On the Runtime Analysis of Selection Hyper-heuristics for Pseudo-Boolean Optimisation

John Alasdair Warwicker
Registration Number: 150127219

Department of Computer Science
University of Sheffield

This dissertation is submitted for the degree of
Doctor of Philosophy

August 2019

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are my own original work and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. Some pieces of this thesis are based on articles that have been published or submitted elsewhere, as specified in Section 1.4.

John Alasdair Warwicker
Registration Number: 150127219
August 2019

Acknowledgements

I would firstly like to express my very great appreciation to my supervisor and mentor, Dr. Pietro Oliveto, for his guidance, support, patience and motivation during my PhD studies. Thanks to him, I have developed the confidence and knowledge required to complete my PhD and write this thesis.

I would also like to offer my special thanks to Dr. Andrei Lissovoi for his insights and help throughout this process. I am grateful also to Professor Benjamin Doerr for hosting us in France and for his advice and co-authorship; I also give thanks to Dr. Dirk Sudholt for his continued advice throughout my studies.

I thank the Algorithms group at the University of Sheffield, all of my labmates and lunch friends who have been with me throughout this process: Adbullah, Adam, Alison, Ben, Cassie, Dave, Donya, Edgar, George, Jorge (Popeyes), José, Mat, Neil, Phil, Tom, Wil, and many more. Thanks are also extended to Beth for providing great motivation throughout, and for all my friends who helped me along the way.

Finally, I would like to thank my family for their continued unconditional support.

Abstract

Rather than manually deciding on a suitable algorithm configuration for a given optimisation problem, hyper-heuristics are high-level search algorithms which evolve the heuristic to be applied. While there are numerous reported successful applications of hyper-heuristics to combinatorial optimisation problems, it is not yet fully understood how well they perform and on which problem classes they are effective. Selection hyper-heuristics (SHHs) employ smart methodologies to select from a pre-defined set of low-level heuristics which to apply in the next decision step. This thesis extends and improves upon the existing foundational understanding of the behaviour and performance of SHHs, providing insights into how and when they can be successfully applied by analysing the time complexity of SHHs on a variety of unimodal and multimodal problem classes.

Through a rigorous theoretical analysis, we show that while four commonly applied simple SHHs from the literature do not learn to select the most promising low-level heuristics, generalising them such that application of the chosen heuristic occurs over a longer period of time allows for vastly improved performance. Furthermore, we prove that extending the size of the set of low-level heuristics can improve the performance of the generalised SHHs, outperforming SHHs with smaller sets of low-level heuristics. We show that allowing the SHH to automatically adapt the length of the learning period may further improve the performance and outperform non-adaptive variants. SHHs selecting between two move-acceptance operators are also analysed on two classes of multimodal benchmark functions. An analysis of the performance of simple SHHs on these functions provides insights into the effectiveness of the presented methodologies for escaping from local optima.

Table of contents

List of Algorithms

Part I

Introduction and Background

Chapter 1

Introduction

1.1 Overview and Motivation

Optimisation problems are some of the most important problems in Mathematics, Computer Science and Engineering. The aim of optimisation problems is to identify the best candidate solution (or set of candidate solutions) with regards to some mathematical measure from a set of feasible solutions. Mathematically, given some function $f : X \to \mathbb{R}$ from some set X to the real numbers, the goal is to find some $x^* \in X$ such that $f(x^*)$ is maximal (for a maximisation problem; vice-versa for a minimisation problem), that is, $f(x^*) \geq f(x)\ \forall x \in X$.

Examples of optimisation problems include the Minimum Spanning Tree (MST) problem, whereby a minimal-weight subset of edges of a connected weighted graph is sought, such that each vertex is connected to at least one edge. Since each solution can be represented as a subset of the set of edges, the problem can be thought of as optimising the function $f : \{0,1\}^n \to \mathbb{R}$ (where a 1-bit in the bit-string corresponds to a certain edge being included in the spanning tree, and a 0-bit corresponds to the edge not being in the spanning tree) which measures the quality of each solution.

Ideally, given an optimisation problem, computer scientists will design a problem-specific algorithm that will find high quality solutions efficiently on all instances of that problem. For example, consider Kruskal's algorithm which provably finds a globally optimal solution in $O(E \log(E))$ steps for every instance of the MST problem, where E is the number of edges in the graph (Cormen et al., 2009). However, often it is not possible to design an efficient algorithm that provably solves a problem because the problem is either not well understood or information about the fitness landscape is unavailable. In such cases, general-purpose algorithms may be applied. General-purpose algorithms are designed to be effective on problems where not much information is known. In particular, all that is often necessary for a problem-independent algorithm to be applied is a way of representing candidate solutions and a way of comparing their quality. Many successful applications of general-purpose algorithms have been reported (Neumann and Witt, 2010).

One of the most important optimisation paradigms found in the natural world is the idea of 'Natural Selection' (or 'Survival of the Fittest'). Introduced by Darwin (1859), the idea of natural selection is that populations 'evolve' over the course of generations through the process of variation of shared characteristics and the 'fitter' individuals (those with a higher chance of reproductive success) will propagate and breed in future generations. Inspired by these ideas, Turing (1948) proposed the idea of 'Evolutionary Search'. The research field known as 'Evolutionary Computation' uses these ideas and takes inspiration from natural optimisation to create automated problem solving and optimisation techniques (Eiben and Smith, 2015).

Given an optimisation problem, a decision still has to be made towards which algorithm to use and how to set its numerous parameters. Indeed, it is well understood than no general-purpose algorithm will be efficient over all problems (see the No Free Lunch theorem of Wolpert and Macready (1997)). Furthermore, each algorithm will have considerably different performance according to how its many parameters are set. For example, in Evolutionary Computation these parameters include (but are not limited to) population size, mutation rate and selection pressure. Setting these parameters well is not trivial and the best choices will vary depending on the function to optimise. In particular, parameter values that lead to good performance on one problem may lead to poor performance on another problem.

Traditionally, two main approaches have been taken to decide which algorithm and related parameters to use for a problem. The first approach has been to perform a series of sophisticated trial-and-error tests on sets of parameter values for trial problem instances. Obviously, this approach is extremely tedious and provides no guarantees about the quality of the ultimately selected parameters. For instance, new questions arise such as how long should the algorithms be allowed to run on the problem in the trial phase and how many problem instances (and which ones) should be used. Furthermore, different algorithms and different parameter values may be more or less effective at different stages of the optimisation process. It is very difficult, if not impossible, to capture this knowledge and understanding through trial-and-error approaches. As a result, the subfield known as 'Parameter Tuning' has risen in order to provide answers to such questions (Eiben et al., 1999). A more recent approach has been to theoretically analyse the performance of different algorithms for a given problem and identify the parameter values than minimise the algorithm's expected optimisation time. However, performing such analyses already for simplified Evolutionary Algorithms, such as the (1+1) Evolutionary Algorithm ((1+1) EA), has proved to be a non-trivial task (e.g., requiring several years to identify that $1/n$ is the optimal mutation rate of the (1+1) EA for linear functions, where *n* is the problem size (Witt, 2013)). Naturally, this approach is prohibitive, if not impossible, for each application of realistic bio-inspired algorithms to sophisticated real-world problems.

Rather than deciding in advance which algorithm to use and how to set its inherent parameters for a problem, the field of hyper-heuristics aims to automate the process by using evolutionary principles to evolve the optimal algorithm for the problem (Cowling et al., 2001). A hyper-heuristic is a heuristic search methodology that acts on the space of heuristics, as opposed to the solution space, to generate or select low-level optimisation heuristics. Using a hyper-heuristic methodology removes the difficult task of manually selecting and configuring the algorithm for the problem at hand. There have been many successful applications of hyper-heuristics for complex combinatorial optimisation problems such as scheduling and timetabling (Burke et al., 2010b; Cowling et al., 2001). However, foundational questions remain regarding their effectiveness and generality. In particular, the following high level questions arise:

- For which classes of problems can a hyper-heuristic evolve an optimal (or good enough) algorithm?
- How long does it take a hyper-heuristic to evolve the optimal (or good enough) algorithm for a particular problem?
- Can a hyper-heuristic evolve an algorithm for a particular problem that can be applied successfully to other problem classes?

Two main classes of hyper-heuristics exist: 'generation' and 'selection' hyper-heuristics (Burke et al., 2010b). Generation hyper-heuristics (or offline hyper-heuristics) aim to evolve new heuristics from components of existing heuristics. Selection hyper-heuristics (or online hyper-heuristics) select, using some high-level methodology, which of a set of low-level heuristics to apply at each decision point of the optimisation process. In this thesis we will develop the theoretical foundations for the analysis of selection hyper-heuristics. Concerning this class of algorithms, we will address the following natural questions:

- Can a selection hyper-heuristic perform better than its low-level heuristics?

In particular, different heuristics (or different parameter settings) may perform well at different stages of the optimisation process so no given heuristic may be the best at every stage of the process.

- Is the performance of the selection hyper-heuristic as good as that of the best heuristic for the problem?

The very few preliminary rigorous performance analyses of hyper-heuristics from the literature (Alanazi and Lehre, 2014, 2016; Lehre and Özcan, 2013; Qian et al., 2016) do not to provide answers to the above questions. In this thesis we provide (often positive) answers to these questions by performing theoretical runtime analyses to derive bounds on the expected runtime (the number of fitness function evaluations performed before the global optimum is found) of simple hyper-heuristics commonly used in the literature. In particular, our main goal is to identify classes of problems where selection hyper-heuristics considerably outperform their constituent heuristics and rigorously prove that they have better expected runtime than well-established general-purpose algorithms from the literature, such as stochastic local search, Evolutionary Algorithms and Simulated Annealing. Since the generation hyper-heuristic approaches are widely different to the selection hyper-heuristic approaches, in this thesis we focus on the latter. According to Burke et al. (2013):

> "This type of studies (*i.e., theoretical studies*) is important for bridging the gap between theory and practice. It is crucial to have theoretical support motivating the development of selection hyper-heuristics."

1.2 Thesis Outline

This thesis is structured as follows.

