, 2009).

Definition 1 (single-objective combinatorial optimisation problem)**.** *A single-objective combinatorial optimisation problem is defined as a triple ($\mathscr{X}$, f, Ω) where:*

- *$\mathscr{X}$ is the search space which is a non-empty finite set of all possible solutions.*
- *$f : \mathscr{X} \to \mathbb{R}$ is the objective function which maps each search point $x \in \mathscr{X}$ in the search space to a value that represents its quality.*
- *$\Omega \subseteq \mathscr{X}$ is the subset of the search space which contains the search points that satisfy the given constraints.*

A single-objective combinatorial optimisation problem is either a maximisation or a minimisation problem. Each instance of the problem comes with a finite search space $\mathscr{X}$. For example, a search space $\mathscr{X}$ for the classical travelling salesman problem is a set of all possible permutations of the cities. Each search point $x \in \mathscr{X}$ that fulfils the given constraints, such that $x \in \Omega$, is called a feasible solution. Each feasible solution is assigned to a value that represents its quality by an objective function f defined on the search space $\mathscr{X}$. A search point is called locally optimal if its quality with respect to a given objective function is the best among its neighbourhoods, where neighbourhoods are a subset of search points that can be reached

directly from a given search point through some modification. While, a search point is called globally optimal if it has the best objective function value among all feasible solutions.

2.2 Optimisation Algorithms

We distinguish between two types of optimisation algorithms for solving combinatorial optimisation problems: *exact algorithms* and *heuristic algorithms*. An alternative classification could identify another main class of the optimisation algorithms that is in the middle between exact algorithms and heuristic algorithms called *approximation algorithms*. Designing exact or approximation algorithms typically require deep insights about the problem at hand. Specifically, the process of designing such algorithms consists of two main steps: (i) identifying the structure of the problem; and (ii) exploiting this structure and finding an algorithmic technique for solving the given problem (Vazirani, 2013). This design process is further complicated by the fact that many combinatorial optimisation problems of practical importance are NP-hard, and the structures of NP-hard problems are not completely understood.

On the other hand, heuristics are stochastic approaches to find a solution of acceptable quality within a reasonable time, but the optimality of solutions is not guaranteed. In some cases, heuristics do not find any solution to a given problem within a reasonable computational time. The applications of heuristics do not assume deep insights about the given problem beyond the description of how to evaluate the objective function. However, problem insights can be exploited to design better heuristics.

The primary objective of research on search heuristics is to define multi-purpose optimisation algorithms capable of guiding the search needed to find a solution that is globally optimal by preventing the search being drawn into local optima (Glover and Laguna, 2013). Search heuristics have to balance between diversifying the search to avoid getting stuck in local op-

tima and intensifying the search with the goal of finding an optimal or a near-optimal solution. This is termed "diversification versus intensification" trade-off. Popular examples of search heuristics are represented by *Iterated Local Search* (Section 2.2.1) and *Evolutionary Algorithms* (Section 2.2.2).

2.2.1 Iterated Local Search

Local search algorithms are simple search heuristics that improve an initially generated solution iteratively by moving from one search point to a neighbouring search point. Hill-climbing is a commonly used local search algorithm that iteratively searches the neighbourhoods of the candidate solution to find a better solution to replace the current candidate solution. Such algorithms can easily get stuck in local optima (see, for example, (Lourenço et al., 2003)). *Iterated Local Search* (ILS) can be considered as an extension of local search algorithms to remedy the shortcoming mentioned above. Algorithm 1 shows a precise description of a generic ILS algorithm (Lourenço et al., 2003). ILS attempts to balance between diversification and intensification trade-off by first applying a perturbative heuristic that could lead to a new solution, but not necessarily in the neighbourhood of the current solution. Then a local search algorithm is applied to the newly generated solution. If the acceptance strategy accepts the new solution, then it replaces the current candidate solution; otherwise, it is rejected.

Algorithm 1 Generic Iterated Local Search

1: Given a finite set $\mathscr{X}$, and an objective function $f : \mathscr{X} \to \mathbb{R}$.
2: Choose an initial solution $x \in \mathscr{X}$
3: $x := LocalSearch(x)$ {apply a local search heuristic}
4: **while** the termination condition is not satisfied **do**
5: $x' := Perturbation(x)$ {apply a perturbative heuristic}
6: $x'' := LocalSearch(x')$
7: **if** x'' is accepted **then**
8: $x := x''$
9: **end if**
10: **end while**

2.2.2 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are general-purpose search heuristics. Their inventors were inspired by the Darwinian concept of "survival of the fittest" in nature (Holland, 1992) (Bäck et al., 1997). Several variants of EAs have been proposed in the literature and successfully applied to numerous optimisation problems (Yu et al., 2008). Most EAs evolve a set of candidate solutions, called a population, as follows. A set of individuals (parents) are chosen from the population according to a selection method which usually selects individuals based on their objective function values. Some variation operators, such as mutation and crossover operators, are applied to produce new individuals (offspring). The newly generated individuals are evaluated with respect to a given objective function. Then the next parents are formed by a set of individuals chosen from the union of both the offspring and current population. This process is repeated until a stopping condition is satisfied.

2.3 Problem-dependent Parameters in Search Heuristics

Search heuristics often have one or more problem-dependent parameters. The choice of the parameters values could lead to major performance differences. The problem-dependent parameters in search heuristics, such as population size and mutation rate in EAs, can be considered as a kind of flexibility in these optimisation algorithms, which makes them applicable to many optimisation problems. On the other hand, the process of finely tuning these parameters is both challenging and time-consuming (Eiben and Smit, 2011). It is plausible that particular problems require specific settings of these parameters to obtain satisfactory solutions (Eiben et al., 2007). The issue of tuning the parameters of search heuristics appears as one of the main challenges when it comes to applying search heuristics to solve a new optimisation problem (Lobo et al., 2007) (Dobslaw, 2010) (Eiben and Smit, 2011). Recently much attention has been paid to define

more autonomous search heuristics that can learn and adapt parameters themselves while solving a given problem. Popular examples of such search heuristics include, *self-adaptive search heuristics* (Lobo et al., 2007), and *hyper-heuristics* (Burke et al., 2013).

2.4 Hyper-heuristics

The term "hyper-heuristic", attributed to Cowling et al. (2001), refers to a multi-purpose optimisation algorithm that either selects a heuristic from a set of predefined low-level heuristics (variation operators) or generates a heuristic based on pre-existing low-level heuristic components for solving a given problem (Burke et al., 2013) (Chakhlevitch and Cowling, 2008) (Özcan et al., 2008). In the context of hyper-heuristics, a primary goal is to develop search heuristics that are easy-to-implement and applicable to various optimisation problems (Burke et al., 2013). As discussed, the problem-dependent parameters in traditional search heuristics require significant expertise to tune them. Hyper-heuristic approaches aim to provide results of acceptable qualities, with or without little modification, on a range of optimisation problems by exploiting the strengths of multiple low-level heuristics. This level of generality to achieve desirable results on various problems is motivated by the fact that there is a growing interest in practice towards easy-to-maintain algorithmic frameworks that could work on different instances of a certain problem or even across various problems, and provide solutions of acceptable qualities (Burke et al., 2013).

2.4.1 A Classification of Hyper-heuristic Methodologies

Hyper-heuristics are broadly divided into two main types: algorithms for selecting heuristics from a prefixed set of low-level heuristics, and algorithms for generating heuristics from a predefined set of low-level heuristic components with the aim of solving a given problem (Burke et al., 2010b). According to the natural type of low-level heuristics, hyper-heuristics may be classified

into perturbative and constructive hyper-heuristics. Perturbative hyper-heuristics generate an initial complete solution(s) and then operate to improve it. While constructive hyper-heuristics start with an empty or partial solution(s) and attempt to produce a complete solution(s). A typical perturbative low-level heuristic can be defined as a function that returns a complete solution given one or more complete solutions. While, a constructive low-level heuristic is a function that returns a partial or complete solution given one or more partial or empty solutions.

Hyper-heuristics can be further broken down according the learning methodologies into three categories: online-learning, offline-learning, and no-learning hyper-heuristics. Online-learning hyper-heuristics interact with the search environment and learn from the feedback while solving a given problem; whereas, offline-learning hyper-heuristics are those hyper-heuristics that learn on some training instances before the actual search starts. No-learning hyper-heuristics do not use any learning mechanism.

We are interested in online-learning perturbative selection hyper-heuristics. Such hyper-heuristics are search techniques that operate on a set of perturbative low-level heuristics and adapt themselves through online learning for solving a given optimisation problem. Most of the previously proposed hyper-heuristics in the literature are perturbative selection hyper-heuristics. Therefore, we focus on this type of hyper-heuristics in our theoretical investigations.

2.5 Selection Hyper-heuristics

This section describes the hyper-heuristic framework considered in this thesis. Figure 2.1 illustrates pictorially the modus operandi of the considered hyper-heuristic framework; a formal description in the form of a pseudo-code is given in Algorithm 2.

Algorithm 2 is a generic selection hyper-heuristic which combines a heuristic selection and move-acceptance strategies as its main components within a single-point based framework. The

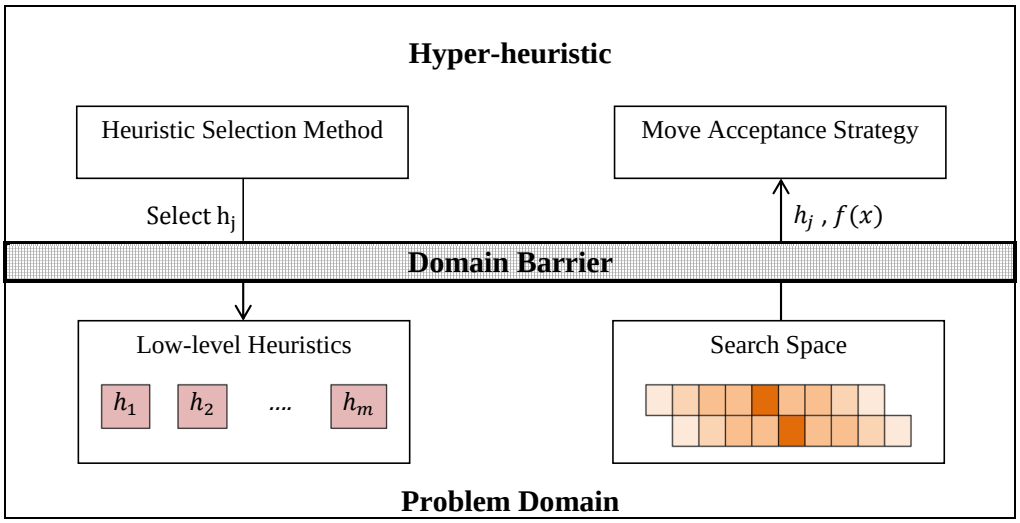

Figure 2.1: Generic selection hyper-heuristic framework.

Algorithm 2 Generic Selection Hyper-heuristic (GSHH)

input:
- a finite search space $\mathscr{X}$
- an objective function $f : \mathscr{X} \to \mathbb{R}$ to be minimised
- a set of m low-level heuristics $H := \{h_1, \ldots, h_m\}$

$\forall i \in [m]$, let $p_i^{(0)}$ be an initial selection probability of the low-level heuristic i.
Choose an initial solution $x_0 \in \mathscr{X}$.
for $t = 0, 1, \ldots$ until the termination condition is satisfied **do**
    Use a heuristic selection method to pick a low-level heuristic i with probability $p_i^{(t)}$
    Generate a new solution $x'_t := h_i(x_t)$ with the chosen heuristic
    if $f(x_t) \geq f(x'_t)$ **then**
        $x_t := x'_t$
    end if
end for

framework of Algorithm 2 consists of a finite search space $\mathscr{X}$, an objective function $f : \mathscr{X} \to \mathbb{R}$, and a set of m low-level heuristics $H := \{h_1, \ldots, h_m\}$. We define the low-level heuristics as follows.

Definition 2. *Given a finite search space $\mathscr{X}$ and an integer $k \geq 1$, a low-level heuristic h of* arity *k is a random mapping*

$$h : \mathscr{X}^k \to \mathscr{X}$$

In the above definition, the integer k represents the number of the input search points. For example, if the low-level heuristic is a mutation heuristic, then $k = 1$ where mutation heuristics process a single solution; whereas, $k \geq 2$ if the low-level heuristic is a crossover heuristic because crossover heuristics require at least two search points as input. Since we consider a single-point based search hyper-heuristic, $k = 1$ for all of the considered low-level heuristics in this thesis.

Initially, Algorithm 2 generates an initial complete solution $x_0 \in \mathscr{X}$. Usually, the initial solution is generated by sampling the search space of a given problem uniformly at random. In some cases, such as for highly constrained problems, it is hard to produce a feasible solution by sampling the search space uniformly at random. In these cases problem-specific constructive heuristics are often used to produce an initial feasible solution (Burke et al., 2007) (Bilgin et al., 2010).

The generic selection hyper-heuristic (GSHH) (Algorithm 2) evolves an initially generated solution as follows. A low-level heuristic h_i is chosen from a set H of predefined low-level heuristics by a *heuristic selection* strategy which defines the selection probabilities of low-level heuristics. The selected low-level heuristic is then applied to the current candidate solution to generate a new solution. The newly generated solution is subsequently evaluated, and a decision is taken as to whether it should replace the current candidate solution, or be discarded. If the stopping condition is not satisfied, then the GSHH generates a new solution following the above generation cycle. A termination condition could be, for example, a fixed time or obtaining a solution with a certain quality. In this thesis, the termination condition, unless stated otherwise, is the first point in time at which Algorithm 2 evaluates an optimal solution. Precisely, let T be the admissible number of iterations. We define T, for minimisation problems, as follows.

$$T := \min\{t | \forall y \in \mathscr{X}, f(x_t) \leq f(y)\} \quad \text{for minimisation} \tag{2.1}$$

Hence T is a random variable represents the number of objective function evaluations performed by the GSHH until an optimal solution is evaluated for the first time. In Chapter 4 of this thesis, we are interested in investigating the runtime for the GSHH with several heuristic selection strategies. The same move-acceptance strategy is therefore used which accepts the new solution if (and only if) it is not worse than the current candidate solution with respect to a given objective function (see lines 4 of Algorithm 2).

2.6 Theoretical Analysis Methods

Here, we present an overview of the necessary theoretical methods that are employed in our analyses.

Search heuristics, as any algorithm, can be evaluated based on both the correctness of the results and the computational resources, such as the time and memory, needed to execute them. As already discussed, the runtime of a search heuristic is defined as the number of objective function evaluations required before an optimal solution is obtained (Neumann and Witt, 2012). Usually, the runtime of search heuristics is expressed as a function of the characteristics of the input to the search heuristic such as the problem size. The success probability of a search heuristic refers to the probability that the search heuristic finds an optimal solution or a solution of certain quality within a given time limit. Runtime analysis is the task of determining how properties of the distribution of the runtime depend on the characteristics of the problem and parameter settings of the search heuristics (Auger and Doerr, 2011) (Neumann and Witt, 2012) (Jansen, 2013).

2.6.1 Notation

Before proceeding, we introduce some notation: for $n > 0$, define $[n] := \{1, 2, \ldots, n\}$. Most often in this thesis, n is the size of the problem (for example, the bit-string length). $\mathbb{E}[Z]$ denotes

the expectation of the random variable Z, and $\mathbb{P}[A]$ denotes the probability of the occurrence of the event A. We use $I \sim \text{Unif}(n)$ to signify that I is sampled uniformly at random from $[n]$. Standard asymptotic notation is used such as big-O, Ω, and Θ (see, for example, (Cormen et al., 2001)). Throughout this thesis, we denote by $\log(.)$ the natural logarithm. An event occurs with high probability (w.h.p.) with respect to a parameter n, if the probability of the event is bounded from below by $1 - O(1/n)$. Denote by $D_{KL}(Z_1||Z_2)$ the Kullback-Liebler divergence between the probability distributions Z_1 and Z_2.

Definition 3 (Kullback-Liebler divergence)**.** *Let Z_1 and Z_2 be two probability distributions defined on the finite set $\mathscr{X}$. Kullback-Liebler divergence from Z_1 to Z_2 is defined as follows:*

$$D_{KL}(Z_1||Z_2) = \sum_x Z_1(x) \log\left(\frac{Z_1(x)}{Z_2(x)}\right)$$

Denote by $T_{\text{GSHH-A, f}}$ the runtime of the GSHH (Algorithm 2) with heuristic selection strategy A on some function f. For $i,t \in \mathbb{N}$, let $p_i^{(t)}$ be the selection probability of low-level heuristic i in the t-th iteration of the GSHH. We refer to the probability of a low-level heuristic generating a strictly better solution with respect to a given search point and an objective function as the *success probability* of the low-level heuristic which is precisely defined as follows.

Definition 4 (Success Probability)**.** *Let $\mathscr{X}$ be a finite search space, and $f : \mathscr{X} \to \mathbb{R}$ be an objective function. Let $x'_t \in \mathscr{X}$ be the offspring of $x_t \in \mathscr{X}$ generated by the low-level heuristic i in the $t-$th iteration of Algorithm 2. The success probability of low-level heuristic i on the search point x_t is, in the case of minimisation problem, defined as follows:*

$$q_i(x_t) := \mathbb{P}\left(f(x_t) > f(x'_t)\right)$$

The success probabilities of low-level heuristics can be estimated as the number of times

the low-level heuristic generated strictly better solution, with respect to a given search point and objective function, proportional to the number of times it has been executed. Formally, we define the estimated success probabilities of low-level heuristics as follows.

Definition 5 (Estimated Success Probability)**.** *Let $s_i^{(t)}$ and $r_i^{(t)}$ be the number of successes and failures, respectively, observed from the i-th low-level heuristics, in terms of generating strictly better solutions with respect to a given objective function, in the t-th iteration of Algorithm 2. The success probability of the i-th low-level heuristic is estimated as follows:*

$$\bar{q}_i(x_t) := \frac{s_i^{(t)}}{s_i^{(t)} + r_i^{(t)}} \tag{2.2}$$

2.6.2 Artificial Fitness Levels

Several runtime analysis techniques have been proposed in the literature (Wegener, 2003) (Lehre and Witt, 2014) (Corus et al., 2014). One of the classical runtime analysis techniques is known as *Artificial fitness levels* (AFLs). AFLs technique is based on a very simple and natural idea: the search space can be artificially divided into subsets based on the objective (fitness) function. The characteristics of these subsets are given in the following definition (Wegener, 2001).

Definition 6 (Fitness Levels)**.** *Let $\mathscr{X}$ be a finite search space, and $f : \mathscr{X} \to \mathbb{R}$ be an objective function for a maximisation problem. A collection of disjoint sets $A_1, A_2, \ldots, A_n$ for $n \in \mathbb{N}$ is called* fitness levels *if the following conditions hold:*

- $\bigcup_{i=1}^{n} A_i = \mathscr{X}$ *and* $A_i \cap A_j = \emptyset$ *for any* $i, j \in [n]$ *and* $i \neq j$
- *for all* $x \in A_i$ *and* $y \in A_j$, $f(x) < f(y)$; $i < j$
- $A_n = \{x \in \mathscr{X} | \forall y \in \mathscr{X}, f(x) \geq f(y)\}$

The first condition implies that the search space is formed by the disjoint fitness levels. The second condition ensures that the objective function value increases as the index of the

fitness level increases. The third condition is satisfied if all the optimal solutions are assigned to the last fitness level. We refer to the search heuristic that never accept worsening solutions as *elitist* search heuristic. Partitioning the search space in such a way leads to a very simple argument to estimate the runtime of elitist search heuristics. To show that, let S be an elitist search heuristic. Let $\mathscr{X}$ be a finite search space partitioned artificially into n fitness levels $A_1, A_2, \ldots, A_n$ according to a given objective function $f : \mathscr{X} \to \mathbb{R}$. Let q_i be the minimum probability of the search heuristic S to improve and upgrade the solution from fitness level A_i towards a higher fitness level A_j for any $j \in \{i+1, i+2, \ldots, n\}$. Denote by $T_{i,f}$ the time required by S to jump from fitness level A_i to a higher fitness level. Thus, $T_{i,f}$ is a random variable and its expected value is:

$$\mathbb{E}[T_{i,f}] \leq \frac{1}{q_i} \tag{2.3}$$

The expected runtime of the search heuristic S on the objective function f is therefore at most the summation of the expected time to leave every fitness level A_i for all $i \in [n-1]$. The above analysis proves the following AFLs theorem due to (Wegener, 2001).

Theorem 7 (AFLs Theorem)**.** *Let $\mathscr{X}$ be a finite search space, and $f : \mathscr{X} \to \mathbb{R}$ be an objective function. For $n \in \mathbb{N}$, let $A_1, \ldots, A_n$ be fitness levels defined according to f. Denote by q_i the minimum probability to leave fitness level i toward a higher fitness level. The expected time to reach A_n is:*

$$\mathbb{E}[T_{n,f}] \leq \sum_{i=1}^{n-1} \frac{1}{q_i} \tag{2.4}$$

Defining appropriate partitions of the search space is a necessary condition to get a good bound on the expected runtime of search heuristics using the above technique. A drawback of Theorem 7 is that it does not take into consideration that the search heuristic might skip more than one fitness level in a single step as well as it may start from a fitness level higher than the first fitness level. Wegener (2003) presented a variant of Theorem 7 that takes into consideration

the initialisation step. Sudholt (2010) proposed an AFL theorem to calculate both an upper bound and a lower bound on the expected runtime of search heuristics. Corus et al. (2014) introduced a new AFL theorem for population-based search heuristics.

2.6.3 Gambler's Ruin Theory

The gambler's ruin problem demonstrates a general stochastic process $\{X_t\}_{t=0,1,\ldots}$. It was probably first formulated by Huygens in 1957 (Shoesmith, 1986). The problem is described, to put it simple, as follows (Feller, 1968). Suppose that there are two players, player A and player B, participating in a gambling game. Player A starts with $x \in \mathbb{N}$ money units and player B starts with $y \in \mathbb{N}$ money units. In each bet, player A wins a money unit from player B with probability p; otherwise, player B wins a money unit from player A with probability $q = 1 - p$. The game continues until player A or player B is ruined. Gambler's ruin theory is a classical runtime analysis tool that has been used in the runtime analysis of search heuristics (Jansen and Wegener, 2001) (Auger and Doerr, 2011). In (Jansen and Wegener, 2001), the runtime of $(1+1)$ EA on some search scenarios in which the algorithm has to perform some random walk to reach the optimal was derived using the gambler's ruin theory. A main gambler's ruin theorem that we will use in Chapter 4 is given in Theorem 8 (Feller, 1968).

Theorem 8. *[Gambler's Ruin] Let p, q, x, and y be as defined above. Let $z := x + y$, and $\delta := \frac{q}{p}$. The probability p_x that player A reaches $v > x$ before attaining zero is:*

$$p_x := \begin{cases} \frac{x}{v} & \text{if} \quad p = q = \frac{1}{2} \\ \frac{\delta^x - 1}{\delta^v - 1} & \text{otherwise} \end{cases} \tag{2.5}$$

Let T_x the number of bets until one player ruin, then

$$\mathbb{E}[T_x] = \begin{cases} x(z-x) & \text{if} \quad p = q = \frac{1}{2} \\ \frac{x}{1-2p} - \frac{z}{1-2p}\frac{1-\delta^x}{1-\delta^z} & \text{otherwise} \end{cases} \tag{2.6}$$

2.6.4 Drift Analysis

The term "drift" refers to the expected change in a single-step of a given stochastic process. For example, in the discussed gambler's ruin problem, the expected change in a single-step is always one. However, different search heuristics could have different drifts. Therefore, drift analysis was introduced to estimate the runtime of search heuristics according to their drift (Hajek, 1982) (He and Yao, 2001).

Drift analysis is a runtime analysis technique that depends on measuring the progress in a potential function a so-called *distance function*. The distance function assigns each state of the search heuristic to a non-negative value that represents its distance to the optimal state. Formally, let $g : \mathscr{X} \to \mathbb{R}$ be a function defined on the state space $\mathscr{X}$ that maps each non-optimal state $X_t \in \mathscr{X}$ to a positive value strictly larger than 0 which represents its distance to the optimal state X_*, and each optimal state $X_* \in \mathscr{X}$ to 0. The function g is called a distance function. The expected progress in g at time t is called the single-step drift and defined as follows (Auger and Doerr, 2011):

$$\Delta_t := \mathbb{E}[g(X_t) - g(X_{t+1}); g(X_t) > 0 | X_t] \tag{2.7}$$

The runtime of a search heuristic is then the time needed to reach a state with a g-value that equals to zero. The idea behind drift analysis is to use the relationship between the drift and the distance function to estimate the runtime of search heuristics. He and Yao (2001) proved the following drift theorem:

Theorem 9. [Additive drift] *Let $(X_t)_{t\in\mathbb{N}}$ be a Markov process over some state space $\mathscr{X}$. Let $g : \mathscr{X} \to \mathbb{R}^+$ be a distance function that assigns a distance to every state from the optimum. Let $T := min_{t=0\cdots\infty}\{t|g(X_t) = 0\}$. If there exists $\Delta_\mu > 0$ such that for any $t \geq 0$*

$$\mathbb{E}[g(X_t) - g(X_{t+1})|g(X_t) > 0] \geq \Delta_\mu \tag{2.8}$$

then

$$\mathbb{E}[T|X_0] \leq \frac{g(X_0)}{\Delta_\mu} \tag{2.9}$$

If there exists $\Delta_\ell > 0$ such that for any $t \geq 0$

$$\mathbb{E}[g(X_t) - g(X_{t+1})|g(X_t) > 0] \leq \Delta_\ell \tag{2.10}$$

then

$$\mathbb{E}[T|X_0] \geq \frac{g(X_0)}{\Delta_\ell} \tag{2.11}$$

The drift of search heuristics may vary during the search process. Hence, using a constant value for the drift over all search points may lead to imprecise bounds for the runtime of search heuristics. Various drift theorems have been therefore proposed in the literature to remedy this shortcoming in the above theorem (Auger and Doerr, 2011). Most of these theorems are based on the Hajek's drift theorem (Hajek, 1982). In chapter 4, we use the Hajek's theorem (Theorem 10) to analyse the learning capabilities and limitations of a reinforcement learning hyper-heuristic. Recently, Lehre and Witt (2014) presented a drift theorem that generalises most of the existing drift theorems (see Theorem 11).

Theorem 10 (Hajek's Theorem (Hajek, 1982))**.** *Given a sequence $(Y^{(t)}, \mathscr{F}^{(t)})$ over an interval $[a,b] \subset \mathbb{R}$, $a > 0$. If there exist $0 < \rho < 1$, η, and $D \geq 1$ such that the following conditions satisfy:*

$D_1 : \mathbb{E}[e^{\eta \cdot Y^{(t+1)}} | Y^{(t)} > a \wedge \mathscr{F}] \leq \rho \cdot e^{\eta \cdot Y^{(t)}}$

$D_2 : \mathbb{E}[e^{\eta \cdot Y^{(t+1)}} | Y^{(t)} \leq a \wedge \mathscr{F}] \leq D \cdot e^{\eta \cdot a}$

Then

$$\Pr(Y^{(t)} > b \,|\mathscr{F}) \leq \rho^t \cdot e^{\eta(Y^{(0)}-b)} + D \cdot e^{\eta(a-b)} \cdot (1-\rho^t)/(1-\rho).$$

Theorem 11. *[General Drift Theorem (Lehre and Witt, 2014)]*

Let $(X_t)_{t\geq 0}$ be a stochastic process over some state space $S \subseteq \{0\} \cup [x_{min}, x_{max}]$, where $x_{min} > 0$. Let $h : [x_{min}, x_{max}] \to \mathbb{R}^+$ be an integrable function and define $g : 0 \cup [x_{min}, x_{max}] \to \mathbb{R}^{\geq 0}$ by $g(x) = \frac{x_{min}}{h(x_{min})} + \int_{x_{min}}^{x} \frac{1}{h(y)} dy$ for $x \geq x_{min}$ and $g(0) = 0$. Let $T_a = min\{t | X_t \leq a\}$ for $a \in \{0\} \cup [x_{min}, x_{max}]$, Then:

- *If $\mathbb{E}[X_t - X_{t+1} | \mathscr{F}_t; X_t \geq x_{min}] \geq h(X_t)$ and $\mathbb{E}[g(X_t) - g(X_{t+1}) | \mathscr{F}_t; X_t \geq x_{min}] \geq \alpha_u$, then:*

$$\mathbb{E}[T | X_0] \leq \frac{g(X_0)}{\alpha_u}.$$

- *If $\mathbb{E}[X_t - X_{t+1} | \mathscr{F}_t; X_t \geq x_{min}] \leq h(X_t)$ and $\mathbb{E}[g(X_t) - g(X_{t+1}) | \mathscr{F}_t; X_t \geq x_{min}] \leq \alpha_\ell$, then:*

$$\mathbb{E}[T | X_0] \geq \frac{g(X_0)}{\alpha_\ell}.$$

- *If $\mathbb{E}[X_t - X_{t+1} | \mathscr{F}_t; X_t \geq x_{min}] \geq h(X_t)$ and there exists $\lambda > 0$ and*

a function $\beta_u : (a, x_{max}] \rightarrow \mathbb{R}^+$ *such that* $\mathbb{E}[e^{-\lambda(g(X_t)-g(X_{t+1}))}|\mathscr{F}_t; X_t > a] \leq \beta_u(X_t)$*, then*

$$\Pr(T_a \geq t*|X_0) < \left(\prod_{r=0}^{t^*-1} \beta_u(X_r)\right) \cdot e^{\lambda(g(X_0)-g(a))} \text{ for } t^* > 0.$$

- *If* $\mathbb{E}[X_t - X_{t+1}|\mathscr{F}_t; X_t \geq x_{min}] \leq h(X_t)$ *and there exists* $\lambda > 0$ *and a function* $\beta_e ll : (a, x_{max}] \rightarrow \mathbb{R}^+$ *such that* $\mathbb{E}[e^{\lambda(g(X_t)-g(X_{t+1}))}|\mathscr{F}_t; X_t > a] \leq \beta_e ll(X_t)$*, then*

$$\Pr(T_a \leq t*|X_0 > a) \leq \left(\sum_{s=1}^{t^*-1}\prod_{r=0}^{s-1} \beta_e ll(X_r)\right) \cdot e^{-\lambda(g(X_0)-g(a))} \text{ for } t^* > 0.$$

2.7 Empirical Benchmark Problems

Although the primary focus of this thesis is on runtime analysis of selection hyper-heuristics, our theoretical studies are complemented with empirical results. We conducted most of the experiments in this thesis using the *Hyper-heuristics Flexible* framework (HyFlex). HyFlex[1] is a multi-domain framework that provides implementations of six well-known combinatorial optimisation problems each associated with a set of problem instances, including: Boolean satisfiability, bin packing, personnel scheduling, permutation flow-shop, travelling salesman problem, and the vehicle routeing problem. HyFlex also contains a set of predefined low-level heuristics for each of the problem domain mentioned above. All the low-level heuristics in HyFlex are variation operators that return a complete solution given one or more complete solutions, which are classified into four categories: mutational (MU), ruin-recreate (RR), local-search (LS), and crossover (CR) low-level heuristics. Table 2.1 summarises the number of the available low-level heuristics for each problem domain in HyFlex (Burke et al., 2011a).

Mutational low-level heuristics stochastically perturb the input solution; thus the consequences of their applications may lead to a worse solution. Local search low-level heuristics

[1] http://www.hyflex.org

Table 2.1: The number of the available low-level heuristics for each problem domain.

Problem Domain	Total No.	MU	RR	LS	CR
SAT	11	5	1	2	2
BP	8	3	2	2	1
PS	12	1	3	5	3
PF	15	5	2	4	4
TSP	13	5	1	3	4
VRP	10	3	2	3	2

search for a locally optimal solution and never return a deteriorating solution with respect to the objective function. Ruin and re-create low-level heuristics deconstruct part of the given solution and then reconstruct a complete solution. Crossover low-level heuristics return a new solution by combining the two input solutions. In HyFlex, each low-level heuristic, except crossover low-level heuristics, consist of a problem-dependent parameter that can be somewhat adjusted by the user. For example, the *depth of search* of local-search low-level heuristics is a user-controlled parameter which determines the number of hill-climbing steps. The parameter takes a value between $[0, 1]$: the smaller the value that the parameter takes, the smaller the hill-climbing steps that the low-level heuristic performs (Burke et al., 2011a).

Chapter 3

Literature Review

This chapter presents a survey of online-learning hyper-heuristics, together with an overview of the theoretical foundation of hyper-heuristics. The goal of this chapter is not only to cover the research materials but also to put the contributions of this thesis into a wider context.

3.1 Introduction

Selection hyper-heuristics are search techniques capable of selecting and applying heuristics from a set of predefined low-level heuristics with the aim of solving a given problem. Historically speaking, the idea of combining multiple low-level heuristics for solving a given problem originated in the 1960*s* (Fisher and Thompson, 1963). Fisher and Thompson (1963) combined different heuristics to solve a job shop scheduling problem. Their experimental examinations showed that the qualities of results obtained by randomly choosing a sequence of heuristics were better than applying any one of the heuristics alone. This provides empirical evidence that combining multiple heuristics can be beneficial. Later on, it has been observed that typical low-level heuristics perform differently between different regions of the search space of a given problem (Remde et al., 2007) (Özcan et al., 2008) (Lehre and Özcan, 2013). Hence, combining different

low-level heuristics could lead to better results than applying each low-level heuristic alone.

As discussed in Section 2.5, a generic selection hyper-heuristic combines two main strategies, heuristic selection and move-acceptance, as its main components within a single-point search based framework (Özcan et al., 2008). Heuristic selection strategies could either be no-learning or learning strategies. Popular examples of no-learning heuristic selection strategies are represented by the *simple random* and *random permutation* strategies (Cowling et al., 2001). The simple random strategy chooses low-level heuristics uniformly at random. The pseudo-code of the considered selection hyper-heuristic with the simple random strategy is given in Algorithm 3. The random permutation strategy generates a uniform at random order of all low-level heuristics, and the next heuristic in the prepared order is applied (see Algorithm 4). Both the simple random and random permutation are commonly used heuristic selection strategies in hyper-heuristics (Cowling et al., 2002; Özcan et al., 2008; Bilgin et al., 2006; Cobos et al., 2011; Bilgin et al., 2012; Mısır et al., 2012a). An overview of online-learning strategies is given in Section 3.3.

Algorithm 3 Simple Random Strategy

1: **Input:**
   - a finite search space $\mathscr{X}$
   - an objective function $f : \mathscr{X} \to \mathbb{R}$
   - a set of m low-level heuristics $H := \{h_1, \ldots, h_m\}$, where for all $i \in [m]$, $h_i : \mathscr{X} \to \mathscr{X}$.
2: Choose an initial solution $x \in \mathscr{X}$.
3: **while** the stopping condition is not satisfied **do**
4: Sample $I \sim \text{Unif}(m)$ {pick a heuristic uniformly at random}
5: $x' := h_I(x)$ {apply the chosen heuristic h_I}
6: **if** $f(x) \geq f(x')$ **then**
7: $x := x'$
8: **end if**
9: **end while**

Move-acceptance strategies could be divided into two broad categories: deterministic and non-deterministic move-acceptance strategies. The former category consists of move-acceptance strategies that return the same decision whenever they are invoked with certain input values. The

Algorithm 4 Random Permutation Strategy

1: Let $\sigma(m)$ be a random permutation of all the low-level heuristics.
2: **while** the stopping condition is not satisfied **do**
3: Let $i := \sigma(m)$ {permute to the next heuristic}
4: $x' := h_i(x)$
5: **if** $f(x) \geq f(x')$ **then**
6: $x := x'$
7: **end if**
8: **end while**

latter category contains move-acceptance strategies that change their decisions based on some criteria such as the time and the quality of the solutions. Popular examples of deterministic move-acceptance strategies are *all moves* (AM), *only improvement* (OI), and *improvement or equal* (IE) (Cowling et al., 2001). The AM acceptance criterion accepts all solutions; whereas, the OI strategy accepts only those solutions that are strictly better than the current candidate solution with respect to a given objective function. The IE strategy accepts the new solution if (and only if) it is not worse than the candidate solution with respect to a given objective function.

Non-deterministic move-acceptance strategies could be further graded into two types: probabilistic and threshold-based move-acceptance strategies. Generally speaking, probabilistic move-acceptance strategies accept all improving solutions, and worsening solutions are accepted with some probability. Threshold-based move-acceptance strategies make their decisions based on satisfaction of some conditions.