In Chapter 2 we introduce the field of Evolutionary Computation, including Evolutionary Algorithms (EAs) and the mathematical methods commonly used for their runtime analyses. We discuss problem-independent search heuristics in detail in Section 2.1, noting the difference between problem-specific and problem-independent search heuristics. In particular, we emphasise the necessity and importance of general-purpose heuristics for combinatorial optimisation. In Section 2.2, EAs are introduced as an example of a large class of widely used problem-independent search heuristics. We introduce the various components of EAs, including stochastic operators such as selection, mutation and recombination. Different EAs use different operators and the performance of each operator may change drastically depending on the characteristics of the optimisation problem. Most operators come with parameters which need to be set appropriately. Different parameter choices may dramatically influence the performance of each operator and good choices often depend on the problem at hand. We describe some of the most common settings for EAs, including operator combinations leading to the well-known Genetic Algorithms. In Section 2.3, we introduce trajectory-based heuristics. The major simplification is that rather than using a population of solutions, they evolve just one (i.e., an individual). While they are considerably simplified models of EAs, they have allowed the design of general mathematical techniques for the analysis. The development of such techniques has gradually allowed the analysis of realistic EAs using populations. In this thesis we will set up the theoretical foundations of hyper-heuristics (HHs) by analysing these trajectory-based search heuristics. Similarly to the history of the theory of EAs, we predict that our analyses will allow the development of techniques to enable future analyses of realistic HHs (see e.g., (Epitropakis and Burke, 2018) for examples of state-of-the-art HHs). In Section 2.4 we introduce some of the most common benchmark problems used to showcase the behaviour of search heuristics for optimising functions with certain properties. These will be used throughout the thesis to evaluate the performance of HHs. Section 2.5 introduces the field of runtime analyses of EAs and randomised search heuristics. In particular, the most common and effective techniques for the analysis of the expected runtime of EAs will be introduced. These include techniques derived from classical probability theory (including tail bounds) in Subsection 2.5.1 and techniques designed with the runtime analysis of EAs in mind (including the general and widely applied drift analysis) in Subsection 2.5.2. The analysis tools introduced in this chapter will be at the heart of our analysis of HHs. Since the theoretical results we present may only hold for large problem sizes, we also discuss how theoretical insights may be applied to perform informative experimental analyses of HHs for realistic problem sizes in Subsection 2.5.3.

In Chapter 3 we present the hyper-heuristics. We begin with a discussion of the motivations for HHs in Section 3.1 and discuss in Subsection 3.1.1 their classification by Burke et al. (2010b) that mainly distinguishes between offline generation HHs and online selection HHs. In this thesis we focus on selection HHs, which select online from a set of low-level heuristics which one to apply at each decision point of the optimisation process. The HHs aim to learn which heuristic to apply by considering both past performance and expected future performance. In Section 3.2 we present some of the more commonly applied selection HH methodologies, including four that were introduced by Cowling et al. (2001) which will form the basis of our analysis in later chapters. Selection HHs are comprised of two main components: heuristic selection methodologies (which we discuss in Subsection 3.2.1), which select the appropriate heuristic from a set of existing low-level heuristics, and move-acceptance operators (which we discuss in Subsection 3.2.2), which decide whether or not to accept the result of the application of the selected heuristic. In Subsection 3.2.3 we report successful applications of selection HHs to complex combinatorial optimisation problems such as scheduling, timetabling, packing and vehicle routing. While there are numerous examples of successful HHs in the literature, the theoretical understanding of HHs lags far behind. In Section 3.3 we outline the currently available theoretical results on HHs, which lay the foundations for the theoretical results contained in this thesis. In order to set the foundations for the future construction of the necessary runtime analysis techniques required to analyse the most sophisticated HHs used in the literature, we will analyse the most simple selection HHs from the literature. We aim to show that good performance can also be achieve with simple HHs. These simple HHs can also be thought of as high-level online parameter setting methodologies. Hence, our results are of independent interest also for the Parameter Setting and Parameter Control communities. We discuss the related areas of Parameter Setting in Section 3.4. We discuss offline Parameter Tuning in Subsection 3.4.1 and online Parameter Control in Subsection 3.4.2.

We start presenting our main contributions in Chapter 4. Since the idea behind selection HHs is to learn which of a set of low-level heuristics is the best choice to apply in the current area of the search space, smart learning techniques are necessary. Cowling et al. (2001) introduced four 'simple' HH learning mechanisms (namely 'Simple Random', 'Random Gradient', 'Greedy' and 'Permutation') which were subsequently analysed theoretically by Alanazi and Lehre (2014). These simple mechanisms make a decision in every iteration on which heuristic to apply (i.e., they can be considered as Reinforcement Learning with the shortest memory length). However, an experimental analysis by Alanazi and Lehre (2014) suggested that trialing the chosen heuristic for a single iteration is not long enough to learn whether or not the choice was smart. Lehre and Özcan (2013) introduced the GAPPATH function as an example where it is necessary to alternate between two operators to motivate the use of hyper-heuristics. We discuss their results and preliminary analyses in Section 4.1. We also introduce a new class of pseudo-Boolean functions, GENERALISEDGAPPATHWITH-TRAPS (GGPT, an extension of the GAPPATH function), where it is necessary to learn to prefer one operator over another to avoid getting trapped at a local optimum. Since the simple mechanisms fail to find the global optimum of this function with overwhelming probability (because they do not learn to use the appropriate heuristic), we introduce a generalised HH framework in Section 4.2 where the chosen heuristic is applied for a 'learning period' of length τ, which is set prior to the run of the algorithm. The new *Generalised Greedy* (which greedily chooses a heuristic and applies it for τ iterations) and *Generalised Random Gradient* (which randomly chooses a heuristic and keeps applying it for τ iterations, restarting the period when a success occurs) HHs are used to show that the inclusion of this learning period is necessary for efficient performance on certain problem classes. In particular, we show that the generalised HHs are able to find the global optimum of GGPT with overwhelming probability, given a correct choice of the learning period.

Once we have shown that it is necessary to equip the HHs with a learning period, we see in Chapter 5 that a correct choice of the length of the learning period can lead to the best-possible performance achievable with the low-level heuristics available (i.e., 1BITFLIP and 2BITFLIP) for the LEADINGONES benchmark function. We call such performance 'optimal' since it is the best that can be achieved with the low-level heuristics available - of course, other algorithms with different components may be faster. In particular, Generalised Random Gradient can outperform its low-level heuristics and match the performance of any other best-possible algorithm using combinations of the same heuristics, up to lower order terms. We begin by proving the best performance achievable on LEADINGONES with combinations of 1BITFLIP and 2BITFLIP in Section 5.1. In Section 5.2 we rigorously prove that the four simple HH mechanisms introduced by Cowling et al. (2001) and analysed in Chapter 4, choosing between these two mutation-based heuristics, all have the same expected runtime ($\approx 0.549n^2$) on LEADINGONES, up to lower order terms. This result implies that they all essentially choose heuristics at random in each iteration. Hence, we provide a rigorous proof of what was conjectured experimentally by Alanazi and Lehre (2014). Furthermore, they exhibit worse performance than the standard RLS algorithm, which is one of the constituent heuristics. On the other hand, the two generalised HHs were designed to better appreciate the performance of each heuristic. In Section 5.3 we rigorously prove that they are able to outperform the simple mechanisms, RLS and the (1+1) EA. Furthermore, Generalised Random Gradient exhibits the best performance achievable with the available low-level heuristics ($\approx 0.423n^2$), up to lower order terms. We complement the theoretical results of the chapter with experimental results that show the effectiveness of the HHs for realistic problem sizes in Section 5.4.

Based on the insights gained in Chapters 4 and 5, we present a new HH called *Generalised Greedy Gradient* in Chapter 6. The Generalised Greedy Gradient HH combines the learning aspects of the Generalised Greedy and Generalised Random Gradient HHs. In particular, Generalised Greedy Gradient makes a greedy heuristic choice and then exploit the chosen heuristic so long as it is successful. We present theoretical results on its performance in Section 6.1. We see in Subsection 6.1.1 that Generalised Greedy Gradient is able to perform efficiently on the GAPPATH and GGPT functions on which the simple HHs failed. Furthermore, we see in Subsection 6.1.2 that Generalised Greedy Gradient is able to achieve the same best-possible performance as Generalised Random Gradient ($\approx 0.423n^2$) with the low-level heuristics available, up to lower order terms, on LEADINGONES. The theoretical results on LEADINGONES are complemented with experiments in Section 6.2 which show the performance of the HHs as the length of the learning period increases. In particular, we see that Generalised Greedy Gradient outperforms the Generalised Random Gradient HH for the same values of the learning period, suggesting that Generalised Greedy Gradient is preferable in practice.

In Chapter 7 we switch our focus to selection operators rather than mutation. We present the first theoretical analyses of selection-based HHs for multimodal optimisation, where the low-level heuristics use different elitist and non-elitist selection operators, commonly referred to as move-acceptance operators in the HH literature. The chapter considerably extends the work of Lehre and Özcan (2013) which is discussed in Section 7.1. They presented a stochastic local search move-acceptance hyper-heuristic (MAHH) which, after each mutation, accepts the new solution regardless of fitness with probability p (i.e., applies the ALLMOVES acceptance operator) and with probability $1-p$ only accepts strict improvements (i.e., applies the ONLYIMPROVING acceptance operator). The capabilities of MAHH on the unimodal benchmark function class ONEMAX are presented in Section 7.2. We identify the range of possible p values for which MAHH performs efficiently (matching the asymptotic performance of RLS and the (1+1) EA). Moreover, we see that if p is too large, then MAHH exhibits at least exponential expected runtime for ONEMAX. Thus, throughout the rest of the chapter we use MAHH with a p value that allows efficient hillclimbing and analyse it when it is also required to escape local optima. As an example where MAHH performs efficiently on a multimodal function, we present an analysis for the CLIFF benchmark function class in Section 7.3. We see that MAHH is able to find the global optimum of the function in the best possible expected asymptotic runtime achievable for unary unbiased search heuristics on any problem where the CLIFF function is very hard for elitist mutation-based algorithms. For easier instances (where the local optima are not too far from the global optimum), MAHH still outperforms by far the elitist (1+1) EA and the well-studied non-elitist METROPOLIS algorithm. The JUMP benchmark function class is presented in Section 7.4 as an example where MAHH is inefficient, giving exponential expected runtime for the large majority of instances. However, MAHH still outperforms well-established non-elitist heuristics such as the METROPOLIS algorithm.

In Chapters 4, 5 and 7, the HHs only choose between two low-level heuristics. In practical applications, HHs are given a choice of many more heuristics to select from. In Chapter 8, we consider the simple and generalised mutation-based HHs introduced in Chapter 4, yet we extend the set of low-level heuristics from size 2 to some arbitrary constant size k. In particular, we consider k mutation heuristics. In Section 8.1 we prove the best expected performance attainable by an algorithm using k mutation operators. We prove that the HHs without a learning period show worse performance with the increase of operators in Section 8.2, while on the other hand, in Section 8.3, Generalised Random Gradient is proven to have improving performance with the increase in operators, matching the leading constant in the best-possible expected runtime with the available heuristics. In particular, we prove that Generalised Random Gradient choosing from k low-level heuristics is able to outperform any algorithm, including the best-possible algorithm, that uses less than k of the available heuristics. In Section 8.4 we see experimentally for different problem sizes that Generalised

Greedy Gradient exhibits the same improved performance as Generalised Random Gradient, while the performance of Generalised Greedy deteriorates with increased sizes of heuristic sets. However, it still outperforms RLS for up to $k \leq 5$ operators.