Simulated annealing strategy is an example of the probabilistic move-acceptance strategies which accepts improving solutions by default, and worsening solutions are still accepted with probability:

$$Exp(-\frac{\Delta}{T}) \tag{3.1}$$

where Δ is the difference between the objective function value of the current candidate solution and the newly generated solution, and T is a parameter of the strategy a so-called *temperature*

(Kirkpatrick et al., 1983) (Bai and Kendall, 2005). Simulated annealing strategy gradually decreases the temperature T according to a cooling schedule that, in turn, decreases the probability of accepting worsening solutions. Naïve strategy is another popular example of the probabilistic move-acceptance strategies (Burke et al., 2010a). It accepts the improving solution, and worsening solutions are also accepted with a predefined probability.

Late Acceptance (LA) is a threshold-based move-acceptance strategy (Burke and Bykov, 2008). LA maintains a list of L previously obtained objective function values, where L is the algorithm parameter. LA accepts the improving solution, while the worsening solution can also be accepted according to the comparison with the L-th solution in the list with respect to a given objective function. Great Deluge (GD), first proposed by (Dueck, 1993), accepts the new solution as long as its objective function value is above a certain threshold level. The threshold level is gradually changed over the time towards the objective function value of the target solution. For example, Kendall and Mohamad (2004) proposed a GD based hyper-heuristic that accepts the worsening solution if its objective function value is above the following threshold level:

$$f_{opt} + \Delta_f \times (1 - \frac{t}{T}) \tag{3.2}$$

where f_{opt} the objective function value of the target solution, Δ_f is the maximum expected change in the objective function value, and T is the admissible number of iterations.

A large number of selection hyper-heuristics have been proposed in the literature and successfully applied to several combinatorial optimisation problems such as *personnel scheduling* (Cowling et al., 2002), *timetabling* (Ahmadi et al., 2003), *vehicle routing problem* (Garrido and Castro, 2009). More applications are discussed in (Burke et al., 2013), (Chakhlevitch and Cowling, 2008), and (Ross, 2014). Selection hyper-heuristics have also been favourably applied to continuous multi-objective optimisation problems, where the goal is to find a set of solutions,

a so-called *Pareto front*, that trade-off between different objective functions at the same time. Interesting examples of selection hyper-heuristic approaches in this context include the tabu-search hyper-heuristic for *space allocation* and *timetabling* problems proposed by Burke et al. (2005b), and the Markov chain based hyper-heuristic proposed by McClymont et al. (2013) for *water distribution network* optimisation.

3.2 Modules of Selection Hyper-heuristic

In (Özcan et al., 2006), three hyper-heuristic frameworks based on an iterated local search approach were proposed and compared to the discussed generic hyper-heuristic framework in Section 2.5. In the first proposed framework, the hyper-heuristic picks a low-level heuristic from a set of both mutation and local search low-level heuristics. If the chosen heuristic was mutation heuristic, then a predefined local search heuristic is applied after the application of the mutation heuristic. In the second framework, the low-level heuristics are grouped into mutation and local search low-level heuristics. In every time step, a mutation heuristic is chosen by a heuristic selection strategy, then after applying the selected heuristic, a pre-chosen local search heuristic is applied to the newly generated solution. In the third framework, the hyper-heuristic applies a heuristic chosen from a set of mutation low-level heuristics and thus producing a new solution. A move-acceptance strategy is used to decide whether to accept or reject the new solution. Then a local search heuristic is chosen from a set of predefined local search low-level heuristics to be applied to the current candidate solution. More on selection hyper-heuristic frameworks can be found in (Burke et al., 2003a) (Özcan et al., 2008) (Ross, 2014) (Burke et al., 2013).

3.3 An Overview of Online-learning Hyper-heuristics

An online learning selection hyper-heuristic operates on a set of low-level heuristics, applying a machine learning technique to predict which low-level heuristic is the most effective to apply at a given point in time. Reinforcement learning is a general machine-learning technique that captures many of the previously proposed online-learning mechanisms in hyper-heuristics (Kaelbling et al., 1996) (Sutton and Barto, 1998). Generally speaking, reinforcement learning mechanisms interact with the search environment to learn and adapt their behaviour according to the feedback during the search process. This section provides a brief overview of frequently used reinforcement learning mechanisms in hyper-heuristics.

Algorithm 5 Greedy Mechanism

1: **while** stopping condition is not satisfied **do**
2: **for** $i = 1, \ldots, m$ **do**
3: $x_i := h_i(x)$ {apply heuristic h_i}
4: **end for**
5: Let $I := \arg\min_{i \in [m]}(f(x_i))$ {for minimisation}
6: **if** $f(x) \geq f(x_I)$ **then**
7: $x := x_I$
8: **end if**
9: **end while**

Cowling et al. (2001) proposed simple and easy-to-implement reinforcement learning mechanisms for hyper-heuristics, including: *greedy*, *random gradient*, and *random permutation decent*. The greedy heuristic selection mechanism applies all low-level heuristics to the same candidate solution at each decision point and chooses the heuristic that yields the best change in the objective function value, as long as its application improves in the candidate solution with respect to a given objective function. Algorithm 5 provides the pseudo-code of the greedy mechanism. The greedy mechanism is an online-learning mechanism with one-step memory length. That is all of the available feedback is discarded after each decision point. In general, a main drawback of the greedy mechanism is that its implementation is computationally expen-

sive regarding the CPU time and memory, and this can be clearly observed when the number of low-level heuristics is large (Remde et al., 2007) (Remde et al., 2012).

Algorithm 6 Random Gradient Mechanism

1: **for** $t = 0, 1, \ldots$ until the stopping condition is satisfied **do**
2: **if** $f(x_{t-1}) > f(x_t)$ **then**
3: $h_t := h_{t-1}$
4: **else**
5: Sample $I \sim \text{Unif}(m)$
6: $h_t := h_I$
7: **end if**
8: $x'_t := h_t(x_t)$
9: **if** $f(x_t) \geq f(x'_t)$ **then**
10: $x_{t+1} := x'_t$
11: **end if**
12: **end for**

The random gradient mechanism chooses a low-level heuristic uniformly at random, then the chosen low-level heuristic is applied continuously as long as it produces strictly better solutions with respect to a given objective function. The precise description of the random gradient mechanism is given in Algorithm 6. The random permutation decent mechanism is similar to random permutation, except that it applies the chosen low-level heuristic repeatedly while an improvement in the candidate solution with respect to a given objective function is found. The pseudo-code of the random permutation decent mechanism is depicted in Algorithm 7. The random gradient and random permutation decent mechanisms have the advantage of low computational cost where their implementations require only maintaining whether the selected heuristic in the previous iteration generated a strictly better solution with respect to a given objective function. Both mechanisms are commonly used learning mechanisms in hyper-heuristics and have been applied to different optimisation problems (see, for example, (Cowling et al., 2001; Özcan et al., 2008; Kiraz and Topcuoglu, 2010; Kiraz et al., 2013). Berberoglu and Uyar (2011) experimentally showed that a hyper-heuristic with the random permutation decent mechanism produced the best results compared to a set of learning hyper-heuristics.

Algorithm 7 Random Permutation Decent Mechanism

Let $\sigma(m)$ be a random permutation of all the low-level heuristics.
for $t = 0, 1, \ldots$ until the stopping condition is satisfied **do**
  if $f(x_{t-1}) > f(x_t)$ **then**
    $h_t := h_{t-1}$
  else
    $h_t := \sigma(m)$ { permute to the next heuristic}
  end if
  $x'_t := h_t(x_t)$
  if $f(x_t) \geq f(x'_t)$ **then**
    $x_{t+1} := x'_t$
  end if
end for

Nareyek (2004) proposed a reinforcement learning mechanism that associates each low-level heuristic with a positive weight. The weights of the low-level heuristics are initialised to the same value and updated during the search process by a pre-chosen weight update scheme. The weight update scheme is an updating rule that assigns a positive or a negative reinforcement value to each low-level heuristic. The weight of each low-level heuristic is restricted within a user-defined interval. At each time step, the proposed learning mechanism uses a selection method to decide which low-level heuristic should be chosen based on their weights. Nareyek experimented with two selection methods: *roulette wheel* and *max*. The roulette wheel selection method chooses a heuristic from a set of low-level heuristics randomly with probabilities proportional to their weights. The max selection method picks a low-level heuristic with maximum weight. The proposed reinforcement learning mechanism involves observing feedback from the environment after applying the chosen low-level heuristic. If an improvement in the candidate solution is found, with respect to a given objective function, then the weight of the selected low-level heuristic is positively reinforced; otherwise, it is negatively reinforced. Various weight update schemes were proposed and tested on the *orc qust* problem and *logistics domains* (Nareyek, 2004). Generally, the best results were obtained by combining a low rate of positive reinforcement value with a strong rate of negative reinforcement value (see Algorithm 8). In

the logistics domains, the experimental results showed that the algorithm performed best with a weight update scheme that increases the weight of the selected heuristic by one if its application led to an improvement in the candidate solution with respect to a given objective function, and decreases the weight by one otherwise; we refer to this weight update scheme as the *additive* update scheme. The discussed reinforcement learning mechanism with the additive weight update scheme is a frequently used learning mechanism in hyper-heuristics (see, for example, (Özcan et al., 2010); (Elhag and Özcan, 2015); and (Remde et al., 2012)). We now discuss some of these studies and the variants that have been proposed in the literature.

Algorithm 8 Reinforcement Learning Mechanism

Input: parameter $w_{\max} \in \mathbb{R}^+$ and $w_{\min} < w_{\max}$.
$\forall i \in [m]$, let $w_i^{(0)} := 1$ be an initial weight of low-level heuristic i.
for $t = 0, 1, \ldots$ until the stopping condition is satisfied **do**
    Let $I := \arg\max_{i=1,\ldots,m}\{w_i^{(t)}\}$
    $x' := h_I(x)$ {apply the chosen heuristic h_I}
    $w_i^{(t+1)} := \begin{cases} \min\{w_i^{(t)} + 1, w_{\max}\} & \text{if} \quad f(x) > f(x'), \text{and} \\ \max\{\sqrt{w_i^{(t)}}, w_{\min}\} & \text{otherwise.} \end{cases}$
    if $f(x) \geq f(x')$ **then**
        $x := x'$
    end if
end for

Özcan et al. (2010) empirically investigated the discussed reinforcement learning mechanism in a *great deluge* based hyper-heuristic on an examination timetabling problem. A new selection method was proposed that chooses a low-level heuristic uniformly at random among those that have weights above the average weight of the low-level heuristics. The proposed reinforcement learning hyper-heuristic was tested with several weight update schemes. The results indicated that the algorithm performed best with the additive weight update scheme and the *max* selection method. The proposed hyper-heuristic was compared to the simple random great deluge hyper-heuristic. Overall their experimental results showed that the reinforcement learning great deluge hyper-heuristic performed better than simple random great deluge hyper-heuristic.

Ozcan et al. (2009) showed that the simple random hyper-heuristic with the *late acceptance* strategy had the best performance when compared to a set of heuristic selection methods including the discussed reinforcement learning mechanism on an examination timetabling problem.

Remde et al. (2012) proposed a variant of Algorithm 8 with a large number of low-level heuristics (243 heuristics). Instead of choosing a single low-level heuristic, the proposed reinforcement learning mechanism selects a subset of the low-level heuristics with the maximum weights. The selected low-level heuristics are then applied to the candidate solution. The weight of the best performing low-level heuristic is rewarded if an improvement in the candidate solution is found; whereas, the reset of the chosen low-level heuristics are negatively reinforced. The proposed algorithm uses the weight update scheme described in Algorithm 8. We observe that if the weights of low-level heuristics are not in the integer domain, then the described weight update scheme in Algorithm 8 leads the reinforcement learning mechanism with the max selection method to get stuck in the first rewarded low-level heuristic due to the fact that for any $x > 1$ we have $\sqrt{x} > 1$. Remde et al. applied the proposed reinforcement learning hyper-heuristic to a personnel scheduling problem. The results showed that the proposed algorithm produced competitive often better results comparing with a set of hyper-heuristics when the admissible runtime is low (Remde et al., 2012).

Kalender et al. (2013) proposed a *greedy-gradient* hyper-heuristic which associates each low-level heuristic with a weight. Instead of predefined reinforcement values, the change in the objective function value is used as a reinforcement value. However, the weight of the chosen heuristic is reset to zero whenever its application led to deteriorating solution with respect to a given objective function. The *max* selection method is used to select a heuristic with maximum weight. If the weights of all low-level heuristics are equal to zero, a greedy mechanism is used as a tie-breaking strategy. The chosen low-level heuristic is applied continuously as along as an improvement in the candidate solution, with respect to a given objective function, is achieved. In

the case of the weights of the low-level heuristics are zero after the greedy step, another problem-dependent weight vector is used, which is computed based on violations of the constraints of the problem at hand.

Bai et al. (2007b) proposed a discounted reinforcement learning mechanism that combines the discussed reinforcement learning mechanism with a discounting factor and a slightly different weight update scheme. The discounting factor is a parameter of the algorithm. If the discounting factor is too large, then the algorithm may evaluate the low-level heuristics based on irrelevant results about their current performance. If the discounting factor is too small, the algorithm may quickly neglect the results of the low-level heuristics and almost choose low-level heuristics uniformly at random.

Generally speaking, Tabu-search heuristics prevent the search from revisiting some search points by maintaining these search points in a list so-called *tabu-list*. Burke et al. (2003b) introduced the tabu-search hyper-heuristic that employs a tabu-list with the aim of preventing the poor-performing low-level heuristics from being chosen too soon. The proposed algorithm works similarly to the described reinforcement learning mechanism above with the additive weight update scheme except that whenever a low-level heuristic produces a worsening solution, it is added to the tabu-list on the basis "first-in/first-out". The tabu-list is also empty whenever the candidate solution is changed.

Later on, Cowling and Chakhlevitch (2003) introduced a variant of the tabu-search hyper-heuristic that adjusts the size of the tabu-list adaptively according to the search progress, so that the size of the tabu-list is decreased if the candidate solution is improved, whereas the size is increased if no improvement in the candidate solution is found. Dowsland et al. (2007) proposed a variant of the tabu-search hyper-heuristic that updates the weight of each low-level heuristic after a predefined number of executions. (Kendall and Hussin, 2005) empirically investigated the effect of making a low-level heuristic tabu for a fixed time or a randomly chosen time.

Kendall and Hussin also proposed a tabu-search hyper-heuristic that applies the chosen low-level heuristic continuously as long as an improvement in the candidate solution is found. Burke et al. (2005b) proposed a tabu-search hyper-heuristic for multi-objective optimisation problems. The proposed hyper-heuristic generates a weight matrix of size $n \times m$, where n is the number of objectives and m is the number of low-level heuristics. In each iteration of the algorithm, a low-level heuristic is chosen based on its weight with respect to the current individual objective function. The proposed hyper-heuristic evaluates the newly generated solution with respect to the objective function. The weights of the chosen low-level heuristic are updated accordingly. The proposed algorithm was assessed with a single and multiple tabu-lists. When a single tabu-list is considered, the chosen heuristic is made tabu if (and only if) it generated a worsening solution with respect to the chosen individual objective function. In the other case, n tabu-lists are used, one for each objective function. The chosen low-level heuristic may be tabu in more than one tabu-list based on its application with respect to each objective function under consideration.

Bai et al. (2007a) proposed a reinforcement learning mechanism that updates the weights of the low-level heuristics periodically. In particular, the run period is divided into short learning periods of identical lengths. In each learning period, the proposed algorithm record for each low-level heuristic: (1) the number of executions; (2) the number of the generated solutions that were accepted by the move acceptance strategy; and (3) the number of new solutions generated by the low-level heuristic. Then, before the next learning period begins, the recorded information in the previous learning period is used to update the weights of the heuristics that will remain fixed during the next learning period. The roulette wheel selection method was used to choose a low-level heuristic with a probability proportionate to their weights.

McClymont and Keedwell (2011) presented a reinforcement learning hyper-heuristic for multi-objective optimisation. The proposed algorithm utilises a Markov model in which the

low-level heuristics forms the state space. The model is fully connected. The weight of each edge represents the transition probability of moving from the current state (current heuristic) to the destination state (any other heuristic, including the heuristic itself). All of the edges' weights are initialised to the same value and updated during the search process. The *roulette wheel* selection strategy is used to choose the next heuristic based on the weights of the current state's edges. The selected low-level heuristic is applied ε times before moving to another heuristic. If the chosen heuristic is considered successful based on a threshold strategy, then the weight of the edge that led to choosing this heuristic is increased by α; otherwise, it is decreased by β where α and β are predefined constants. Several variants of the Markov chain hyper-heuristic were applied to both single-objective and multi-objective optimisation problems (see, for example, (McClymont et al., 2013); (Kheiri et al., 2015)).

3.4 Theoretical Background of Hyper-heuristics

Despite the fact that selection hyper-heuristics have attracted many empirical research studies (see the numerous references cited in this thesis), there are few theoretical investigations of hyper-heuristics. Quoting Burke et al. (2013):

> *" It is crucial to have theoretical support motivating the development of selection hyper-heuristics."* (Burke et al., 2013)

The importance of the theoretical investigations of search heuristics is manifold. For example,

- Important results such as lower bounds on the runtime of search heuristics and those results that hold for a class of algorithms cannot be shown empirically.
- Theoretical investigation, such as runtime analysis, often shows how the performance of a search heuristic depends on the design of the search heuristic. This may provide

guidelines for practitioners to find the most appropriate settings for their algorithms such as identifying the most appropriate parameters values (Auger and Doerr, 2011) (Neumann and Witt, 2012) (Jansen, 2013).

- In the field of Evolutionary Algorithms, there is good evidence that runtime analysis, show rigorously when and why Evolutionary Algorithms perform well. This, in turn, sheds light on their capabilities and limitations which aid the design of better algorithms (Sudholt, 2009) (Jansen and Wegener, 2001).

In the literature, some theoretical studies are comparing between *pure* and *mixed* strategy search heuristics. A pure-strategy search heuristic uses a single search method (e.g. variation operator); whereas, a mixed-strategy search heuristic chooses between different search methods, like hyper-heuristics. He et al. (2012) presented a comparative study between a pure-strategy and mixed-strategy $(1+1)$ EA. The pure-strategy $(1+1)$ EA applies a single mutation operator in each time step; whereas, the mixed-strategy $(1+1)$ EA chooses a mutation operator with fixed probability from a set of predefined mutation operators. The results showed that the asymptotic hitting time and convergence rate of the mixed-strategy $(1+1)$ EA is never worse than the pure-strategy $(1+1)$ EA which uses the worst mutation operator. In this context, He et al. (2013) showed that if the heuristics are *complementary* to each other (such that for each heuristic there exist some search states in which its drift is strictly better than the other heuristics), then there exists a search heuristic that mixes those heuristics such that the expected runtime of the search heuristic is better than the expected runtime of any pure-strategy search heuristic that uses any one of the given heuristics alone.

He et al. (2014) presented a theoretical assessment of solution qualities using pure and mixed strategies $(N+1)$ EA for solving a knapsack problem. In this study, mixed-strategy $(N+1)$ EA refers to the $(N+1)$ EA that combines two repairing methods, so that when a repairing method

is needed, the algorithm chooses one of these methods uniformly at random. A repairing method is an algorithm to reform the infeasible solutions into feasible ones. The results show that the generated solution by both pure and mixed strategy $(N+1)$ EA were not promising (He et al., 2014).

Lehre and Özcan (2013) presented the first runtime analysis of a selection hyper-heuristic and investigated whether mixing low-level heuristics and move-acceptance strategies can improve the search process on given theoretical benchmark problems. The results demonstrated that combining mutation operators could lead to exponentially faster search comparing with using anyone of the mutation operators alone. The results also suggested that combining different move-acceptance strategies is necessary in some cases. Additionally, the runtime of a hyper-heuristic with the random permutation and random permutation decent mechanisms were derived on a benchmark problem.

Furthermore, there exist some theoretical results in related fields that could be interpreted in the context of hyper-heuristics. For instance, Böttcher et al. (2010) showed that adapting the mutation rate according to the objective function value can speed up the runtime of an EA on a benchmark problem. Dang and Lehre (2016) presented one of the first studies on the runtime analysis of self-adaptive EAs. In this study, a self-adaptive EA controls the mutation rates during the search process. The results showed that self-adaptive non-elitist EAs can be exponentially faster than those that use a fixed mutation rate or choose the mutation rate uniformly at random. Additionally, Doerr et al. (2016) showed that the runtime of a self-adaptive randomised local search heuristic is asymptotically optimal on a benchmark problem.

3.5 Non-stationary Multi-armed Bandit Learning Mechanisms

The so-called multi-armed bandit problems provide a general model for learning problems in different areas such as machine learning, statistics, and engineering (see, for example, (Thompson, 1933); (Berry and Fristedt, 1985); (Sutton and Barto, 1998)). Basically, the multi-armed bandits problem can be described as follows: a gambler is allowed to sequentially choose between and play a set of available slot machines (arms) to maximise his/her total rewards within a fixed time. On the one hand, the gambler might explore the arms to discover the arm with the highest expected reward, but he/she might fail to exploit the best arm enough time to maximise the total rewards. On the other hand, the gambler might want to exploit the best arm that has been observed so far, but he/she might ignore the other arms that have higher expected rewards. Therefore, the gambler must balance between exploring the arms and exploiting the most profitable arms. The multi-armed bandit problem turned out to be a general paradigmatic model of the trade-off between exploration and exploitation in sequential decision making with uncertainty. It has attracted significant interest since it was first motivated by applications in clinical experiments by Thompson (1933). Many learning mechanisms for the multi-armed bandit problems have been proposed in the literature (Bubeck et al., 2012). However, most of these learning mechanisms are designed based on strong assumptions about the statistics of the arms that may not be applicable to real problems (Sutton and Barto, 1998).

3.5.1 Thompson Sampling

In the early days of the multi-armed bandit problem, Thompson introduced a reinforcement learning mechanism for the multi-armed bandit problem referred to as Thompson Sampling (Thompson, 1933). Thompson was interested in identifying the most appropriate action to use among two available actions. This was in the context of a clinical experiment in which the goal

is to decide between two treatments based on a small number of observations before more feedback can be collected. Conceptionally, Thompson Sampling is based on the idea of probability matching: each action is randomly chosen with a probability consistent with its probability of being optimal. This means that Thompson Sampling chooses an arm j in time t with probability $p_j^{(t)} = \mathbb{P}(j = i^*)$ given that i^* is the optimal arm (i.e. arm i^* has the maximum expected reward). A formal presentation of Thompson Sampling for the Bernoulli multi-armed bandit problem (i.e. the reward is either 0 or 1) is given in Algorithm 9 (Thompson, 1933).

Algorithm 9 Thompson Sampling for Bernoulli Multi-armed Bandit

Input: a set of k arms.
$\forall i \in [k]$, let $\alpha_i := 1$ and $\beta_i := 1$ be the success and failure counters.
for $t = 0, 1 \ldots$ until the stopping condition is satisfied **do**
    Sample $U_i^{(t)} \sim Beta(\alpha_i, \beta_i)\ \forall i \in [k]$
    Let $I := \arg\max_{i \in [k]} \{U_i^{(t)}\}$
    Pull arm I and observe the reward r_i
    if $r_i = 1$ **then**
        $\alpha_i := \alpha_i + 1$
    else
        $\beta_i := \beta_i + 1$
    end if
end for

Algorithm 9 observes after each trial as feedback from the environment whether the trial resulted in a success or a failure. Thompson Sampling maintains prior distributions on the mean of the reward of each arm. For the Bernoulli rewards, it has been observed that Beta distribution is an appropriate distribution of the priors (Thompson, 1933) (Agrawal and Goyal, 2013). In this case, if the prior distribution of arm i is $Beta(\alpha_i, \beta_i)$, then after a single trial the posterior distribution of arm i is either $Beta(\alpha_i + 1, \beta_i)$ or $Beta(\alpha_i, \beta_i + 1)$ based on whether the reward is 1 or 0.

Despite the fact that it was absent from the artificial intelligence literature, there has recently been renewed interest in Thompson Sampling and theoretical results have emerged that justify its use (Agrawal and Goyal, 2013) (Chapelle and Li, 2011) (Granmo, 2010). Kaufmann et al.

(2012) proved that Thompson Sampling is asymptotically optimal on the so-called Bernoulli multi-armed bandit problem. Its effectiveness on more complex reward models was shown theoretically in (Granmo, 2010). The idea of Thompson Sampling has also been successfully applied to several real-world problems (see, for example, (Chapelle and Li, 2011) (Graepel et al., 2010) (Tang et al., 2013)).

3.5.2 Upper Confidence Bound Algorithms

Lai and Robbins (1985) introduced a class of reinforcement learning algorithms that associate each arm with a quantity called *upper confidence bound*. Although the proposed algorithms are asymptotically optimal, in terms of balancing between the exploration and exploitation on particular multi-armed bandit problems, they are computational inefficient in terms of the computational resources needed to execute them (Auer et al., 2002). Later on, Auer et al. (2002) proposed a variant of the upper confidence bound algorithms so-called *upper confidence bound 1* (UCB for hand-short) which are computationally more efficient and easier to implement. The UCB algorithm associates each arm with two quantities: the mean of the rewards and the number of times the arm has been played. The pseudo-code of the UCB algorithm for Bernoulli multi-armed bandit is given in Algorithm 10 (Auer et al., 2002).

Algorithm 10 UCB Algorithm for Bernoulli Multi-armed Bandit

Input: a set of k arms.
$\forall i \in [k]$, let α_i and β_i be the success and failure counters
$\forall i \in [k]$, pull arm i once.
for $t = 0, 1 \ldots$ until the stopping condition is satisfied **do**
    Select arm i that maximises $\frac{\alpha_i}{\alpha_i+\beta_i} + \sqrt{\frac{2\log(t)}{\alpha_i+\beta_i}}$
    Observe the reward r_i
    if $r_i = 1$ **then**
        $\alpha_i := \alpha_i + 1$
    else
        $\beta_i := \beta_i + 1$
    end if
end for

(Auer et al., 2002) introduced a variant of the UCB algorithm known as UCB-tuned and experimentally showed that it achieve competitive, often better, results than UCB algorithm on certain multi-armed bandit models. At time $t \in \mathbb{N}$, the UCB-tuned algorithm picks the arm that maximises the following confidence bound:

$$\frac{\alpha_i}{\alpha_i + \beta_i} + \sqrt{\frac{\log(t)}{\alpha_i + \beta_i} \min\left\{\frac{1}{4}, \sigma_i^2 + \sqrt{\frac{2\log(t)}{\alpha_i + \beta_i}}\right\}} \tag{3.3}$$

where α_i, β_i, and t are as defined in Algorithm 10, and σ_i^2 is the sample variance.

The confidence bound in the UCB algorithm is computed based on the entire history of invocations of each arm. This implies that, as the number of pulls of each arm increases, the new results have less impact on the confidence bound estimate. Several variants of the UCB algorithm have been proposed in the literature with the goal of adapting the UCB algorithm to changing environments. As presented in (Garivier and Moulines, 2011), the past results of the arms can be discarded by associating the UCB algorithm with a discount factor that gradually decreases the importance of the old results, which in turn increases the impact of the recent results on the upper confidence bound estimate. However, this introduces a new parameter to the UCB algorithm which may need tuning in order to perform well in terms of handling the exploration and exploitation trade-off. The UCB algorithm was also hybridised with a statistical change-point detection test called the Page-Hinkley (PH) (Page, 1954) to detect abrupt changes in the environment (see, for example, (Fialho et al., 2009)). Suppose that a set of n rewards are observed from the current best arm. The PH test is concerned with deciding whether these observations refer to the same distribution, taking into consideration the normal variations, or some changes occurred. Algorithm 11 precisely describes the PH test. In Fialho et al.'s work, the UCB algorithm was restarted whenever the PH test detects a change in the environment (Fialho et al., 2009).

Algorithm 11 Page-Hinkley Test

Input: a set of n observations $x_1, x_2, \ldots, x_n$, and parameter λ and δ.
Let $\bar{x} := \frac{1}{n}\sum_{i=1}^{n} x_i$.
Let $m_n := \sum_{i=1}^{n}(x_i - \bar{x} + \delta)$.
Let $M_n := \max_{t=1,\ldots,n}\{m_t\}$.
Let $PH_{test} := M_n - m_n$.
if $PH_{test} > \lambda$ **then**
    result:= change detected
else
    result:= no change detected
end if
Output: result

3.6 The Cross-domain Heuristic Search Challenge

Before presenting the contributions of this thesis, it is useful to discuss the results of a well-known competition in the field of hyper-heuristics called *cross-domain heuristic search challenge* (CHeSC) 2011, as it will be used as a benchmark for the proposed algorithms in this thesis. Before the CHeSC 2011 competition, the organisers designed the HyFlex framework to support comparisons between the competing algorithms. As discussed in Section 2.7, HyFlex is a software framework that provides a common interface to enable researchers to evaluate their algorithms across several problem domains. The task of the competitors was to design their high-level selection hyper-heuristics, while HyFlex provided all of the problem-dependent implementations, including the initialisation methods, objective functions, and low-level heuristics.

Before the competition began, Burke et al. (2010a) investigated the use of a set of hyper-heuristic approaches on three problem domains within the HyFlex framework: one-dimensional bin packing, permutation flow shop, and personnel scheduling. The goal of this study was to be a proof of concept for the HyFlex as a benchmark for evaluating the generality of hyper-heuristics and other search heuristics across different combinatorial optimisation problems. The overall results revealed that the most promising results were achieved by a simple iterated local search

hyper-heuristic which iteratively applies a mutation or ruin-recreate low-level heuristic chosen uniformly at random, followed by the sequence of all the available local search heuristics based on a predefined order.

International competitors submitted twenty hyper-heuristic approaches to CHeSC 2011 [1]. Regarding the competition's best-ranking hyper-heuristics, the winning hyper-heuristic, called *AdapHH*, is based on a dynamic relay technique that hybridises the best-performing heuristics (Mısır et al., 2012b). The second best hyper-heuristic is the *VNS-HH* (variable neighbourhood search based hyper-heuristic) which applies a predefined sequence of mutation and local search heuristics (Hsiao et al., 2011). The *ML* hyper-heuristic is ranked as the third best hyper-heuristic, which applies a mutation low-level heuristic followed by the application of a set of local search heuristics (Larose, 2011). The description of the other competing hyper-heuristics can be found on the competition website. Since the competition, several studies have used the CHeSC 2011 as a benchmark to evaluate their algorithms. Burke et al. (2011b) assessed the performance of two adaptive iterated local search hyper-heuristics using HyFlex. Drake et al. (2012) proposed a variant of the *choice function* hyper-heuristic (Cowling et al., 2000) and applied it to the HyFlex problems. In (Ochoa et al., 2012b), adaptive population-based EAs were proposed and evaluated on the HyFlex problem domains. Kheiri et al. (2015) tested a hyper-heuristic utilising a hidden Markov chain model using the HyFlex framework.

[1] www.asap.cs.nott.ac.uk/external/chesc2011

Chapter 4

Theoretical Analyses of Reinforcement Learning Hyper-heuristics

This chapter presents the first studies investigating theoretically commonly used reinforcement learning mechanisms in hyper-heuristics. Additionally, it introduces a benchmark problem for the theoretical and empirical evaluations of hyper-heuristics.

4.1 Introduction

The goal of using reinforcement learning mechanisms in hyper-heuristics is to identify the most appropriate low-level heuristics to apply at any given time during the search process. As discussed in the literature review (Section 3.3), commonly used heuristic selection methods in hyper-heuristics are simple reinforcement learning mechanisms such as random gradient, greedy, and additive reinforcement learning mechanisms (Özcan et al., 2008) (Cowling et al., 2001) (Özcan et al., 2010). Runtime analysis of hyper-heuristics with random gradient, greedy, and additive reinforcement learning mechanisms are non-existing. Several empirical studies have reported inconclusive results about the performance of hyper-heuristics with the commonly

used additive reinforcement learning mechanism (Nareyek, 2004) (Özcan et al., 2010) (Remde et al., 2012) (Ozcan et al., 2009). A question arises: when the additive reinforcement learning mechanism does not improve the performance when compared to the simple random strategy which chooses low-level heuristics uniformly at random?

This chapter presents the first study investigating theoretically the runtime of a hyper-heuristic with the aforementioned reinforcement learning mechanisms. We consider the generic selection hyper-heuristic (GSHH) described in the preliminaries chapter (Section 2.5), and investigate whether the random gradient and greedy mechanisms can improve the expected runtime of the GSHH with the simple random strategy on two benchmark problems. Additionally, we investigate theoretically the research question mentioned above and show the conditions under which the additive reinforcement learning mechanism chooses asymptotically low-level heuristics uniformly at random. An artificial benchmark problem is introduced as a secondary contribution, in order to gain insight into the performance of reinforcement learning hyper-heuristics. The proposed problem allows evaluating the performance of reinforcement learning hyper-heuristics with respect to different scenarios in a controlled environment.

4.2 Benchmark Problems

We analyse the runtime of the GSHH (see Algorithm 2 in Section 2.5) on two benchmark problems. The first problem is a well-known test problem in runtime analysis, the so-called LEADINGONES problem. The second problem is a proposed artificial benchmark problem. The description of both problems are given in the following sections.

4.2.1 LEADINGONES

We propose a search scenario on a frequently used theoretical benchmark problem called LEADINGONES.

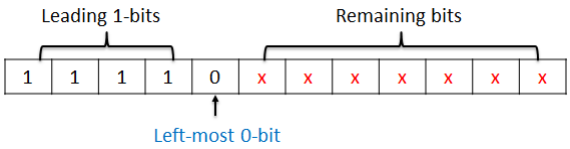

Figure 4.1: LEADINGONES

Definition 12. *For all* $x \in \{0,1\}^n$,

$$\text{LEADINGONES}(x) := \sum_{i=1}^{n} \prod_{j=1}^{i} x_j$$

The LEADINGONES value of a bit-string is the number of consecutive leading 1-bits in the bit-string. In order to improve the candidate solution, at least the left-most 0-bit should be flipped and the leading 1-bits should remain the same (see Figure 4.1). The reason for using the LEADINGONES problem is that, in the considered search scenario, the probability of generating a better solution, with respect to the LEADINGONES function, decreases as the hyper-heuristic moves to the optima. This property is common on some NP-hard problems of practical interest where the probability of generating better solutions, with respect to a given objective function, decreases as the optimiser moves toward the optima.

Most of runtime analyses conducted in this chapter use a state-of-the-art runtime analysis technique known as drift analysis (Hajek, 1982) (He and Yao, 2001) (Lehre and Witt, 2014). Recall that in drift analysis, a potential function (a so-called *distance function*) has to be defined. The distance function assigns each non-optimal search point to a value strictly larger than zero and each optimal search point to zero. The distance function value of a given search point thereby represents its "distance" to the optimal search point. Here, we choose the distance function $g : \{0,1\}^n \to \mathbb{R}$ to be:

$$g(x) := n - \text{LEADINGONES}(x) \tag{4.1}$$

for any bit-string $x \in \{0,1\}^n$. We choose X_t as the function $g(x_t)$ of the search point x_t, i.e.

$$X_t := g(x_t) \tag{4.2}$$

given that x_t is the current search point at time $t \in \mathbb{N}$, such that X_t denotes the number of non-leading 1-bits in iteration t of the GSHH. Let x_{min} and x_{max} be the minimum and maximum value, respectively, that X_t can take, i.e.

$$X_t \in \{0\} \cup \{x_{\min}, x_{\max}\}$$

We analyse the runtime of the GSHH on the LEADINGONES problem based (unless stated otherwise) on the following assumptions:

- It is assumed that all the bits after the left-most 0-bit are distributed uniformly at random, such that each bit is 1 with probability $1/2$ and 0 otherwise. According to Lemma (1) in (Lehre and Witt, 2012), this is a correct assumption which also makes it easier to estimate the expected number of consecutive 1-bits after left-most 0-bit.

- The hyper-heuristic has a set of m low-level heuristics $H := \{h_1, h_2, \ldots, h_m\}$. We assume that

  - h_1 is a mutation operator which flips one bit position chosen uniformly at random.
  - h_2 is a mutation operator which flips two bits uniformly at random.
  - $h_3 - h_m$ are variation operators that always return the same input bit-string.

  We are interested in analysing the limitations and capabilities of the considered learning mechanisms rather than effectively solving the LEADINGONES problem. Therefore, we propose this scenario where only two among m low-level heuristics can improve the so-

lution to investigate the learning capabilities of the considered learning mechanisms to identify the appropriate low-level heuristics. This also simulates the typical case where low-level heuristics most often fail to generate better solutions in most of time during the search process.

- We choose $x_{\min} := 2$ and $x_{\max} := n-1$, such that $\forall t \in \mathbb{N}$, $2 \leq X_t \leq n-1$. The distance to the optimal solution 1^n is at most $n-1$, i.e. $g(x) \leq n-1$ for any search point $x \in \{0,1\}^n$. The reason of choosing $x_{\min} = 2$ is that the success probability of the second variation operator is zero if the distance to the optimal is less than two as explained below.

Based on the above assumptions, the success probabilities of the low-level heuristics $h_1, \ldots, h_m$ in iteration t of the GSHH with respect to a given search point $x_t \in \{0,1\}^n$ and the LEADINGONES function are as follows.

1. The success probability of the first low-level heuristic h_1 is:

$$q_1(x_t) := \frac{1}{n} \quad \forall t \in \mathbb{N}$$

Recall that in order to generate a strictly better solution with respect to the LEADINGONES function at least the left-most 0-bit must be flipped and the leading 1-bits remain the same. Since the first low-level heuristic flips one bit uniformly at random, the probability that the flipped bit is the left-most 0-bit is $(1/n)$.

2. The success probability of the second low-level heuristic h_2 is:

$$q_2(x_t) := \frac{2(X_t - 1)}{n^2} \quad \forall t \in \mathbb{N}$$

The second low-level heuristic generates a strictly better solution, with respect to the LEADINGONES function, if the flipped bits are the left-most 0-bit and any one of the

remaining bits. Hence, by noting that the probability of flipping one of the remaining bits uniformly at random is $\frac{X_t-1}{n}$ and the probability of flipping the left-most 0-bit is $\frac{1}{n}$, the success probability of the second low-level heuristic is $\frac{2(X_t-1)}{n^2}$. Note that if $X_t < 2$, then the success probability of the second low-level heuristic is zero.

3. For all $j \in \{3,\ldots,m\}$, the success probability of the j-th low-level heuristic is:

$$q_j(x_t) := 0 \quad \forall t \in \mathbb{N}$$

As mentioned above, $\forall j \in \{3,\ldots,m\}$, low-level heuristic j, when applied, returns the same input bit-string.

4.2.2 Uniform Bernoulli Benchmark Problem

This section proposes a benchmark problem inspired by the artificial search scenarios proposed in (Fialho, 2011). The proposed problem, called Uniform Bernoulli (UB), allows evaluating the performance of reinforcement learning hyper-heuristics with respect to different scenarios. We now present the formal description of the UB problem.

There is a finite search space $\mathscr{X}$. A search point x is a bit-string of size n (i.e. $x \in \{0,1\}^n$). Let $UB : \mathscr{X} \to \mathbb{R}$ be an objective function defined as follows:

$$UB(x) := n - \sum_{i=1}^{n} x_i \tag{4.3}$$

The goal of algorithms for the UB problem is to minimise the function value of the UB function. Note that the definition of the UB function is similar, yet not identical, to the so-call ONEMAX (see, for example, (Auger and Doerr, 2011)). The difference between the UB problem and the ONEMAX problem is that the former is a minimisation problem; whereas, the latter is a

maximisation problem. Additionally, let $\mathscr{X}$ be partitioned into n states according to the UB function such that the solution in state X_i if it has UB-value equal i, i.e.

$$X_i = \{x | UB(x) = i\} \quad \text{for } i \in [n] \tag{4.4}$$

A set of m low-level heuristics $H := \{h_1, \ldots, h_m\}$ is available to be applied at any given time during the search process. For $j \in [m]$, low-level heuristic h_j is a function defined as follows:

$$h_j : \mathscr{X} \to \mathscr{X} \tag{4.5}$$

An algorithm for the UB problem has to decide which low-level heuristic should be applied at time $t \in \mathbb{N}$. For $i \in [n]$ and $j \in [m]$, if the solution in state X_i, then low-level heuristic j improves the solution, with respect to the UB function, with probability $q_j^{(i)}$. In order to improve the solution, with respect to the UB function, the number of 0-bits should be decreased by at least one. We assume that low-level heuristic j, when applied, flips exactly one 0-bit with probability $q_j^{(i)}$ such that the difference between the available low-level heuristics is their success probabilities.

4.3 General Bounds on the Optimisation Time

This section presents general bounds for the expected runtime of the GSHH on both the LEADINGONES and UB problems. These bounds hold for the GSHH with any arbitrary reinforcement learning mechanism. It is useful to first show such general results to give hints on how reinforcement learning mechanisms influence the runtime of the GSHH on the considered benchmark problems.

4.3.1 Results for LEADINGONES

The GSHH optimises the LEADINGONES function by choosing a variation operator i from a set $H := \{h_1, h_2, \ldots, h_m\}$ of m variation operators with probability $p_i \in [0,1]$. The chosen operator is then applied to the current candidate solution. The newly generated solution is accepted if it is not found to be worse than the current candidate solution with respect to the LEADINGONES function. This process is repeated until the optimal solution 1^n is found. The following theorem shows the runtime of the GSHH on the LEADINGONES problem with the prescribed settings in Section 4.2.1.

Theorem 13. *For $j \in [m]$, let $p_j \in]0,1]$ be the selection probability of the j-th low-level heuristic. The expected runtime of the GSHH on the* LEADINGONES *problem is*

$$\mathbb{E}[T_{GSHH,\,LO}] \leq \begin{cases} \frac{n^2}{p_1} & \text{if} \quad p_1 > \frac{2p_2}{\log(n)+1} \\ \frac{n^2}{2p_2} \cdot (\log(n)+1) & \text{otherwise} \end{cases} \tag{4.6}$$

The proof of Theorem 13 is based on a runtime analysis technique called artificial fitness-levels (Wegener, 2001). As discussed in Section 2.6.2, in order to use this technique, one has to define fitness-levels partitioning the search space according to a given objective function. Here, we used the canonical fitness-levels as defined below.

Proof. Define the fitness-levels as follows:

$$A_i := \{x | \textsc{LeadingOnes}(x) = i\}, \quad \text{for } i \in [n] \tag{4.7}$$

For $j \in [m]$ and $i \in [n]$, let $q_j^{(i)}$ be the success probability of low-level heuristic j in fitness level i, and p_j be the selection probability of the j-th low-level heuristic. We have $q_1^{(i)} = \frac{1}{n}$, $q_2^{(i)} = \frac{2(n-i-1)}{n^2}$ if $i \leq n-2$; otherwise, $q_2^{(i)} = 0$, and for all $k \in \{3, \ldots, m\}$, $q_k^{(i)} = 0$. Thus, it

follows by Theorem 7 that:

$$\mathbb{E}[T_{GSHH,LO}] \leq \min\left\{\sum_{i=1}^{n-1}\frac{1}{p_1 q_1^{(i)}}, \sum_{i=1}^{n-1}\frac{1}{p_2 q_2^{(i)}}\right\} \tag{4.8}$$

$$= \min\left\{\frac{1}{p_1}\sum_{i=1}^{n-1} n, \frac{1}{p_2}\sum_{i=1}^{n-1}\frac{n^2}{2(n-i-1)}\right\} \tag{4.9}$$

$$\leq \min\left\{\frac{n^2}{p_1}, \frac{n^2}{2p_2}H_n\right\} \tag{4.10}$$

$$= \min\left\{\frac{n^2}{p_1}, \frac{n^2}{2p_2}(\log(n)+1)\right\} \tag{4.11}$$

By noting that

$$\frac{n^2}{p_1} < \frac{n^2}{2p_2}(\log(n)+1) \tag{4.12}$$

if

$$p_1 > \frac{2p_2}{\log(n)+1} \tag{4.13}$$

then the theorem immediately follows. □

In all fitness levels $i > n/2$, the first variation operator h_1 has the maximum success probability of improving the solution towards a higher fitness level where:

$$\frac{1}{n} \geq \frac{2(n-i-1)}{n^2} \quad \text{for all} \quad i > \frac{n}{2} \tag{4.14}$$

Theorem 13 suggests that if the first variation operator is chosen with probability larger than $\frac{2p_2}{\log(n)+1}$, where p_2 is the selection probability of the second variation operator, then the expected runtime of the GSHH is at most n^2/p_1; otherwise, it is no more than $\frac{n^2}{2p_2}(\log(n)+1)$. This implies that if the selection probability of the first low-level heuristic is lower bounded by a

constant with respect to n that satisfies the above condition, then the expected runtime of the GSHH with any arbitrary learning mechanism on the LEADINGONES problem is $O(n^2)$.

4.3.2 Results for the UB problem

The GSHH optimises the UB function as follows. At time $t \in \mathbb{N}$, the GSHH picks a low-level heuristic $i \in [m]$ with probability $p_i^{(t)} \in [0,1]$. The selected low-level heuristic is then applied to the candidate solution. The newly generated solution is accepted if (and only if) it is not worse than the current candidate solution. This process is repeated until the GSHH evaluates an optimal solution for the first time. The following theorem shows the worst-case runtime of the GSHH, with any learning mechanism, on the UB problem.

Theorem 14. *If there exists an $\varepsilon \in]0,1]$ such that the success probability $q_i(x_t) \geq \varepsilon$ for all $i \in [m]$, and $x_t \in \mathscr{X}$, then the expected runtime of the GSHH on the UB problem is at most*

$$\mathbb{E}[T_{GSHH,\,UB}] \leq \frac{n-1}{\varepsilon} \tag{4.15}$$

Theorem 14 suggests that if the success probabilities of low-level heuristics are lower bounded by a constant with respect to n, then the expected runtime of the GSHH with any heuristic selection method on the UB problem is asymptotically $O(n)$.

Proof. Define $X_t := UB(x_t)$ given that x_t is the search point in the t-th iteration of the GSHH. For $j \in [m]$ and $t \in \mathbb{N}$, let $p_j^{(t)}$ be the selection probability of low-level heuristic j, and $q_j(x_t)$ be the success probability of low-level heuristic j in the t-th iteration of the GSHH. By the law of total expectation, the expected change in distance is

$$\mathbb{E}[X_t - X_{t+1} | X_t > 0] = \sum_{j=1}^{m} p_j^{(t)} q_j^{(t)} \tag{4.16}$$

We now use the fact that, $\forall j \in [m]$ and $t \geq 0$, $q_j^{(t)} \geq \varepsilon$.

$$\mathbb{E}[X_t - X_{t+1} | X_t > 0] \geq \varepsilon \sum_{j=1}^{m} p_j^{(t)} = \varepsilon \tag{4.17}$$

The expected runtime of the GSHH on the UB problem is therefore by Theorem 9 upper bounded by

$$\mathbb{E}[T_{GSHH,UB}] \leq \frac{n-1}{\varepsilon} \tag{4.18}$$

□

4.4 Runtime Analysis of the GSHH with Classical Heuristic Selection Methods

This section presents runtime analyses of the GSHH with the *simple random*, *random permutation*, *greedy*, and *random gradient* mechanisms. Both the simple random and random permutation strategies are employed as a baseline for our comparisons as they do not learn. The greedy and random gradient are reinforcement learning mechanisms with short memory, such that they rapidly forget the feedback from the search environment, as discussed in Section 3.3. The reason of evaluating the performance of the greedy and random gradient mechanisms is that both mechanisms are commonly used in hyper-heuristics and their theoretical background is non-existing (see, for example, (Cowling et al., 2001); (Özcan et al., 2008); (Berberoglu and Uyar, 2011)).

4.4.1 Results for LEADINGONES

This section, uses the runtime as performance measure, and investigates the performance of the GSHH with the heuristic selection strategies mentioned earlier on the LEADINGONES problem. For the purpose of simplicity, we assumed that the number of low-level heuristics is two, resulting in only the first two variation operators being used. Our analysis employs a state-of-the-art runtime analysis technique called drift analysis to estimate the runtime of the GSHH on the LEADINGONES problem. The *drift* of the GSHH constitutes the progress in moving towards the optimum in an iteration of the GSHH.

As discussed in Section 3.1, the simple random strategy chooses low-level heuristics uniformly at random. Let $p_i^{(t)}$ be the selection probability of low-level heuristic i in the t-th iteration of the GSHH, which, with the simple random strategy fixes at $p_i^{(t)} := \frac{1}{m}$ for all $i \in [m]$ and $t \in \mathbb{N}$ given that m is the number of low-level heuristics. Lemma 15 shows the expected drift of the GSHH with the simple random strategy on the LEADINGONES problem. The result shows that the second operator contributes to the drift of the GSHH with the simple random strategy as long as $X_t \geq \frac{n+2}{2}$, where the success probability of the second operator is larger than the success probability of the first operator if $X_t \geq \frac{n+2}{2}$ i.e.

$$\frac{2(X_t - 1)}{n^2} \geq \frac{1}{n} \quad \text{if } X_t \geq \frac{n+2}{2} \tag{4.19}$$

This implies that if there exists a learning mechanism that is capable of identifying the most suitable variation operator, then the GSHH benefits from mixing the two variation operators. This particular observation, hyper-heuristics can benefit from mixing different heuristics, has already been shown in (Lehre and Özcan, 2013) (He et al., 2012). We now present our runtime analysis of the GSHH with the simple random, random permutation, greedy, and random gradient mechanisms. The results will be discussed afterword.

Lemma 15. *The GSHH with the simple random strategy, and* $H := \{h_1, h_2\}$*, on the* LEADINGONES *problem has drift bounded by*

$$\frac{n+2(X_t-1)}{2n^2} \leq \mathbb{E}[X_t - X_{t+1} | X_t \geq x_{min}] \leq \frac{n+2(X_t-1)}{n^2} \tag{4.20}$$

Proof. Denote by X_t the distance to the optimal solution 1^n in iteration t of the GSHH i.e. $X_t := n - \text{LEADINGONES}(x_t)$ given that x_t is the candidate solution in iteration t of the GSHH. Let R be the random variable representing the number of consecutive 1-bits after the left-most 0-bit in the bit-string. Since every bit after the left-most 0-bit is 1 with probability $1/2$, the expected value of R is:

$$\mathbb{E}[R|X_t = y] = \sum_{i=1}^{y-1} i \left(\frac{1}{2}\right)^{i+1} = \frac{1}{2}\sum_{i=1}^{y-1} i \left(\frac{1}{2}\right)^{i} \tag{4.21}$$

$$= \frac{1}{2}\left(\frac{(y-1)(1/2)^{y+1} - y(1/2)^y + (1/2)}{1/4}\right) = 1 - 2^{-y}(y+1). \tag{4.22}$$

By noting that $y = X_t$, the following inequality holds:

$$0 \leq 1 - 2^{-y}(y+1) \leq 1 \tag{4.23}$$

where $1 \leq X_t \leq n-1$. Therefore, it holds $0 \leq \mathbb{E}[R \mid X_t] \leq 1$, so, the expected progress in an improving step is at least 1 and at most 2. This also implies that $\mathbb{E}[X_0] \geq n-2$. Recall that the simple random strategy chooses a low-level heuristic uniformly at random in each iteration of the GSHH. Thus, by the law of total expectation, the expected change in distance is bounded from below by

$$\mathbb{E}[X_t - X_{t+1} | X_t \geq x_{min}] \geq \frac{1}{2}(q_1(x_t) + q_2(x_t)) \tag{4.24}$$

and from above by

$$\mathbb{E}[X_t - X_{t+1} | X_t \geq x_{min}] \leq q_1(x_t) + q_2(x_t) \tag{4.25}$$

where $q_1(x_t) = 1/n$, and $q_2(x_t) = 2(X_t - 1)/n^2$. □

Theorem 16. *The expected runtime of GSHH with the simple random strategy, and* $H := \{h_1, h_2\}$*, on the* LEADINGONES *problem is upper bounded by*

$$\mathbb{E}[T_{GSHH\text{-}SR,\ LO} | X_0] \leq n^2 \log(3) + O(n) \tag{4.26}$$

Proof. The drift of the GSHH with the simple random strategy is by Lemma 15 lower bounded as:

$$\mathbb{E}[X_t - X_{t+1} | X_t > x_{min}] \geq \frac{n + 2X_t - 2}{2n^2} =: h(X_t). \tag{4.27}$$

Since the first derivative of the expected drift of the GSHH with the simple random strategy $h(X_t)$ is $h'(x) = 1/n^2 > 0$, the expected runtime of the GSHH is by Theorem 11 at most

$$\begin{aligned}
\mathbb{E}[T_{\text{GSHH-SR, LO}} | X_0] &\leq \frac{x_{min}}{h(x_{min})} + \int_{x_{min}}^{X_0} \frac{1}{h(y)} dy && (4.28)\\
&\leq \frac{4n^2}{n+2} + \int_2^{n-1} \frac{2n^2}{n+2y-2} dy && (4.29)\\
&\leq n^2 \left(\frac{4}{n+2} + \log\left(\frac{3n-4}{n+2} \right) \right). && (4.30)
\end{aligned}$$

The theorem now follows by noting that $\frac{4}{n+2} = O(1/n)$ and $\log(\frac{3n-4}{n+2}) < \log(3)$. □

Theorem 17. *The expected runtime of the GSHH with the simple random strategy, and* $H := \{h_1, h_2\}$*, on the* LEADINGONES *problem is lower bounded by*

$$\mathbb{E}[T_{GSHH\text{-}SR,\ LO} | X_0] \geq \frac{n^2}{6} \log(3). \tag{4.31}$$

Proof. The expected single-step drift of the GSHH with the simple random strategy is by Lemma 15 at most:

$$\mathbb{E}\left[X_t - X_{t+1} | X_t > x_{min}\right] \leq \frac{n+2X_t}{n^2} =: h(X_t) \tag{4.32}$$

Hence,

$$\begin{aligned}
\mathbb{E}[g(X_t) - g(X_{t+1}) | X_t \geq x_{min}] &\leq \int_{X_{t+1}}^{X_t} \frac{1}{h(y)} dy && (4.33)\\
&\leq \frac{n^2}{2} \log\left(\frac{n+2X_t}{n+2X_{t+1}}\right) && (4.34)\\
&\leq \frac{n^2}{2} \log\left(1 + \frac{2(X_t - X_{t+1})}{n+2X_{t+1}}\right) && (4.35)\\
&\leq \frac{n^2}{2} \left(\frac{2}{n}\mathbb{E}[X_t - X_{t+1} | X_t]\right) && (4.36)\\
&\leq \frac{n(n+2X_t)}{n^2} \leq 3, && (4.37)
\end{aligned}$$

because the largest value that X_t can take is $n-1$. The theorem now follows by the second statement in Theorem 11 with $\alpha = 3$.

$$\mathbb{E}\left[T_{\text{GSHH-SR, LO}} | X_0\right] \geq \frac{g(X_0)}{3} \tag{4.38}$$

where,

$$\begin{aligned}
g(X_0) &= \frac{2n^2}{n+2} + \int_2^{n-1} \frac{n^2}{n+2y-2} dy && (4.39)\\
&\geq \frac{n^2}{2} \log\left(\frac{3n-4}{n+2}\right) && (4.40)\\
&\geq \frac{n^2}{2} \log(3). && (4.41)
\end{aligned}$$

□

Before analysing the performance of the GSHH with the considered reinforcement learning

mechanisms, we analyse runtime of the GSHH with the random permutation strategy which generates a random order of all low-level heuristics, then permutes to the next low-level heuristic in the cyclic order (Cowling et al., 2001). The following lemma shows the expected drift of the GSHH with the random permutation strategy on the LEADINGONES problem.

Lemma 18. *The expected drift of the GSHH with the random permutation strategy, and* $H := \{h_1, h_2\}$*, on the* LEADINGONES *problem is:*

$$\frac{n+2X_t-2}{n^2} \leq \mathbb{E}[X_t - X_{t+2} | X_t \geq x_{min}] \leq \frac{2n+4(X_t-1)}{n^2} \tag{4.42}$$

Proof. We consider the drift of two steps for more accurate runtime analysis:

Case 1: If the first variation operator is in the top of the list.

$$\mathbb{E}[X_t - X_{t+2} | X_t \geq x_{min}] \geq q_1(X_t)(1-q_2(X_{t+1})) \tag{4.43}$$

$$+ (1-q_1(X_t))q_2(X_t) + 2q_1(X_t)q_2(X_{t+1}) \tag{4.44}$$

$$= \frac{n+2X_t-2}{n^2} \tag{4.45}$$

Case 2: If the second variation operator is in the top of the list.

$$\mathbb{E}[X_t - X_{t+2} | X_t \geq x_{min}] \geq q_2(X_t)(1-q_1(X_{t+1})) \tag{4.46}$$

$$+ (1-q_2(X_t))q_1(X_t) + 2q_2(X_t)q_1(X_{t+1}) \tag{4.47}$$

$$= \frac{n+2X_t-2}{n^2} \tag{4.48}$$

It can be seen that the drift in both cases are the same, then the expected drift of the GSHH is at

least

$$\mathbb{E}[X_t - X_{t+2} | X_t \geq x_{min}] \geq \frac{n + 2X_t - 2}{n^2} \tag{4.49}$$

and at most

$$\mathbb{E}[X_t - X_{t+2} | X_t \geq x_{min}] \leq \frac{2n + 4(X_t - 1)}{n^2} \tag{4.50}$$

□

Theorem 19. *The expected runtime of the GSHH with the random permutation strategy, and* $H := \{h_1, h_2\}$*, on the* LEADINGONES *problem is upper bounded as*

$$\mathbb{E}\left[T_{GSHH\text{-}RP,\ LO} | X_0\right] \leq n^2 \left(\log(3) + o(1)\right) \tag{4.51}$$

Proof. The expected two-step drift of the GSHH with the random permutation strategy is by Lemma 18 at least

$$\mathbb{E}[X_t - X_{t+2} | X_t \geq x_{min}] \leq \frac{n + 2X_t - 2}{n^2} =: h(X_t) \tag{4.52}$$

Since we use the drift of two steps, the expect runtime of the GSHH with the random permutation strategy is by Theorem 11 at most

$$\mathbb{E}\left[T_{\text{GSHH-RP, LO}} | X_0\right] \leq 2 \left(\frac{2n^2}{n+2} + \int_2^{n-1} \frac{n^2}{n+2y-2} dy \right) \tag{4.53}$$

$$\leq n^2 \left(\frac{4}{n+2} + \log(\frac{3n-4}{n+2}) \right) \tag{4.54}$$

where $\frac{3n-4}{n+2} < 3$. Thus, we have

$$\mathbb{E}\left[T_{\text{GSHH-RP, LO}}|X_0\right] \leq n^2\left(\log(3)+o(1)\right) \tag{4.55}$$

□

Theorem 20. *The expected runtime of the GSHH with the random permutation strategy, and* $H := \{h_1, h_2\}$*, on the* LEADINGONES *problem is lower bounded by*

$$\mathbb{E}\left[T_{\textit{GSHH-RP, LO}}|X_0\right] \geq \frac{n^2}{6}\left(\log(3)+o(1)\right) \tag{4.56}$$

Proof. By Lemma 18, we have

$$\mathbb{E}[X_t - X_{t+2}|X_t \geq x_{min}] \leq \frac{2n+4(X_t-1)}{n^2} =: h(X_t) \tag{4.57}$$

Thus

$$\begin{aligned}
\mathbb{E}[g(X_t) - g(X_{t+1})|X_t \geq x_{min}] &\leq \int_{X_{t+1}}^{X_t} \frac{n^2}{2n+4(y-1)}dy && (4.58)\\
&\leq \frac{n^2}{4}\left(\log\left(\frac{n+2X_t-2}{n+2X_{t+1}-2}\right)\right) && (4.59)\\
&\leq \frac{n^2}{4}\left(\log\left(1+\frac{2(X_t-X_{t+1})}{n^2}\right)\right) && (4.60)\\
&\leq \frac{n^2}{4}\left(\frac{2}{n}\frac{2n+4X_t-4}{n^2}\right) && (4.61)
\end{aligned}$$

Since $X_t \leq n-1$, it follows that

$$\mathbb{E}[g(X_t) - g(X_{t+1})|X_t \geq x_{min}] \leq \frac{n}{2}\cdot\frac{6n}{n^2} = 3 \tag{4.62}$$

Recall that we use the drift of two steps, so by Theorem 11 we have

$$\mathbb{E}\left[T_{\text{GSHH-RP, LO}}|X_0\right] \geq \frac{2}{3}\left(\frac{n^2}{n+2} + \int_2^{n-1} \frac{n^2}{2n+4(y-1)}dy\right) \tag{4.63}$$

$$\geq \frac{2}{3}\left(\frac{n^2}{4}\left(\frac{4}{n+2} + \log\left(\frac{3n-4}{n+2}\right)\right)\right) \tag{4.64}$$

$$\geq \frac{n^2}{6}\left(\log(3) + o(1)\right) \tag{4.65}$$

□

Cowling et al. (2001) proposed the greedy strategy which applies all low-level heuristics to the same candidate solution and chooses the heuristic that achieves the best change in the objective function. The following analysis shows the runtime of the GSHH with the greedy strategy on the LEADINGONES problem.

Lemma 21. *The GSHH with the greedy strategy, and $H := \{h_1, h_2\}$, has drift bounded by*

$$\frac{2X_t(n-2)+n(n-2)+2}{n^3} \leq \mathbb{E}[X_t - X_{t+1}|X_t \geq x_{min}] \leq \frac{4X_t(n-2)+2n(n-2)+4}{n^3} \tag{4.66}$$

Proof. The greedy strategy applies both variation operators to the same candidate solution, then decides which operator to choose. Therefore, the expected change in distance is at least:

$$\mathbb{E}[X_t - X_{t+1}|X_t \geq x_{min}] \geq 1 - (1-q_1(x_t))(1-q_2(x_t)) \tag{4.67}$$

$$\geq \frac{2X_t(n-2)+n(n-2)+2}{n^3} \tag{4.68}$$

The second statement in Lemma 21 can be proved in symmetrical way to the first statement taking into account the lower and upper value of the expected progress in an improving step. □

Theorem 22. *The expected runtime of the GSHH with the greedy mechanism, and $H := \{h_1, h_2\}$,*

on the LEADINGONES *problem is upper bounded by:*

$$\mathbb{E}\left[T_{GSHH\text{-}G,\,LO}|X_0\right] \leq n^2\left(\log(3)+o(1)\right) \tag{4.69}$$

Proof. The expect drift of the GSHH with the greedy strategy is by Lemma 21 at least

$$\mathbb{E}[X_t - X_{t+1}|X_t \geq x_{min}] \geq \frac{2X_t(n-2)+n(n-2)+2}{n^3} \tag{4.70}$$

Thus, the expected runtime of the GSHH with the greedy strategy on the LEADINGONES problem is by Theorem 11 upper bounded as

$$\mathbb{E}\left[T_{\text{GSHH-G, LO}}|X_0\right] \leq \frac{2n^2}{n+2} + \int_2^{n-1} \frac{n^3}{2y(n-2)+n(n-2)+2}dy \tag{4.71}$$

$$\leq \frac{n^2}{2}\left(\frac{4}{n+2} + \log(\frac{3n^2-8n+6}{n^2+2n-6})\right) \tag{4.72}$$

where,

$$\frac{3n^2-8n+6}{n^2+2n-6} = \frac{n^2+2n-6}{n^2+2n-6} + \frac{2n^2-10n+12}{n^2+2n-6} \tag{4.73}$$

$$= 1+2(\frac{n^2-5n+6}{n^2+2n-6}) \tag{4.74}$$

$$\leq 1+2(1) = 3 \tag{4.75}$$

Since the greedy strategy applies both operators in every step, it holds:

$$\mathbb{E}\left[T_{\text{GSHH-G, LO}}|X_0\right] \leq 2\left(\frac{n^2}{2}(\frac{4}{n+2}+\log(3))\right) \tag{4.76}$$

$$\leq n^2\left(\log(3)+o(1)\right) \tag{4.77}$$

□

Theorem 23. *The expected runtime of the GSHH with the greedy mechanism, and* $H := \{h_1, h_2\}$*, on the* LEADINGONES *problem is lower bounded by:*

$$\mathbb{E}\left[T_{GSHH\text{-}G,\,LO}|X_0\right] \geq \frac{n^2}{6}\left(\log(3) + o(1)\right) \tag{4.78}$$

Proof. From Lemma 21, the expected drift of the GSHH with greedy mechanism is:

$$\mathbb{E}[X_t - X_{t+1}|X_t \geq x_{min}] \leq \frac{4X_t(n-2) + 2n(n-2) + 4}{n^3} \tag{4.79}$$

Hence,

$$\begin{aligned}
\mathbb{E}[g(X_t) - g(X_{t+1})|X_t \geq x_{min}] &\leq \int_{X_{t+1}}^{X_t} \frac{n^3}{4y(n-2) + 2n(n-2) + 4} dy && (4.80)\\
&\leq \frac{n^2}{4}\left(\log\left(\frac{4X_t(n-2) + 2n(n-2) + 4}{4X_{t+1}(n-2) + 2n(n-2) + 4}\right)\right) && (4.81)\\
&\leq \frac{n^2}{4}\left(\log\left(\frac{4X_t n + 2n^2}{4X_{t+1} n + 2n^2}\right)\right) && (4.82)\\
&\leq \frac{n^2}{4}\left(\log\left(1 + \frac{4n(X_t - X_{t+1})}{2n^2}\right)\right) && (4.83)\\
&\leq \frac{n^2}{4}\left(\frac{4n}{2n^2} \cdot \frac{4X_t(n-2) + n(n-2) + 4}{n^3}\right) && (4.84)
\end{aligned}$$

where $X_t \leq n-1$, then:

$$\mathbb{E}[g(X_t) - g(X_{t+1})|X_t \geq x_{min}] \leq \frac{n}{2} \cdot \frac{6n^2 - 10n}{n^3} < 3 \tag{4.85}$$

The expected runtime of the GSHH with the greedy mechanism on the LEADINGONES problem

is, therefore, at least:

$$\mathbb{E}\left[T_{\text{GSHH-G, LO}}|X_0\right] \geq 2\left(\frac{g(X_0)}{3}\right) \tag{4.86}$$

$$\geq \frac{2}{3}\left(\frac{n^2}{n+2} + \int_2^{n-1} \frac{n^3}{4y(n-2)+2n(n-2)+4} dy\right) \tag{4.87}$$

$$\geq \frac{2}{3}\left(\frac{n^2}{4}\left(\frac{8}{n+2} + \log\left(\frac{3n^2-8n+6}{n^2+2n-6}\right)\right)\right) \tag{4.88}$$

Since $\frac{3n^2-8n+6}{n^2+2n-6} \leq 3$ as shown in (4.73), it follows that:

$$\mathbb{E}\left[T_{\text{GSHH-G, LO}}|X_0\right] \geq \frac{2}{3}\left(\frac{n^2}{4}(\log(3)+o(1))\right) \tag{4.89}$$

$$\geq \frac{n^2}{6}(\log(3)+o(1)) \tag{4.90}$$

□

The random gradient strategy is a reinforcement learning mechanism that chooses a low-level heuristic uniformly at random, then the chosen heuristic is repeatedly applied as long as an improvement in the candidate solution is found (Cowling et al., 2001). Lemma 24 shows the expected drift of the GSHH with the random gradient mechanism on the LEADINGONES problem.

Lemma 24. *The two step drift of the GSHH with the random gradient mechanism if* $X_t \leq \frac{n}{2}+1$ *is:*

$$\frac{n(n-2)+X_t(2n-1)-1}{2n^3} \leq \mathbb{E}[X_t - X_{t+2}|X_t \geq x_{min}] \leq \frac{6n+4(X_t-1)}{2n^2} \tag{4.91}$$

If $X_t > \frac{n}{2}+1$*, then the drift is:*

$$\frac{3n+2X_t-2}{2n^2} \leq \mathbb{E}[X_t - X_{t+2}|X_t \geq x_{min}] \leq \frac{n(n-2)+X_t(2n-1)-1}{n^3} \tag{4.92}$$

Proof. The random gradient mechanism updates the selection probabilities of low-level heuristics according to the outcome of the previous iteration, hence we considered all the possible drift in two steps.