In the previous chapters, the HH's learning period was fixed during the run. Although there are often wide asymptotic ranges to choose from to achieve optimal performance, it is not necessarily obvious how to set this prior to the run of the HH. Furthermore, the best value of the learning period for the problem at hand might change during the optimisation process. The work presented in Chapter 9 further generalises the Generalised Random Gradient HH by allowing it to adapt the learning period on the fly throughout the optimisation process. In particular, we incorporate an adaptive rule to automatically update the length of the learning period within the HH framework. The resulting, more general HH, called *Adaptive Random Gradient*, is introduced in Section 9.1. The update rule is inspired by the 1/5th rule commonly used in continuous optimisation where the mutation rate is increased or decreased with the aim of obtaining one successful mutation out of every five. We consider a heuristic to be successful if it attains σ fitness improvements in τ steps. In our adaptive rule, the value of the learning period is increased if the chosen heuristic is unsuccessful during the learning period, and decreased by a factor σ times geometrically smaller if it is successful. Thus, the aim is to achieve σ successes for each failure, giving a $1-1/\sigma$ success ratio. The value of σ is easier to set than the value of the learning period, as a wider range of possible values allow optimal performance for LEADINGONES. In Section 9.2, we prove that Adaptive Random Gradient, equipped with the $1-1/\sigma$ success rule, achieves optimal performance for LEADINGONES when choosing between two stochastic mutation-based heuristics, up to lower order terms. Furthermore, we see that the value of the learning period is updated successfully and the best value for the learning period is non-static. Hence, a static value of the learning period is not the best choice, justifying the necessity of the adaptive update rule. We supplement the theoretical results with experiments in Section 9.3 that show that Adaptive Random Gradient can outperform Generalised Random Gradient with the best static τ value found in the experimental analysis in Chapter 5. Furthermore, the best low-level heuristic is used throughout the optimisation process with a high probability, as desired.

Chapter 10 concludes the thesis by providing a critical overview of the presented results and a discussion of the future of the field of the theoretical analysis of HHs.

1.3 Thesis Contributions

This thesis considerably extends the foundational understanding of selection hyper-heuristics for combinatorial optimisation. In this section, we highlight its main contributions.

- We prove that four commonly used 'simple' selection hyper-heuristics cannot learn to prefer certain heuristics throughout the optimisation process and thus fail to find the global optimum of a pseudo-Boolean function class (called GGPT), where learning is necessary, with overwhelming probability.
- The same four hyper-heuristics are shown to have the same expected performance on the benchmark function LEADINGONES, implying that they all essentially choose heuristics at random.

These results are achieved by a careful analysis up to the leading constant in the expected runtime. The four simple hyper-heuristics are not capable of appreciating how well a low-level heuristic performs, because the heuristics are not given the opportunity to showcase their capabilities. For this reason, we extend the amount of time a chosen heuristic is allowed to run and call this amount of time the 'learning period'. The idea is that the *generalised* hyper-heuristics (as we refer to them) can now measure the performance of the low-level heuristics over a larger period of time, thus better appreciating their quality in the current area of the search space.

- We show that the two hyper-heuristics (i.e., Generalised Greedy and Generalised Random Gradient) which contain a sufficiently large learning period, are able to learn to prefer certain operators and can thus find the global optimum of GGPT with overwhelming probability.
- We prove that the hyper-heuristics equipped with a learning period of appropriate length are able to optimise LEADINGONES very efficiently, and Generalised Random Gradient is able to achieve the best-possible expected runtime achievable on LEADINGONES with the available low-level heuristics.

This result allows us to appreciate the effectiveness of the inclusion of the learning period and that the generalised hyper-heuristics are preferable to well-established algorithms already for simple unimodal functions.

By combining the learning methodologies of the two introduced generalised hyper-heuristics, we introduce a new hyper-heuristic: Generalised Greedy Gradient.

- We prove that Generalised Greedy Gradient is able to achieve the best-possible performance achievable with the two mutation-based low-level heuristics available on LEADINGONES. Experimentally, Generalised Greedy Gradient outperforms the two previous generalised hyper-heuristics.

Many theoretical analyses of hyper-heuristics to date have considered hyper-heuristics in their simplest settings, choosing between two low-level heuristics. We present the first time-complexity analyses of selection hyper-heuristics choosing between sets of low-level heuristics larger than two, as used in practice.

- We prove that the inclusion of more heuristics leads to improved performance of the generalised hyper-heuristics on the LEADINGONES benchmark function and that Generalised Random Gradient choosing from a set of k low-level mutation-based operators is faster than the best-possible algorithm that uses less than k mutation operators.

The inclusion of the learning period within the hyper-heuristic framework introduced new questions, including the problem of how to set it correctly to achieve efficient performance. We present an 'adaptive' hyper-heuristic (Adaptive Random Gradient) which uses an innovative update rule to adapt the size of the learning period throughout the run of the hyper-heuristic.

- We prove that Adaptive Random Gradient achieves the best achievable performance on the LEADINGONES benchmark function with the low-level heuristics available, choosing the 'optimal' low-level heuristic with high probability in each iteration. Experimentally, the hyper-heuristic outperforms the previous generalised hyper-heuristics with static learning periods.

Most previous analyses of hyper-heuristics have focused on heuristic selection mechanisms that select between different variation operators as low-level heuristics. We present the first analysis of selection hyper-heuristics choosing between different move-acceptance operators for multimodal optimisation.

- We prove that a hyper-heuristic choosing between elitist and non-elitist acceptance operators is able to outperform a range of elitist and non-elitist algorithms, including the well-studied METROPOLIS algorithm, on two classes of benchmark multimodal functions (i.e., CLIFF and JUMP).

Additionally, many of the analysis methods used in this thesis are innovative and have not been applied in such a way before. In particular, many of the previous theoretical analyses of hyper-heuristics present asymptotic runtime results, whereas we present tight bounds on the leading constants of the expected runtimes of hyper-heuristics on a wide range of pseudo-Boolean benchmark functions.

1.4 Underlying Publications and Joint Work

The contents of this thesis are based on the following publications, for which I am lead author. I wrote the papers and proofs after discussions and help from coauthors. Author's names are sorted alphabetically.

Chapter 4 is based on the following paper (Lissovoi et al., 2017):

1. Lissovoi, A., Oliveto, P. S., and Warwicker, J. A. (2017). On the Runtime Analysis of Selection Hyper-heuristics for Pseudo-Boolean Optimisation. In *Proceedings of the Annual Conference on Genetic and Evolutionary Computation (GECCO '17)*, pages 849-856. ACM.

Chapters 5 and 8 are based on its extension (Lissovoi et al., 2019):

2. Lissovoi, A., Oliveto, P. S., and Warwicker, J. A. (2019). Simple Hyper-heuristics Control the Neighbourhood Size of Randomised Local Search Optimally for LeadingOnes. *Evolutionary Computation* (to appear).

Chapter 7 is based on the following paper (Lissovoi et al., 2018):

3. Lissovoi, A., Oliveto, P. S., and Warwicker, J. A. (2018). On the Time Complexity of Algorithm Selection Hyper-heuristics for Multimodal Optimisation. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI '19)*, To appear.

Chapter 9 is based on the following paper (Doerr et al., 2018):

4. Doerr, B., Lissovoi, A., Oliveto, P. S., Warwicker, J. A. (2018). On the Runtime Analysis of Selection Hyper-heuristics with Adaptive Learning Periods. In *Proceedings of the Annual Conference on Genetic and Evolutionary Computation (GECCO '18)*, pages 1015-1022. ACM.

Chapter 2

Evolutionary Computation for Combinatorial Optimisation

Randomised search heuristics (RSHs) have been used successfully to solve optimisation problems for a number of years in cases where efficient problem-specific algorithms do not exist or are too difficult to design. In this chapter, we give an overview of the field of general purpose RSHs for combinatorial optimisation. Given an optimisation problem, it is difficult to decide which algorithm to apply and how to set its inherent parameters and operators such that the algorithm will deliver optimal, or good enough, performance. As a proof of concept, we discuss Evolutionary Algorithms (EAs) in detail. EAs are randomised search heuristics which aim to 'evolve' a population of candidate solutions towards the global optimum by iteratively applying stochastic variation operators. Different EAs are defined by their parameters and operators, and different combinations of parameters and operators give rise to different variations of EAs. Furthermore, different choices of parameters will lead to different performances on the same problem instances. We will discuss the role of the inherent parameters and operators of EAs in Section 2.2 and define precisely the heuristics considered in the rest of this thesis in Section 2.3. In order to evaluate the performance of hyper-heuristics throughout the thesis, we analyse them on various benchmark functions with significant landscape characteristics. Such analyses allow us to derive insights into problem classes where the hyper-heuristics are efficient or inefficient. We present the benchmark functions used for the analysis throughout this thesis in Section 2.4 with a discussion of the characteristics they present and important results from the literature of EAs applied to the same functions.

Throughout this thesis we seek bounds on the 'expected runtime' (i.e., the expected time until a globally optimal solution has been produced) of hyper-heuristics and RSHs such as EAs. A variety of mathematical and probabilistic techniques for the runtime analysis of

RSHs are available in the literature. In Section 2.5 we will discuss the ones used in this thesis in detail with relevant examples which we will build upon in further chapters.

2.1 Problem-Independent Search Heuristics

Traditionally in Computer Science, given a computational problem an algorithm is designed to effectively solve the problem. If for every input the algorithm returns the correct output then the algorithm is said to be *correct*. If the time required to solve the problem scales as a polynomial function of the size of the problem then the algorithm is said to be *efficient* at solving that problem. Conversely, if the required time is a superpolynomial function of the problem size then the algorithm is said to be *inefficient* on that problem.

Examples of efficient algorithms are Kruskal's and Prim's algorithms for finding minimum weight spanning trees in graphs which run in $O(E \log E)$ computation time (where E is the number of edges of the graph) (Cormen et al., 2009). Another example is Mergesort which sorts a sequence of n elements in $\Theta(n \log n)$ computational steps (Cormen et al., 2009). Apart from being efficient, Mergesort also runs in the best time possible for comparison-based sorting algorithms. In this sense, the algorithm is 'optimal'.

However, it is not always possible to design a provably correct and efficient algorithm for the problem at hand. For instance, unless P = NP (which is widely considered to be unlikely), there exist no efficient algorithms for any NP-hard problems. Traditionally, approximation algorithms have been sought in cases such as these. Such algorithms guarantee 'nearly optimal' solutions efficiently for the worst-case input. Since the design and analysis of correctness of approximation algorithms is tailored towards the worst-case input, it is often unclear how fast the algorithms are in general. Furthermore, it is difficult to compare the performance of two different approximation algorithms beyond worst-case performance.

In other cases, the problem may be not as well-understood as classical NP-hard problems from combinatorial optimisation. This is typical for industrial scheduling and routing optimisation problems which change over time and include many more constraints than well-studied problems. In such cases there may not be enough time or it may be too expensive to design a provably correct and efficient algorithm. Finally, the exact problem specification may not be available while only the quality of candidate solutions may be accessible. In such cases, the design of a problem-specific algorithm is even impossible. Such a scenario is modelled by the fields of black-box complexity (Droste et al., 2003) and query complexity (Immerman, 1999).

In these cases, general-purpose heuristics (i.e., heuristics that are not designed to solve a specific problem) are often applied. These are algorithms that can be applied to any optimisation problem, as long as the quality of different candidate solutions can be compared, with the hope of quickly gaining a solution of reasonable quality. Traditional problem-independent search heuristics include local search methods (Michiels et al., 2007), Tabu search (Glover, 1986) and Simulated Annealing (Kirkpatrick et al., 1983). The No Free Lunch theorem of Wolpert and Macready (1997) roughly states that over the space of all possible problems, the average performance of all non-revisiting black-box algorithms (those without any problem-specific knowledge incorporated, and that do not revisit already tested solutions) is the same. Nonetheless, it is important to understand on which classes of problems families of general-purpose heuristics perform efficiently and on which classes they perform inefficiently.