Case 1: An operator is chosen uniformly at random.

$$\mathbb{E}[X_t - X_{t+2} | X_t \geq x_{min}] \geq \left(1 - \frac{p(X_t)+q(X_t)}{2}\right)\left(\frac{p(X_t)+q(X_t)}{2}\right) \tag{4.93}$$

$$+ \left(1 - \frac{p(X_t)+q(X_t)}{2}\right)\frac{q(X_t)}{2} + \frac{p(X_t)(1-p(X_{t+1}))}{2} \tag{4.94}$$

$$+ \frac{q(X_t)(1-q(X_{t+1}))}{2} + \frac{2p(X_t)p(X_{t+1})}{2} + \frac{2q(X_t)q(X_{t+1})}{2} \tag{4.95}$$

$$= \frac{(n+2X_t-2)(4n^2-n-2X_t+2)}{4n^4} =: h_1(X_t) \tag{4.96}$$

Case 2: The first operator is chosen with probability one.

$$\mathbb{E}[X_t - X_{t+2} | X_t \geq x_{min}] \geq p(X_t)(1-p(X_{t+1})) \tag{4.97}$$

$$+ (1-p(X_t))\frac{p(X_t)+q(X_t)}{2} + 2p(X_t)p(X_{t+1}) \tag{4.98}$$

$$= \frac{3n+2X_t-2}{2n^2} =: h_2(X_t) \tag{4.99}$$

Case 3: The second operator is chosen with probability one.

$$\mathbb{E}[X_t - X_{t+2} | X_t \geq x_{min}] \geq q(X_t)(1-q(X_{t+1})) \tag{4.100}$$

$$+ (1-q(X_t))\frac{p(X_t)+q(X_t)}{2} + 2q(X_t)q(X_{t+1}) \tag{4.101}$$

$$= \frac{n(n-2)+X_t(2n-1)-1}{2n^3} =: h_3(X_t) \tag{4.102}$$

The expected progress in an improving step is at least 1 and at most 2 as shown in Lemma 15,

then $\mathbb{E}[X_t - X_{t+2} | X_t \geq x_{min}]$ if $X_t \leq \frac{n}{2} + 1$:

$$h_3(X_t) \leq \mathbb{E}[X_t - X_{t+2} | X_t \geq x_{min}] \leq 2h_2(X_t) \tag{4.103}$$

Otherwise,

$$h_2(X_t) \leq \mathbb{E}[X_t - X_{t+2} | X_t \geq x_{min}] \leq 2h_3(X_t) \tag{4.104}$$

□

Theorem 25. *The expected runtime of the GSHH with the random gradient mechanism, and* $H := \{h_1, h_2\}$*, on the* LEADINGONES *problem is upper bounded by:*

$$\mathbb{E}\left[T_{GSHH\text{-}RG,\ LO} | X_0\right] \leq 2n^2 \left(\log(5/2) + o(1)\right) \tag{4.105}$$

Proof. If $X_t \leq \frac{n+2}{2}$ the expected drift is by Lemma 24 lower bounded as:

$$\mathbb{E}[X_t - X_{t+2} | X_t \geq x_{min}] \geq \frac{n(n-2) + X_t(2n-1) - 1}{2n^3} =: h_1(X_t) \tag{4.106}$$

Otherwise, the expected drift is:

$$\mathbb{E}[X_t - X_{t+2} | X_t \geq x_{min}] \geq \frac{3n + 2X_t - 2}{2n^2} =: h_2(X_t) \tag{4.107}$$

Since $h_1'(x)$ and $h_2'(x)$ are larger than zero, the expected runtime of the GSHH with the random

gradient mechanism is by Theorem 11 at most

$$\mathbb{E}\left[T_{\text{GSHH-RG, LO}}|X_0\right] \leq \frac{x_{min}}{h_1(x_{min})} + \int_{x_{min}}^{\frac{n}{2}+1} \frac{1}{h_1(y)}dy + \int_{\frac{n}{2}+2}^{n-1} \frac{1}{h_2(y)}dy \tag{4.108}$$

$$\leq \frac{4n^2}{n+2} + \int_{2}^{\frac{n}{2}+1} \frac{2n^3}{n(n-2)+y(2n-1)-1}dy + \int_{\frac{n}{2}+2}^{n-1} \frac{2n^2}{3n+2y-2}dy \tag{4.109}$$

$$\leq n^2\left(\frac{4}{n+2} + \log\left(\frac{5n-4}{4n+2}\right) + \log\left(\frac{-4n^2+n+4}{-2n^2-4n+6}\right)\right) \tag{4.110}$$

where,

$$\frac{-4n^2+n+4}{-2n^2-4n+6} = \frac{4n^2-n-4}{2n^2+4n-6} \tag{4.111}$$

$$= \frac{2n^2+4n-6}{2n^2+4n-6} + \frac{2n^2-5n+2}{2n^2+4n-6} \tag{4.112}$$

$$= 1 + \frac{2n^2-5n+2}{2n^2+4n-6} < 2 \tag{4.113}$$

and since $\log\left(\frac{5n-4}{4n+2}\right) \leq \log\left(\frac{5}{4}\right)$, then

$$\mathbb{E}\left[T_{\text{GSHH-RG, LO}}|X_0\right] \leq n^2\left(\log(2) + \log(5/4) + o(1)\right) \tag{4.114}$$

We used the drift of two steps, thus the expected runtime of the GSHH with the random gradient mechanism is upper bounded as:

$$\mathbb{E}\left[T_{\text{GSHH-RG, LO}}|X_0\right] \leq 2n^2\left(\log(5/2) + o(1)\right) \tag{4.115}$$

□

Theorem 26. *The expected runtime of the GSHH with the random gradient mechanism, and*

$H := \{h_1, h_2\}$, *on the* LEADINGONES *problem is lower bounded by:*

$$\mathbb{E}[T_{GSHH\text{-}RG,\ LO}|X_0] \geq \frac{n^2}{9}(4+3\log(10/3)) \tag{4.116}$$

Proof. The drift of the GSHH with the random gradient mechanism is by Lemma 24 at most:

$$\mathbb{E}[X_t - X_{t+2}|X_t \geq x_{min}] \leq \frac{n(n-2)+X_t(2n-1)-1}{n^3} \tag{4.117}$$

$$\begin{aligned}
\mathbb{E}[g(X_t) - g(X_t t+1)|X_t \geq x_{min}] &\leq \int_{X_{t+1}}^{X_t} \frac{2n^3}{2n(n-2)+2y(2n-1)-2} dy && (4.118)\\
&\leq \frac{n^2}{2}\left(\log\left(\frac{n^2+2n(X_t-1)}{n^2+2n(X_{t+1}-1)}\right)\right) && (4.119)\\
&\leq \frac{n^2}{2}\left(\log\left(\frac{n+2X_t}{n+2X_{t+1}}\right)\right) && (4.120)\\
&\leq \frac{n^2}{2}\left(\frac{2}{n}\mathbb{E}[X_t - X_{t+1}|X_t]\right) && (4.121)\\
&\leq n \cdot \frac{6n^2-4n}{2n^3} < 3 && (4.122)
\end{aligned}$$

where $X_t \leq n-1$. Then the expected runtime of this algorithm on the LEADINGONES problem is at least:

$$\begin{aligned}
\mathbb{E}[T_{\text{GSHH-RG, LO}}|X_0] &\geq 2(\frac{g(X_0)}{3}) && (4.123)\\
&\geq \frac{2}{3}\left(\frac{2n}{3} + \int_2^{\frac{n}{2}+1} \frac{2n^2}{6n+4(y-1)} dy + \int_{\frac{n}{2}+2}^{n-1} \frac{2n^3}{2n(n-2)+2y(2n-1)-2} dy\right) && (4.124)\\
&\geq \frac{n^2}{9}\left(4+3\left(\log\left(\frac{5n+2}{3n+2}\right)+\log\left(\frac{6n^2-10n}{4n^2+3n-6}\right)\right)\right) && (4.125)
\end{aligned}$$

Mechanism	Upper Bound	Lower Bound
Simple Random	$n^2 \log(3) + O(n)$	$\frac{n^2}{6} \log(3)$
Random Permutation	$n^2(\log(3) + o(1))$	$\frac{n^2}{6}(\log(3) + o(1))$
Greedy	$n^2(\log(3) + o(1))$	$\frac{n^2}{6}(\log(3) + o(1))$
Random Gradient	$2n^2(\log(5/2) + o(1))$	$\frac{n^2}{9}(4 + 3\log(10/3))$

Table 4.1: Runtime of the GSHH with various heuristic selection mechanisms on the LEADINGONES problem.

where,

$$\frac{6n^2 - 10n}{4n^2 + 3n - 6} = \frac{4n^2 - 3n - 6}{4n^2 + 3n - 6} + \frac{2n^2 - 7n + 6}{4n^2 + 3n - 6} \leq 2 \tag{4.126}$$

Thus,

$$\mathbb{E}[T_{\text{GSHH-RG, LO}} | X_0] \geq \frac{n^2}{9}(4 + 3(\log(5/3) + \log(2)) \tag{4.127}$$

$$\geq \frac{n^2}{9}(4 + 3\log(10/3)) \tag{4.128}$$

□

Overall, the above theoretical analyses show that the runtime of the GSHH with the simple random, random permutation, greedy, and random gradient mechanisms is almost the same on the LEADINGONES problem. Table 4.1 shows the expected runtime of the GSHH with the above heuristic selection mechanisms on the LEADINGONES problem. The results show that both the upper and lower bound on the expected runtime of the GSHH with the greedy and random permutation on LEADINGONES is the same. As shown in Table 4.1 the expected runtime of the GSHH with all the considered heuristic selection strategies is asymptotically $\Theta(n^2)$ on the LEADINGONES problem. This implies that using the GSHH with the simple random strategy,

which chooses heuristics uniformly at random, is similar to use the random gradient and greedy strategies with the prescribed settings on the LEADINGONES problem.

4.4.2 Results for the UB Problem

This section provides runtime analyses of the GSHH with the simple random, greedy, and random gradient strategies on the UB problem. The analyses are based on the artificial fitness-levels technique. Recall that in order to use the artificial fitness-level technique, one has to define the fitness-levels according to a given objective function, and estimate the minimum probability to leave each fitness-level towards a higher fitness-level. We define the fitness-levels as follows.

$$A_i = \{x | UB(x) = i\} \quad \text{for } i \in [n] \tag{4.129}$$

The following theorem shows an upper bound on the runtime of the GSHH with the simple random strategy on the UB problem.

Theorem 27. *The expected runtime of the GSHH with the simple random strategy on the UB problem is:*

$$\mathbb{E}[T_{\textit{GSHH-SR, UB}}] \leq m \sum_{i=1}^{n-1} \frac{1}{\sum_{j=1}^{m} q_j^{(i)}} \tag{4.130}$$

Proof. For $j \in [m]$ and $i \in [n]$, denote by $q_j^{(i)}$ the success probability of low-level heuristic j in fitness-level i. For $i \in [n]$, let $s^{(i)}$ be the probability of improving the solution in fitness level i towards a higher fitness level. Recall that the simple random strategy chooses each low-level heuristic with probability $(1/m)$ given that m is the number of low-level heuristics. Thus, by the

law of total probability, we have

$$s^{(i)} = \sum_{j=1}^{m} \frac{q_j^{(i)}}{m} \tag{4.131}$$

Apply Theorem 7 with $s^{(i)}$ and the theorem follows. □

The following theorem shows the runtime of the GSHH with the greedy mechanism on the UB problem.

Theorem 28. *The expected runtime of the GSHH with the greedy mechanism on the UB problem is:*

$$\mathbb{E}\left[T_{GSHH\text{-}G,\ UB}\right] \leq m(n-1) + \sum_{i=1}^{n-1} \frac{1}{1-\prod_{j=1}^{m}(1-q_j^{(i)})} \tag{4.132}$$

Proof. The probability of the GSHH with the greedy mechanism to upgrade the solution from fitness level i towards a higher fitness level is by the law of total probability at least

$$s^{(i)} \geq 1 - \left(\prod_{j=1}^{m}(1-q_j^{(i)})\right) \tag{4.133}$$

Recall that the greedy mechanism applies all the available low-level heuristics to the same candidate solution, then chooses the low-level heuristic that makes the best change in the candidate solution with respect to a given objective function. Therefore, it makes m objective function evaluations in each time step. Hence it follows by Theorem 7 that

$$\mathbb{E}[T_{\text{GSHH-G, UB}}] \leq \sum_{i=1}^{n-1} \left(m + \frac{1}{1-\left(\prod_{j=1}^{m}(1-q_j^{(i)})\right)}\right) \tag{4.134}$$

□

The following theorem presented the expected runtime of the GSHH with the random gradient mechanism on the UB problem.

Theorem 29. *For $i,k \in [n]$ and $j \in [m]$, If $q_j^{(i)} = q_j^{(k)}$, then the expected runtime of the GSHH with the random gradient mechanism on the UB problem is:*

$$\mathbb{E}\left[T_{\textit{GSHH-RG, UB}}\right] \leq (n-1)\left(1+\frac{m}{\sum_{j=1}^{m} q_j}(1-\frac{\sum_{j=1}^{m} q_j^2}{\sum_{j=1}^{m} q_j})\right) \tag{4.135}$$

Proof. We prove by induction that for all fitness level $i \in [n]$:

$$\mathbb{P}(S_j^{(i)} = 1) = \frac{q_j^{(i)}}{\sum_{k=1}^{m} q_k^{(i)}} \tag{4.136}$$

where $\mathbb{P}(S_j^{(i)} = 1)$ is the success probability of the j-th heuristic in fitness level i.

base case: when $i = 1$, the random gradient strategy chooses a low-level heuristic uniformly at random, the probability that the j-th heuristic improves the solution in fitness level 1 towards a higher fitness level is:

$$\mathbb{P}(S_j^{(1)} = 1) = \frac{q_j^{(1)}}{\sum_{k=1}^{m} q_k^{(1)}} \tag{4.137}$$

Induction step: Let R^i be a random variable that takes one when the selected heuristic fails to improve the candidate solution, and zero otherwise. Suppose that $i = z$, by the law of total probability:

$$\mathbb{P}(S_j^{(z)} = 1) = R^{(z)}(\frac{q_j^{(z)}}{\sum_{k=1}^{m} q_k^{(z)}}) + (1-R^{(z)})\Pr(S_j^{(z)} = 1) \tag{4.138}$$

$$= R^{(z)}(\frac{q_j^{(z)}}{\sum_{k=1}^{m} q_k^{(z)}}) + (1-R^{(z)})(\frac{q_j^{(z)}}{\sum_{k=1}^{m} q_k^{(z)}}) \tag{4.139}$$

$$= \frac{q_j^{(z)}}{\sum_{k=1}^{m} q_k^{(z)}}(R^{(z)} + 1 - R^{(z)}) \tag{4.140}$$

$$= \frac{q_j^{(z)}}{\sum_{k=1}^{m} q_k^{(z)}} \tag{4.141}$$

By induction, (4.136) is true for all $i \in [n]$. Thus by the law of the total expectation:

$$\mathbb{E}[R^{(i)}] = \sum_{j=1}^{m} \Pr(S_j^{(k)} = 1)\mathbb{E}[R^{(i)} \mid S_j^{(i)} = 1] \tag{4.142}$$

$$= \sum_{j=1}^{m} \frac{q_j^{(i)}}{\sum_{k=1}^{m} q_k^{(i)}}(1 - q_j^{(i)}) \tag{4.143}$$

$$= \frac{1}{\sum_{k=1}^{m} q_k^{(i)}} \left(\sum_{j=1}^{m} q_j^{(i)}(1 - q_j^{(i)}) \right) \tag{4.144}$$

Once the selected heuristic fails to improve the candidate solution, the random gradient mechanism chooses a heuristic uniformly at random. The expected waiting time to reach a higher fitness level by selecting a heuristic uniformly at random is:

$$\mathbb{E}[T_i] = \frac{1}{\frac{1}{m}\sum_{j=1}^{m} q_j^{(i)}} \tag{4.145}$$

Since the random gradient mechanism needs one iteration to examine the chosen heuristic, the expected runtime of random gradient hyper-heuristic is at most:

$$\mathbb{E}\left[T_{\text{GSHH-RG, UB}}\right] \leq \sum_{i=1}^{n-1} \left(1 + \mathbb{E}[R^{(i)}](\frac{m}{\sum_{j=1}^{m} q_j^{(i)}}) \right) \tag{4.146}$$

$$\leq (n-1)\left(1 + \frac{m}{\sum_{j=1}^{m} q_j}(1 - \frac{\sum_{j=1}^{m} q_j^2}{\sum_{j=1}^{m} q_j}) \right) \tag{4.147}$$

□

The above analysis shows general bounds on the expected runtime of the GSHH with the simple random, greedy, and random gradient on the UB problem. The results suggest that if the search space is stationary such that $\forall i,k \in [n]$ and $j \in [m]$, $q_j^{(i)} = q_j^{(k)}$, and the following

condition holds

$$\frac{1}{m} < \frac{\sum_{j=1}^{m} q_j^2}{\left(\sum_{j=1}^{m} q_j\right)^2} \tag{4.148}$$

then the GSHH with the random gradient mechanism has a better upper bound than the simple random strategy on the UB problem where

$$\mathbb{E}[T_{\text{GSHH-SR, UB}}] > \mathbb{E}[T_{\text{GSHH-RG, UB}}] \tag{4.149}$$

if

$$m\sum_{i=1}^{n-1} \frac{1}{\sum_{j=1}^{m} q_j^{(i)}} > (n-1)\left(1 + \frac{m}{\sum_{j=1}^{m} q_j}(1 - \frac{\sum_{j=1}^{m} q_j^2}{\sum_{j=1}^{m} q_j})\right) \tag{4.150}$$

$$m(n-1)\frac{1}{\sum_{j=1}^{m} q_j} > (n-1)\left(1 + \frac{m}{\sum_{j=1}^{m} q_j}(1 - \frac{\sum_{j=1}^{m} q_j^2}{\sum_{j=1}^{m} q_j})\right) \tag{4.151}$$

by simplifying the above equation, we have

$$\frac{1}{\sum_{j=1}^{m} q_j} - \frac{1}{m} > \frac{1}{\sum_{j=1}^{m} q_j} - \frac{\sum_{j=1}^{m} q_j^2}{\left(\sum_{j=1}^{m} q_j\right)^2} \tag{4.152}$$

which gives

$$\frac{1}{m} < \frac{\sum_{j=1}^{m} q_j^2}{\left(\sum_{j=1}^{m} q_j\right)^2} \tag{4.153}$$

To conclude, this section presented runtime analysis of the GSHH with the simple random, random gradient, and greedy mechanisms on the UB problem. We derived upper bounds on the expected runtime of the GSHH with these mechanisms. The results suggest that under certain conditions the GSHH with the random gradient mechanism has a better upper bound on the

expected runtime than the simple random strategy on the UB problem.

4.5 Limits to Learning in Reinforcement Learning Hyper-heuristics

Typical low-level heuristics may perform differently between different regions of the search space of a given problem. The goal of reinforcement learning mechanisms in hyper-heuristics is to identify the most suitable low-level heuristics to apply at a given point in time. This section investigates theoretically when, and why, the commonly used *additive* reinforcement learning mechanism fails to perform in a more effective manner than the simple random strategy. The pseudo-code of GSHH with the additive reinforcement learning mechanism is given in Algorithm 12. We use the GSHH with the simple random strategy as a baseline algorithm with which Algorithm 12 is compared as it chooses low-level heuristics uniformly at random.

Algorithm 12 Generic Selection Hyper-heuristic with the Additive Reinforcement Learning Mechanism (GSHH-RL)

Input:
- a finite search space $\mathscr{X}$
- an objective function $f : \mathscr{X} \to \mathbb{R}$
- a set of m low-level heuristics $H := \{h_1, \ldots, h_m\}$
- parameters $w_{\max} \in \mathbb{R}^+$ and $w_{\min} \in [0, w_{\max}[$.

Let $w_i^{(0)} := 1$ be an initial weight of low-level heuristic i, $\forall i \in [m]$.
Let $\alpha, \beta \in \mathbb{R}^+$ be the rewarding and punishing rates respectively.
Choose an initial solution $x_0 \in \mathscr{X}$.
for $t = 0, 1, \ldots$ until the termination condition is satisfied **do**
  Pick h_i from H with probability $p_i^{(t)} := \frac{w_i^{(t)}}{\sum_{j=1}^{m} w_j^{(t)}}$
  $x'_t := h_i(x_t)$ {apply the chosen heuristic h_i}
  $w_i^{(t+1)} := \begin{cases} \min\{w_i^{(t)} + \alpha, w_{\max}\} & \text{if} \quad f(x_t) > f(x'_t), \text{and} \\ \max\{w_i^{(t)} - \beta, w_{\min}\} & \text{otherwise.} \end{cases}$
  if $f(x_t) \geq f(x'_t)$ **then**
    $x_t := x'_t$
  end if
end for

At each time $t \in \mathbb{N}$, Algorithm 12 associates each low-level heuristic $i \in [m]$ with a positive weight $w_i^{(t)}$. Initially, all the weights are the same (i.e. $\forall i, j \in [m]$, $w_i^{(0)} = w_j^{(0)}$). The weight of

each low-level heuristic is restricted within a user-defined interval $[w_{\min}, w_{\max}]$, i.e.

$$w_i^{(t)} \in [w_{\min}, w_{\max}] \quad \forall t \in \mathbb{N} \text{ and } i \in [m]$$

The reason of restricting the weights of low-level heuristics within a predefined interval is discussed in Section 4.5.1. Algorithm 12 has to decide which heuristic to apply at time t, based on a distribution $\mathbf{p}^{(t)} := (p_1^{(t)}, \ldots, p_m^{(t)})$ (see line 6 of Algorithm 12). The distribution $\mathbf{p}^{(t)}$ is defined by a selection method based on the weights of low-level heuristics at time t. Algorithm 12 applies a weight update scheme to update the weights of low-level heuristics according to the feedback from the search environment. In this section, we are interested in analysing the behaviour of Algorithm 12 with the most frequently used weight update scheme called *additive* weight update scheme (see line 8 in Algorithm 12). The additive weight update scheme increases the weight of the chosen low-level heuristic by one if its application resulted in an improvement in the candidate solution with respect to a given objective function; otherwise, the weight is decreased by one (i.e. $\alpha := 1 =: \beta$ in Algorithm 12).

4.5.1 Impact the Bounds of the weights

A challenge for reinforcement learning mechanisms in hyper-heuristics is to explore the performance of the low-level heuristics with respect to the current search point, and exploit the the most effective low-level heuristics based on past observations. The additive reinforcement learning mechanism restricts the weight of each low-level heuristic within a predefined interval $[w_{\min}, w_{\max}]$. This allows the additive reinforcement learning mechanism to maintain a non-zero selection probability for each heuristic, where the bounds of the weight also imply bounds on the selection probabilities of low-level heuristics. These bounds, however, are user-defined parameters which may need tuning. Proposition 30 shows that if these bounds are constants

with respect to the number of low-level heuristics m (for example, m could be a function of the problem size), then the GSHH with the *additive* reinforcement learning mechanism assigns asymptotically, with respect to the number of low-level heuristics, the same selection probability to each low-level heuristic similar to the simple random strategy.

Proposition 30. *If $w_{\min}$ and $w_{\max}$ are constants with respect to m, then $\forall i \in [m]$ and $t \geq 0$ the GSHH-RL chooses low-level heuristic i with probability $p_i^{(t)} = \Theta\left(\frac{1}{m}\right)$.*

Proof. Let $p_i^{(t)} := \frac{w_i^{(t)}}{\sum_{i=1}^{m} w_j^{(t)}}$ be the probability of choosing heuristic i in iteration t of the GSHH-RL. Due to the fact that $w_{\min} \leq w_i^{(t)} \leq w_{\max}$ for all $i \in [m]$ and $t \geq 0$, the following inequality holds true.

$$\frac{w_{\min}}{w_{\min} + (m-1)w_{\max}} \leq p_i^{(t)} \leq \frac{w_{\max}}{w_{\max} + (m-1)w_{\min}} \tag{4.154}$$

which is equivalent to

$$\frac{1}{1 + (m-1)\frac{w_{\max}}{w_{\min}}} \leq p_i^{(t)} \leq \frac{1}{1 + (m-1)\frac{w_{\min}}{w_{\max}}} \tag{4.155}$$

If $w_{\min}$ and $w_{\max}$ are constant with respect to the number of low-level heuristics m (i.e. $\frac{w_{\max}}{w_{\min}} = O(1)$), then it follows for all $i \in [m]$ and $t \geq 0$ that

$$p_i^{(t)} = \Theta\left(\frac{1}{m}\right) \tag{4.156}$$

□

Proposition 30 indicates that if the bounds of the weight are constants with respect to the number of low-level heuristics m, then no selection probability is larger than (c/m) for some constant $c > 0$. Hence, in this scenario, the GSHH with both the additive reinforcement learning

mechanism and the simple random strategy assign asymptotically the same selection probability to each low-level heuristic.

4.5.2 Analysis of the Additive Weight Update Schemes

This section investigates theoretically the behaviour of the additive reinforcement learning mechanism. The following proposition shows that if the success probabilities of low-level heuristics are less than $1/2$, then $\forall t \in \mathbb{N}$, the probability that the total weights of low-level heuristics is much larger than the initial total weights in the t-th iteration of the GSHH-RL is small.

Proposition 31. *Let $W^{(t)} := \sum_{i=1}^{m} w_i^{(t)}$ be the total weights of low-level heuristics in the t-th iteration of Algorithm 12. For all $t \in \mathbb{N}$, $i \in [m]$, and $x_t \in \mathscr{X}$, let $q_i(x_t)$ be the success probability of low-level heuristic i in iteration t. If $\alpha = 1$, $\beta = 1$, and there exists a constant $\varepsilon \in]0,1]$ such that $q_i(x_t) \leq \frac{1}{2}(1-\varepsilon)$ for all $t \in \mathbb{N}$, $i \in [m]$, $x_t \in \mathscr{X}$ then for any positive integer k and every iteration $t \geq 0$*

$$\mathbb{P}\left(W^{(t)} \geq W^{(0)} + k\right) \leq \frac{3}{\varepsilon^2}(1-\varepsilon)^k \tag{4.157}$$

Proposition 31 holds true for the GSHH-RL with any move-acceptance strategy and selection method. It also indicates that if the success probability of a low-level heuristic is less than $\frac{1}{2}(1-\varepsilon)$ and the additive weight update scheme is used, then, with high probability, its weight is not much larger than the minimum weight.

Proof. Let $W^{(t)}$ be as defined above, and $q := \frac{1}{2}(1-\varepsilon)$. Let $(\mathscr{F}_t)_{t\in\mathbb{N}}$ be the filtration induced by the weights $w_1^{(t)}, \ldots, w_m^{(t)}$. We use Theorem 10 with $a := W^{(0)}$ and $b := W^{(0)} + k$ for some integer $k \geq 0$. The first condition D_1 in Theorem 10 is satisfied if the following equation holds

true for all $t \geq 0$:

$$\mathbb{E}[e^{\eta W^{(t+1)}} | W^{(t)} > W^{(0)} \wedge \mathscr{F}_t] \leq \rho e^{\eta W^{(t)}} \tag{4.158}$$

By the law of total expectation, the left-hand side quantity is at most

$$\mathbb{E}[e^{\eta W^{(t+1)}} | W^{(t)} > W^{(0)} \wedge \mathscr{F}_t] \leq q e^{\eta (W^{(t)} + \alpha)} + (1-q) e^{\eta (W^{(t)} - \beta)} \tag{4.159}$$

We now use the fact that $\alpha = 1 = \beta$,

$$\mathbb{E}[e^{\eta W^{(t+1)}} | W^{(t)} > W^{(0)} \wedge \mathscr{F}_t] = e^{\eta W^{(t)}} \left(q e^{\eta} + (1-q) e^{-\eta} \right) \tag{4.160}$$

From (4.160), if $q e^{\eta} + (1-q) e^{-\eta} \leq \rho < 1$ then condition D_1 in Theorem 10 is satisfied. We choose $\eta := \log(\frac{1}{1-\varepsilon})$, and get

$$q e^{\eta} + (1-q) e^{-\eta} = q \left(\frac{1}{1-\varepsilon} \right) + (1-q)(1-\varepsilon) \tag{4.161}$$

$$= \frac{1}{2} + \frac{1}{2}(1+\varepsilon)(1-\varepsilon) \tag{4.162}$$

$$= 1 - \frac{\varepsilon^2}{2} \tag{4.163}$$

Therefore, we choose $\rho := 1 - \frac{\varepsilon^2}{2}$. Condition D_2 in Theorem 10 is satisfied if the following inequality holds true.

$$D \geq \mathbb{E}[e^{\eta (W^{(t+1)} - W^{(t)})} | W^{(t)} \leq W^{(0)} \wedge \mathscr{F}_t] \tag{4.164}$$

By the law of total expectation,

$$\mathbb{E}[e^{\eta W^{(t+1)}-W^{(t)}}|W^{(t)} \leq W^{(0)} \wedge \mathscr{F}_t] \leq qe^{\eta} + (1-q)e^{0} \tag{4.165}$$

$$= \frac{1}{2} + \frac{1}{2}(1+\varepsilon) \tag{4.166}$$

$$= 1 + \frac{\varepsilon}{2} \tag{4.167}$$

We, therefore, choose $D := 1 + \frac{\varepsilon}{2}$. It now follows by Theorem 10 that

$$\mathbb{P}\left(W^{(t)} \geq W^{(0)} + k\right) \leq \rho^t e^{\eta\left(W^{(0)} - \left(W^{(0)}+k\right)\right)} + De^{\eta\left(W^{(0)} - \left(W^{(0)}+k\right)\right)} \frac{1-\rho^t}{1-\rho} \tag{4.168}$$

we now use the fact that $\eta = \log(\frac{1}{1-\varepsilon})$.

$$\mathbb{P}\left(W^{(t)} \geq W^{(0)} + k\right) \leq e^{-\eta k}\left(\rho^t + D\frac{1-\rho^t}{1-\rho}\right) \tag{4.169}$$

$$\leq (1-\varepsilon)^k\left(\frac{\rho^t}{1-\rho} + D\frac{1-\rho^t}{1-\rho}\right) \tag{4.170}$$

$$= (1-\varepsilon)^k\left(\frac{D + \rho^t(1-D)}{1-\rho}\right) \tag{4.171}$$

$$\leq (1-\varepsilon)^k\left(\frac{D}{1-\rho}\right) \tag{4.172}$$

By plugging in the values of D and ρ, we have

$$\mathbb{P}\left(W^{(t)} \geq W^{(0)} + k\right) \leq (1-\varepsilon)^k\left(\frac{1+\frac{\varepsilon}{2}}{\frac{\varepsilon^2}{2}}\right) \tag{4.173}$$

$$\leq (1-\varepsilon)^k\left(\frac{2+\varepsilon}{\varepsilon^2}\right) \tag{4.174}$$

$$\leq \frac{3}{\varepsilon^2}(1-\varepsilon)^k \tag{4.175}$$

which proves the claimed result. □

The following corollary shows that if the success probabilities of low-level heuristics are less than $\frac{1}{2}(1-\varepsilon)$, then the GSHH-RL, with high probability, chooses asymptotically low-level heuristics uniformly at random.

Corollary 32. *Suppose that* $w_{\min} = O(1)$*, and there exists a constant* $\varepsilon \in]0,1]$ *such that* $q_i(x_t) \leq \frac{1}{2}(1-\varepsilon)$ *for all* $t \in \mathbb{N}$*, and* $i \in [m]$*, then for any* $\delta > 0$ *,* $i \in [m]$*, and* $t \geq 0$*, with probability at least* $1 - \frac{3}{\varepsilon^2}(1-\varepsilon)^{m^\delta}$*,* $p_i^{(t)} = O\left(m^{-(1-\delta)}\right)$ *and* $\mathbb{E}[p_i^{(t)}] = O(1/m)$.

If the success probabilities of low-level heuristics are small (such that less than $1/2$), and the rewarding rate α is no larger than the punishing rate β, then the GSHH with both the additive reinforcement learning mechanism and the simple random strategy assign asymptotically the same selection probability to each low-level heuristic. Corollary 32 also shows that the probability that the GSHH-RL with the roulette wheel selection method, chooses roughly low-level heuristics with probability $\frac{1}{m}$ increases as the number of low-level heuristics m increases.

Proof. The probability that the GSHH-RL chooses low-level heuristic i in iteration t is

$$p_i^{(t)} = \frac{w_i^{(t)}}{\sum_{j=1}^{m} w_j^{(t)}} \leq \frac{w_i^{(t)}}{w_i^{(t)} + (m-1)w_{\min}} \tag{4.176}$$

Suppose, without loss of generality, that $W^{(t)} \leq W^{(0)} + k$ for some integer $k \geq 0$, then from the fact that $W^{(0)} = \sum_{i=1}^{m} w_i^{(0)} = m$, we have

$$\begin{aligned} w_i^{(t)} &\leq W^{(0)} + k - (m-1)w_{\min} \\ &= w_{\min} + k \end{aligned} \tag{4.177}$$

By Proposition 31, with probability at least $1 - \frac{3}{\varepsilon^2}(1-\varepsilon)^k$, the following holds.

$$p_i^{(t)} \leq \frac{w_{\min} + k}{k + m w_{\min}}$$

Let $k := m^{\delta}$ for any $\delta > 0$. By noting that $w_{\min}$ is a constant, then it follows that

$$p_i^{(t)} \leq \frac{\frac{w_{\min}}{k} + 1}{1 + \frac{mw_{\min}}{k}} = \frac{1}{m} + \frac{k}{mw_{\min}} = O\left(\frac{1}{m^{1-\delta}}\right) \tag{4.178}$$

which proves the first statement of the corollary. Now, it follows by the law of total expectation that the expected selection probability of the i-th low-level heuristic in iteration t of the GSHH-RL is bounded from above by

$$\mathbb{E}[p_i^{(t)}] \leq \sum_{k=0}^{\infty} \mathbb{P}\left(w_i^{(t)} \geq w_i^{(0)} + k\right) \frac{w_{\min} + k}{k + mw_{\min}} \tag{4.179}$$

From Proposition 31, we have $\mathbb{P}\left(w_i^{(t)} \geq w_i^{(0)} + k\right) \leq \frac{3}{\varepsilon^2}(1-\varepsilon)^k$, and hence

$$\mathbb{E}[p_i^{(t)}] \leq \sum_{k=0}^{\infty} \frac{3}{\varepsilon^2}(1-\varepsilon)^k \frac{\frac{w_{\min}}{k} + 1}{1 + \frac{mw_{\min}}{k}} \tag{4.180}$$

$$\leq 3\sum_{k=0}^{\infty} \frac{(1-\varepsilon)^k}{\varepsilon^2} \frac{w_{\min} + 1}{\frac{mw_{\min}}{k}} \tag{4.181}$$

$$\leq \frac{3(w_{\min} + 1)}{\varepsilon^2 mw_{\min}} \sum_{k=0}^{\infty} (1-\varepsilon)^k k \tag{4.182}$$

$$\leq \frac{3(w_{\min} + 1)}{\varepsilon^2 mw_{\min}} \frac{1-\varepsilon}{\varepsilon^2} \tag{4.183}$$

It now follows immediately by noting that $w_{\min}$ and ε are constants. □

The additive reinforcement learning mechanism with the *max* selection method chooses the low-level heuristic with the maximum weight. In the case of ties, a low-level heuristic among those heuristics that have the maximum weight is chosen uniformly at random. The following proposition shows a lower bound on the probability that the weight of a low-level heuristic is larger than the minimum weight when the additive weight update scheme is used with the *max* selection method.

Proposition 33. *Consider Algorithm 12 with* $w_{\min} := 0$, $\beta := 1 =: \alpha$, *and the max selection*

method. Let $q_i(x_t)$ be the success probability of low-level heuristic i in iteration t. Let q_i be a lower bound on the success probability of low-level heuristic i such that $q_i \leq q_i(x_t)$ for all $x_t \in \mathscr{X}$. Suppose that $w_i^{(t)} := c$ for some $c > w_{\min}$. Let A be the event that $w_i^{(t')} = c+k$ for any $t' > t$ and integer $k \geq 1$ before that $w_i^{(t')} = w_{\min}$.

$$\mathbb{P}(A) \geq \begin{cases} \frac{c}{c+k} & \text{if } q_i = \frac{1}{2} \\ \frac{(\frac{q_i}{1-q_i})^c - 1}{(\frac{q_i}{1-q_i})^{c+k} - 1} & \text{otherwise} \end{cases} \tag{4.184}$$

The progress in the weight of the chosen low-level heuristic i forms a simple random walk with the following distribution.

$$\mathbb{P}\left(w_i^{(t+1)} = w_i^{(t)} + 1\right) \geq q_i \quad \text{and} \quad \mathbb{P}\left(w_i^{(t+1)} = w_i^{(t)} - 1\right) \leq 1 - q_i \tag{4.185}$$

Suppose, for example, that the weight of the chosen low-level heuristic i in iteration t is $w_i^{(t)} := c+k-1$. Then, by the law of total probability, we have

$$\mathbb{P}\left(w_i^{(t+1)} = c+k | w_i^{(t)} = c+k-1\right) \geq q_i + (1-q_i)\mathbb{P}\left(w_i^{(t+2)} = c+k | w_i^{(t+1)} = c+k-2\right)$$

This scenario is similar to the well-known *gambler's ruin* problem (see Section 2.6.3). Proposition 33 is an application of Theorem 2.

Proof. Apply Theorem 2 with $p \geq q_i$, and $q \leq 1 - q_i$, and the proposition follows immediately.

□

4.5.3 Runtime Analysis

Here, we investigate the runtime of the GSHH with the additive reinforcement learning mechanism on the LEADINGONES problem. The reasons of using the LEADINGONES problem were

described in Section 4.2.1. We assume that there are m variation operators, but only the first variation operator has success probability larger than 0 and all the other variation operators have success probability 0. In such a simple search scenario, if the additive reinforcement learning mechanism is unable to identify the suitable heuristic, then it might not be able to identify the appropriate heuristics on more complicated problems. We compared the additive reinforcement learning mechanism with the simple random strategy as it chooses low-level heuristics uniformly at random.

The following theorem shows the runtime of the GSHH with the additive reinforcement learning mechanism and the roulette wheel selection method on the LEADINGONES problem.

Theorem 34. *The expected runtime of the GSHH-RL on the* LEADINGONES *problem is upper bounded by*

$$\mathbb{E}[T_{\textit{GSHH-RL, LO}}] \leq mn^2$$

and lower bounded by

$$\mathbb{E}[T_{\textit{GSHH-RL, LO}}] \geq \frac{(m+5)n(n-2)}{12}$$

Proof. Recall that $X_t := n - \text{LEADINGONES}(x_t)$ is the distance to the optimal solution in iteration t of the GSHH-RL. From Lemma 15, the expected progress in an improving step is at least 1 and at most 2. For all $t \in \mathbb{N}$ and $i \in [m]$, let $p_i^{(t)}$ be the selection probability of low-level heuristic i in iteration t of the GSHH-RL, and $q_i(x_t)$ be the success probability of low-level heuristic i in iteration t of the algorithm. We have $q_1(x_t) = \frac{1}{n}$ and $q_i(x_t) = 0$ for all $i \in \{2, \ldots, m\}$ and $x_t \in \mathscr{X}$.

Since $w_i^{(t)} = 1$ for all $i \in \{2, \ldots, m\}$ and $t \geq 0$, the following holds true for all $t \geq 0$.

$$p_1^{(t)} \geq \frac{1}{m} \tag{4.186}$$

By the law of total expectation, the expected progress in distance in a single step is

$$\mathbb{E}[X_t - X_{t+1} | X_t > 0] \geq \sum_{i=1}^{m} p_i^{(t)} q_i(x_t) \tag{4.187}$$

By plugging in the values of $p_i^{(t)}$ and $q_i(x_t)$ for all $i \in [m]$, we have

$$\mathbb{E}[X_t - X_{t+1} | X_t > 0] \geq \frac{1}{mn} =: \Delta_u \tag{4.188}$$

The distance from the initial solution is at most $n-1$ (i.e. $X_0 \leq n-1$). The expected runtime of the GSHH with the additive reinforcement learning mechanism is therefore by Theorem 9 upper bounded by

$$\mathbb{E}[T_{\text{GSHH-RL, LO}} | X_0] \leq \frac{n-1}{\Delta_u} \leq mn^2 \tag{4.189}$$

By the law of total probability, the selection probability of the first variation operator is

$$p_1^{(t)} \leq \mathbb{P}(w_1^{(t)} < w_{\min} + k) \frac{w_{\min} + k}{k + m w_{\min}} + \mathbb{P}(w_1^{(t)} \geq w_{\min} + k) \frac{w_{\max}}{w_{\max} + (m-1) w_{\min}} \tag{4.190}$$

By noting that $q_1^{(t)} \leq \frac{1}{2}(1-\varepsilon)$ if $1 - \varepsilon \geq \frac{2}{n}$, then it follows by Proposition 31 for any positive

integer k that

$$p_1^{(t)} \leq \left(1 - \frac{3}{\varepsilon^2}(1-\varepsilon)^k\right) \frac{w_{\min}+k}{k+mw_{\min}} + \frac{3}{\varepsilon^2}(1-\varepsilon)^k \frac{w_{\max}}{w_{\max}+(m-1)w_{\min}} \tag{4.191}$$

$$\leq \left(1 - \frac{3}{(1-\frac{2}{n})^2}\left(\frac{2}{n}\right)^k\right) \frac{w_{\min}+k}{k+mw_{\min}} + \frac{3}{(1-\frac{2}{n})^2}\left(\frac{2}{n}\right)^k \frac{w_{\max}}{w_{\max}+(m-1)w_{\min}} \tag{4.192}$$

If $k=5$, then the probability that $p_1^{(t)} < \frac{w_{\min}+k}{k+mw_{\min}}$ is $1 - \frac{3}{(1-\frac{2}{n})^2}\left(\frac{2}{n}\right)^2 \approx 1$. Hence, we have

$$p_1^{(t)} \leq \left(1 - \frac{3}{(1-\frac{2}{n})^2}\left(\frac{2}{n}\right)^5\right) \frac{w_{\min}+5}{5+mw_{\min}} + \frac{3}{(1-\frac{2}{n})^2}\left(\frac{2}{n}\right)^5 \frac{w_{\max}}{w_{\max}+(m-1)w_{\min}} \tag{4.193}$$

$$\approx \frac{w_{\min}+5}{5+mw_{\min}} \tag{4.194}$$

which gives

$$p_1^{(t)} \leq \frac{6}{5+m} \tag{4.195}$$

By the law of total expectation, the expected change in distance is at most

$$\mathbb{E}[X_t - X_{t+1} | X_t > 0] \leq 2\left(p_1^{(t)} q_1(x_t)\right) \leq \frac{12}{n(m+5)} =: \Delta_\ell \tag{4.196}$$

Finally, it follows by Theorem 9 that

$$\mathbb{E}[T_{\text{GSHH-RL, LO}} | X_0] \geq \frac{n-2}{\Delta_\ell} \geq \frac{m+5}{12} n(n-2) \tag{4.197}$$

□

The following theorem shows the expected runtime of the GSHH-RL with the max selection method on the LEADINGONES problem.

Theorem 35. *The expected runtime of the GSHH with the additive reinforcement learning mech-*

anism and the max selection method on the LEADINGONES *problem is upper bounded by*

$$\mathbb{E}[T_{GSHH\text{-}RL,\,LO}] \leq mn^2 \tag{4.198}$$

and lower bounded by

$$\mathbb{E}[T_{GSHH\text{-}RL,\,LO}] \geq \frac{m\,(n-2)^2\,(n-2)}{2(n+6m)} \tag{4.199}$$

Proof. In the worst case, the reinforcement learning mechanism with the max selection method chooses the first variation operator with probability $1/m$ as the weights of the other low-level heuristics are never larger than the lower bound of the weight. Recall that $X_t := n - \text{LEADINGONES}(x_t)$ is the distance to the optimal solution in iteration t of the GSHH-RL. Thus, by the law of total expectation, the expected progress in distance in a single step is

$$\mathbb{E}[X_t - X_{t+1} | X_t > 0] = \sum_{i=1}^{m} p_i^{(t)} q_i(x_t) \geq \frac{1}{mn} =: \Delta_\mu \tag{4.200}$$

Since $X_0 \leq n-1$, the expected runtime of the GSHH-RL with the max selection method is by Theorem 9 upper bounded as

$$\mathbb{E}[T_{\text{GSHH-RL, LO}}] \leq \frac{n-1}{\Delta_\mu} \leq mn^2 \tag{4.201}$$

The selection probability of first variation operator in iteration t of the GSHH-RL with the max selection method is

$$p_1^{(t)} = 1 \cdot \mathbb{P}\left(w_1^{(t)} \geq w_{\min} + 1\right) + \frac{1}{m}\mathbb{P}\left(w_1^{(t)} < w_{\min} + 1\right) \tag{4.202}$$

By noting that $\frac{1}{n} \leq \frac{1}{2}(1-\varepsilon)$ if $1-\varepsilon \geq \frac{2}{n}$, then it follows by Proposition 30 that

$$\mathbb{P}\left(w_1^{(t)} \geq w_{\min}+1\right) \leq \frac{3}{\varepsilon^2}(1-\varepsilon) \leq \frac{3}{\left(1-\frac{2}{n}\right)^2}\frac{2}{n} \tag{4.203}$$

Thus,

$$p_1^{(t)} \leq \frac{6}{n\left(1-\frac{2}{n}\right)^2} + \frac{1}{m}\left(1-\frac{6}{n\left(1-\frac{2}{n}\right)^2}\right) \tag{4.204}$$

$$\leq \frac{n^2+6mn-10n+4}{m(n-2)^2} \tag{4.205}$$

$$< \frac{n^2+6mn}{m(n-2)^2} \tag{4.206}$$

By Lemma 15 the expected progress in an improving step is at most 2. The expected drift of the GSHH-RL with the max selection method is therefore bounded from above by

$$\mathbb{E}[X_t - X_{t+1} | X_t > 0] \leq 2\left(p_1^{(t)} q_1(x_t)\right) \tag{4.207}$$

$$\leq \frac{2}{n}\left(\frac{n^2+6mn}{m(n-2)^2}\right) \tag{4.208}$$

$$\leq \frac{2(n+6m)}{m(n-2)^2} =: \Delta_\ell \tag{4.209}$$

It now follows by Theorem 9 that

$$\mathbb{E}[T_{\text{GSHH-RL, LO}}] \geq \frac{n-2}{\Delta_\ell} \geq \frac{m(n-2)^2(n-2)}{2(n+6m)} \tag{4.210}$$

which concludes the proof. □

The following theorem shows the runtime of the GSHH with the simple random strategy with the prescribed settings above on the LEADINGONES problem.

Theorem 36. *The expected runtime of the GSHH with the simple random strategy on the*

LEADINGONES *problem is upper bounded as*

$$\mathbb{E}[T_{GSHH-SR,LO}] \leq mn^2 \tag{4.211}$$

and lower bounded as

$$\mathbb{E}[T_{GSHH-SR,LO}] \geq \frac{mn(n-2)}{2} \tag{4.212}$$

Proof. For all $i \in [m]$ and $t \in \mathbb{N}$, let $p_i^{(t)} := \frac{1}{m}$ be the selection probability of low-level heuristic i in iteration t of the GSHH with the simple random strategy. Let $X_t := n - \text{LEADINGONES}(x_t)$ be the distance to the optimal solution in iteration t. By the law of total expectation, the expected progress in distance in a single step of the GSHH with the simple random strategy is

$$\mathbb{E}[X_t - X_{t+1} | X_t > 0] \geq \sum_{i=1}^{m} p_i^{(t)} q_i(x_t) \geq \frac{1}{nm} \tag{4.213}$$

By Lemma 15, the expected progress in an improving step is at least 1 and at most 2. The expected runtime of the GSHH with the simple random strategy is therefore by Theorem 9 bounded from above by

$$\mathbb{E}[T_{\text{GSHH-SR, LO}} | X_0] \leq \frac{n-1}{1/mn} \leq mn^2 \tag{4.214}$$

and lower bounded by

$$\mathbb{E}[T_{\text{GSHH-SR, LO}} | X_0] \geq \frac{mn(n-2)}{2} \tag{4.215}$$

□

The above analyses show that the runtime of the GSHH with the simple random strategy and the additive reinforcement learning mechanism with both the roulette wheel and max selection methods are asymptotically the same. The results show that if the number of low-level heuristics is a constant, with respect to n, then the expected runtime of the GSHH with both mechanisms is $\Theta\left(n^2\right)$; otherwise, it is $\Theta\left(mn^2\right)$.

4.6 Experimental Supplements

A group of experiments were conducted to investigate the performance of the GSHH with the additive reinforcement learning mechanism and the simple random strategy on well-known combinatorial optimisation problems: bin packing, permutation flow-shop, and the travelling salesman problem. We evaluated the performance of the GSHH in terms of the qualities of results within a fixed time on the problems noted above. We also evaluated the runtime of the GSHH with the considered heuristic selection methods on the LEADINGONES problem. The purpose of the experiments is not only to shed light on the hidden constants in the theoretical results but also to verify whether assuming that the success probabilities of typical low-level heuristics are less than $(1/2)$ is a reasonable assumption where we analysed theoretically the performance of the additive reinforcement learning mechanism based on this assumption in Section 4.5. The experiments were conducted by the author using the HyFlex framework (Ochoa et al., 2012a). The reason of using HyFlex is that it is a well-known benchmark in the field of hyper-heuristics as discussed in Section 3.6. The experimental results were summarised in the following sections.

4.6.1 Results on the LEADINGONES Problem

We applied the GSHH with the simple random, random permutation, greedy, and random gradient mechanisms to the LEADINGONES problem with the described settings in Section 4.4.1. A run is continued until the optimal solution is found (the 1^n bit-string). Various lengths of the bit-

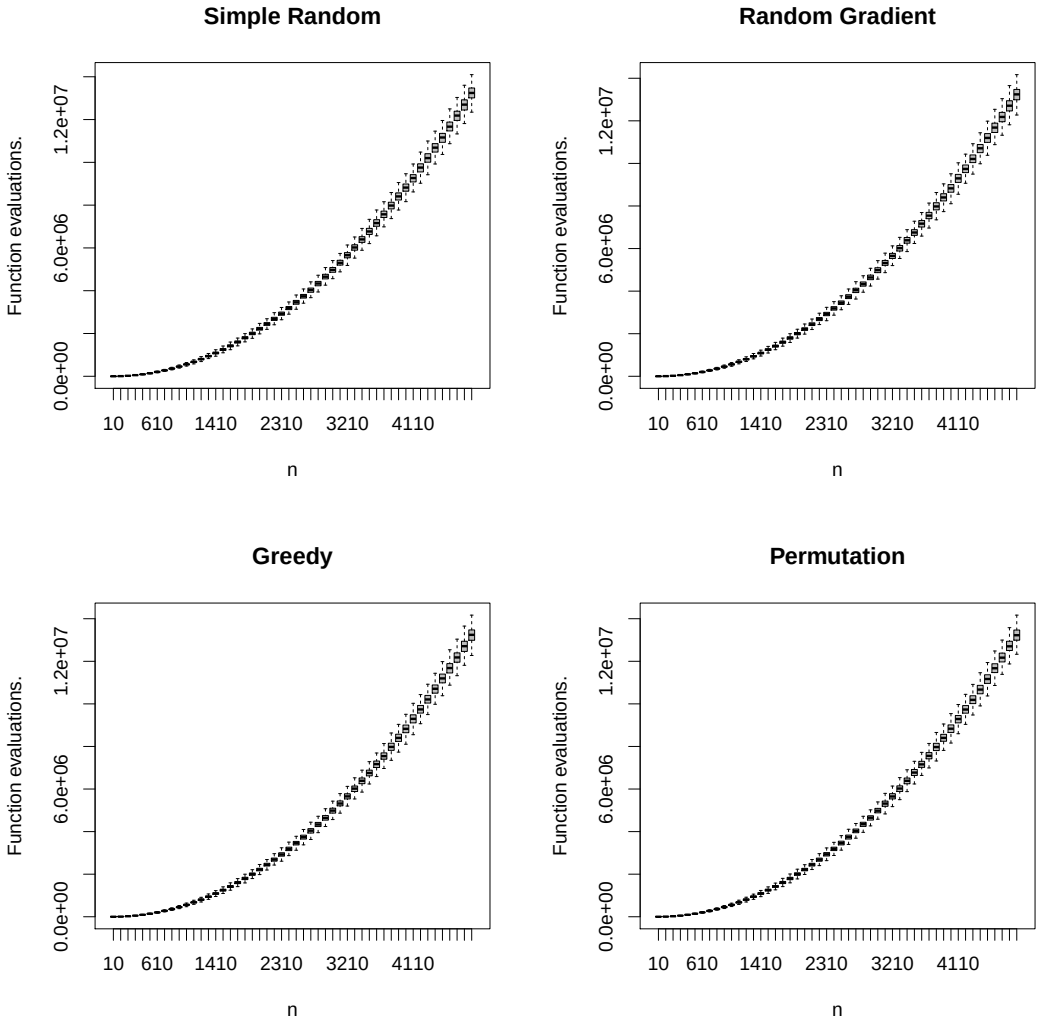

Figure 4.2: Runtime of the GSHH with the simple random, random permutation, greedy, and random gradient strategies on the LEADINGONES problem.

string were considered. A single run was repeated 1000 times. The considered algorithms were run simultaneously on the same initial randomly generated solution. The results are summarised in the following figures.

Figure 4.2 shows the runtime of the GSHH with the simple random, random permutation, greedy, and random gradient mechanisms on the LEADINGONES problem. The results show that the runtime of the GSHH with the heuristic selection strategies noted above is almost identical on the LEADINGONES problem with the described settings in Section 4.4.1. These empirical results are consistent with the theoretical results where we have shown that the expected runtime

of the GSHH with the above heuristic selection strategies is almost the same. This suggests that the learning schemes in the random gradient and greedy mechanisms do not improve the run-time of the GSHH on the example problem in comparison to the no-learning heuristic selection strategies.

4.6.2 Results on NP-hard Problems

This section empirically evaluates the performance of the GSHH, in terms of the qualities of the results, with the additive reinforcement learning mechanism, comparing it to the simple random strategy on several instances of the bin packing, permutation flow-shop scheduling, and travelling salesman problems.

As mentioned in Section 2.7, HyFlex provides implementations of a set of well-known combinatorial optimisation problems, including: one denominational bin packing (BP), permutation flow-shop (PF), and the travelling salesman problem (TSP). HyFlex also provides a set of low-level heuristics for each problem domain, classified under four categories: (1) mutation, (2) hill-climbing, (3) ruin and re-create, and (4) crossover low-level heuristics. As this current study focuses on a single-point search based hyper-heuristic framework, the crossover low-level heuristics were excluded from the experiments, where they required at least two solutions as input. In HyFlex, all of the low-level heuristics (apart from crossover low-level heuristics), are associated with a problem-dependent parameter that can be adjusted by the user. In this experiment, we were interested in comparing the performance of the GSHH with the additive reinforcement learning mechanism and the simple random strategy. Therefore, the values of problem-dependent parameters in the low-level heuristics were set as specified by HyFlex. We compared the additive reinforcement learning mechanism with the simple random strategy on six instances per problem domain: BP, PF, and TSP. The bounds of the weight in the additive reinforcement learning mechanism were set so that $w_{\min} := 1$ and $w_{\max} := 40$. Each hyper-

heuristic was allowed to evaluate the objective function 2×10^6 times. A single run was repeated 120 times, each run with a different seed generated uniformly at random. All the HyFlex problems are minimisation problems, such that the objective of hyper-heuristics is to minimise the function value of a given objective function. The results are presented in the following figures.

Figure 4.3 shows the objective function values of the GSHH with the simple random (SR) strategy, the additive reinforcement learning mechanism with both the roulette wheel (RL) and the max (RLM) selection methods on six instances of the bin packing problem. Overall, the results have shown that the simple random and the additive reinforcement learning mechanism, with both roulette wheel and max selection methods, have roughly the same performance on all instances. The median objective function value of the GSHH with both RL and RLM mechanisms were slightly better than that with the SR strategy on instance 1. On the other hand, the GSHH with the SR strategy achieved slightly better median objective function values on instance 2 and instance 6. Comparable results were obtained when we applied the aforementioned hyper-heuristics to six instances of the permutation flow-shop scheduling problem as shown in Figure 4.4. Figure 4.5 confirms the performance similarity between the additive reinforcement learning mechanisms and the simple random strategy on six instances of the travelling salesman problem.

4.6.3 An Analysis of Additive Reinforcement Learning Mechanisms

In Section 4.5, we investigated theoretically the behaviour of the additive reinforcement learning mechanism based on the assumption that the success probabilities of the low-level heuristics are less than $1/2$. Here, we experimentally estimate the success probabilities of typical low-level heuristics on the considered optimisation problems: bin packing, permutation flow-shop scheduling, and travelling salesman problem. The goal of this current experiment is to verify whether the above assumption is reasonable, as well as to evaluate the behaviour of the additive

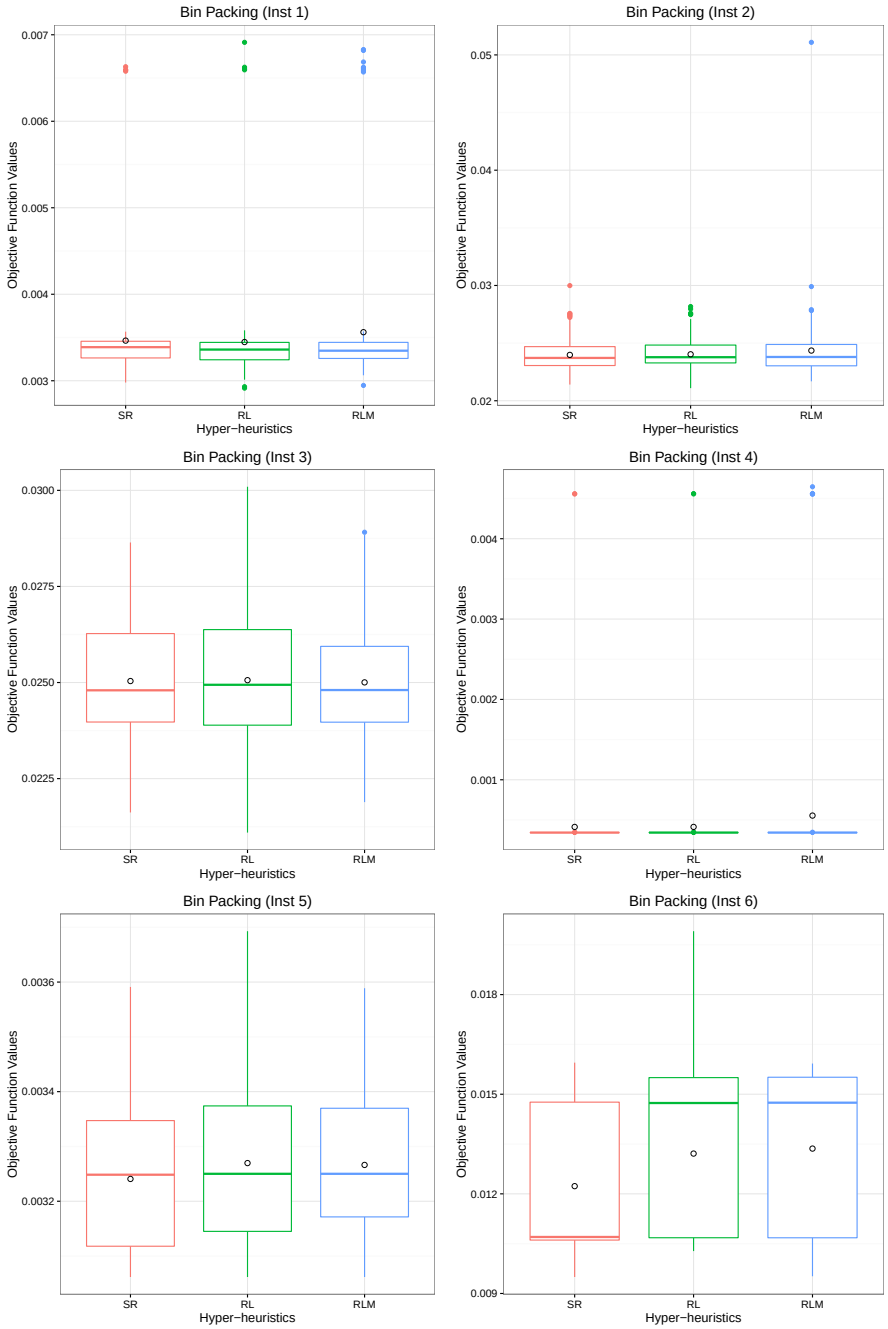

Figure 4.3: The objective function values of the GSHH with the simple random strategy (SR), the additive reinforcement learning mechanism with the roulette wheel selection method (RL), and the additive reinforcement learning mechanism with the max selection method (RLM) on six instances of the *bin packing* (BP) problem. The small circle shows the mean.

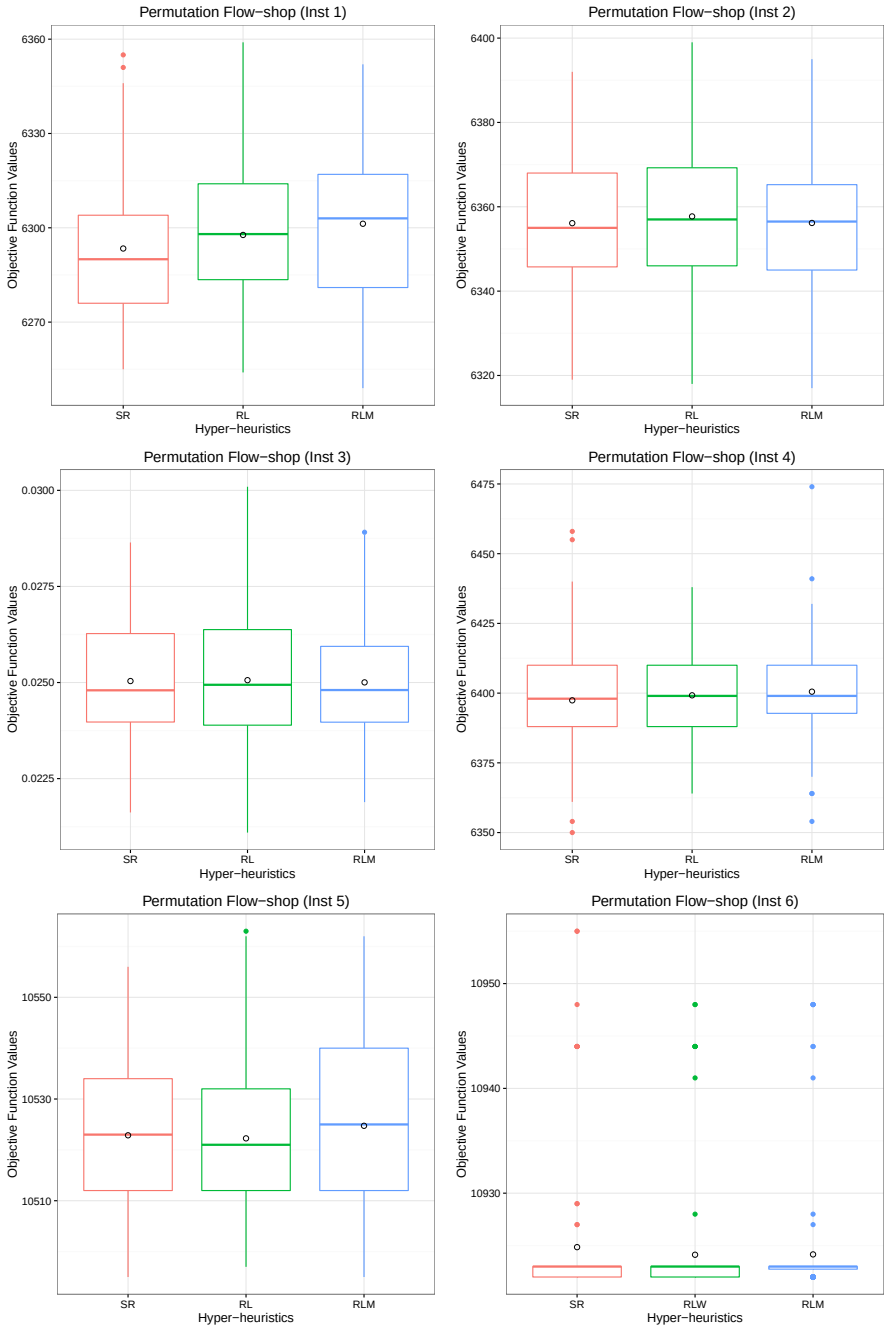

Figure 4.4: The objective function values of the GSHH with the simple random strategy (SR), the additive reinforcement learning mechanism with the roulette wheel selection method (RL), and the additive reinforcement learning mechanism with the max selection method (RLM) on six instances of the *permutation flow-shop* (PF) problem. The small circle shows the mean.

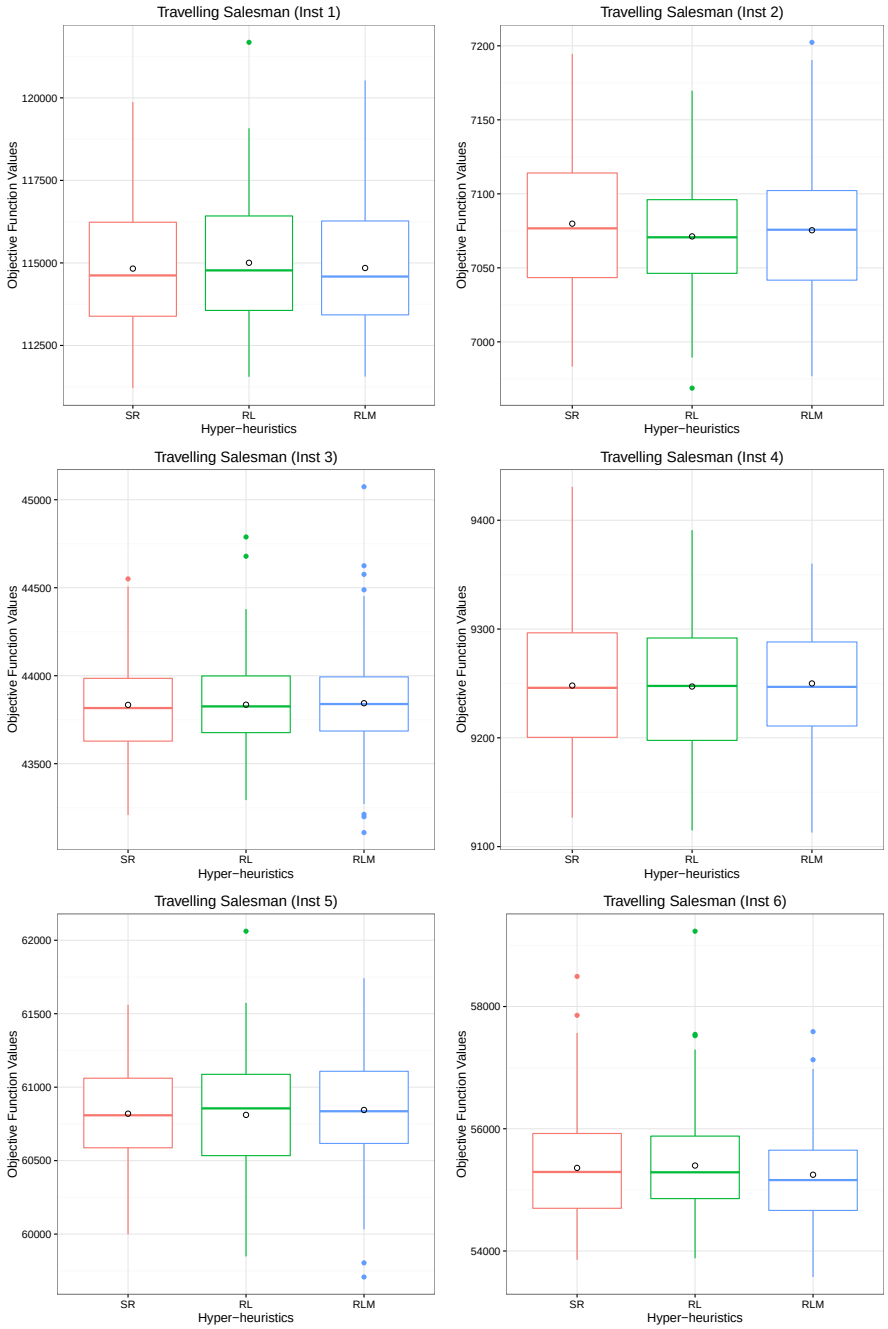

Figure 4.5: The objective function values of the GSHH with the simple random strategy (SR), the additive reinforcement learning mechanism with the roulette wheel selection method (RL), and the additive reinforcement learning mechanism with the max selection method (RLM) on six instances of the *travelling salesman problem* (TSP). The small circle shows the mean.

reinforcement learning mechanism.

We estimated the success probabilities of the available low-level heuristics in HyFlex for the problems noted above. As discussed in Section 2.6.1, the success probability of a low-level heuristic can be estimated as the number of times the low-level heuristic generated strictly better solutions, with respect to a given objective function, proportional to the number of times it has been invoked. It is plausible that for any two search points x and y, if the distance of the search point x to the optima is less than the distance of the search point y, then typical low-level heuristics have success probabilities on the search point x less than that on the search point y, such that the success probabilities of typical low-level heuristics decrease as the distance towards the optima decreases. Therefore, we divided the entire run period into several sub-periods of different lengths and estimated the success probabilities of low-level heuristics in each sub-period. Particularly, we estimated the success probabilities of low-level heuristics in the first 10^2, 10^3, 10^4 iterations, and then in every 10^5 iterations until the termination condition is satisfied. The length of the run is 2×10^6. Each run was repeated 120 times. In each iteration, all of the available low-level heuristics were applied to the same candidate solution. Then the best generated solution is accepted if it is not worse than the current candidate solution. We estimated the success probabilities of the available low-level heuristics in HyFlex on the first instance of the aforementioned problems: bin packing, permutation flow-shop scheduling, and travelling salesman problem. The results are presented in the following figures.

Figure 4.6 shows the estimated success probabilities of the low-level heuristics in the $10^2, 10^3, 10^4$ and 10^5 iterations on instance 1 of the bin packing problem. The results show that the estimated success probabilities of the most successful low-level heuristics are close to $1/2$ in the first 100 iterations, and then these probabilities decrease as the search progresses. The results also show that success probabilities of all low-level heuristics after the first 10^5 iterations are very small (i.e. at most $1.00E-04$). The estimated success probabilities of the low-level heuristics on the

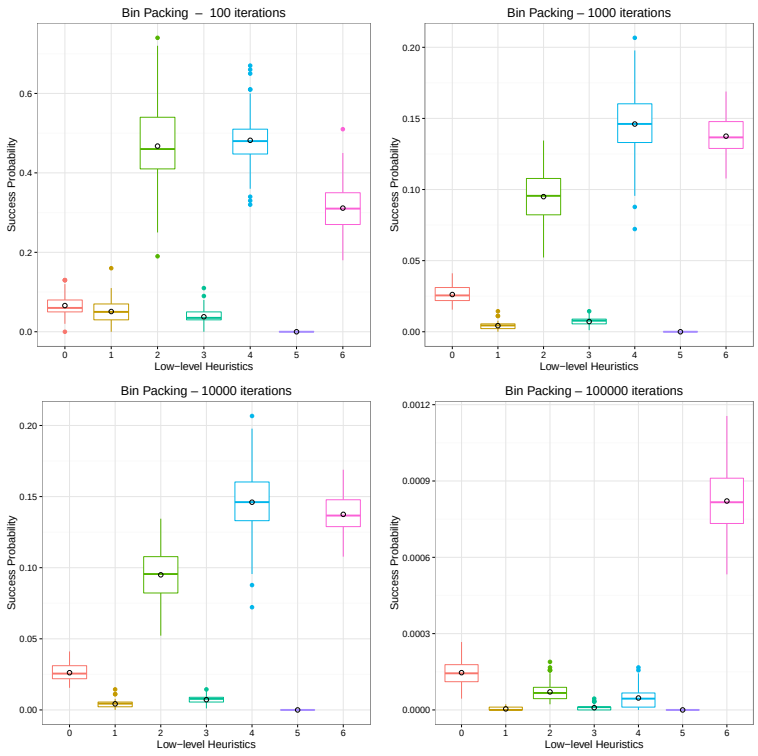

Figure 4.6: Success probabilities of low-level heuristics on instance 1 of the bin-packing problem estimated over 120 runs.

first instance of the permutation flow-shop problem are much smaller than $1/2$ in the first 100 iterations as shown Figure 4.7. Similarly, the estimated success probabilities of the low-level heuristics on instance 1 of the travelling salesman problem are also small as shown in Figure 4.8.

The experimental results are consistent with our theoretical results in Section 4.5. We have shown that if the success probabilities of low-level heuristics are smaller than $\frac{1}{2}(1-\varepsilon)$, then the additive reinforcement learning mechanism chooses asymptotically low-level heuristics uniformly at random the same as the simple random strategy. Since the success probabilities of low-level heuristics in the considered problems (as typical low-level heuristics) are small as shown in the above figures, Figure 4.9 shows that the number of executions of each low-level heuristic is almost the same with the additive reinforcement learning mechanism. Additionally, we tracked the trajectory of the objective function values of the GSHH with the additive rein-

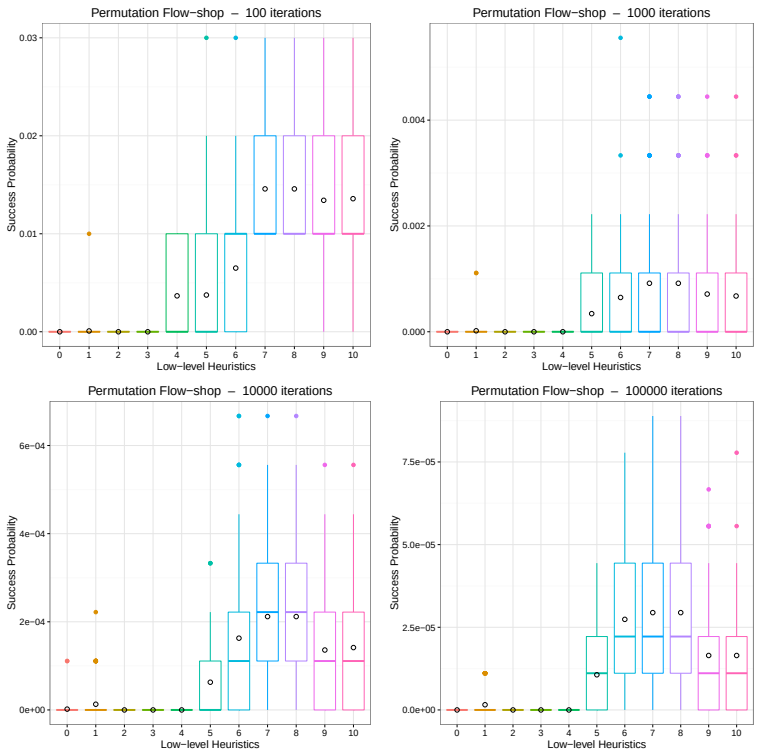

Figure 4.7: Success probabilities of low-level heuristics on instance 1 of the permutation flow-shop problem estimated over 120 runs.

forcement learning mechanisms and the simple random strategy on the aforementioned problems. Figure 4.10 shows the trajectory of the objective function values of the GSHH with both mechanisms on instance 1 of the bin packing problem. The results indicated that the additive reinforcement learning mechanism and simple random strategy have almost identical trajectories.

4.7 Summary of Results

This chapter has presented the first studies investigating the performance of a hyper-heuristic with random gradient, greedy, and additive reinforcement learning mechanisms theoretically. It also has proposed a benchmark problem that allows evaluating the performance of reinforcement learning hyper-heuristics in a controlled environment. We investigated whether the random gradient, greedy, and additive reinforcement learning mechanisms can improve the expected

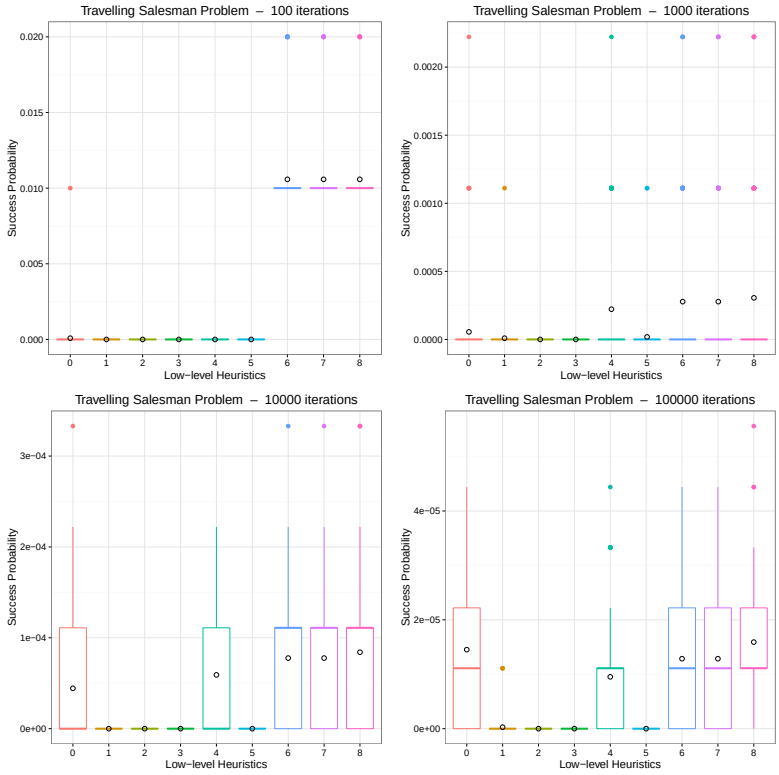

Figure 4.8: Success probabilities of low-level heuristics on instance 1 of the travelling salesman problem estimated over 120 runs.