More recently, general-purpose heuristics inspired by biological processes have appeared and have been applied successfully to optimisation problems. Examples of bio-inspired search heuristics used to solve complex computational problems include Ant Colony Optimisation (Dorigo and Stützle, 2004) (which takes inspiration from the search and communication behaviour of ants) and Artificial Immune Systems (Castro et al., 2002) (which take inspiration from the principles of the immune system). The most popular subset of bio-inspired computation is Evolutionary Computation (and Evolutionary Algorithms). As an example of a class of general-purpose heuristics, we discuss Evolutionary Algorithms in detail in the next sections.

2.2 Evolutionary Algorithms

Evolution as a natural process can be seen as a way of optimising (or adapting) certain species to be the best they can be by propagating the best-suited genes in future generations. This evolutionary paradigm easily lends itself to the field of optimisation. *Evolutionary Algorithms* (or EAs) are a subclass of bio-inspired algorithms which take inspiration from the ideas of evolution and survival of the fittest (Darwin, 1859). By repeatedly applying stochastic *variation* and *selection* operators to a *population* of *individuals*, EAs aim to evolve it towards the solution of optimal quality (referred to as the *fitness* of the population) using the ideas of Darwinian variation and natural selection. EAs are general-purpose heuristics and are quite easy to implement. Given a representation of individuals (i.e., candidate solutions) and a way of comparing the quality of such individuals (i.e., by assigning to each solution some fitness value), an EA can be applied to the computational problem at hand.

Taking inspiration from the biological sciences, a lot of the terminology regarding EAs is used synonymously with the biological nomenclature. All EAs share specific characteristics and the way they are implemented defines the different types of EAs. All EAs are initialised

Algorithm 1 Overview of an Evolutionary Algorithm - $(\mu(+/,)\lambda)$ EA

```
1: Initialise a population of $\mu$ individuals
2: repeat
3:     Select parents from the population
4:     Create $\lambda$ new individuals through variation of individuals
5:     Select $\mu$ individuals for the next generation
6: until termination condition;
```

with a *population* of *individuals* (candidate solutions), each with an assigned *fitness* based on their solution quality. *Variation* operators, such as the *recombination* of two individuals or *mutation* of an individual, will act on the *parent* population to produce a new population of *offspring* solutions and the next *generation* of individuals is selected from the parent and offspring populations in the *selection* phase. The variation operators will introduce diversity within the population and the selection operators will act to increase the quality of solutions in the population (Eiben and Smith, 2015). Through the repeated application of variation and selection, the population of individuals is guided towards solutions of higher fitness. Algorithm 1 and Figure 2.1 describe this process.

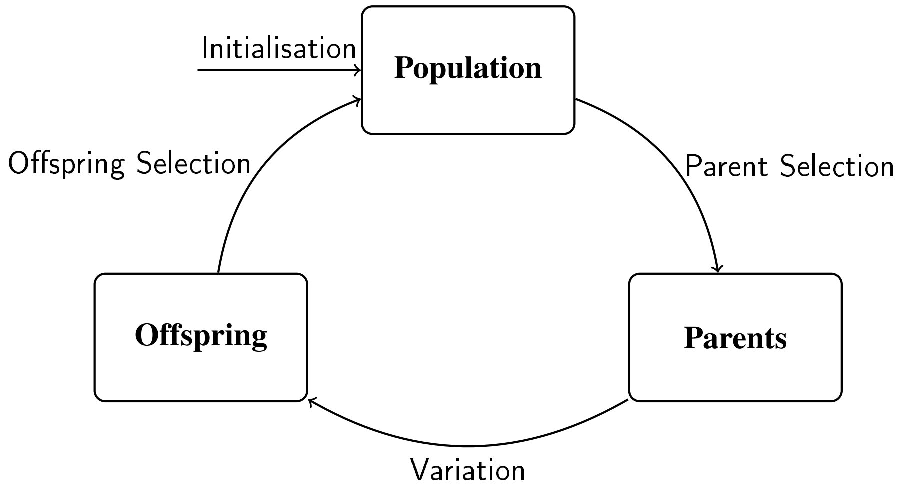

Fig. 2.1 Overview of the process of an Evolutionary Algorithm.

We now discuss the most common choices for each of the steps of an EA (i.e., lines 1,3,4,5,6 of Algorithm 1).

Initialisation (Line 1): Since EAs are general-purpose heuristics that are often applied when there is little to no a priori knowledge of the fitness landscape, initialising the population is usually a simple process. One of the most common initialisation techniques is random initialisation, where candidate solutions are generated uniformly at random.

Selection (Lines 3 & 5): Selection mechanisms choose which parents from the population are selected for reproduction, and which individuals in the population are selected for the next generation. Examples include uniform selection, where individuals are selected at random, or rank selection, where individuals with higher fitness have a better chance of being selected.

Recombination (Line 4): The recombination operator (sometimes referred to as *crossover*) is a variation operator that acts on two or more individuals of the population and creates one or more offspring individuals. Along with mutation, recombination acts on the genotype space and thus depends on the representation of the individuals. The most common recombination operator is uniform crossover which acts similarly to sex in biological reproduction, whereby two parent individuals produce one offspring individual. The genotype of the offspring individual comprises of information chosen uniformly at random from the genotype of each parent. The idea of recombination is for the offspring individual to inherit beneficial properties from their parents.

Mutation (Line 4): Mutation is a unary variation operator which slightly perturbs the genotype of one individual to create a new individual. The idea of mutation is to introduce diversity into the population and explore areas of the search space that cannot be found by recombination operators.

Termination Condition (Line 6): The termination condition decides when to end the run of the algorithm. Typically, this is either after some fixed, predetermined length of time, or after some criteria is met; for example, if the fitness has reached a certain level. In the theoretical study of EAs, the termination criterion is undefined and researchers are interested in the first point in time when the algorithm samples an individual with a specific fitness; usually the global optimum. However, fixed-budget analyses exist where the goal is to find the expected solution quality after a certain number of iterations or fitness function evaluations (see e.g., Jansen and Zarges 2012).

Different classes of EAs are obtained according to the way solutions are represented and the choice of selection and variation operators.

In Genetic Algorithms (GAs) (Holland, 1975), individuals are typically represented by discrete-valued vectors (i.e., combinatorial optimisation). Crossover is used as the main variation operator and mutation rates are fixed throughout the run. Stochastic selection (e.g., rank selection) is used to select parents for reproductions and all offspring have a non-zero probability of being selected for future generations (i.e., the selection pressure).

On the other hand, in Evolution Strategies (ESs) (Schwefel, 1975) which are used for continuous optimisation, individuals are represented as real-valued vectors which are mutated using Gaussian perturbation. The mutation rate (i.e., step sizes) are adapted during the run, with the aim of decreasing the rate as the algorithm approaches the optimum.

By using more sophisticated representations, more complex structures may be evolved. For instance, Evolutionary Programming (Fogel et al., 1966) using finite state machines to represent solutions has been used to evolve chess programs (Fogel et al., 2004). In Genetic Programming (Koza, 1992), solutions are represented as tree structures with the aim of evolving computer programs.

As well as deciding on which representation and operators will be used for the problem at hand, there are also numerous related parameters that have to be set (e.g., parent population size, offspring population size, crossover rate, mutation rate, selection pressure). Different parameter configurations for an EA on a specific problem or class of problems can lead to drastically different performance and thus it is important to set them correctly. In the next chapter, we will introduce hyper-heuristics as methodologies to automate the choice of which heuristic to apply and related parameter settings with the aim of removing such a burden from the user.

2.3 Trajectory-Based Heuristics

In this section, we introduce the heuristics which will be used throughout this thesis as low-level components of hyper-heuristics. They are all *trajectory-based heuristics* that evolve only a single individual as opposed to using a population of individuals (see e.g., Nallaperuma et al. 2019). In the context of Evolutionary Algorithms, consider Algorithm 1 with the parameter setting $\mu = 1$. Naturally, if there is no parent population, it is not possible to recombine individuals in the search space. Thus, any evolution must be created through repeated applications of mutation and selection. Traditional trajectory-based heuristics such as local search, Simulated Annealing and Tabu search have been reported to be very effective in practice, yet their performance is not necessarily well understood (Hoos and Stützle, 2004; Jansen, 2011). Compared to EA paradigms which involve the repeated variation and selection of populations of candidate solutions, trajectory-based EAs present simplified stochastic processes since the progress of only one solution has to be tracked to analyse their behaviour.

The analysis of trajectory based EAs (i.e., (1+1) EA (Droste et al., 2002), (1+λ) EA (Jansen et al., 2005)) has allowed to gradually build mathematical techniques for the analysis of more sophisticated EAs (e.g., $(\mu + 1)$ EA (Witt, 2006)). Nowadays, the analysis of realistic GAs as used in practice is possible (Corus et al., 2018). In this thesis we follow a similar

Algorithm 2 Trajectory-Based Heuristic

```
1: Initialise $x \in S$
2: repeat
3:     $x' \leftarrow \textsc{Mutate}(x)$
4:     $x \leftarrow \textsc{Select}(x, x')$
5: until termination condition;
```

approach. By analysing simple hyper-heuristics equipped with ‘simple’ low-level heuristics we aim to setup the foundations with the aim of developing techniques which in the future may be extended to the analysis of more sophisticated hyper-heuristics.

Algorithm 2 gives a basic framework for trajectory-based search heuristics over some search space S.

Different trajectory-based heuristics differ by how the new points in the search space are generated and by how the next candidate solution is selected. We refer to these respectively as the mutation and selection operator in order to be consistent with the EA nomenclature.

Mutation Operator

When mutating bit-strings, there are two main types of mutation operators:

- **Local Mutations** - In a local mutation, the parent solution is perturbed slightly such that the mutated individual remains within some neighbourhood of the parent individual. These operators are used by traditional heuristics such as local search and Simulated Annealing.
- **Global Mutations** - Global mutations consist of perturbing the genotype of the parent solution such that any other individual within the search space can be created with a non-zero probability. These operators are typical within bio-inspired optimisation algorithms such as Evolutionary Algorithms.

Although with both mutation operators small perturbations are likely, global mutations can reach the entire search space. Local mutations can instead only create solutions within some neighourhood of the original solution. Hence, only using local mutations leads to trajectory-based heuristics that are unable to escape from local optima if elitist selection is used. Restart strategies are often used to mitigate this problem.

Throughout this thesis, we will analyse search heuristics using local mutation operators. We will further analyse the role of the mutation operators and why it is important to use different mutation-based heuristics in different areas of the search space in Chapters 4, 5, 6, 8 and 9.

Selection Operator

The selection operator in a trajectory-based heuristic is used to decide whether the mutated solution should replace, or not replace, its parent. In the context of trajectory-based Evolutionary Algorithms, the two main selection operators are:

- **Elitist Selection** - the mutated offspring is selected for the next generation if its fitness is greater (or greater or equal) to the fitness of the parent.