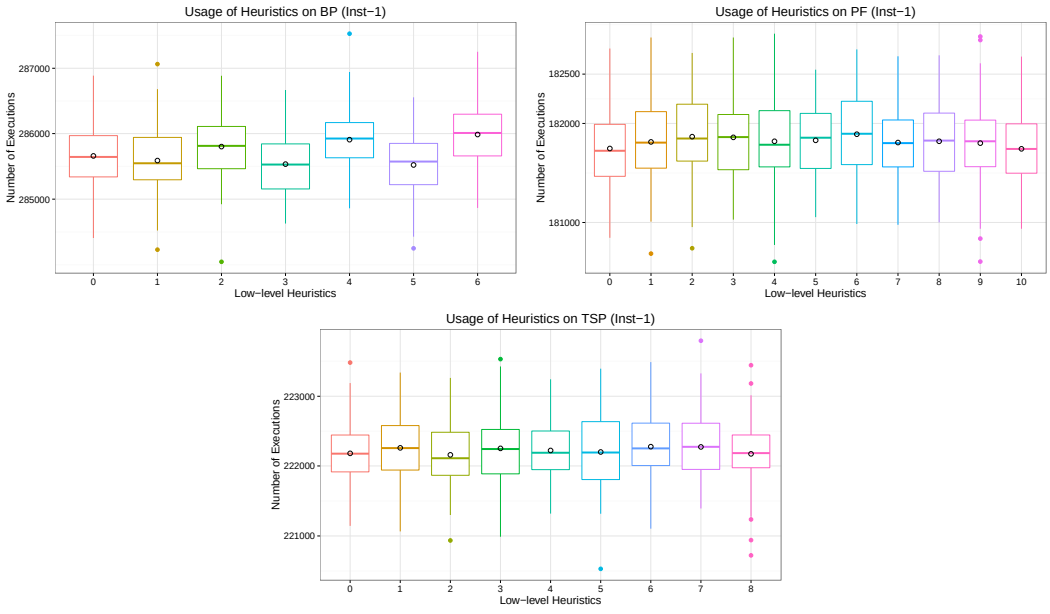

Figure 4.9: Usage of low-level heuristics with the GSHH with the additive reinforcement learning mechanism and max selection method in instance 1 of the bin packing, permutation flowshop, and the travelling salesman problem.

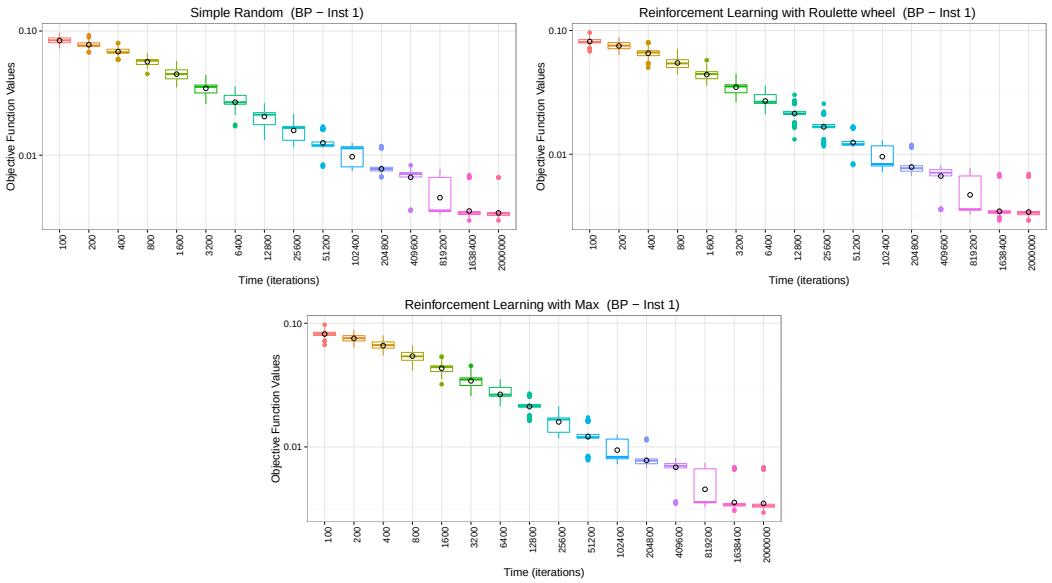

Figure 4.10: The trajectory of the objective function values of the GSHH with the simple random and the additive reinforcement learning mechanism with both roulette wheel and max selection methods on instance 1 of the bin packing problem.

runtime of a hyper-heuristic with the simple random strategy which chooses low-level heuristics uniformly at random on a benchmark problem. Additionally, We investigated theoretically when the additive reinforcement learning mechanism fails to perform in a more efficient manner than the simple random strategy.

The results have shown that the expected runtime of a hyper-heuristic with the greedy, random gradient, and simple random mechanisms are almost the same on a test problem the so-called LEADINGONES problem. The results rigorously demonstrated that the random gradient and greedy mechanisms do not necessarily improve the performance when compared to the simple random strategy. Additionally, we have shown two conditions under each one of them the additive reinforcement learning mechanism chooses asymptotically low-level heuristics uniformly at random similar to the simple random strategy: (1) if the probability of generating a strictly better solution with respect to a given objective function is less than $1/2$; (2) if the bounds of the weight are constant with respect to the number of low-level heuristics. We analysed the expected runtime of a hyper-heuristic with the additive reinforcement learning mechanism and the

simple random strategy on the LEADINGONES problem. The results showed that the expected runtime of the considered hyper-heuristic with both mechanisms is asymptotically the same. We conducted experiments on well-known combinational optimisation problems: bin packing, permutation flow-shop scheduling, and the travelling salesman problem. The experimental results have shown that a hyper-heuristic with both the additive reinforcement learning mechanism and the simple random strategy have roughly the same performance on several instances of these problems. We also estimated the probability of generating a strictly better solution with respect to a given objective function on the problems mentioned above. The results showed that, in most of the time, the estimated probability of generating a strictly better solution with respect to a given objective function were much less than $1/2$.

This chapter suggests that, under a mild assumption, the additive reinforcement learning mechanism is not necessarily capable of distinguishing between the performances of low-level heuristics, and asymptotically chooses heuristics uniformly at random the same as the simple random strategy.

Chapter 5

Adaptive Reinforcement Learning Mechanisms for Hyper-heuristics

This chapter introduces three reinforcement learning hyper-heuristics, as well as presents evaluations of their performances on a range of optimisation problems.

5.1 Introduction

Thompson Sampling is a reinforcement learning mechanism, which was initially proposed by Thompson in 1933 (Thompson, 1933). However, it almost disappeared from the literature of artificial intelligent. Recently, several studies have reported impressive success of Thompson Sampling on real-world problems (Chapelle and Li, 2011) (Graepel et al., 2010) (Tang et al., 2013). As discussed in Section 3.5.1, Thompson Sampling is theoretically an optimal learning mechanism for handling the exploration versus exploitation dilemma on a *stationary* multi-armed bandit problem (Kaufmann et al., 2012) (Agrawal and Goyal, 2012). This chapter adapts Thompson Sampling and introduced it to the field of hyper-heuristics. The proposed Thompson Sampling mechanism utilises a sliding time-window to adapt its behaviour according to the

recent past feedback from the search environment.

We investigated the efficiency of Thompson Sampling with a large number of low-level heuristics. The advantages of mixing a large number of different low-level heuristics are manifold. Using a large number of different low-level heuristics can increase the probability that any non-optimal search point can be improved by at least one of the existing low-level heuristics. This, in turn, could lead to high-quality solutions (Remde et al., 2012). Hyper-heuristics with a large number of different low-level heuristics are also applicable to those problems where instances differ significantly. We introduced a variant of the proposed Thompson Sampling hyper-heuristic that applies two Thompson Sampling mechanisms at the same time to classify/manage a large set of low-level heuristics for solving a given optimisation problem (Section 5.2.2).

This chapter also contributes a modification of the additive reinforcement learning mechanism. We have shown in the previous chapter that if the probability of generating a strictly better solution, with respect to a given objective function, is small (i.e. less than $1/2$), then the commonly used additive reinforcement learning mechanism chooses asymptotically low-level heuristics uniformly at random. Following our theoretical analysis, Section 5.2.3 introduces a reinforcement learning hyper-heuristic that adapts the reinforcement values according to the probability of generating a strictly better solution while solving a given problem.

5.2 Proposed Algorithms

This section presents the main algorithmic components and techniques of the proposed hyper-heuristics. The framework of the proposed hyper-heuristics is similar to the framework described in Section 2.5 which consists of a finite search space $\mathscr{X}$, an objective function $f : \mathscr{X} \rightarrow \mathbb{R}$, a set $H := \{h_1, \ldots, h_m\}$ of m low-level heuristics, and two high-level strategies (i.e. heuristic selection and move-acceptance). The proposed hyper-heuristics evolve an initially generated

solution $x \in \mathscr{X}$ by selecting a heuristic(s) from a set of predefined low-level heuristics H. Then the chosen low-level heuristic(s) is applied to the current candidate solution. The newly generated solution is evaluated, and a move-acceptance strategy is then used to decide whether the new solution should replace the candidate solution or it should be rejected.

5.2.1 Adaptive Thompson Sampling for Hyper-heuristics

This section introduces an adaptive Thompson Sampling hyper-heuristic. Thompson Sampling is a reinforcement learning mechanism that chooses each heuristic with probability matching its probability of being the most suitable heuristic based on all past observations (Thompson, 1933). We propose a Thompson Sampling mechanism that utilises a sliding time-window to only maintain the recent observations about the performance of the low-level heuristics. This enables past, and potentially irrelevant, observations about the performance of low-level heuristics to be discarded which, in turn, emphasises the importance of recent observations on the estimation of the effectiveness of low-level heuristics with respect to the current search point. A formal presentation of the proposed hyper-heuristic in the form of a pseudo-code is given in Algorithm 13.

Let $\alpha_i^{(t)}$ and $\beta_i^{(t)}$ be the number of successes and failures, respectively, observed from low-level heuristic i, in terms of generating strictly better solutions with respect to a given objective function, within a time-window at the t-th iteration of Algorithm 13. The proposed Thompson Sampling mechanism associates each low-level heuristic i with a utility score $U_i^{(t)}$ which is a random variable, in iteration t of Algorithm 13, sampled from a Beta distribution with parameters $\alpha_i^{(t)}$ and $\beta_i^{(t)}$. As discussed in Section 3.5.1, the reason of using Beta distribution is that it makes it easier to maintain the success distributions of the low-level heuristics, where if low-level heuristic i is applied in iteration t and its application led to an improvement in the candidate solution, with respect to a given objective function, then the success distribution of the low-level

Algorithm 13 Thompson Sampling Hyper-heuristic (TSHH)

Input:
- a finite search space $\mathscr{X}$
- an objective function $f : \mathscr{X} \to \mathbb{R}$ to be minimised
- a set $H := \{h_1, \ldots, h_m\}$ of m mutation and local search low-level heuristics. Let $MU \subset H$ be a subset of mutation low-level heuristics, and $LS \subset H$ be a subset of local search low-level heuristics.

Parameter: an integer $w \in \mathbb{N}^+$ {size of the sliding time-window}
$\forall i \in [m]$, set $\alpha_i^{(0)} := 1$, and $\beta_i^{(0)} := 1$.
$\forall i \in [m]$, let $U_i^{(0)}$ be the utility score of low-level heuristic i.
Choose an initial solution $x \in \mathscr{X}$.
for $t = 0, 1, \ldots$ until the termination condition is satisfied **do**
  Sample $U_i^{(t)} \sim \text{Beta}(\alpha_i^{(t)}, \beta_i^{(t)})$, $\forall i \in [m]$
  Let $I := \arg\max_{i \in MU} \{U_i^{(t)}\}$ {select a mutation heuristic with the maximum utility score}
  Let $J := \arg\max_{j \in LS} \{U_j^{(t)}\}$ {select a local search heuristic with the maximum utility score}
  $x' := h_I(x)$ {apply the chosen mutation heuristic h_I}
  $x'' := h_J(x')$ {apply the chosen local search heuristic h_J}
  $\forall k \in [m]$, let $\eta_k^{(t)} := \begin{cases} 1 & \text{if } k \in \{I, J\} \text{ and } f(x) > f(x'') \\ 0 & \text{otherwise} \end{cases}$
  $\forall k \in [m]$, let $\ell_k^{(t)} := \begin{cases} 1 & \text{if } k \in \{I, J\} \text{ and } f(x) \leq f(x'') \\ 0 & \text{otherwise} \end{cases}$
  $\forall i \in [m]$, let $\alpha_i^{(t+1)} := \alpha_i^{(t)} + \eta_i^{(t)}$
  $\forall i \in [m]$, let $\beta_i^{(t+1)} := \beta_i^{(t)} + \ell_i^{(t)}$
  if $t \geq w$ **then**
    $\forall i \in [m]$, let $\alpha_i^{(t+1)} := \alpha_i^{(t+1)} - \eta_i^{(t-w)}$
    $\forall i \in [m]$, let $\beta_i^{(t+1)} := \beta_i^{(t+1)} - \ell_i^{(t-w)}$
  end if
  if $f(x) \geq f(x'')$ **then**
    $x := x''$
  end if
end for

heuristic i will be $\text{Beta}(\alpha_i^{(t)} + 1, \beta_i^{(t)})$ distribution; otherwise, it will be $\text{Beta}(\alpha_i^{(t)}, \beta_i^{(t)} + 1)$.

The proposed hyper-heuristic evolves an initially generated solution $x \in \mathscr{X}$ by (i) choosing a heuristic from a set of mutation low-level heuristics MU and a heuristic from a set of local search low-level heuristics LS; (ii) applying the chosen heuristics successively to the current candidate solution, thus generating a new solution $x'' \in \mathscr{X}$; (iii) evaluating the newly generated solution x'' and deciding whether it should be accepted or discarded. In the t-th iteration of Algorithm 13, the utility score of low-level heuristic i, for all $i \in [m]$, is sampled from a

Beta($\alpha_i^{(t)}, \beta_i^{(t)}$) distribution, and the heuristic with the maximum utility score, in each subset of the low-level heuristics, is then chosen (see lines $7-9$ of Algorithm 13). The parameters $\alpha_i^{(t)}$ and $\beta_i^{(t)}$ are initialised to one and updated during the search process based on the performance of low-level heuristics (see lines $12-15$ in Algorithm 13). Usually, the performance of low-level heuristics are unknown beforehand, and hence we initialise these parameters ($\alpha_i^{(t)}$ and $\beta_i^{(t)}$) to one where Beta$(1,1)$ has the same distribution as Uniform$(0,1)$. The sliding time-window is implemented by the "first-in/first-out" principle. Each slot in the time window contains the index of the selected low-level heuristic, and whether it generated a better solution with respect to a given objective function. The proposed hyper-heuristic only maintains the observations about the low-level heuristics in the last w iterations where w is the size of the window (see lines $16-19$ in Algorithm 13). The size of the window w is a parameter of the algorithm. If w is too large, the algorithm may estimate the effectiveness of the low-level heuristics based on irrelevant observations about their current performances, and if w is too small, the algorithm may forget too quickly the observations about the performances of the low-level heuristics. Algorithm 13 accepts the newly generated solution if (and only if) it is not worse than the current candidate solution with respect to a given objective function (see lines $20-22$ in Algorithm 13).

5.2.2 Hierarchical Thompson Sampling Hyper-heuristic with a Large Number of Low-level Heuristics

This section introduces a new variant of the adaptive Thompson Sampling hyper-heuristic for tackling optimisation problems with a large number of low-level heuristics. In this scenario where the number of low-level heuristics is very large, usually a subset of low-level heuristics do not contribute much to the search progress (see, for example, (Remde et al., 2007)). We introduce a Thompson Sampling hyper-heuristic that partitions the set of low-level heuristics into a number of subsets, and applies two Thompson Sampling mechanisms to decide which

low-level heuristic should be applied to a given search point. The pseudo-code of the proposed hyper-heuristic is given in Algorithm 14.

Algorithm 14 Hierarchical Thompson Sampling Hyper-heuristic (HTSHH)

Input:
- a finite search space $\mathscr{X}$
- an objective function $f : \mathscr{X} \to \mathbb{R}$ to be minimised
- a set of m low-level heuristics $H := \{h_1, \ldots, h_m\}$.

Parameters $k \in \mathbb{N}$, $w \in \mathbb{N}^+$ {size of the sliding-time-window}
Let H be partitioned into k subsets $H_1, \ldots, H_k$.
$\forall i \in [m]$, set $\alpha_i^{(0)} := 1$, and $\beta_i^{(0)} := 1$.
$\forall j \in [k]$, set $s_j^{(0)} := 1$, and $r_j^{(0)} := 1$.
Choose an initial solution $x \in \mathscr{X}$.
for $t = 0, 1, \ldots$ until the termination condition is satisfied **do**
  Sample $A_j^{(t)} \sim \text{Beta}(s_j^{(t)}, r_j^{(t)})$, $\forall j \in [k]$
  Let $J := \arg\max_{j \in [k]} \{A_j^{(t)}\}$ and select the subset H_J
  Sample $U_i^{(t)} \sim \text{Beta}(\alpha_i^{(t)}, \beta_i^{(t)})$, $\forall i \in H_J$
  Let $I := \arg\max_{i \in H_J} \{U_i^{(t)}\}$
  $x' := h_I(x)$ {apply the chosen heuristic h_I}
  $\forall j \in [m]$, let $\ell_j^{(t)} := \begin{cases} 1 & \text{if } j = I \text{ and } f(x) > f(x') \\ 0 & \text{otherwise} \end{cases}$
  $\forall j \in [m]$, let $\eta_j^{(t)} := \begin{cases} 1 & \text{if } j = I \text{ and } f(x) \leq f(x') \\ 0 & \text{otherwise} \end{cases}$
  $\alpha_i^{(t+1)} := \alpha_i^{(t)} + \ell_i^{(t)}$
  $\beta_i^{(t+1)} := \beta_i^{(t)} + \eta_i^{(t)}$
  $\forall j \in [k]$, let $\kappa_j^{(t)} := \begin{cases} 1 & \text{if } I \in H_j \text{ and } f(x) > f(x') \\ 0 & \text{otherwise} \end{cases}$
  $\forall j \in [k]$, let $\nu_j^{(t)} := \begin{cases} 1 & \text{if } I \in H_j \text{ and } f(x) \leq f(x') \\ 0 & \text{otherwise} \end{cases}$
  $s_j^{(t+1)} := s_j^{(t)} + \kappa_j^{(t)}$
  $r_j^{(t+1)} := r_j^{(t)} + \nu_j^{(t)}$
  if $t \geq w$ **then**
    $\forall i \in [m]$, let $\alpha_i^{(t+1)} := \alpha_i^{(t+1)} - \ell_i^{(t-w)}$
    $\forall i \in [m]$, let $\beta_i^{(t+1)} := \beta_i^{(t+1)} - \eta_i^{(t-w)}$
    $\forall j \in [k]$, let $s_j^{(t+1)} := s_j^{(t+1)} - \kappa_j^{(t-w)}$
    $\forall j \in [k]$, let $r_j^{(t+1)} := r_j^{(t+1)} - \nu_j^{(t-w)}$
  end if
  if $f(x) \geq f(x')$ **then**
    $x := x'$
  end if
end for

The proposed hyper-heuristic divides the set of low-level heuristics uniformly at random into

k subsets. Then it applies two Thompson Sampling mechanisms. The first learning mechanism evaluates the overall performance of the low-level heuristics in each subset and decides which subset should be chosen at time t. Denote by $s_i^{(t)}$ and $r_i^{(t)}$ the number of successes and failures, respectively, observed from the low-level heuristics in the i-th subset, in terms of generating strictly better solutions with respect to a given objective function, within a time-window at iteration t of Algorithm 14. Each subset $i \in [k]$ is associated with a utility score. The utility score of each subset is a random variable, in the t-th iteration of Algorithm 14, sampled from a Beta$(s_i^{(t)}, r_i^{(t)})$ distribution. The subset that has the maximum utility score is then chosen. The second Thompson Sampling mechanism decides which heuristic in the selected subset should be applied to the current candidate solution. As in Algorithm 13, the newly generated solution is accepted if it is not worse than the current candidate solution with respect to a given objective function; otherwise, it is rejected. The Thompson Sampling mechanisms are combined with a sliding time-window to only maintain the recent observations about the performances of the low-level heuristics.

5.2.3 Multiplicative Weights Update Hyper-heuristic

This section introduces a multiplicative weight update hyper-heuristic; its precise description is given in Algorithm 15. The proposed hyper-heuristic associates each low-level heuristic with a positive weight. The weights of low-level heuristics are initialised to the same value and updated by a weight update scheme during the search process. We introduce a new multiplicative weight update (MWU) scheme that adjusts the negative reinforcement value according to the current weight of the chosen low-level heuristic and an estimated probability of generating a strictly better solution with respect to a given objective function.

Recall that the success probabilities of low-level heuristics are estimated as the number of times of the low-level heuristic produced strictly better solutions, with respect to a given

Algorithm 15 Multiplicative Weights Update Hyper-heuristic (MWUHH)

Input:
- a finite search space $\mathscr{X}$
- an objective function $f : \mathscr{X} \to \mathbb{R}$ to be minimised
- a set $H := \{h_1, \ldots, h_m\}$ of m mutation and local search low-level heuristics, where $\forall i \in [m]$, $h_i : \mathscr{X} \to \mathscr{X}$. - parameters $w_{\max} \in \mathbb{R}^+$, $\beta_{\max} < 1$, $\gamma \in \mathbb{R}^+$

$\forall i \in [m]$, let $w_i^{(0)} := 1$ be an initial weight of the low-level heuristic i.
Let $\alpha > 1$ and $0 < \beta < 1$ be the rewarding and punishing rates respectively.
$\forall i \in [m]$, set $s_i := 1$ and $r_i := 1$.
Choose an initial solution $x \in \mathscr{X}$.
for $t = 0, 1, \ldots$ until the termination condition is satisfied **do**
  Let $J := \arg\max_{j \in [m]} \{w_j^{(t)}\}$
  Let $K := \arg\max_{k \in [m]/\{J\}} \{w_k^{(t)}\}$
  Pick a heuristic h_I uniformly at random from a set $\{h_J, h_K\}$
  $x'_t := h_I(x_t)$ {apply the chosen heuristic h_I}
  if $f(x) > f(x')$ **then**
    $w_I^{(t+1)} := \min\{w_I^{(t)} \alpha, w_{\max}\}$
    $s_I := s_I + 1$
  else
    $\ell := \arg\max_{i \in [m]} \{\frac{s_i}{s_i + r_i}\}$
    $\eta := \frac{s_\ell}{s_\ell + r_\ell}$
    $\beta := \min\{(\frac{1}{w_I^{(t)}})^\eta, \beta_{\max}\}$
    $w_I^{(t+1)} := \max\{w_I^{(t)} \beta, w_{\min}\}$
    $r_I := r_I + 1$
  end if
  if $s_I + r_I > \gamma$ **then**
    $s_I := s_I \cdot \frac{\gamma - 1}{\gamma}$
    $r_I := r_I \cdot \frac{\gamma - 1}{\gamma}$
  end if
  if $f(x_t) \geq f(x'_t)$ **then**
    $x_{t+1} := x'_t$
  else
    if $x_t = x'_{t-1}$ and $f(x_{t-1}) < f(x'_{t-1}))$ **then**
      $x_{t+1} := x_{t-1}$
    else
      $x_{t+1} := x'_t$
    end if
  end if
end for

objective function, proportional to the number of times it has been executed. Precisely, let $s_i^{(t)}$ be the number of times the low-level heuristic i generated strictly better solutions, and $r_i^{(t)}$ be the number of times it failed to generate better solutions with respect to a given objective function until the t-th iteration of Algorithm 15. The success probability of the i-th low-level heuristic in iteration t is estimated as follows:

$$\bar{q}_i^{(t)} := \frac{s_i^{(t)}}{s_i^{(t)} + r_i^{(t)}}$$

Suppose that the i-th low-level heuristic has the maximum estimated success probability in iteration t of Algorithm 15 (i.e. $\bar{q_i}^{(t)} \geq \bar{q_j}^{(t)}$ for all $j \in [m]$). Let Y_i be the random variable representing the number of executions of the i-th low-level heuristic until it generates a strictly better solution with respect to a given objective function. The expected value of Y_i in iteration t of Algorithm 15 is:

$$\mathbb{E}[Y_i] = \frac{1}{\bar{q}_i^{(t)}}$$

The MWU scheme adapts the negative reinforcement values so that the i-th low-level heuristic, which has the maximum estimated success probability, does not lose its entire weight before it is executed $1/\bar{q}_i^{(t)}$ times (see lines $15-18$ of Algorithm 15).

The success probabilities of typical low-level heuristics may vary between different regions of the search space of a given problem. Algorithm 15, therefore, employs a discounting factor to estimate the success probabilities of low-level heuristics based on their recent performance (see lines $21-23$ in Algorithm 15). The discounting factor γ is a parameter of the algorithm. If γ is large, the recent results of the low-level heuristics are weighted less in the estimation of their success probabilities, and if γ is small, the algorithm may not estimate the success

probabilities of low-level heuristics accurately. As discussed in Section 4.6.3, the estimated success probabilities of typical low-level heuristics can be extremely small (i.e. equal to zero in some search regions of the search space of a given problem). Therefore, the proposed algorithm bounds the punishing rate β from above by a predefined value $\beta_{\max} < 1$ such that the punishing rate β remains within the interval $]0, \beta_{\max}]$.

The proposed selection method is a tournament strategy that picks a low-level heuristic uniformly at random among the two low-level heuristics with the maximum weights (see lines $7-9$ in Algorithm 15). The selected low-level heuristic is then applied to the current candidate solution. Algorithm 15 mixes mutation and local search low-level heuristics in a single set H. If the generated solution is not worse than the current candidate solution with respect to a given objective function, then it is immediately accepted and replaces the existing solution (see lines $25-27$ of Algorithm 15). If the generated solution was worse than the candidate solution, then it is passed to the next iteration, and a low-level heuristic is chosen and applied to it. The newly generated solution is accepted if (and only if) it is not worse than the current candidate solution; otherwise, it is rejected.

5.3 Benchmark Problems

5.3.1 HyFlex Problems

We evaluate the proposed hyper-heuristics on benchmark problems of the well-known CHeSC 2011. As discussed in Section 2.7, HyFlex (version v1.0) was designed to facilitate the CHeSC 2011 competition which provides implementations of several instances of six well-known combinatorial optimisation problems: Boolean satisfiability (SAT), bin packing (BP), personnel scheduling (PS), permutation flow-shop (PF), travelling salesman problem (TSP), and vehicle routing problem (VRP). In the CHeSC 2011 competition, the competing hyper-heuristics were

evaluated on five instances per problem domain: SAT, BP, PS, PF, TSP, and VRP. 31 runs of the same hyper-heuristic on each problem instance (thirty instances in total) were conducted. A ranking method inspired by the Formula One ranking method was used Burke et al. (2011a). In each instance, the eight best hyper-heuristics are given the ranks of $10, 8, 6, 5, 4, 3, 2$, and 1, respectively, while the remaining hyper-heuristics are given ranks of zero. In the case of ties (i.e. more than one hyper-heuristic achieving the same median objective function value), a tie method is used to assign appropriate ranks to the tied hyper-heuristics (further details are available on the competition website[1]). The winner of the competition is the hyper-heuristic achieving the highest overall cumulative ranks on all problem domains.

5.3.2 Test Models

We proposed test models to gain insights into the behaviour of the proposed Thompson Sampling mechanism in a controlled search environment. The proposed models are inspired by the Boolean artificial search scenario in (Fialho, 2011), and can be described as follows. There are m low-level heuristics available to work on a given solution. For $i \in [m]$, let q_i be the success probability of the i-th low-level heuristic. Suppose that the success probability of one low-level heuristic is p and the success probabilities of the other low-level heuristics are $q < p$. We refer to the heuristic that has success probability p as the *best* low-level heuristic. As discussed in Section 3.1, the success probabilities of typical low-level heuristics may vary between the regions of the search space of a given problem. Switching search models are thus proposed, in which the entire run period is divided into five sub-periods, which are referred to as *search phases*. All search phases contain identical run length. We investigated how the proposed Thompson Sampling mechanism adapts the selection probability of the best low-level heuristic and experimented with six models based on the following assumptions. (1) The hyper-heuristic has a set

[1] www.asap.cs.nott.ac.uk/external/chesc2011

Table 5.1: The success probabilities (SP) of the low-level heuristics in the proposed test models.

SP	Model 1	Model 2	Model 3	Model 4	Model 5	Model 6
p	0.400	0.200	0.180	0.100	0.100	0.010
q	0.300	0.100	0.150	0.050	0.010	0.005

$H := \{h_1, \ldots, h_{10}\}$ of ten heuristics. Usually, hyper-heuristics operate on a small number of low-level heuristics, we, therefore, choose the number of low-level heuristics $m = 10$. (2) The best low-level heuristic in search phase $1, 2, 3, 4$ and 5 is heuristic $1, 2, 3, 4$ and 5 respectively. (3) The success probabilities of low-level heuristics in each model are given in Table 5.1.

5.4 Experimental Results

We conducted a set of experiments on a broad range of optimisation problems and test models. The purpose of the experiments was to investigate the performance of the proposed hyper-heuristics in terms of the quality of the results within a fixed time. We compared the performance of the proposed algorithms with a large number of hyper-heuristics, including a state-of-the-art algorithm, within the CHeSC 2011 competition. The reason of using CHeSC 2011 is that it is a well-established benchmark in the field of hyper-heuristics (see, for example, (Burke et al., 2011a) (Drake et al., 2012) (Kheiri et al., 2015)). We evaluated the learning capabilities of the proposed Thompson Sampling mechanism to identify the most suitable low-level heuristics on the proposed test models (see Section 5.3.2). The proposed models allow evaluating the learning capability of the proposed Thompson Sampling mechanism with respect to different scenarios. We summarised our general experimental settings below.

5.4.1 General Experimental Settings

We used the same settings considered in CHeSC 2011 competition to compare the performance of the proposed algorithms to the CHeSC 2011 competition hyper-heuristics. The experiments

in this current study were run on an Intel i7 CPU at 3.60 GHz with 16 GB RAM. The CHeSC 2011 competition website provides a benchmarking program computing the equivalent time to 600 seconds on the organisers' machines. The program reported that 415 seconds on our machine is equal to 600 seconds on the competition organisers machines.

We tested the adaptive Thompson Sampling hyper-heuristic (TSHH) (Algorithm 13) and compared it to the CHeSC 2011 competition hyper-heuristics. The TSHH was tested 31 times on each of the five CHeSC 2011 instances per problem domain: Boolean satisfiability (SAT), bin packing (BP), personnel scheduling (PS), permutation flow-shop (PF), travelling salesman problem (TSP), and vehicle routing problem (VRP). The subset of the mutation low-level heuristics in the TSHH consists of both the mutation and ruin-recreate low-level heuristics and the local search subset contains the available local search low-level heuristics in HyFlex. The sliding time-window size in the TSHH was set to 2×10^5 based on initial experiments. In HyFlex, each low-level heuristic (apart from crossover low-level heuristics) consists of a problem-dependent parameter that takes a value between $[0,1]$. The TSHH employs the proposed Thompson Sampling mechanism to control these parameters and adapt their values while solving a given problem. Particularly, each parameter value i, for $i \in \{0.1, 0.2, \ldots, 0.9\}$, is associated with a utility score that, in the t-th iteration of the algorithm, is sampled from a Beta$(\delta_i^{(t)}, \chi_i^{(t)})$ distribution given that $\delta_i(t)$ is the number of times the hyper-heuristic generated a strictly better solution under the i-th parameter value, and $\chi_i(t)$ is the number of times the hyper-heuristic, under the i-th parameter value, failed to produce a strictly better solution with respect to a given objective function.

The TSHH was also assessed on the proposed test models (see Section 5.3.2), and compared to the following algorithms:

- The simple random hyper-heuristic (SRHH) (Cowling et al., 2001): we employed the SRHH as a baseline for these comparisons, as it selects low-level heuristics uniformly at

random.

- The additive reinforcement learning hyper-heuristic (RLHH) (Nareyek, 2004): the discussed additive reinforcement learning mechanism in the previous chapter with the max selection method was evaluated on the proposed test models and compared to the TSHH.

- The Markov chain hyper-heuristic (MCHH) proposed in (Kheiri and Keedwell, 2015): the MCHH is based on a Markov chain model in which the low-level heuristics form the state space. The weight of each edge represents the transition probability of moving from the current state (such that the current heuristic) to the destination state (such that any alternative heuristic, including the heuristic itself). The model is fully connected such that every pairs of nodes have non-zero transition probability. All the weights of the edges are initialised to the same value. If the application of the chosen heuristic resulted in an improvement in the candidate solution with respect to a given objective function, then the weight of the edge leading to the selection of this heuristic is increased by one. The *roulette wheel* selection method is employed to select the following heuristic, based on the weights of the edges of the current state.

- The upper confidence bound (UCB) and the UCB-tuned algorithms (Auer et al., 2002): the UCB and UCB-tuned are two popular *confidence-interval* based algorithms, their efficiency in terms of handling the exploration versus exploitation dilemma were already proved, as discussed in Section 3.5.2. Therefore, we compare the TSHH with both algorithms on the proposed test models.

The size of the window w in the proposed hyper-heuristic was set to 4000 based on preliminary experiments. The upper confidence bounds of the UCB and UCB-tuned are computed based on the entire history of the run. This implies that as the number of executions of low-level heuristics increases the new results of low-level heuristics are weighted as less important on the estima-

Table 5.2: The rank of the adaptive Thompson Sampling hyper-heuristic (TSHH) and the CHeSC 2011 competition hyper-heuristics.

Method	SAT	BP	PS	PF	TSP	VRP	Overall
AdapHH	**34.75**	**45.00**	7·00	31·00	34·75	14·00	166·50
TSHH	0·00	24·00	**47.00**	**44.00**	**38.00**	7·00	160·00
VNS-TW	34·25	2·00	30·50	27·00	13·75	4·00	111·50
ML	14·50	8·00	25·50	32·50	10·00	21·00	111·50
PHunter	10·50	3·00	9·50	6·00	22·75	**33.00**	84·75
EPH	0·00	7·00	7·50	16·00	29·75	12·00	72·25
HAHA	32·75	0·00	21·50	0·83	0·00	14·00	69·08
NAHH	14·00	17·00	1·00	19·50	10·00	6·00	67·50
KSATS-HH	24·00	9·00	5·50	0·00	0·00	22·00	60·50
ISEA	6·00	27·00	11·50	1·50	8·00	4·00	58·00
HAEA	0·50	2·00	1·00	5·33	8·00	27·00	43·83
ACO_HH	0·00	19·00	0·00	6·33	7·00	1·00	33·33
GenHive	0·00	12·00	4·50	5·00	2·00	6·00	29·50
XCJ	5·50	11·00	0·00	0·00	0·00	5·00	21·50
AVEG-Nep	12·00	0·00	0·00	0·00	0·00	9·00	21·00
SA_ILS	0·75	0·00	14·00	0·00	0·00	4·00	18·75
DynILS	0·00	9·00	0·00	0·00	9·00	0·00	18·00
GISS	0·75	0·00	8·00	0·00	0·00	6·00	14·75
MCHH-S	4·75	0·00	0·00	0·00	0·00	0·00	4·75
SelfSearch	0·00	0·00	1·00	0·00	2·00	0·00	3·00
Ant-Q	0·00	0·00	0·00	0·00	0·00	0·00	0·00

tion of the upper confidence bound of UCB and UCB-tuned algorithms (See Section 3.5.2). Therefore, both UCB and UCB-tuned algorithms were combined with a sliding-time-window to accommodate the dynamic change in the success probabilities of low-level heuristics.

5.4.2 Adaptive Thompson Sampling Hyper-heuristic

Table 5.2 shows the rank of the TSHH compared to the CHeSC 2011 hyper-heuristics according to the ranking method used in the CHeSC 2011.

Overall, the proposed hyper-heuristic achieved the second best rank across all of the problem domains. In the personnel scheduling, permutation flow-shop, and travelling salesman problem domains, the TSHH achieved the best rank compared to the CHeSC 2011 hyper-heuristics, including the state-of-the-art algorithm. However, it did not perform very well in either the

Boolean satisfiability and vehicle routing problems, thus raising an interesting subject for future work.

We followed Kheiri et al. (2015), and normalised the median of the objective function values of the proposed Thompson Sampling hyper-heuristic and all the competitors' hyper-heuristics per problem domain using the following formula:

$$\frac{d_i - d_{\min}}{d_{\max} - d_{\min}} \tag{5.1}$$

where d_i is the median of objective function values of the hyper-heuristic, $d_{\max}$ is the maximum median objective function value, and $d_{\min}$ is the minimum median objective function value. The above formula is a standard way to normalise the median of the objective function values of the hyper-heuristics. Figure 5.1 shows the normalised median of the objective function values of the TSHH and the CHeSC 2011 hyper-heuristics per problem domain.

Test Models

Figure 5.2 shows the selection probability of the best heuristic with the TSHH and the considered algorithms on the proposed test models. In Model 1, the results reveal that the TSHH identified the best low-level heuristic in each search phase. Comparable results are also obtained when the UCBTHH is applied on the same test model. While the UCBHH chooses the best heuristic less frequently than both the TSHH and the UCBTHH, the estimated selection probability of the best heuristic with the UCBHH is larger than that with the MCHH, RLHH, and SRHH. Almost identical results are obtained in Model 2. The results indicate that the MCHH can identify the best heuristic, but is too slow to learn the changes in the success probabilities of heuristics. In Model 3, the difference between the success probability of the best heuristic and the other heuristics is relatively small (i.e. 0.03). The results show that the ability of the

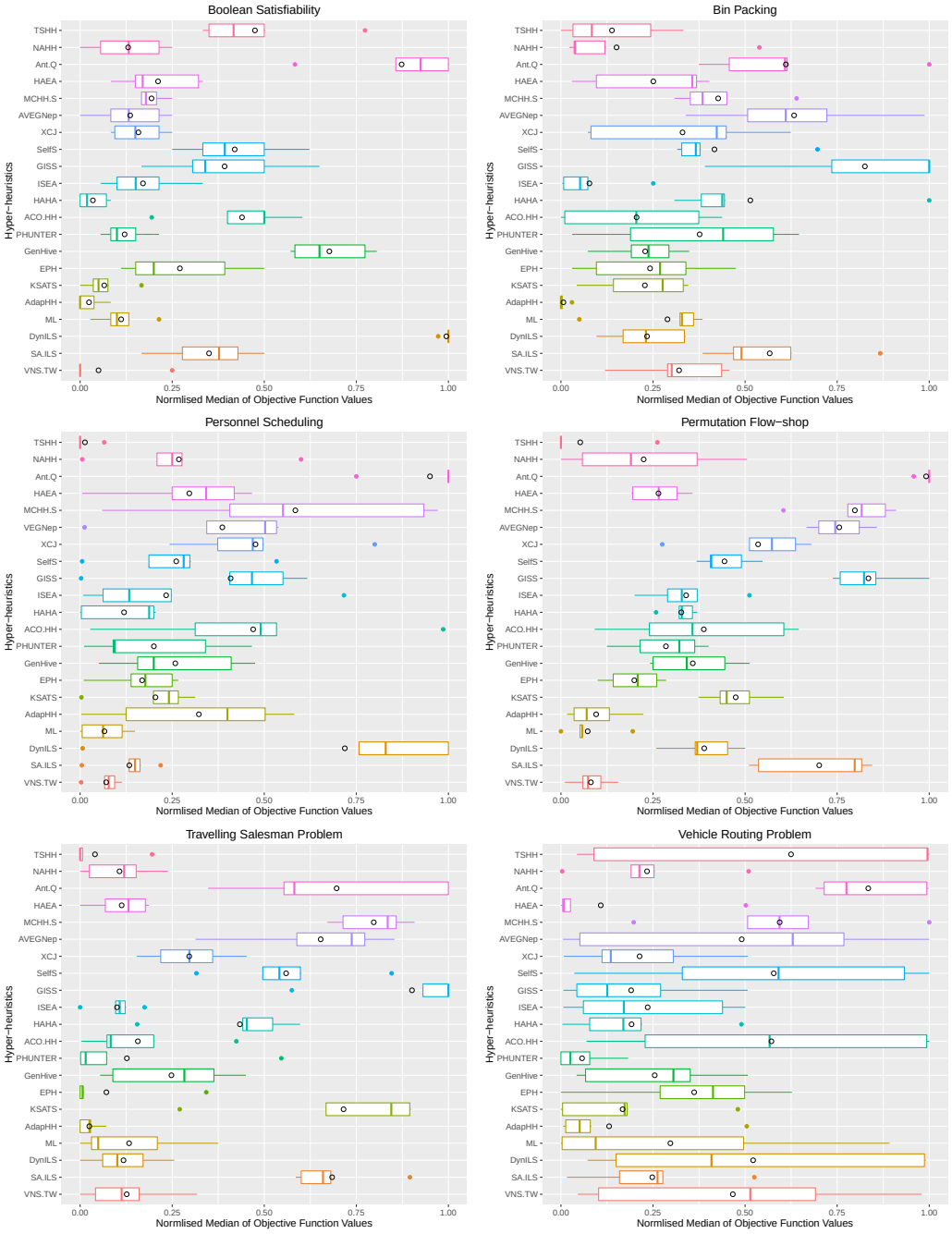

Figure 5.1: Box plots of the normalised median of the objective function values of the adaptive Thompson Sampling hyper-heuristic (TSHH) and the CHeSC 2011 competition hyper-heuristics on five instances per problem domain in HyFlex (smaller is better). The small circles indicate the mean.

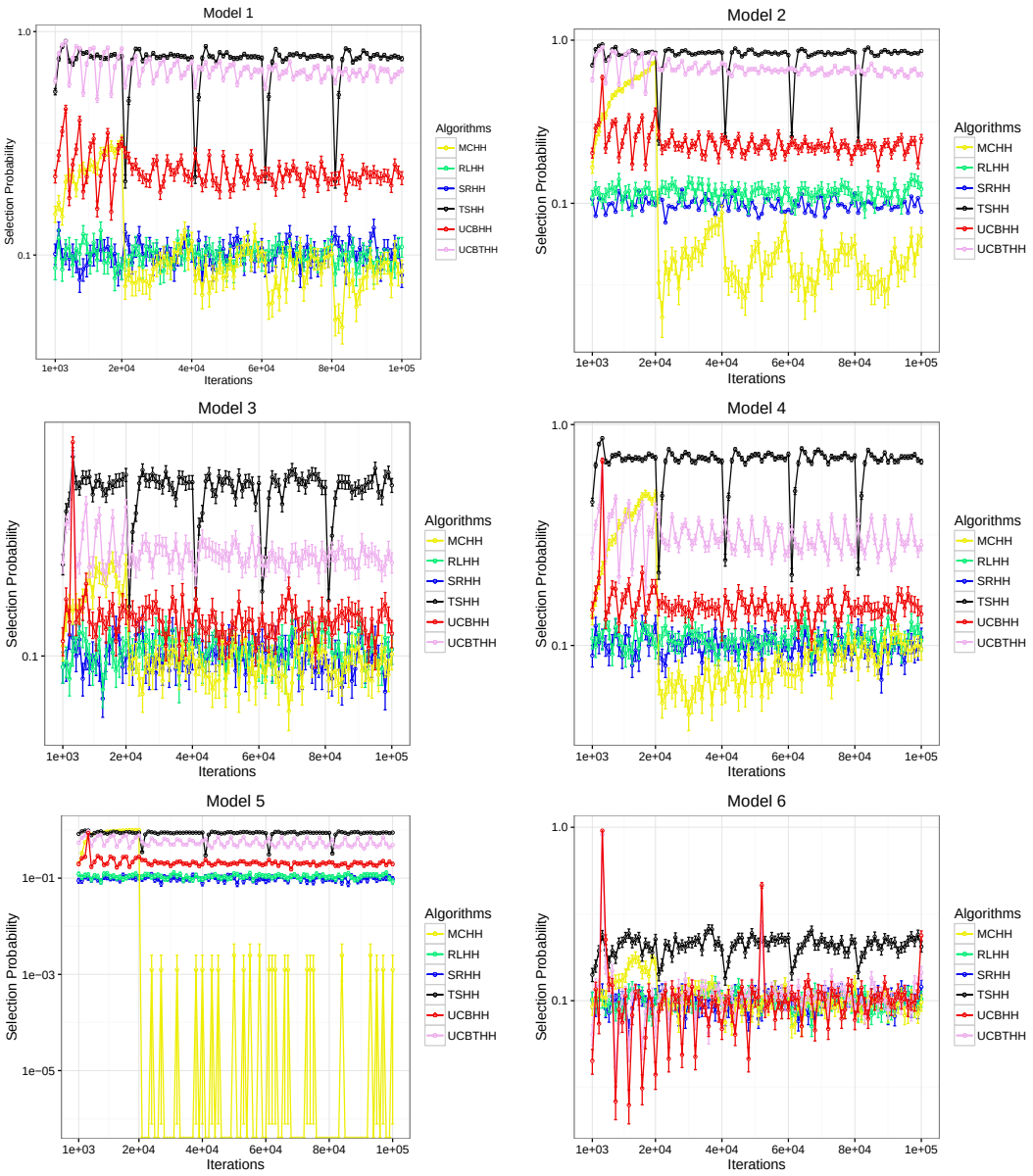

Figure 5.2: The selection probability of the best low-level heuristic, with error bars, with the Markov chain hyper-heuristic (MCHH), additive reinforcement learning hyper-heuristic (RLHH), simple random hyper-heuristic (SRHH), Thompson Sampling hyper-heuristic (TSHH), UCB hyper-heuristic (UCBHH), and UCB-Tuned hyper-heuristic (UCBTHH) on the proposed test models estimated over 800 runs. The length of the run is 10^5 iterations.

TSHH to identify the best heuristic in each search phase as being more effective than with other hyper-heuristics. In Models 4 and 5, the results of the RLHH and SRHH are almost the same with respect to the selection probability of the best heuristic. In Model 6, the success probabilities of the heuristics are somewhat small, where the success probability of the best heuristic is $p = 0.010$ and the success probability of other heuristics is $q = 0.005$. However, the success probabilities of typical low-level heuristics have the potential to be considerably smaller (see Section 4.6.3). The TSHH is the only hyper-heuristic was capable of choosing the most suitable low-level heuristic with probability higher than uniform at random in each search phase as shown in Figure 5.2. In the first search phase, the MCHH increases the selection probability of the best heuristic. However, it almost chooses heuristics uniformly at random in the other search phases. A high variance can be observed in the estimated selection probability of the best heuristic with the UCBHH. Overall, the results show that the selection probability of the most suitable heuristic with the TSHH is better than that with the other hyper-heuristics in all the proposed models.

The effectiveness of the TSHH with respect to the number of low-level heuristics was also assessed. We investigated the impact of the number of low-level heuristics on the performance of the TSHH in terms of identifying the best low-level heuristic. We tested and compared the TSHH with the considered hyper-heuristics on Model 2 (see Section 5.3.2) with different numbers of low-level heuristics. The reason of choosing Model 2 is that it is easy to learn model where the success probabilities of the low-level heuristics and the difference between the success probability of the best heuristic and the other heuristics are relatively large (where $p = 0.2$ and $q = 0.1$). Therefore, if the considered hyper-heuristics were not able to identify the best low-level heuristic in this model, then they may not be able to identify the most appropriate heuristics in more complicated models.

Figure 5.3 shows the selection probability of the best heuristic with the considered hyper-

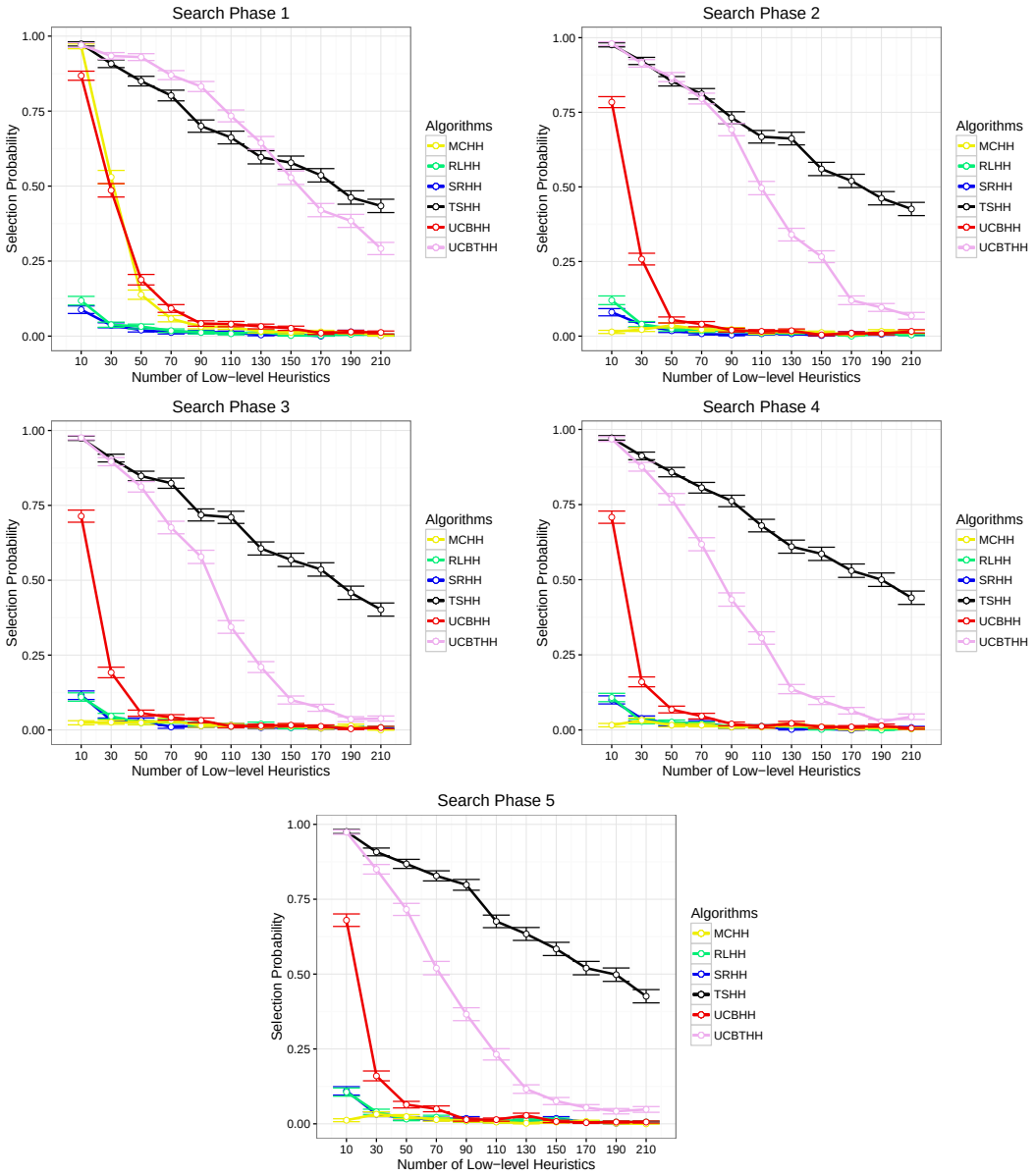

Figure 5.3: The selection probability of the best heuristic at the end of each search phase in Model 2 with the Markov chain hyper-heuristic (MCHH), reinforcement learning hyper-heuristic (RLHH), simple random hyper-heuristic (SRHH), Thompson Sampling hyper-heuristic (TSHH), UCB hyper-heuristic (UCBHH), and UCB-Tuned hyper-heuristic (UCBTHH) estimated over 500 runs. The length of the run is 10^6 iterations.

heuristics at the end of each search phase in Model 2 with m heuristics, for ($m = 10, \ldots, 210$). In general, the results show that the selection probability of the best heuristic decreases as the number of low-level heuristics increases with all the considered hyper-heuristics. The TSHH chooses roughly the best heuristic with probability higher than $1/2$ at the end of each search phase when the number of low-level heuristics is less than 200; and slightly less than $1/2$ otherwise. Competitive results were achieved by the UCBTHH in the first search phase. While, in the other search phases, if the number of low-level heuristics is large (i.e. larger than 200), then the UCBTHH chooses roughly low-level heuristics uniformly at random. The results show that if the number of low-level heuristics is smaller than 50, then (i) the TSHH and UCBTHH provide roughly the same selection probability of the best heuristic in each search phase; (ii) the UCBHH is capable of learning and increasing the selection probability of the best heuristic in each search phase; (iii) in the first search phase the MCHH is able to increase the selection probability of the best heuristic, but it almost get stuck with it where it chooses the best heuristics with probability less than that with the SRHH. On the other hand, if the number of low-level heuristics is larger than 50, then the estimated selection probability of the best low-level heuristic is roughly the same with the UCBHH, MCHH, RLHH, and SRHH. This implies that the learning mechanisms in the UCBHH, MCHH, and RLHH do not necessary improve their performances if the number of low-level heuristics is large. Overall, the results reveal that the TSHH is able to learn the change in the success probabilities of the low-level heuristics and increase the selection probability of the best heuristic in each search phase of the considered model.

5.4.3 Hierarchical Thompson Sampling Hyper-heuristic

We tested the Hierarchical Thompson Sampling hyper-heuristic (HTSHH) with 1000 low-level heuristics on several instances of the permutation flow-shop and the travelling salesman prob-

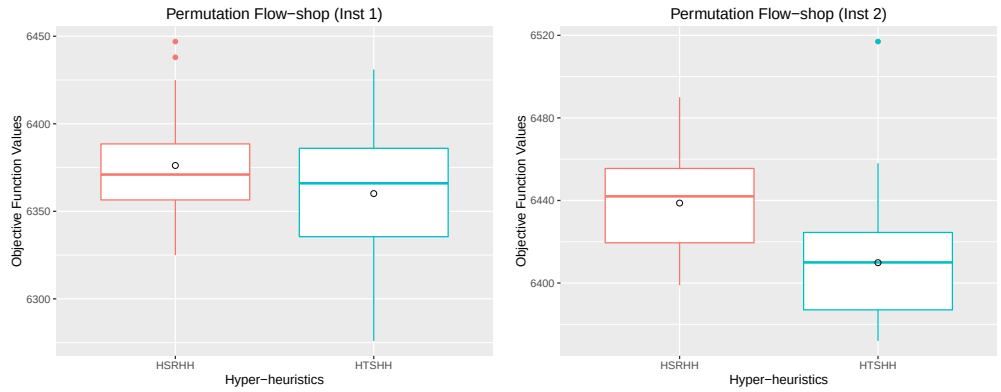

Figure 5.4: The objective function values of the Hierarchical Thompson Sampling hyper-heuristic (HTSHH) and Hyper-random hyper-heuristic (HRHH) on two instances of the permutation flow-shop problem.

lems. The purpose of the experiments was to evaluate the performance of the HTSHH in terms of the quality of the results. We used the HyFlex framework which provides 15 low-level heuristics for the permutation flow-shop problem and 13 low-level heuristics for the travelling salesman problem. The remaining low-level heuristics were artificial low-level heuristics that return the same input solution. We considered a challenging search scenario on the above problems where only a few low-level heuristics can improve the solution while the success probability of the remaining low-level heuristics is zero. The HTSHH divided the set of the low-level heuristics uniformly at random into 50 subsets. The size of the window is 4×10^5. The HTSHH was compared to the so-called Hyper-random algorithm which chooses a low-level heuristic uniformly at random, then the selected low-level heuristic is repeatedly applied as long as an improvement in the candidate solution, with respect to a given objective function, is found (Remde et al., 2007). The length of the run is 2×10^6 objective function evaluations. Each run was repeated 31 times. The results are presented in the following figures.

Figure 5.4 shows the objective function values of the HTSHH and HRHH on two instances of the permutation flow-shop problem. The results indicate that the HTSHH achieves better average and median of the objective function values compared to the HRHH. Figure 5.5 shows the objective function values that are obtained by the HTSHH and HRHH on two instances of the

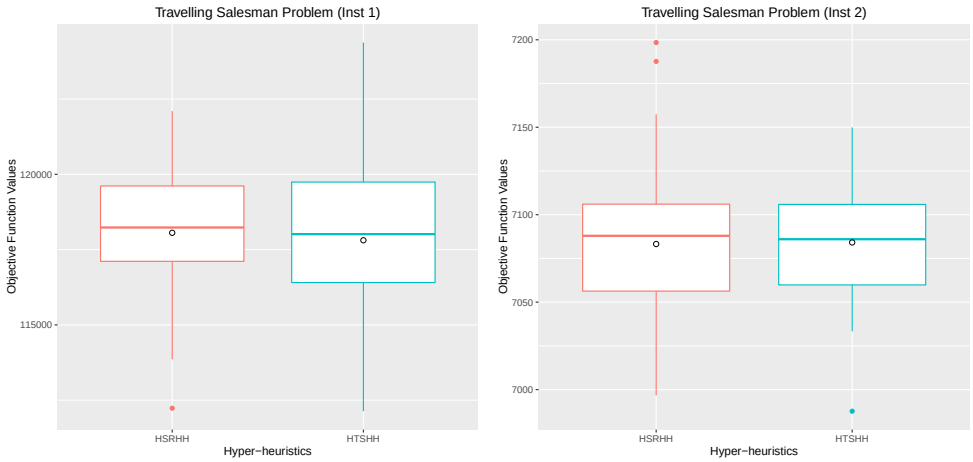

Figure 5.5: The objective function values of the Hierarchical Thompson Sampling hyper-heuristic (HTSHH) and Hyper-random hyper-heuristic (HRHH) on two instances of the travelling salesman problem.

travelling salesman problem. The results show that the HTSHH achieves slightly better average and median objective function value in the first instance, while both algorithms have roughly the same average and median objective function values in the other instance. We performed a non-parametric statistical test, a so-called *Wilcoxon rank sum* test, with confidence interval 95 percent to examine whether the results of the algorithms are statistically different. The results indicate that the difference between the results of the algorithms is significant in instance 2 of the permutation flow-shop problem, while the p-value is above the critical level 0.05 in the other instances which means there is no significant difference between the results of the algorithms on the other instances of the considered problems.

5.4.4 Multiplicative Weights Update Hyper-heuristic

We evaluated the multiplicative weight update hyper-heuristic (MWUHH) on the CHeSC 2011 instances of the bin packing problem and compared it to the CHeSC 2011 competition hyper-heuristics. The goal of this current experiment is to investigate the performance of the MWUHH in terms of the quality of the results within a fixed time. We used the same settings considered in CHeSC 2011. The MWUHH was tested 31 times on each of the CHeSC 2011 instance of the bin

packing problem. The discounting factor γ was set to 10^5, the maximum negative reinforcement rate $\beta_{\max}$ was set to 0.999 based on initial experiments. The subset of the mutation low-level heuristics in MWUHH consists of both the mutation and ruin-recreate low-level heuristics in HyFlex. The results are presented in the following table.

Table 5.3 shows the median and the best objective function values of the MWUHH, adaptive Thompson Sampling hyper-heuristic (TSHH), and all the CHeSC 2011 hyper-heuristics on five instances of the bin packing problem. The results reveal that the MWUHH achieves the best median objective function value compared to all the CHeSC 2011 hyper-heuristics, including the state-of-the-art algorithm, on three instances of the bin packing problem. It achieves the second best median objective function value on the other instances. We also compared the performance of the MWUHH to the adaptive Thompson Sampling hyper-heuristic (TSHH) on the CHeSC 2011 instances of the bin packing problem. Figure 5.6 shows the objective function values of the MWUHH and the TSHH on the CHeSC 2011 instances of the bin packing problem.

The results show that the average and the median objective function values of the MWUHH are better than these of the TSHH on four instances of the bin packing problem. The TSHH achieves better average and median objective function value than the MWUHH in one instance. A low variance in the results of the MWUHH can be observed. We investigated whether the results of the algorithms are statistically different. We used the so-called *Wilcoxon rank sum* test, and the results show that the difference between the results of the algorithms is significant on all the considered instances of the bin packing problem.

5.5 Summary of Results

This chapter has introduced three reinforcement learning hyper-heuristics: Adaptive Thompson Sampling, Hierarchical Thompson Sampling, and the Multiplicative Weight Update hyper-

Table 5.3: The median and the best objective function values of the multiplicative weight update hyper-heuristic (MWUHH), the adaptive Thompson Sampling hyper-heuristic (TSHH), and the CHeSC 2011 competition hyper-heuristics on the bin-packing problem.

Algorithms	Inst 7		Inst 1		Inst 9		Inst 10		Inst 11	
	Median	Best	Median	Best	Median	Best	Median	Best	Median	Best
MWUHH	0.018436812	0.015845157	**0.002687247**	**0.002470279**	**4.76E-04**	**4.19E-04**	0.108350727	**0.108323273**	**1.18E-03**	**3.70E-04**
TSHH	0.0337	0.0286	0.0076	0.0036	0.0060	0.0038	0.1082	0.1082	0.0053	0.0021
VNS-TW	0.037	0.0298	0.0072	0.0036	0.0167	0.0136	0.1088	0.1087	0.0278	0.0238
SA-ILS	0.0787	0.0734	0.0115	0.0081	0.0146	0.0123	0.1104	0.11	0.0296	0.0239
DynILS	0.0403	0.0349	0.0077	0.0036	0.0102	0.0069	0.1087	0.1086	0.0129	0.0086
ML	0.0421	0.0323	0.0075	0.0067	0.0146	0.0124	0.1085	0.1084	0.0218	0.0178
AdapHH	**0.0161**	**0.0131**	0.0036	0.0028	0.0036	**0.0004**	**0.1083**	**0.1083**	0.0035	0.0031
KSATS	0.0192	0.0162	0.0078	0.0071	0.0115	0.0094	0.1089	0.1088	0.022	0.0188
EPH	0.0504	0.043	0.0036	0.0034	0.0113	0.008	0.1087	0.1083	0.0224	0.0136
GenHive	0.0299	0.0133	0.0071	0.0067	0.0104	0.0069	0.1086	0.1084	0.0229	0.0157
PHUNTER	0.0479	0.0397	0.0036	0.0034	0.0201	0.0178	0.1091	0.1088	0.0395	0.0318
ACO-HH	0.0477	0.0404	0.0032	0.0029	0.0039	0.0024	0.1099	0.1097	0.0149	0.0108
HAHA	0.0883	0.0617	0.0073	0.0035	0.0145	0.0037	0.1102	0.1095	0.0279	0.0076
ISEA	0.0342	0.0219	0.0033	0.0029	0.0037	0.0017	0.1086	0.1085	0.0064	0.0033
GISS	0.0692	0.0591	0.0084	0.0077	0.0322	0.0278	0.1126	0.111	0.0592	0.0515
SelfS	0.0664	0.0531	0.0074	0.0067	0.0144	0.0102	0.1097	0.1091	0.0239	0.0177
XCJ	0.022	0.0168	0.0115	0.0075	0.0157	0.0145	0.1086	0.1085	0.0285	0.0247
AVEGNep	0.0874	0.053	0.0077	0.0072	0.0181	0.0138	0.1114	0.1104	0.0375	0.028
MCHH-S	0.0623	0.0492	0.0073	0.0067	0.0146	0.0113	0.1098	0.1092	0.0286	0.0218
HAEA	0.0452	0.0349	0.0036	0.0033	0.0138	0.0083	0.1087	0.1087	0.024	0.0168
Ant-Q	0.0491	0.041	0.0165	0.0083	0.021	0.0158	0.1099	0.1093	0.0377	0.0317
NAHH	0.055	0.0473	0.0035	0.003	0.0047	0.0027	0.1088	0.1083	0.0055	0.0046

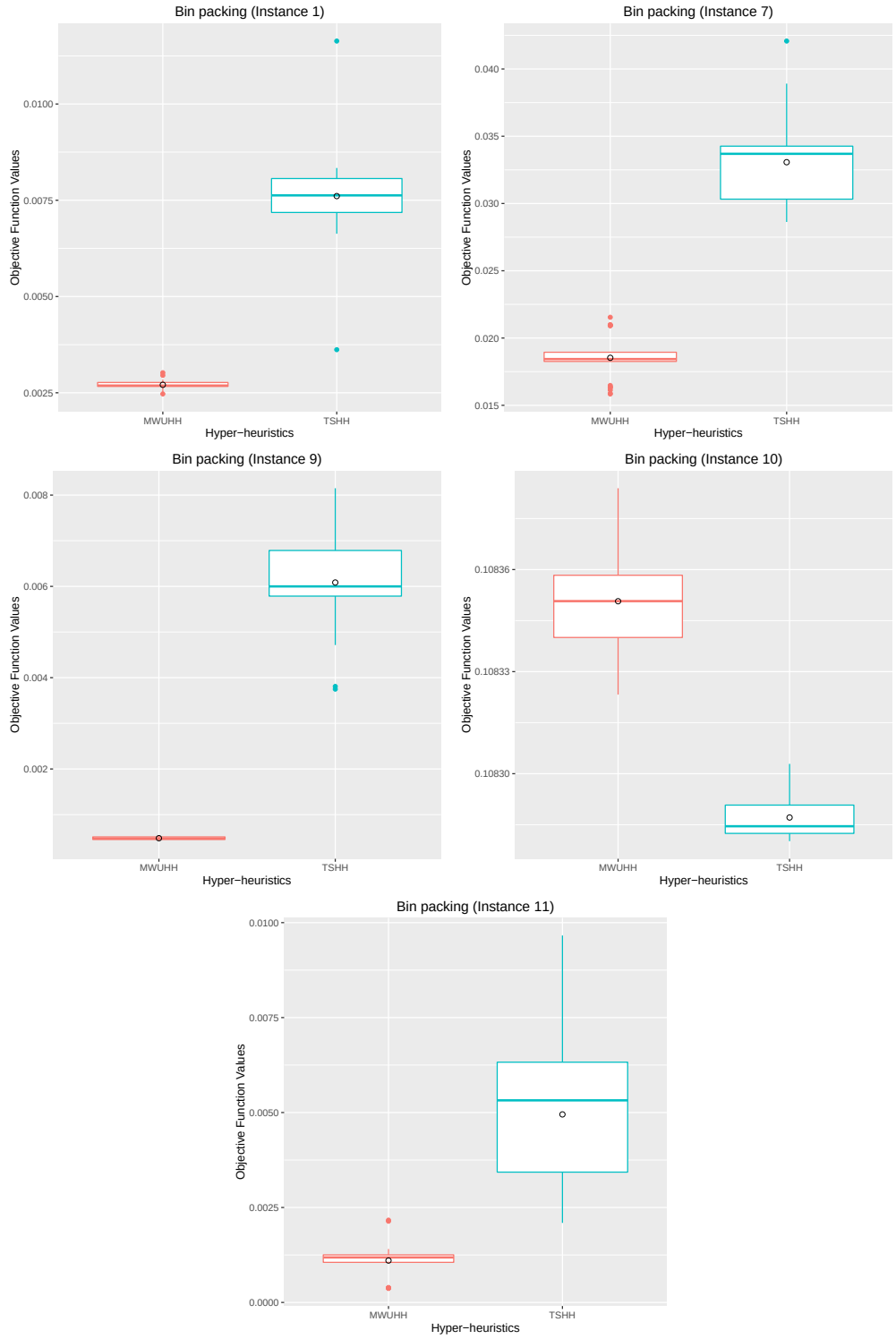

Figure 5.6: Box plots of the objective function values of the multiplicative weight update hyper-heuristic (MWUHH) and the adaptive Thompson Sampling hyper-heuristic (TSHH) on the CHeSC 2011 instances of the bin packing problem.

heuristic. We proposed an adaptive Thompson Sampling mechanism for hyper-heuristics and extensively evaluate its performance on both test models and a set of well-known optimisation problems. The results have revealed that the proposed Thompson Sampling mechanism effectively identifies the suitable heuristics on test models. The generality of the Adaptive Thompson Sampling hyper-heuristic has been assessed on a broad range of combinatorial optimisation problems: Boolean satisfiability (SAT), bin packing (BP), personnel scheduling (PS), permutation flow-shop (PF), travelling salesman problem (TSP), and vehicle routing problem (VRP). The results have shown that it produces competitive, often better, results compared to a large number of hyper-heuristics including a state-of-the-art algorithm.

We introduced a Thompson Sampling hyper-heuristic for solving optimisation problems with a large number of low-level heuristics. We tested the proposed algorithm and compared it to the so-called Hyper-random algorithm on four instances of the permutation flow-shop and the travelling salesman problem using HyFlex. We proposed a search scenario on HyFlex in which 1000 low-level heuristics are available for the hyper-heuristics to apply at any given time during the search process. The proposed low-level heuristics were artificial heuristics that return the same input search point. The results have shown that the proposed algorithm performed best in one instance of the permutation flow-shop problem. Although there is no significant difference between the results of both algorithms on the other instances, the median and the average objective function values obtained by the proposed hyper-heuristic were slightly better than those obtained by the Hyper-random algorithm. This chapter has empirical evidence that Thompson Sampling is an effective learning mechanism in hyper-heuristics, and has provided suggestions as to how it can be improved.

Additionally, we introduced the multiplicative weight update hyper-heuristic. The proposed hyper-heuristic was tested and compared to a large number of hyper-heuristics within the CHeSC 2011 competition on the bin packing problem. The results have shown that the pro-

posed algorithm achieved better or comparable results compared to a state-of-the-art algorithm.

Chapter 6

On the Complexity of Identifying the Most Suitable heuristics

6.1 Introduction

Reinforcement learning hyper-heuristics need to acquire information about the performance of low-level heuristics, in order to predict which heuristic is the most suitable to apply to a given search point. As already discussed, reinforcement learning hyper-heuristics have to balance between applying the current most appropriate low-level heuristic, based on past observations (exploitation), and gathering information concerning the performance of uncertain low-level heuristics (exploration), in order to refine their decisions in the future search points. This chapter investigates the lower bound on the time that each reinforcement learning hyper-heuristic need to identify the most suitable low-level heuristic with a given error probability.

The proposed problem in this chapter is similar to the so-called *best arm identification problem* which has proved to be useful in the context of machine learning (see, for example, (Audibert and Bubeck, 2010; Mellor, 2014; Kaufmann et al., 2015)). In the best arm identification problem, an agent is given k arms. Each arm when the agent pulls it, returns a reward. The

expected reward of each arm is fixed but unknown. The task of the agent is to identify the arm which has the maximum expected reward. Two models of the best arm identification problem have been considered in the literature of multi-armed bandit: *fixed budget* and *fixed confidence*. In the former, the agent is given a predefined number of admissible pulls, with the goal of maximising the confidence probability of identifying the best arm. In the latter, the confidence probability of identifying the best arm is predefined, with the task of the agent being to minimise the number of pulls, and so identify the best arm with the given confidence probability. The best arm identification problem provides a model for several applications ranging from product testing (Birattari et al., 2010) to network optimisation problems (Audibert and Bubeck, 2010). Additional information concerning the best arm identification problem can be found on (Mannor and Tsitsiklis, 2004; Kaufmann et al., 2015).

The aim of formulating the problem of identifying the most suitable low-level heuristic with a given error probability is to investigate the distinguishability of reinforcement learning hyper-heuristics to identify the most suitable low-level heuristics. As in the previous chapters, the discussion is restricted to a general hyper-heuristic framework in which the observable behaviour of low-level heuristics is the change in the objective function values after their applications. Typically, the success probabilities of low-level heuristics vary between the regions of the search space of a given problem. Therefore, it may be too strong to assume that the success probabilities of low-level heuristics are drawn from fixed distributions. We used classical theoretic-information tools, such as Kullback-Leibler divergence and Pinsker's inequality, to rigorously prove a general distribution-independent lower bound for the exploration time that needs to be performed by each reinforcement learning hyper-heuristic, to identify the most suitable low-level heuristic with a given error probability.