Within the framework of hyper-heuristics, the following elitist move acceptance operators are used. The ONLYIMPROVING (OI) operator will only accept mutated offspring with a strictly higher fitness than the parent solution. The IMPROVINGANDEQUAL (IE) operator is the equivalent of the standard plus selection operator where only mutated offspring with higher or equal fitness to the parent are accepted.

- **Non-elitist Selection** - the offspring is selected for the next generation according to some probability. In particular, such selection operators can accept worsening moves with a non-zero probability. This probability may depend on the fitness difference between the parent and offspring solution.

In hyper-heuristics, the ALLMOVES (AM) operator accepts all moves independent of fitness. Move acceptance operators which accept all improving moves, and accept worsening moves with some probability based on the fitness decrease, are known as MONTECARLO (MC) mechanisms (Ayob and Kendall, 2003; Özcan et al., 2008).

A well-known fitness-dependent selection operator appears in the well-known METROPOLIS algorithm (Metropolis et al., 1953). The selection operator will accept all improving moves (whereby the offspring solution has higher fitness than the parent solution), yet will accept worsening moves with probability $\alpha(n)^{f(y)-f(x)}$ for some function $\alpha(n) \geq 1$, where n is the problem size, and $f(x)$ and $f(y)$ are the fitness of the parent and offspring solution, respectively. The function $\alpha(n)$ is typically referred to as the *temperature*.

2.3.1 Algorithmic Definitions

In this thesis, we analyse the performance of trajectory-based heuristics applied to maximisation problems on the class of pseudo-Boolean functions $f : \{0,1\}^n \to \mathbb{R}$, where each individual is represented as a bit-string in the set $S = \{0,1\}^n$. Each point of the set $S = \{0,1\}^n$ is assigned a real number as its objective value, and the goal is to find some individual $x \in \{0,1\}^n$ that maximises $f(x)$. The binary search space $S = \{0,1\}^n$ is one of the three most used search space representations (Jansen, 2013) (as well as the real space and permutation space). We will only consider the binary search space $\{0,1\}^n$ within this thesis.

Algorithm 3 Random Walk

```
1: Initialise $x \in \{0,1\}^n$
2: repeat
3:     $x \leftarrow$ Flip one bit of $x$ chosen uniformly at random.
4: until termination condition;
```

Algorithm 4 Randomised Local Search (RLS)

```
1: Initialise $x \in \{0,1\}^n$
2: repeat
3:     $x' \leftarrow$ Flip one bit of $x$ chosen uniformly at random.
4:     if $f(x') \geq f(x)$ then
5:         $x \leftarrow x'$
6:     end if
7: until termination condition;
```

By implementing the aforementioned mutation and selection operators in different ways within Algorithm 2 , we detail some commonly used trajectory-based heuristics which will be studied in this thesis. We begin with the well-known Random Walk algorithm (Algorithm 3) which uses local mutations and accepts all new solutions regardless of their fitness.

The Random Walk algorithm does not evolve the search towards any optimal solutions since candidate solutions are not compared and as such has no practical applications (other than baseless exploration). By adding elitism (i.e., plus selection or IE), we get the well-known Randomised Local Search (RLS) algorithm (Algorithm 4).

Extensions of RLS include variants which increase the neighbourhood size of the local mutations. We will consider RLS$_k$ (Algorithm 5) and KBITFLIP (Algorithm 6) throughout this thesis. The difference between the two is that RLS$_k$ flips k distinct bits (and thus mutates to a solution with a Hamming distance of k from the parent), while kBITFLIP can possibly flip the same bit twice within the k bit-flips.

While RLS with local mutations is efficient at hillclimbing (making improvements to the candidate solution by repeated perturbation of the solution), it is unable to escape from local optima. For this reason it is often used in combination with a restart (or parallel run) strategy. If instead of local mutations, the algorithm uses global mutations, we get the well-known $(1+1)$ Evolutionary Algorithm ((1+1) EA) (Algorithm 7).

The $(1+1)$ EA is the simplest Evolutionary Algorithm and its analysis by Droste et al. (2002) was one of the first rigorous theoretical works regarding EAs in the literature. A mutation rate of $1/n$ (where each bit of the bit-string is flipped with probability $1/n$) is widely used and provably the best for linear functions, giving expected runtime $en\log n \pm \Theta(n)$ in all cases (Witt, 2013). However, different fitness functions might require a different mutation

Algorithm 5 RLS$_k$

```
1: Initialise $x \in \{0,1\}^n$
2: repeat
3:     $x' \leftarrow$ Flip $k$ distinct bits of $x$ chosen uniformly at random.
4:     if $f(x') \geq f(x)$ then
5:         $x \leftarrow x'$
6:     end if
7: until termination condition;
```

Algorithm 6 kBITFLIP

```
1: Initialise $x \in \{0,1\}^n$
2: repeat
3:     $x' \leftarrow$ Flip $k$ bits of $x$ chosen uniformly at random.
4:     if $f(x') \geq f(x)$ then
5:         $x \leftarrow x'$
6:     end if
7: until termination condition;
```

rate to achieve optimal performance (for example, the best static mutation rate for the (1+1) EA on the unimodal benchmark function class LEADINGONES is $\approx 1.59/n$ (Böttcher et al., 2010)).

Another well-known trajectory-based heuristic is the METROPOLIS algorithm (Algorithm 8) (Metropolis et al., 1953), which is a Monte Carlo algorithm (Motwani and Raghavan, 1995) where worsening moves are accepted with a probability relating to the fitness difference between the parent and offspring solutions, and a constant function $\alpha(n)$.

As a non-elitist algorithm, METROPOLIS is able to escape local optima by accepting worsening moves. If the rate at which worsening moves are accepted changes over time (usually it is preferable to accept worsening moves early in the optimisation process for better exploration, while less preferable towards the end when exploitation of good solutions is desired), then we get the well-known Simulated Annealing algorithm (Kirkpatrick et al., 1983).

In the next section, we discuss the role of benchmark functions in the study of the performance of heuristic search algorithms and present the ones that will be used in this thesis. The aim behind the design of the benchmark functions is to present simple landscapes that capture characteristics that are expected to appear in real world applications. In particular, they allow to isolate function characteristics where a given heuristic is efficient against those where it is not. Thus, it is important to analyse trajectory-based heuristics on simple

Algorithm 7 $(1+1)$ Evolutionary Algorithm ((1+1) EA)

```
1: Initialise $x \in \{0,1\}^n$
2: **repeat**
3:     $x' \leftarrow$ Flip each bit of $x$ uniformly at random with probability $1/n$.
4:     **if** $f(x') \geq f(x)$ **then**
5:         $x \leftarrow x'$
6:     **end if**
7: **until** *termination condition*;
```

Algorithm 8 Metropolis Algorithm (Metropolis et al., 1953)

```
1: Choose $x \in \{0,1\}^n$ uniformly at random
2: Set $\alpha(n) \geq 1$
3: **while** termination criteria not satisfied **do**
4:     $x' \leftarrow$ Flip one bit of $x$ chosen uniformly at random.
5:     $\Delta f \leftarrow f(x') - f(x)$
6:     **if** $\Delta f \geq 0$ **then** $x \leftarrow x'$
7:     **else**
8:         Choose $r \in [0,1]$ uniformly at random
9:         **if** $r \leq \alpha(n)^{\Delta f}$ **then** $x \leftarrow x'$
10:        **end if**
11:    **end if**
12: **end while**
```

benchmark functions in order to gain an understanding of how well they are expected to perform in a variety of real-world situations.

2.4 Standard Benchmark Problem Classes

Throughout this thesis, we will analyse the performance of a variety of trajectory-based heuristics on pseudo-Boolean fitness functions $f : \{0,1\}^n \to \mathbb{R}$, where elements of the Boolean hyper-cube are assigned a real number as their fitness. Figure 2.2 shows a two-dimensional mapping of the Boolean hyper-cube. The vertical position of each bit-string is determined by the number of 1-bits in the bit-string, and the horizontal position of each bit-string is determined by the number of 1-bits before the first 0-bit.

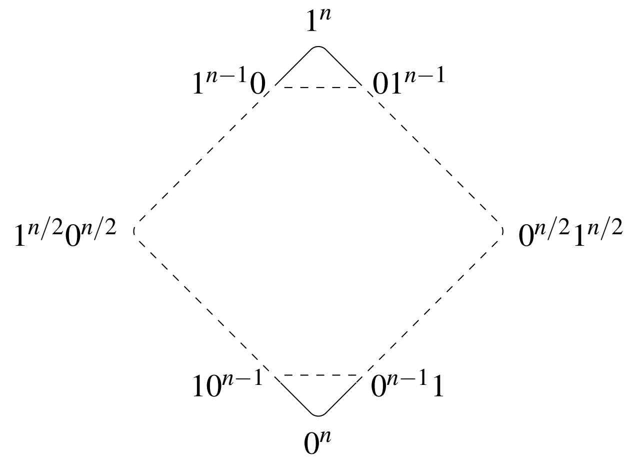

Fig. 2.2 Two-dimensional mapping of the Boolean hyper-cube.

When analysing the performance of heuristics to optimise pseudo-Boolean functions, it is often helpful to consider a metric for the search space (i.e., a measure on how similar or different two elements of the search space are). The most common metric is to consider the number of differing bit positions in the bit-string. This is known as the *Hamming distance* of two bit-strings. We can also consider the idea of a *Hamming Neighbourhood*, which is a set of bit-strings with a Hamming distance of 1 to a desired bit-string.

Definition 1 (Hamming Distance) *Given two bit-strings $x, y \in \{0,1\}^n$, the Hamming distance between them is given by* $\mathrm{H}(x,y) = \sum_{i=1}^{n} |x_i - y_i|$. *The Hamming neighbourhood of a bit-string x comprises of all the bit-strings $x^\star$ where* $\mathrm{H}(x,x^\star) = 1$.

Using this metric, we can formalise the idea of *local optima*, which are bit-strings with no Hamming neighbours of higher fitness, and *global optima*, the bit-strings with the highest fitness in the search space. Global optima are in fact local optima with the highest fitness.

Definition 2 (Local and Global Optima) *Given a function $f : \{0,1\}^n \to \mathbb{R}$, the set of local optima is given by* $\{y \,|\, x,y \in \{0,1\}^n, \mathrm{H}(x,y) = 1 \implies f(y) \geq f(x)\}$. *The global optimum is given by* $x_{\text{opt}} = \text{argmax}\{f(y) \,|\, y \in \{0,1\}^n\}$.

Since theoretically analysing randomised search heuristics on real-world combinatorial problems was prohibitive, initial analyses considered simple functions. This would allow researchers to build up a variety of mathematical and probabilistic techniques for the analysis which could then be applied to more complicated fitness function landscapes. Furthermore, these *toy problems* often have significant properties for understanding the behaviour and performance of randomised search heuristics (Oliveto et al., 2007). Any results stemming from the analysis of randomised heuristics on such toy problems, either positive or negative, helps us to understand on which classes of problems certain algorithms perform well or not. Moreover, since the functions are well-understood, it is not too hard to track the performance of randomised search heuristics when applied to such a function. While the global optimum of these functions is often found at the bit-string 1^n, the black-box algorithms we will analyse have no knowledge of this (and hold no bias on flipping 0-bits or 1-bits or flipping bits in certain positions). However, our knowledge of the location of the global optima allows us to better understand how the heuristic progresses. We now introduce some of the main classes of toy problems.

Linear functions have often been used to analyse the performance of randomised search heuristics on easy optimisation problems.

Definition 3 (Linear Functions) *A function* $f : \{0,1\}^n \to \mathbb{R}$ *is called a linear function if*

$$f(x) = C + \sum_{i=1}^{n} W_i \cdot x_i$$

for some constants $C, W_i \in \mathbb{R}$.

We often assume (without loss of generality) that $C = 0$, and note that if $W_i > 0 \; \forall \; i$, then $x_{\text{opt}} = 1^n$. The (1+1) EA optimises this class of functions in expected $en \log n \pm \Theta(n)$ in all cases (Witt, 2013). One of the most well-studied linear functions is OneMax, which has all weights set to 1 ($W_i = 1 \; \forall i$).

Definition 4 (OneMax) *Let* $x \in \{0,1\}^n$. *Then*

$$\text{ONEMAX}(x) = \sum_{i=1}^{n-1} x_i.$$

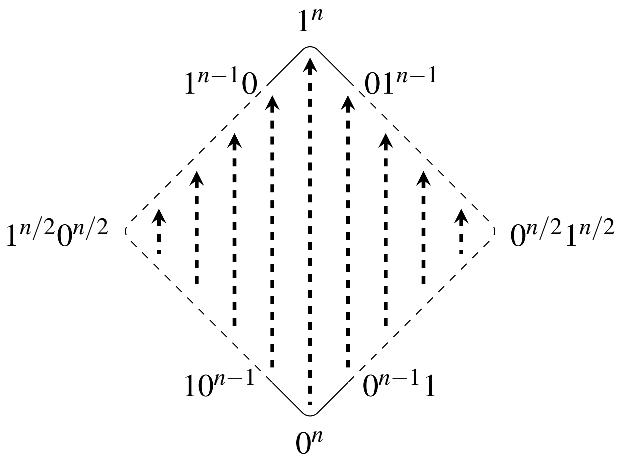

Fig. 2.3 Two-dimensional mapping of the ONEMAX fitness function on the Boolean hypercube. Arrows represent the direction of increasing fitness.

The ONEMAX problem class (Figure 2.3) provides a consistent fitness gradient to the global optimum. As is typical in optimisation problems, improving solutions are harder to identify as the global optimum is being approached. ONEMAX is used as a benchmark function to evaluate the hillclimbing ability of randomised search heuristics. Lehre and Witt (2012) proved that the ONEMAX class is the easiest (i.e., is solved fastest in expectation) among all functions with a unique global optimum for unary unbiased black-box algorithms (i.e., mutation-based EAs), and its black box complexity is $\Omega(n \log n)$ for unary unbiased black-box algorithms (Lehre and Witt, 2012) (i.e., algorithms using only operators that act on a single solution with no bias for 0-bits or 1-bits cannot beat the $\Omega(n \log n)$ expected runtime). We will derive a bound of $O(n \log n)$ for the (1+1) EA on ONEMAX in the Section 2.5 as an example application of modern runtime analysis techniques. Anil and Wiegand (2009) proved an upper bound (which was later proved to be tight) of $O(n/\log n)$ for the generalised black-box complexity of ONEMAX (see (Droste et al., 2003) for a rigorous definition of black-box complexity).

For analytical convenience, the global optimum of the ONEMAX function is placed at the 1^n bit-string, such that the fitness increases with the number of 1-bits in the candidate bit-string. Any results derived on ONEMAX will hold for any instance of the function class, i.e., the optimum is set to some bit-string and the fitness function returns the number of matching bit positions between the global optimum and the candidate solution. We call such a function class GENERALISEDONEMAX. However, since the algorithms we study are unbiased and do not favour flipping 0-bits or 1-bits (i.e., are unbiased), we only study ONEMAX for ease of analysis.

Definition 5 (GENERALISEDONEMAX) *Let $x, x_{opt} \in \{0,1\}^n$. Then*

$$\text{GENERALISEDONEMAX}(x) = n - \mathrm{H}(x, x_{opt}).$$

Linear functions also belong to the class of unimodal functions, which are functions with exactly one local optimum.

Definition 6 (Unimodal Functions) *A function f is called unimodal if and only if $\forall x \neq x_{opt}$, $\exists y$ with $\mathrm{H}(x,y) = 1$ and $f(y) > f(x)$.*

Mühlenbein (1992) postulated that the (1+1) EA could solve all unimodal functions in expected $O(n \log n)$ time. This was disproved by Droste et al. (2002) who presented a unimodal function (called PATH$_k$) for which the (1+1) EA takes exponential expected time ($\Theta(n^{3/2}2^{\sqrt{n}})$) to optimise.

Along with ONEMAX, one of the most well-studied unimodal functions is LEADINGONES, which counts the number of leading 1-bits in a bit-string before the first 0-bit. The LEADINGONES function was introduced by Rudolph (1997).

Definition 7 (LEADINGONES) *Let $x \in \{0,1\}^n$. Then*

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

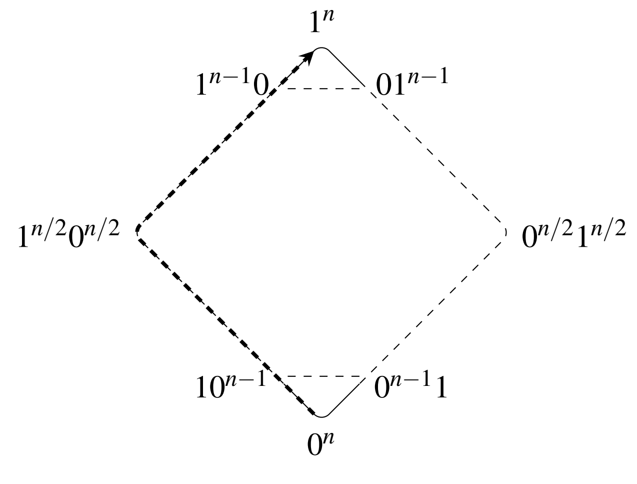

Fig. 2.4 Two-dimensional mapping of the LEADINGONES fitness function on the Boolean hyper-cube. The arrow represents increasing fitness.

The LEADINGONES benchmark function (Figure 2.4) was presented as a difficult function for the (1+1) EA, since in expectation it takes the (1+1) EA longer to optimise than all linear

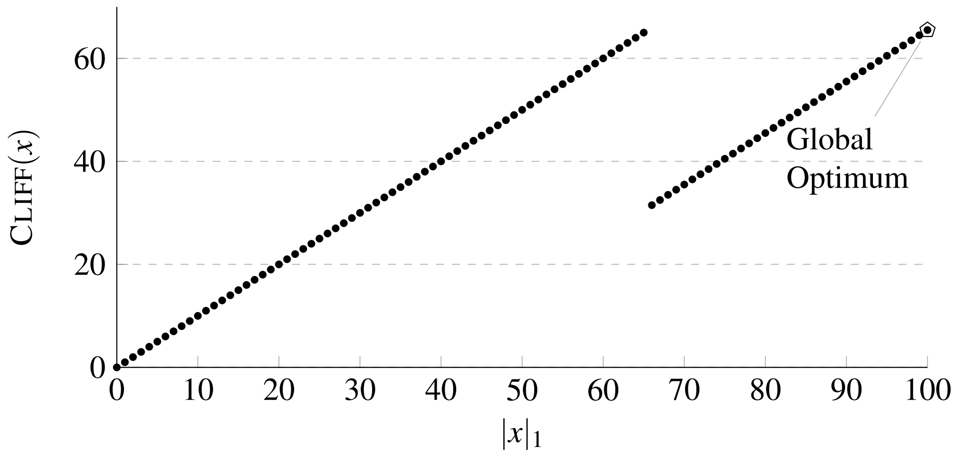

Fig. 2.5 An example of $\text{CLIFF}_d(x)$ with $n = 100$ and $d = 35$.

functions (i.e., the (1+1) EA has expected runtime $\Theta(n^2)$ on LEADINGONES) (Droste et al., 2002). Improving solutions are found only by mutating the first 0-bits after the leading 1-bits (and leaving the prefix 1-bits undisturbed). Lehre and Witt (2012) proved that the unary unbiased black-box complexity of LEADINGONES is $\Omega(n^2)$. However, in the general case the black-box complexity of LEADINGONES is $O(n \log\log n)$ (Afshani et al., 2013). In this thesis we will often use LEADINGONES as a unimodal function to show how hyper-heuristics can be very fast in the exploitation of heuristics.

The final class of fitness function we consider during this thesis is multimodal functions, which are a class of functions with more than one local optima. Multimodal functions allow us to analyse the capability of EAs at escaping local optima. We now present two simple, similar multimodal problem classes that differ in whether the fitness gradient after the local optima guides the search toward or away from the global optimum.

The CLIFF class of functions (Figure 2.5) was proposed as an example problem where non-elitist evolutionary algorithms outperform elitist ones (Jägersküpper and Storch, 2007).

$$\text{CLIFF}_d(x) := \begin{cases} \text{ONEMAX}(x) & \text{if } |x|_1 \leq n - d, \\ \text{ONEMAX}(x) - d + 1/2 & \text{otherwise.} \end{cases}$$

CLIFF functions lead the optimisation process towards the local optima through an increasing fitness gradient, from which a fitness-decreasing mutation can be taken to find another fitness-improving slope which leads to the global optimum. The parameter d (usually $1 < d < n/2$) controls the fitness decrease from the local optimum to the new slope and the length of the second slope. Jägersküpper and Storch (2007) proved that while the elitist (1+λ) EA has runtime larger than $n^{n/4}$ with overwhelming probability to optimise CLIFF

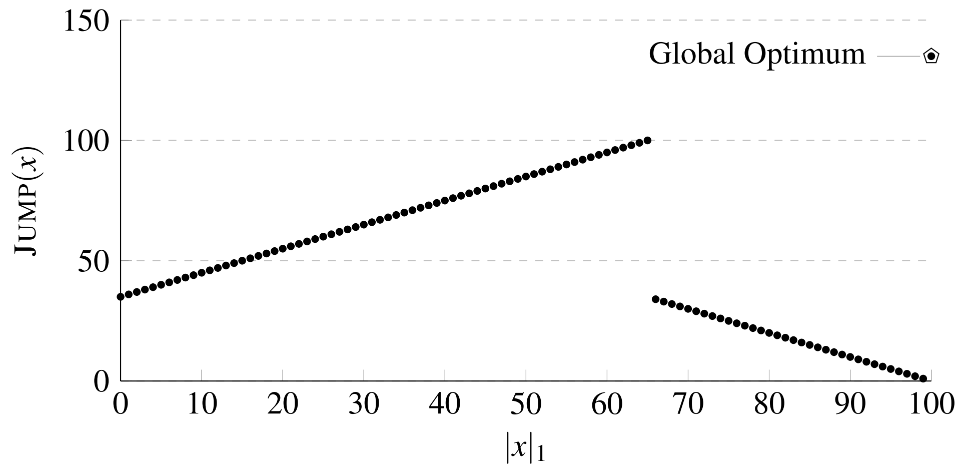

Fig. 2.6 An example of JUMP$_m(x)$ with $n = 100$ and $m = 35$.