6.2 Problem Formulation

We investigate the problem of identifying the most suitable low-level heuristic among a set of predefined low-level heuristics. This section presents specifically our formulation of this problem.

There are a finite search space $\mathscr{X}$, an objective function $f : \mathscr{X} \to \mathbb{R}$, and a set $H := \{h_1, \ldots, h_m\}$ of m low-level heuristics. When a low-level heuristic is applied, the hyper-heuristic receives as a feedback whether the applied low-level heuristic generated a strictly better solution with respect to a given objective function. Recall that the success probability of a low-level heuristic is the probability that the heuristic generates a strictly better solution with respect to a given search point and objective function.

Suppose, without loss of generality, that low-level heuristic i has success probability $q_i^{(t)} := p^{(t)}$, and for all $j \in [m]$ and $j \neq i$ the success probability of low-level heuristic j is $q_j^{(t)} := r^{(t)}$. Assume that $p^{(t)} \geq r^{(t)}$ for all $t \in \mathbb{N}$. We call the low-level heuristic that has success probability equals to $p^{(t)}$ the most suitable low-level heuristic. Let A be a reinforcement learning hyper-heuristic that evaluates the performance of the low-level heuristics for T iterations, and then identifies the most suitable low-level heuristic with probability at least $1 - \delta$ for some $\delta > 0$. Suppose, without loss of generality, that A observes results for two low-level heuristics in each time step $t \in \mathbb{N}$, such that

$$A : \{0,1\}^{2T} \to \{1, \ldots, m\} \tag{6.1}$$

We are interested in the exploration time T that A needs to identify the most suitable low-level heuristic with a given error probability without knowing how A operates.

6.3 Best Low-level Heuristic Identification: General Lower Bound

This section presents general lower bounds on the runtime that every reinforcement learning hyper-heuristic must perform to identify the most suitable low-level heuristic with a given error probability. The following theorem shows a general lower bound on the time that each reinforcement learning hyper-heuristic needs to identify, with a given error probability, which low-level heuristic is most suitable among two low-level heuristics.

Theorem 37. *Given a finite set $\mathscr{X}$, an objective function $f : \mathscr{X} \to \mathbb{R}$, and a set $H := \{h_1, h_2\}$ of two low-level heuristics. Let A, $p^{(t)}$, $r^{(t)}$, and T be as defined above. If there exist constants ε, α, and δ such that $p^{(t)} - r^{(t)} \leq \varepsilon$ and $\log\left(\frac{p^{(t)}(1-r^{(t)})}{r^{(t)}(1-p^{(t)})}\right) \leq \alpha$ for all $t \in \mathbb{N}$, then the expected time that A must perform to identify the best low-level heuristic with probability at least $1-\delta$ is lower bounded by*

$$\mathbb{E}[T] \geq \frac{2(1-2\delta)^2}{\varepsilon \log(2) \alpha}$$

Theorem 37 indicates that as the difference between the success probabilities of the low-level heuristics ε decreases the expected lower bound on time that each reinforcement learning hyper-heuristic must perform to identify the most suitable low-level heuristic increases.

Proof. Recall that, at time $t \in \mathbb{N}$, the success probability of the most suitable low-level heuristic is $p^{(t)}$, and the success probability of the other low-level heuristic is $r^{(t)}$. Let Z_1 and Z_2 be two probability distributions defined as follows:

$$Z_1 := \begin{cases} 1 & \text{with probabillity } p^{(t)}(1-r^{(t)}) \\ 0 & \text{with probability } p^{(t)}r^{(t)} + (1-p^{(t)})(1-r^{(t)}) \\ -1 & \text{with probability } (1-p^{(t)})r^{(t)} \end{cases}$$

$$Z_2 := \begin{cases} 1 & \text{with probabillity } r^{(t)}(1-p^{(t)}) \\ 0 & \text{with probability } p^{(t)}r^{(t)} + (1-p^{(t)})(1-r^{(t)}) \\ -1 & \text{with probability } (1-r^{(t)})p^{(t)} \end{cases}$$

Let A be a reinforcement learning hyper-heuristic capable of identifying the most suitable low-level heuristic with probability at least $1-\delta$ after T iterations, such that

$$\mathbb{P}_{Z_1^T}(A=1) \geq 1-\delta \quad \text{and} \quad \mathbb{P}_{Z_2^T}(A=2) \geq 1-\delta \tag{6.2}$$

Let η be a random variable defined as follow.

$$\eta := \begin{cases} 1, & \text{if } A=1 \\ 0, & \text{otherwise} \end{cases} \tag{6.3}$$

Thus, the expected value of η under the probability distribution Z_1 is at least

$$\mathbb{E}_{Z_1^T}[\eta] \geq 1-\delta \tag{6.4}$$

and under the probability distribution Z_2 is at most

$$\mathbb{E}_{Z_2^T}[\eta] \leq \delta \tag{6.5}$$

This implies that

$$|\mathbb{E}_{Z_1^T}[\eta] - \mathbb{E}_{Z_2^T}[\eta]| \geq 1-2\delta \tag{6.6}$$

Note that $0 \leq \eta \leq 1$. Thus, by Lemma (1.2) in (Tulsian, 2001) the following equation holds true:

$$1 - 2\delta \leq |\mathbb{E}_{Z_1^T}[\eta] - \mathbb{E}_{Z_2^T}[\eta]| \leq \frac{1}{2}||Z_1^T - Z_2^T||_1 \tag{6.7}$$

which gives

$$||Z_1^T - Z_2^T||_1 \geq 2(1-2\delta) \tag{6.8}$$

It follows by Pinsker's inequality that

$$D_{KL}(Z_1^T||Z_2^T) \geq \frac{2(1-2\delta)^2}{\log(2)} \tag{6.9}$$

By the definition of the KL-divergence (see Definition 3), the left-hand side is computed as follows.

$$\begin{aligned} D_{KL}(Z_1^T||Z_2^T) = &\sum_{t=1}^{T} p^{(t)}(1-r^{(t)}) \log\left(\frac{p^{(t)}(1-r^{(t)})}{r^{(t)}(1-p^{(t)})}\right) \\ &+ \left(p^{(t)}r^{(t)} + (1-p^{(t)})(1-r^{(t)})\right) \log\left(\frac{p^{(t)}r^{(t)} + (1-p^{(t)})(1-r^{(t)})}{p^{(t)}r^{(t)} + (1-p^{(t)})(1-r^{(t)})}\right) \\ &+ (1-p^{(t)})r^{(t)} \log\left(\frac{(1-p^{(t)})r^{(t)}}{(1-r^{(t)})p^{(t)}}\right) \end{aligned}$$

We now use the fact that $\log(1) = 0$, and $\log(\frac{p^{(t)}}{r^{(t)}}) = -\log(\frac{r^{(t)}}{p^{(t)}})$.

$$D_{KL}(Z_1^T||Z_2^T) = \sum_{t=1}^{T} (p^{(t)} - r^{(t)}) \log(\frac{p^{(t)}}{r^{(t)}}) + (p^{(t)} - r^{(t)}) \log(\frac{1-r^{(t)}}{1-p^{(t)}}) \tag{6.10}$$

$$= \sum_{t=1}^{T} (p^{(t)} - r^{(t)}) \log(\frac{p^{(t)}(1-r^{(t)})}{r^{(t)}(1-p^{(t)})}) \tag{6.11}$$

By noting that $p^{(t)} - r^{(t)} \leq \varepsilon$ and $\log(\frac{q^{(t)}(1-r^{(t)})}{r^{(t)}(1-q^{(t)})}) \leq \alpha$ for all $t \in \mathbb{N}$, it follows that

$$D_{KL}(Z_1^T || Z_2^T) \leq T\varepsilon\alpha \tag{6.12}$$

Finally, by plugging (6.12) in (6.9), we have

$$T\varepsilon\alpha \geq \frac{2(1-2\delta)^2}{\ln(2)} \tag{6.13}$$

□

6.3.1 General Lower Bound

This section shows a general lower bound on the required time to identify the best low-level heuristic.

Recall that A is a reinforcement learning hyper-heuristic capable of identifying the most suitable low-level heuristic after T iterations with probability at least $1-\delta$ for some $\delta > 0$. Suppose, without loss of generality, that at each time $t \in \mathbb{N}$, A picks one low-level heuristic and observes result for both the chosen low-level heuristic and the most suitable heuristic. For $j \in \{2,\ldots,m\}$, the expected time that A needs to learn that low-level heuristic j has success probability $r^{(t)} < p^{(t)}$ is by Theorem 37 lower bounded by

$$\mathbb{E}[T] \geq \frac{2(1-2\delta)^2}{\varepsilon\log(2)\alpha} \tag{6.14}$$

Thus, the expected time that A needs to identify the most suitable low-level heuristic with probability $1-\delta$ is at least

$$\mathbb{E}[T] \geq \sum_{j=2}^{m} \frac{2(1-2\delta)^2}{\varepsilon\log(2)\alpha} \tag{6.15}$$

The above analysis proves the following theorem.

Theorem 38. *Given a finite set $\mathscr{X}$, an objective function $f : \mathscr{X} \to \mathbb{R}$, and a set $H := \{h_1, h_2\}$ of two low-level heuristics. Let A, $p^{(t)}$, $r^{(t)}$, and T be as defined above. If there exist constants ε, α, and δ such that $p^{(t)} - r^{(t)} \leq \varepsilon$ and $\log\left(\frac{p^{(t)}(1-r^{(t)})}{r^{(t)}(1-p^{(t)})}\right) \leq \alpha$ for all $t \in \mathbb{N}$, then the expected time that A must perform to identify the best low-level heuristic with probability at least $1-\delta$ is lower bounded by*

$$\mathbb{E}[T] \geq \frac{2(m-1)(1-2\delta)^2}{\varepsilon \log(2) \alpha}$$

Theorem 38 shows that the lower bound on the time that each reinforcement learning hyper-heuristic needs to identify the most suitable low-level heuristic with a given error probability depends on the number of low-level heuristics m and the difference between the success probabilities of low-level heuristics ε. As the number of low-level heuristics increases the lower bound on the exploration time T increases. While the exploration time T decreases as the difference between the success probability of low-level heuristics ε increases. The results suggest that there can be some scenarios where the time that a reinforcement learning hyper-heuristic need to identify the most suitable low-level heuristic, with a given error probability, is more than the required optimisation time to find the target solution.

6.4 Summary of Results

This chapter has formulated the problem of identifying the most suitable low-level heuristic with a given error probability. The goal of proposing this problem was to investigate generally the capabilities of reinforcement learning hyper-heuristics to identify the most suitable low-level heuristics. We showed a general lower bound on the exploration time that every reinforcement learning hyper-heuristic must perform to determine the most suitable low-level heuristic with

a given error probability. The results suggest that there can be some scenarios where no reinforcement learning hyper-heuristics can identify the most suitable low-level heuristic with a given error probability within an optimisation. However, the presented results in this chapter are for certain scenarios on the proposed problem. More scenarios which may lead to different results should be considered to provide more insights into the learning capabilities of reinforcement learning hyper-heuristics.

Chapter 7

Conclusions and Future Work

Hyper-heuristics are search algorithms that operate on the space of heuristics with the goal of solving a wide range of optimisation problems. A typical hyper-heuristic evolves an initially generated solution by selecting and applying a heuristic to the solution at hand. Then an acceptance method decides whether the newly generated solution should replace the existing solution or reject it. This process is repeated until the termination conditions are satisfied. It has been observed that heuristics perform differently during the search process (Özcan et al., 2008) (Lehre and Özcan, 2013). Various learning mechanisms have been proposed for hyper-heuristics in the literature, as discussed in Chapter 3. Given a set of heuristics, the goal of learning mechanisms is to identify the most suitable heuristic to apply to a given search point. Reinforcement learning mechanisms interact with the search environment with the aim of adapting the selection of heuristics while solving a given problem. A challenge for reinforcement learning mechanisms is that they have to balance between exploiting the most suitable heuristics based on previous observations and exploring the effectiveness of other heuristics. This thesis provided different contributions for reinforcement learning hyper-heuristics which are summarised in the following section.

7.1 Summary of Results

The theoretical foundation of reinforcement learning hyper-heuristics was almost non-existing. This thesis built the foundation for this area in the field of hyper-heuristics. We started with an analysis of commonly used reinforcement learning mechanisms in hyper-heuristics; namely: greedy, random gradient, and additive reinforcement learning mechanism. This was the first study investigating the performance of the above reinforcement learning mechanisms theoretically. The results have rigorously shown that the random gradient and greedy mechanisms do not necessary improve the performance when compared to uniform at random selection hyper-heuristics. Additionally, we showed certain conditions under which the additive reinforcement learning mechanism chooses heuristics asymptotically uniformly at random. The results indicate that, the commonly used random gradient, greedy, and additive reinforcement learning mechanisms do not effectively learn as they expected to learn.

Following our theoretical analysis, we introduced a modified additive reinforcement learning mechanism. The proposed hyper-heuristic adapts the reinforcement values according to the estimated probability of generating a better solution, with respect to a given objective function, while solving a given problem. The proposed hyper-heuristic achieves better or comparable results compared to a state-of-the-art algorithm on a well-known benchmark problem. This thesis proposed an adaptive Thompson Sampling mechanism for hyper-heuristics. Thompson Sampling was introduced in the field of bandit theory. It provides theoretical guarantees in terms of handling the exploration versus exploitation trade-off on the so-called stationary multi-armed bandit problem (Kaufmann et al., 2012). The proposed Thompson Sampling mechanism was associated with a sliding time-window to discard past and irrelevant observations about the performance of heuristics. We extensively evaluated the proposed Thompson Sampling hyper-heuristic on both test models and a wide range of optimisation problems. The proposed

hyper-heuristic outperforms a state-of-the-art algorithm on three well-known optimisation problems. It efficiently identifies the most suitable heuristics on test models. This is the first study investigating Thompson Sampling in the field of hyper-heuristics.

With respect to the time that reinforcement learning hyper-heuristics need to identify the suitable heuristics, we proposed the problem of identifying the most suitable heuristic with a given error probability. We theoretically investigated the lower bound on the runtime that each reinforcement learning hyper-heuristic needs to determine the most suitable heuristic with a given error probability. Reinforcement learning hyper-heuristics have to gather information about the performance of heuristics in order to predict which heuristic is the most suitable to apply to a given search point, as discussed in Section 6.1. We showed general lower bounds on the runtime that each reinforcement learning hyper-heuristic must perform to identify the most suitable heuristic with a given error probability. This is the first time such a result was presented; there were no general results that hold "for all" reinforcement learning hyper-heuristics. The results indicated that the time that each reinforcement learning hyper-heuristic needs to identify the most suitable heuristic, with a given error probability, depends on both the number of heuristics and the probabilities of generating better solutions. Overall, the results revealed a general limitation to learning achieved by every reinforcement learning hyper-heuristic.

7.2 Future Work

This thesis provided theoretical analyses of reinforcement learning hyper-heuristics as well as proposed a benchmark problem that allows evaluating their performances with respect to various situations. These built the foundation for this area in the field of hyper-heuristics. However, this is only the first step. Theoretical results for reinforcement learning hyper-heuristics on classes of functions and more typical scenarios are still needed. Moreover, most of our theoretical results

were for classical reinforcement learning hyper-heuristics. Future work should consider more advanced reinforcement learning mechanisms.

Along the same line, move-acceptance strategies can influence the performance of reinforcement learning hyper-heuristics. In this thesis, the central focus was on the learning capabilities of reinforcement learning mechanisms. Further work could also investigate how different move-acceptance strategies influence the performance of reinforcement learning hyper-heuristics. For example, (1) to what extent the cooling/heating schedule affects the runtime of reinforcement learning hyper-heuristics with the *simulated annealing* strategy on a given problem? (2) how the mutation rates in mutation low-level heuristics affect on the performance of a reinforcement learning hyper-heuristic with the *all-moves* acceptance strategy?

Another direction for future work lies within the work on Thompson Sampling hyper-heuristic. The results have shown that the proposed hyper-heuristic did not perform very well on the Boolean satisfiability and vehicle routing problems. Further work could, therefore, investigate the potential for improving the performance of the proposed hyper-heuristic on these problems. Additionally, in a related field, a class of the so-called portfolio algorithms combine a set of search heuristics, such as hyper-heuristics, and interleave their run with the goal of solving a given problem (Huberman et al., 1997) (Gomes and Selman, 2001). It is interesting to investigate whether the proposed Thompson Sampling mechanism compares favourably to the state-of-the-art learning mechanisms in the field of portfolio algorithms.

References

Agrawal, S. and Goyal, N. (2012). Analysis of thompson sampling for the multi-armed bandit problem. In *COLT*, pages 39–1.

Agrawal, S. and Goyal, N. (2013). Further optimal regret bounds for thompson sampling. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*, pages 99–107.

Ahmadi, S., Barone, R., Cheng, P., Cowling, P., and McCollum, B. (2003). Perturbation based variable neighbourhood search in heuristic space for examination timetabling problem. *Proceedings of multidisciplinary international scheduling: theory and applications (MISTA 2003)*, pages 13–16.

Asta, S., Özcan, E., and Curtois, T. (2016). A tensor based hyper-heuristic for nurse rostering. *Knowledge-Based Systems*, 98:185–199.

Audibert, J.-Y. and Bubeck, S. (2010). Best arm identification in multi-armed bandits. In *COLT-23th Conference on Learning Theory-2010*, pages 13–p.

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256.

Auger, A. and Doerr, B. (2011). *Theory of Randomized Search Heuristics: Foundations and Recent Developments*, volume 1. World Scientific.

Bäck, T., Fogel, D., and Michalewicz, Z. (1997). Handbook of evolutionary computation. *Release*, 97(1):B1.

Bai, R., Blazewicz, J., Burke, E. K., Kendall, G., and McCollum, B. (2007a). A simulated annealing hyper-heuristic methodology for flexible decision support. *School of CSiT, University of Nottingham, UK, Tech. Rep*.

Bai, R., Burke, E. K., Gendreau, M., Kendall, G., and McCollum, B. (2007b). Memory length in hyper-heuristics: An empirical study. In *2007 IEEE Symposium on Computational Intelligence in Scheduling*, pages 173–178. IEEE.

Bai, R. and Kendall, G. (2005). An investigation of automated planograms using a simulated annealing based hyper-heuristic. In *Metaheuristics: Progress as real problem solvers*, pages 87–108. Springer.

Berberoglu, A. and Uyar, A. Ş. (2011). Experimental comparison of selection hyper-heuristics for the short-term electrical power generation scheduling problem. In *European Conference on the Applications of Evolutionary Computation*, pages 444–453. Springer.

Berry, D. A. and Fristedt, B. (1985). *Bandit Problems: Sequential Allocation of Experiments (Monographs on Statistics and Applied Probability)*. Springer.

Bilgin, B., Demeester, P., Misir, M., Vancroonenburg, W., and Berghe, G. V. (2012). One hyper-heuristic approach to two timetabling problems in health care. *Journal of Heuristics*, 18(3):401–434.

Bilgin, B., Demeester, P., Mısır, M., Vancroonenburg, W., Berghe, G. V., and Wauters, T. (2010). A hyper-heuristic combined with a greedy shuffle approach to the nurse rostering competition. In *Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT 10)*.

Bilgin, B., Özcan, E., and Korkmaz, E. E. (2006). An experimental study on hyper-heuristics and exam timetabling. In *Practice and Theory of Automated Timetabling VI*, pages 394–412. Springer.

Birattari, M., Yuan, Z., Balaprakash, P., and Stützle, T. (2010). F-race and iterated f-race: An overview. In *Experimental methods for the analysis of optimization algorithms*, pages 311–336. Springer.

Böttcher, S., Doerr, B., and Neumann, F. (2010). Optimal fixed and adaptive mutation rates for the leadingones problem. In *International Conference on Parallel Problem Solving from Nature*, pages 1–10. Springer.

Bubeck, S., Cesa-Bianchi, N., et al. (2012). Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends in Machine Learning*, 5(1):1–122.

Burke, E., Curtois, T., Hyde, M., Kendall, G., Ochoa, G., Petrovic, S., Vázquez-Rodríguez, J. A., and Gendreau, M. (2010a). Iterated local search vs. hyper-heuristics: Towards general-purpose search algorithms. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE.

Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P., and Schulenburg, S. (2003a). Hyper-heuristics: An emerging direction in modern search technology. In *Handbook of metaheuristics*, pages 457–474. Springer.

Burke, E., Kendall, G., Silva, D. L., O'Brien, R., and Soubeiga, E. (2005a). An ant algorithm hyperheuristic for the project presentation scheduling problem. In *2005 IEEE Congress on Evolutionary Computation*, volume 3, pages 2263–2270. IEEE.

Burke, E. K. and Bykov, Y. (2008). A late acceptance strategy in hill-climbing for exam timetabling problems. In *PATAT 2008 Conference, Montreal, Canada*.

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

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

Burke, E. K., Gendreau, M., Ochoa, G., and Walker, J. D. (2011b). Adaptive iterated local search for cross-domain optimisation. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1987–1994. ACM.

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

Burke, E. K., Kendall, G., and Soubeiga, E. (2003b). A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics*, 9(6):451–470.

Burke, E. K., McCollum, B., Meisels, A., Petrovic, S., and Qu, R. (2007). A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research*, 176(1):177–192.

Burke, E. K., Silva, J. D. L., and Soubeiga, E. (2005b). Multi-objective hyper-heuristic approaches for space allocation and timetabling. In *Metaheuristics: Progress as Real Problem Solvers*, pages 129–158. Springer.

Chakhlevitch, K. and Cowling, P. (2008). Hyperheuristics: recent developments. In *Adaptive and multilevel metaheuristics*, pages 3–29. Springer.

Chapelle, O. and Li, L. (2011). An empirical evaluation of thompson sampling. In *Advances in neural information processing systems*, pages 2249–2257.

Chong, E. K. and Zak, S. H. (2013). *An introduction to optimization*, volume 76. John Wiley & Sons.

Cobos, C., Mendoza, M., and León, E. (2011). A hyper-heuristic approach to design and tuning heuristic methods for web document clustering. In *2011 IEEE Congress of Evolutionary Computation (CEC)*, pages 1350–1358. IEEE.

Cormen, T., Leiserson, C., and Rivest, R. C. S. (2001). Introduction to algorithms. *MIT Press*, 5(3):55.

Cormen, T. H. (2009). *Introduction to algorithms*. MIT press.

Corus, D., Dang, D.-C., Eremeev, A. V., and Lehre, P. K. (2014). Level-based analysis of genetic algorithms and other search processes. In *Parallel Problem Solving from Nature–PPSN XIII*, pages 912–921. Springer.

Cowling, P. and Chakhlevitch, K. (2003). Hyperheuristics for managing a large collection of low level heuristics to schedule personnel. In *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, volume 2, pages 1214–1221. IEEE.

Cowling, P., Kendall, G., and Soubeiga, E. (2001). A hyperheuristic approach to scheduling a sales summit. In *Practice and Theory of Automated Timetabling III*, pages 176–190. Springer.

Cowling, P., Kendall, G., and Soubeiga, E. (2002). Hyperheuristics: A tool for rapid prototyping in scheduling and optimisation. In *Applications of Evolutionary Computing*, pages 1–10. Springer.

Cowling, P. I., Kendall, G., and Soubeiga, E. (2000). A hyperheuristic approach to scheduling

a sales summit. In *Selected papers from the Third International Conference on Practice and Theory of Automated Timetabling III*, pages 176–190. Springer-Verlag.

Crowston, W. B., Glover, F., Trawick, J. D., et al. (1963). Probabilistic and parametric learning combinations of local job shop scheduling rules. Technical report, DTIC Document.

Dang, D.-C. and Lehre, P. K. (2016). Self-adaptation of mutation rates in non-elitist populations. In *International Conference on Parallel Problem Solving from Nature*, pages 803–813. Springer.

Dobslaw, F. (2010). A parameter tuning framework for metaheuristics based on design of experiments and artificial neural networks. In *International Conference on Computer Mathematics and Natural Computing*. WASET.

Doerr, B., Doerr, C., and Kötzing, T. (2016). Provably optimal self-adjusting step sizes for multi-valued decision variables. In *International Conference on Parallel Problem Solving from Nature*, pages 782–791. Springer.

Dokeroglu, T. and Cosar, A. (2016). A novel multistart hyper-heuristic algorithm on the grid for the quadratic assignment problem. *Engineering Applications of Artificial Intelligence*, 52:10–25.

Dowsland, K. A., Soubeiga, E., and Burke, E. (2007). A simulated annealing based hyper-heuristic for determining shipper sizes for storage and transportation. *European Journal of Operational Research*, 179(3):759–774.

Drake, J. H., Özcan, E., and Burke, E. K. (2012). An improved choice function heuristic selection for cross domain heuristic search. In *Parallel Problem Solving from Nature-PPSN XII*, pages 307–316. Springer.

Droste, S., Jansen, T., and Wegener, I. (2002). Optimization with randomized search heuristics - the (a) nfl theorem, realistic scenarios, and difficult functions. *Theoretical Computer Science*, 287(1):131–144.

Dueck, G. (1993). New optimization heuristics: the great deluge algorithm and the record-to-record travel. *Journal of Computational physics*, 104(1):86–92.

Eiben, A. E., Michalewicz, Z., Schoenauer, M., and Smith, J. E. (2007). Parameter control in evolutionary algorithms. In *Parameter setting in evolutionary algorithms*, pages 19–46. Springer.

Eiben, A. E. and Smit, S. K. (2011). Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19–31.

Elhag, A. and Özcan, E. (2015). A grouping hyper-heuristic framework: Application on graph colouring. *Expert Systems with Applications*, 42(13):5491–5507.

Feller, W. (1968). *An introduction to probability theory and its applications*, volume 1. Wiley.

Fialho, Á., Da Costa, L., Schoenauer, M., and Sebag, M. (2009). Dynamic multi-armed bandits and extreme value-based rewards for adaptive operator selection in evolutionary algorithms. In *International Conference on Learning and Intelligent Optimization*, pages 176–190. Springer.

Fialho, Á. R. S. (2011). *Adaptive operator selection for optimization*. PhD thesis, Citeseer.

Fisher, H. and Thompson, G. L. (1963). Probabilistic learning combinations of local job-shop scheduling rules. *Industrial scheduling*, 3(2):225–251.

Garivier, A. and Moulines, E. (2011). On upper-confidence bound policies for switching bandit

problems. In *International Conference on Algorithmic Learning Theory*, pages 174–188. Springer.

Garrido, P. and Castro, C. (2009). Stable solving of cvrps using hyperheuristics. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 255–262. ACM.

Glover, F. and Laguna, M. (2013). *Tabu Search*. Springer.

Gomes, C. P. and Selman, B. (2001). Algorithm portfolios. *Artificial Intelligence*, 126(1):43–62.

Graepel, T., Candela, J. Q., Borchert, T., and Herbrich, R. (2010). Web-scale bayesian click-through rate prediction for sponsored search advertising in microsoft's bing search engine. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 13–20.

Granmo, O.-C. (2010). Solving two-armed bernoulli bandit problems using a bayesian learning automaton. *International Journal of Intelligent Computing and Cybernetics*, 3:207–234.

Hajek, B. (1982). Hitting-time and occupation-time bounds implied by drift analysis with applications. *Advances in Applied probability*, pages 502–525.

He, J., He, F., and Dong, H. (2012). Pure strategy or mixed strategy? In *Evolutionary Computation in Combinatorial Optimization*, pages 218–229. Springer.

He, J., Hou, W., Dong, H., and He, F. (2013). Mixed strategy may outperform pure strategy: An initial study. In *2013 IEEE Congress on Evolutionary Computation*, pages 562–569. IEEE.

He, J., Mitavskiy, B., and Zhou, Y. (2014). A theoretical assessment of solution quality in evolutionary algorithms for the knapsack problem. In *Evolutionary Computation (CEC), 2014 IEEE Congress on*, pages 141–148. IEEE.

He, J. and Yao, X. (2001). Drift analysis and average time complexity of evolutionary algorithms. *Artificial Intelligence*, 127(1):57–85.

Holland, J. H. (1992). Genetic algorithms. *Scientific american*, 267(1):66–72.

Hsiao, P.-C., Chiang, T.-C., and Fu, L.-C. (2011). A variable neighborhood search-based hyper-heuristic for cross-domain optimization problems in chesc 2011 competition. In *Fifty-Third Conference of OR Society (OR53), Nottingham, UK*.

Huberman, B. A., Lukose, R. M., and Hogg, T. (1997). An economics approach to hard computational problems. *Science*, 275(5296):51–54.

Jansen, T. (2013). *Analyzing Evolutionary Algorithms: The Computer Science Perspective*. Springer.

Jansen, T. and Wegener, I. (2001). Evolutionary algorithms-how to cope with plateaus of constant fitness and when to reject strings of the same fitness. *IEEE Transactions on Evolutionary Computation*, 5(6):589–599.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285.

Kalender, M., Kheiri, A., Özcan, E., and Burke, E. K. (2013). A greedy gradient-simulated annealing selection hyper-heuristic. *Soft Computing*, 17(12):2279–2292.

Kaufmann, E., Cappé, O., and Garivier, A. (2015). On the complexity of best arm identification in multi-armed bandit models. *The Journal of Machine Learning Research*.

Kaufmann, E., Korda, N., and Munos, R. (2012). Thompson sampling: An asymptotically optimal finite-time analysis. In *Algorithmic Learning Theory*, pages 199–213. Springer.

Kendall, G. and Hussin, N. M. (2005). A tabu search hyper-heuristic approach to the examination timetabling problem at the mara university of technology. In *Practice and Theory of Automated Timetabling V*, pages 270–293. Springer.

Kendall, G. and Mohamad, M. (2004). Channel assignment optimisation using a hyper-heuristic. In *Cybernetics and Intelligent Systems, 2004 IEEE Conference on*, volume 2, pages 791–796. IEEE.

Kheiri, A. and Keedwell, E. (2015). Markov chain selection hyper-heuristic for the optimisation of constrained magic squares.

Kheiri, A., Keedwell, E., Gibson, M. J., and Savic, D. (2015). Sequence analysis-based hyper-heuristics for water distribution network optimisation. *Procedia Engineering*, 119:1269–1277.

Kiraz, B., Etaner-Uyar, A., and Özcan, E. (2013). Selection hyper-heuristics in dynamic environments. *Journal of the Operational Research Society*, 64(12):1753–1769.

Kiraz, B. and Topcuoglu, H. R. (2010). Hyper-heuristic approaches for the dynamic generalized assignment problem. In *2010 10th International Conference on Intelligent Systems Design and Applications*, pages 1487–1492. IEEE.

Kirkpatrick, S., Vecchi, M., et al. (1983). Optimization by simmulated annealing. *science*, 220(4598):671–680.

Lai, T. L. and Robbins, H. (1985). Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22.

Larose, M. (2011). A hyper-heuristic for the chesc 2011. In *The 53rd Annual Conference of the UK Operational Research Society (OR53)*.

Lehre, P. K. and Özcan, E. (2013). A runtime analysis of simple hyper-heuristics: to mix or not to mix operators. In *Proceedings of the twelfth workshop on Foundations of genetic algorithms XII*, pages 97–104. ACM.

Lehre, P. K. and Witt, C. (2012). Black-box search by unbiased variation. *Algorithmica*, 64(4):623–642.

Lehre, P. K. and Witt, C. (2014). Concentrated hitting times of randomized search heuristics with variable drift. In *Algorithms and Computation*, pages 686–697. Springer.

Lobo, F., Lima, C. F., and Michalewicz, Z. (2007). *Parameter setting in evolutionary algorithms*, volume 54. Springer Science & Business Media.

Lourenço, H. R., Martin, O. C., and Stützle, T. (2003). *Iterated local search*. Springer.

Mannor, S. and Tsitsiklis, J. N. (2004). The sample complexity of exploration in the multi-armed bandit problem. *Journal of Machine Learning Research*, 5(Jun):623–648.

Marín-Blázquez, J. G. and Schulenburg, S. (2007). A hyper-heuristic framework with xcs: Learning to create novel problem-solving algorithms constructed from simpler algorithmic ingredients. In *Learning classifier systems*, pages 193–218. Springer.

McClymont, K., Keedwell, E., Savić, D., and Randall-Smith, M. (2013). A general multi-objective hyper-heuristic for water distribution network design with discolouration risk. *Journal of Hydroinformatics*, 15(3):700–716.

McClymont, K. and Keedwell, E. C. (2011). Markov chain hyper-heuristic (mchh): an online selective hyper-heuristic for multi-objective continuous problems. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 2003–2010. ACM.

Mellor, J. C. (2014). Decision making using thompson sampling.

Mısır, M., Verbeeck, K., De Causmaecker, P., and Berghe, G. V. (2012a). The effect of the set of low-level heuristics on the performance of selection hyper-heuristics. In *International Conference on Parallel Problem Solving from Nature*, pages 408–417. Springer.

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

Nareyek, A. (2004). Choosing search heuristics by non-stationary reinforcement learning. In *Metaheuristics: Computer decision-making*, pages 523–544. Springer.

Neumann, F. and Witt, C. (2012). Bioinspired computation in combinatorial optimization: algorithms and their computational complexity. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion*, pages 1035–1058. ACM.

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

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

Oliveto, P. S., He, J., and Yao, X. (2007). Time complexity of evolutionary algorithms for combinatorial optimization: A decade of results. *International Journal of Automation and Computing*, 4(3):281–293.

Özcan, E., Bilgin, B., and Korkmaz, E. (2006). Hill climbers and mutational heuristics in hyperheuristics. *Parallel Problem Solving from Nature-PPSN IX*, pages 202–211.

Özcan, E., Bilgin, B., and Korkmaz, E. E. (2008). A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis*, 12(1):3–23.

Ozcan, E., Bykov, Y., Birben, M., and Burke, E. K. (2009). Examination timetabling using late acceptance hyper-heuristics. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 997–1004. IEEE.

Özcan, E., Misir, M., Ochoa, G., and Burke, E. K. (2010). A reinforcement learning-great-deluge hyper-heuristic for examination timetabling. *International Journal of Applied Metaheuristic Computing (IJAMC)*, 1(1):39–59.

Page, E. (1954). Continuous inspection schemes. *Biometrika*, 41(1/2):100–115.

Pisinger, D. and Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & operations research*, 34(8):2403–2435.

Remde, S., Cowling, P., Dahal, K., and Colledge, N. (2007). Exact/heuristic hybrids using rvns and hyperheuristics for workforce scheduling. In *Evolutionary Computation in Combinatorial Optimization*, pages 188–197. Springer.

Remde, S., Cowling, P., Dahal, K., Colledge, N., and Selensky, E. (2012). An empirical study of hyperheuristics for managing very large sets of low level heuristics. *Journal of the operational research society*, 63(3):392–405.

Ross, P. (2005). Hyper-heuristics. In *Search methodologies*, pages 529–556. Springer.

Ross, P. (2014). Hyper-heuristics. In *Search Methodologies*, pages 611–638. Springer.

Shoesmith, E. (1986). Huygens' solution to the gambler's ruin problem. *Historia mathematica*, 13(2):157–164.

Steiglitz, K. and Papadimitriou, C. H. (1982). Combinatorial optimization: Algorithms and complexity. *Prentice Hall, New Jersey., UV Vazirani (1984). On two geometric problems related to the travelling salesman problem. J. Algorithms*, 5:231–246.

Sudholt, D. (2009). The impact of parametrization in memetic evolutionary algorithms. *Theoretical Computer Science*, 410(26):2511–2528.

Sudholt, D. (2010). General lower bounds for the running time of evolutionary algorithms. In *Parallel Problem Solving from Nature, PPSN XI*, pages 124–133. Springer.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Tang, L., Rosales, R., Singh, A., and Agarwal, D. (2013). Automatic ad format selection via contextual bandits. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 1587–1594. ACM.

Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294.

Tulsian, M. (2001). Information and coding theory. *http://ttic.uchicago.edu/ madhurt/courses/infotheory2014/l5.pdf*.

Vazirani, V. V. (2013). *Approximation algorithms*. Springer Science & Business Media.

Wegener, I. (2001). *Theoretical aspects of evolutionary algorithms*. Springer.

Wegener, I. (2003). *Methods for the analysis of evolutionary algorithms on pseudo-boolean functions*. Springer.

Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82.

Yu, T., Davis, L., Baydar, C., and Roy, R. (2008). *Evolutionary computation in practice*, volume 88. Springer.