(with $d = n/3$), the non-elitist (1,λ) EA was proved to have expected $O(n^{25})$ runtime on the same instance. Mutation based EAs (such as the (1+1) EA) have an expected runtime of $\Theta(n^d)$ (for $d \leq n/2$) (Jägerskküpper and Storch, 2007) (i.e., exponential in the size of the cliff).

The JUMP class of functions (Figure 2.6) is similar to CLIFF, yet both fitness gradients lead towards the local optima. The JUMP function was introduced by Jansen and Wegener (2002) as an example where crossover-based algorithms outperform mutation-based algorithms.

$$\text{JUMP}_m(x) := \begin{cases} m + \text{ONEMAX}(x) & |x|_1 \leq n - m \text{ or } |x|_1 = n, \\ n - \text{ONEMAX}(x) & \text{otherwise.} \end{cases}$$

To reach the global optimum of JUMP, it is necessary to not only accept a fitness-decreasing mutation from the local optimum to the second slope, but also disregard the fitness gradient in further iterations. The parameter m (usually $1 < m < n/2$) acts similarly to the parameter d in the CLIFF function. The expected runtime of the (1+1) EA on JUMP$_m$ is $\Theta(n^m + n \log n)$ (Droste et al., 2002), while a steady state GA with crossover and a diversity mechanism that allows crossover to effectively recombine the required bits can optimise the function in expected polynomial time for $m = O(\log n)$ (Jansen and Wegener, 2002). The best expected runtime proven for a standard GA without diversity is $O(n^{m-1})$ (Dang et al., 2017). In Chapter 7, we will show when, by switching between elitist and non-elitist selection strategies, hyper-heuristics can be very effective at escaping from local optima for multimodal functions.

In order to understand and compare the performance of different EAs (or EAs with differing parameter values), we must be able to present rigorous results on their expected runtime. In the next section, we introduce the methods used in this thesis for the time-complexity analyses.

2.5 Runtime Analysis of Evolutionary Algorithms

General-purpose search heuristics have been applied succesfully for a number of years. Despite these successes, rigorous answers to the following questions are required for a greater understanding:

- How long (in expectation) does it take for a given algorithm to optimise a given function?
- On which classes of functions are a given algorithm efficient/inefficient?
- How does the setting of the algorithm's parameters affect its performance?

The field of theoretical analysis of stochastic search algorithms (such as EAs) aims to provide rigorous answers to these questions. The goal is to provide tights bounds on the expected runtime of stochastic algorithms on a variety of problem classes. Although initial theoretical studies of Evolutionary Algorithms proved performance results for simple algorithms applied to simple problems (Droste et al., 2002), nowadays analysis of standard randomised search heuristics such as Genetic Algorithms (Corus et al., 2018) and Artificial Immune Systems (Jansen et al., 2011) are possible as well as analyses of standard NP-hard problems from combinatorial optimisation (Neumann and Witt, 2010).

In this thesis, we will perform rigorous runtime analyses to answer our research questions regarding the performance of hyper-heuristics. In this section we will introduce the standard techniques used for the analysis of EAs which we will use throughout the thesis. We will provide examples of their applications to derive results which we will build upon in the following chapters.

When analysing algorithms, we are interested in their *correctness* (i.e. do they converge to the global optimum) and *runtime* (i.e., how long will it take to reach the global optimum). The property of convergence is not (usually) difficult to show. All that needs to be shown is that any solution can be reached from the current population of solutions with a non-zero probability (e.g., with global mutations), and that once it is in the population, it will remain there forever (e.g., with elitism) (Rudolph, 1998). Analysing the runtime of an algorithm, however, is a more difficult task and requires a series of mathematical and analytical tools.

In the field of runtime analysis, the main research question is: when does the algorithm first sample the required solution (i.e., the global optimum)?

Definition 8 (Optimisation Time) *Let $\{X_t\}_{t\geq 0}$ be a stochastic process on the state space $\{0,1\}^n$ and $f : \{0,1\}^n \to \mathbb{R}$ a fitness function. The optimisation time* T *is the first point in time when the maximal fitness value is sampled; i.e.,*

$$\mathrm{T} := \min\left\{ t \geq 0 \mid X_t = \underset{X_t \in \{0,1\}^n}{\arg\max}\, f(X_t) \right\}.$$

The measure of time we are concerned with is the number of fitness function evaluations performed by the algorithm, as this is often the most computationally expensive step of each iteration in real world applications. Since we consider randomised algorithms, the exact optimisation time will change with each run of the algorithm. Hence, the aim is to find the *expected* optimisation time (or *expected runtime*) of an algorithm, which is the mean number of fitness function evaluations required until the global optimum is sampled for the first time (i.e., when it appears in the population for the first time). The expected optimisation time is often presented in asymptotic notation with regards to the problem size n (see e.g., Cormen et al. 2009).

Definition 9 (Asymptotic Notation) *Let $f, g : \mathbb{N}_0 \to \mathbb{R}$ be two functions. Then*

- *$f = O(g)$ if and only if there exist positive constants c and n_0 such that $0 \leq f(n) \leq c \cdot g(n)$ for all $n \geq n_0$;*
- *$f = \Omega(g)$ if and only if $g = O(f)$;*
- *$f = \Theta(g)$ if an only if $f = O(g)$ and $f = \Omega(g)$;*
- *$f = o(g)$ if and only if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$;*
- *$f = \omega(g)$ if and only if $g = o(f)$.*

When presenting theoretical results with asymptotic notation throughout this thesis, it is assumed that $n \geq n_0$ holds for some constant $n_0 > 0$. While it is possible that the presented asymptotic results only hold for large problem sizes, we often supplement theoretical results with experiments to understand the performance of the algorithms for different problem sizes.

It is often stated that an algorithm that optimises a function in superpolynomial time is *inefficient* and an algorithm that solves a problem in polynomial time is *efficient*. We now formalise the definition of such terms.

Definition 10 (Polynomial/Superpolynomial/Exponential) *Let $f : \mathbb{N}_0 \to \mathbb{R}$ be a function. Then*

- *f is polynomial ($f = poly(n)$) if $f(n) = O(n^k)$ for some $k \in \mathbb{R}_0^+$;*
- *f is superpolynomial if $f(n) = \omega(n^k)$ for every constant $k \in \mathbb{R}_0^+$;*
- *f is exponential if $f(n) = \Omega(2^{n^\varepsilon})$ for some constant $\varepsilon \in \mathbb{R}^+$;*
- *f is polynomially small ($f = 1/poly(n)$) if $1/f$ is polynomial;*
- *f is superpolynomially small if $1/f$ is superpolynomial;*
- *f is exponentially small if $1/f$ is exponential.*

Sometimes, the expected runtime of a stochastic algorithm does not give a realistic indication of how efficient an algorithm is for a problem (i.e., the expected runtime may be much higher than the derived runtime in practice). In these cases, it may be useful to prove that the runtime will not exceed some value with high probability.

Definition 11 (Probabilistic Notions) *Let A be some event. We say that*

- *The event A occurs with high probability (w.h.p.) if $1 - \Pr(A)$ is at most polynomially small;*
- *The event A occurs with overwhelming probability (w.o.p.) if $1 - \Pr(A)$ is exponentially small.*

Now that we have outlined methods to define the success of Evolutionary Algorithms, we introduce some of the mathematical and probabilistic techniques used to achieve the necessary rigorous runtime results. While this chapter thus far presents the requirements for understanding our presented theorem statements in the latter chapters, a basic knowledge of the remainder of this chapter is necessary to understand the proofs that are used to derive these results.

2.5.1 Tools from Probability Theory

We now present some results from the fields of mathematics, statistics and probability theory that are useful for analysing stochastic search heuristics (see e.g., Doerr 2011; Feller 1968; Mitzenmacher and Upfal 2005). A basic understanding of the concepts of *probability space* and *random variables* is assumed.

We begin by stating some basic results. The following two inequalities are simple yet useful when aiming to simplify certain mathematical formulae.

Theorem 2.1 *For all $n \in \mathbb{N}_1$:*

$$\left(1-\frac{1}{n}\right)^n \leq \frac{1}{e} \leq \left(1-\frac{1}{n}\right)^{n-1}$$
$$\left(1+\frac{1}{n}\right)^n \leq e \leq \left(1+\frac{1}{n}\right)^{n+1}.$$

The next theorem bounds the probability of at least one of a series of arbitrary events occurring in some probability space.

Theorem 2.2 (Union Bound (Mitzenmacher and Upfal, 2005)) *Let $E_1, \ldots, E_n$ be arbitrary events in some probability space. Then*

$$\Pr\left(\bigcup_{i=1}^{n} E_i\right) \leq \sum_{i=1}^{n} \Pr(E_i).$$

The *first hit probability* (we will often use the following theorem in the thesis) gives the probability of one event occurring before another.

Theorem 2.3 (First Hit Probability (Mitzenmacher and Upfal, 2005)) *In any trial, let the outcome A occur with probability* $\Pr(A)$ *and the outcome B occur with probability* $\Pr(B)$. *Then in a sequence of trials, the probability that A occurs before B is*

$$\Pr(A \textit{ before } B) = \frac{\Pr(A)}{\Pr(A)+\Pr(B)}.$$

Let X be a random variable with fixed parameters $n \in \mathbb{N}$ (denoting the number of independent trials) and $p \in [0,1]$ (denoting the success probability of each trial, where each trial can either be successful or unsuccessful). Such a variable is known as a *binomial* random variable. In particular, the probability of k successes in n trials is $\Pr(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$, where $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ is the binomial coefficient. Theorem 2.4 gives the expectation of a binomial variable (i.e., the expected number of successes in n trials).

Theorem 2.4 (Mitzenmacher and Upfal 2005) *Let X be a binomial random variable describing n independent trials, each with success probability p. Then* $\mathrm{E}(X) = np$.

We also consider geometric random variables. Given a number of independent Bernoulli trials (i.e., each trial has two outcomes: success or failure) with fixed success probability p, let X be the number of trials before a success occurs. The probability of waiting k trials for success is given by $\Pr(X = k) = (1-p)^{k-1}p$. The so called *Waiting Time Argument* provides us with the expectation of a geometric random variable (i.e., the expected time for a success).

Theorem 2.5 (Waiting Time Argument (Mitzenmacher and Upfal, 2005)) *Let X be a geometric random variable with success probability p. Then* $\mathrm{E}(X) = 1/p$.

The Waiting Time Argument is used often in the analysis of Evolutionary Algorithms. Since we can often say that a stochastic mutation is 'successful' with a certain probability, the Waiting Time Argument gives the expected time for such a successful mutation to occur. A well-known problem where the Waiting Time Argument can be applied is the *Coupon Collector's Problem* (see e.g., Motwani and Raghavan 1995). In this problem, a collector is attempting to obtain a set of n different coupons and every time she purchases a coupon, she receives one uniformly at random from the set of n coupons. The question is to find the expected time until the collector has collected all n coupons. The following theorem gives this result, and the proof is from Doerr (2011).

Theorem 2.6 (Coupon Collector (Mitzenmacher and Upfal, 2005)) *The expected time to collect all coupons is nH_n, where H_n is the n^{th} Harmonic number.*

Proof: The expected time to collect a new coupon, given that the collector already has i coupons, is $\left(\frac{n-i}{n}\right)^{-1} = \frac{n}{n-i}$ by the Waiting Time Argument. Since the collector starts with 0 coupons and must collect n, the expected time to collect all n coupons is

$$\sum_{i=0}^{n-1} \frac{n}{n-i} = n \cdot \sum_{i=0}^{n-1} \frac{1}{n-i} = n \cdot \sum_{i=1}^{n} \frac{1}{i} = nH_n.$$

□

The Coupon Collector's problem is useful in the analysis of randomised search heuristics using local mutations. In particular, it is easy to find similarities with functions of unitation (i.e., functions where flipping a 0-bit into a 1-bit will change the fitness) where flipping a 0-bit into a 1-bit is akin to collecting a new coupon. As an example, see the following analysis of RLS (Algorithm 4) on OneMax.

Theorem 2.7 *Starting at 0^n, the expected time for* RLS *to optimise* OneMax *is $n \ln n + O(n)$ (Droste et al., 2002).*

Proof: Let i denote the number of 1-bits in the current solution. RLS uses local mutations and thus can only reach Hamming neighbours through mutation. The number of Hamming neighbours with more 1-bits and thus higher fitness is $n-i$ (while i have lower fitness). Thus, the probability of finding a Hamming neighbour of higher fitness in a single mutation is $\frac{n-i}{n}$. If a Hamming neighbour of worse fitness is created by mutation, it will not be accepted and the current solution will be retained. This mimics exactly the Coupon Collector's problem and thus the expected time for RLS to find the global optimum of OneMax starting with a bit-string with 0 1-bits will be $nH_n = n \ln n + O(n)$. □

When analysing a stochastic algorithm, it is important to consider every possibility of what could happen. The following theorem states that the expected value of a random variable is equal to the sum of the expected values of that random variable conditioned on a second random variable (e.g., a random variable that can be one of a finite number of events).

Theorem 2.8 (Law of Total Expectation (Mitzenmacher and Upfal, 2005)) *Let X be a random variable and $A_1, A_2, \dots, A_n$ mutually disjoint partitions of the sample space. Then*

$$\mathrm{E}(X) = \sum_{i=1}^{n} \mathrm{E}(X \mid A_i) \Pr(A_i).$$

As an example application of the Law of Total Expectation in the runtime analysis of Evolutionary Algorithms, consider analysing an EA with random initialisation on some problem. We can use the Law of Total Expectation to find the exact expected runtime, given that we are able to calculate the expected runtime from each possible point of initialisation of the search space and the probability of each initialisation occurring. Doerr and Doerr (2016) used such an approach to calculate the exact expected runtime for the (1+1) EA and RLS on OneMax and LeadingOnes. Moreover, Jansen and Zarges (2012) used the law of total expectation to prove *fixed budget* results for the (1+1) EA and RLS on OneMax and LeadingOnes. Rather than proving results on the expected runtime of randomised search heuristics, they derived results on the expected fitness after a fixed number of iterations (known as the *budget*).

In some occasions, the expected runtime does not provide a sufficiently clear picture of the algorithm's performance. As an example, consider the runtime of RLS with random initialisation on the following function.

$$\textsc{OneMaxTrap}(x) = \begin{cases} 2 & \text{if } x = 0^n, \\ \textsc{OneMax}(x) & \text{otherwise.} \end{cases}$$

The probability of initialising a solution at 0^n is 2^{-n}. However, from this point, RLS will not be able to make any improving move - the expected runtime from this position is infinite. Hence, by the Law of Total Expectation, the total expected runtime will be infinite. Nevertheless, we note that with overwhelming probability $1 - 2^{1-n}$, RLS will not initialise its random solution at 0^n (or at any bit-string with exactly one 1-bit and then flip this bit to return to the 0^n bit-string) and will be able to find the global optimum in expected conditional $O(n \log n)$ time (by a simple application of the Coupon Collector's Problem). Thus, the infinite expected runtime result does not give an overview of the true optimisation process. Although the expected runtime may be high, sometimes the algorithm may still be efficient with a high probability. We can use tail bounds to prove such results.

Tail Bounds and Deviations from the Mean

While it is often easy to make claims about the expectation of a certain event occurring (i.e., the value of a random variable or a sum of random variables), we often want to know some probability bound on the event exceeding, or falling below, its expectation. Tail inequalities allow us to derive bounds on the probability that an event deviates by a certain amount from its expectation. Regarding randomised search heuristics, tail inequalities can be used to derive bounds on the probability that the global optimum of a certain function will be reached (or will not be reached) within a certain number of iterations.

Markov's inequality gives a bound on the probability that any non-negative random variable exceeds its expectation by an arbitrary multiplicative constant.

Theorem 2.9 (Markov's Inequality (Mitzenmacher and Upfal, 2005)) *Let X be a non-negative random variable. Then for all $\lambda \geq 1$,*

$$\Pr(X \geq \lambda \cdot \mathrm{E}(X)) \leq \frac{1}{\lambda}.$$

It is often necessary to derive tighter bounds than those provided by Markov's inequality. If more information about the random variable is available, then such results may be derived. Chernoff bounds consider sums of independent random variables taking values in $[0,1]$. As an example application in the theory of Evolutionary Algorithms, consider providing bounds on the probability that a certain number of successful mutations occur within a certain number of iterations. We now present two variants of Chernoff bounds.

Theorem 2.10 (Chernoff Bounds (Mitzenmacher and Upfal, 2005)) *Let $X_1, \ldots, X_n$ be independent random variables taking value 1 with probability p_i and 0 otherwise. Let $X = \sum_{i=1}^{n} X_i$, $E(X) = \sum_{i=1}^{n} p_i$ and $0 < \delta \leq 1$. Then the following Chernoff bounds hold:*

$$\Pr(X \geq (1+\delta) \cdot \mathrm{E}(X)) \leq \exp\left(-\mathrm{E}(X) \cdot \delta^2 \cdot \frac{1}{3}\right); \tag{2.1}$$

$$\Pr(X \leq (1-\delta) \cdot \mathrm{E}(X)) \leq \exp\left(-\mathrm{E}(X) \cdot \delta^2 \cdot \frac{1}{2}\right). \tag{2.2}$$

2.5.2 Drift Analysis

One of the more general and powerful methods used to analyse the runtime of randomised search heuristics is *drift analysis*. The *drift* is the expected progress towards a target (e.g., the global optimum) achieved by the randomised search heuristic in one step given its current distance to the target. Drift analysis concerns calculating (or bounding) the drift ($\Delta(t)$) of a randomised process at each state in the optimisation process. In particular, we wish to find the expected progress of the algorithm in a single step (or iteration) and use this information to bound the expected runtime of the algorithm. Formally, given a stochastic process $\{X_t\}_{t\geq 0}$ over some search space S, and a distance function $d : S \to \mathbb{R}_0^+$:

$$\Delta(t) := \mathrm{E}[d(X_t) - d(X_{t+1}) \mid d(X_t) > 0] \tag{2.3}$$

with $\Delta(t)$ representing the expected decrease in distance to the global optimum by the algorithm at time t. Naturally, a comparison can be made with velocity in classical mechanics, which is a measure of the rate of change of position with respect to time.

Drift analysis was introduced by Hajek (1982) but was first considered for the analysis of randomised algorithms by He and Yao (2001). Naturally, it is quite difficult to obtain exact results for $\Delta(t)$, so bounds on the drift are sought. Different drift theorems are available depending on how the drift can be bounded.

Additive Drift

If we are only able to bound the drift by a constant function that holds throughout the optimisation process and does not depend on the current state (but possibly on the problem size n), i.e., $\Delta(t) \geq \delta$, we are able to use the *Additive Drift* theorem. This was first presented by He and Yao (2001).

Theorem 2.11 (Additive Drift (He and Yao, 2001)) *Let $\{X_t\}_{t\geq 0}$ be a sequence of random variables over a finite set of states S and let* T *be the random variable that denotes the first point in time for which $X_t = 0$. If there exists positive constants $\delta_u \geq \delta_l > 0$ such that for all t, X_t we have*

$$\delta_u \geq \mathrm{E}[X_t - X_{t+1} \mid X_t] \geq \delta_l,$$

then the expected optimisation time E(T) *satisfies*

$$\frac{X_0}{\delta_u} \leq \mathrm{E}[\mathrm{T} \mid X_0] \leq \frac{X_0}{\delta_l}$$

and

$$\frac{\mathrm{E}[X_0]}{\delta_u} \leq \mathrm{E(T)} \leq \frac{\mathrm{E}[X_0]}{\delta_l}.$$

An application of using the additive drift theorem is in the following example.

Theorem 2.12 *The expected runtime of the (1+1) EA on* LEADINGONES *is $O(n^2)$ (Oliveto and Yao, 2011).*

Proof: Let $X_t = n - i$ denote the distance to the global optimum at the point the current solution has i leading 1-bits before the first 0-bit. Since the (1+1) EA is elitist, worsening moves will not be accepted. An improvement is made if the first 0-bit flips to a 1-bit (with probability $\frac{1}{n}$) while leaving the current i leading 1-bits unflipped (with probability $\left(1-\frac{1}{n}\right)^i \geq \left(1-\frac{1}{n}\right)^{n-1} \geq \frac{1}{e}$). Hence, the drift in one step (with $\Delta^+(t)$ denoting the positive drift) will be

$$\Delta(t) = \mathrm{E}(\Delta^+(t)) \geq \frac{1}{n} \cdot \left(1 - \frac{1}{n}\right)^i \geq \frac{1}{en} = \delta_l.$$

Pessimistically, we assume that the algorithm is initialised with 0 leading 1-bits (i.e., $\mathrm{E}[X_0] = n$). Hence, we can apply the additive drift theorem to bound the expected time T for the (1+1) EA to optimise LEADINGONES:

$$\mathrm{E(T)} \leq \frac{n}{\frac{1}{en}} = en^2 = O(n^2).$$

$\square$

Böttcher et al. (2010) presented the following theorem which allows for precise expected runtime results for randomised algorithms on LEADINGONES; the theorem allows us to give a tighter bound on the expected runtime on LEADINGONES for the (1+1) EA.

Theorem 2.13 (Böttcher et al. 2010; Buzdalov and Buzdalova 2015; Doerr 2018a) *The expected time needed to find the optimum of* LEADINGONES *given random initialisation is*

$$\frac{1}{2}\sum_{i=1}^{n} A_{n-i},$$

where A_{n-i} is the expected time needed to find an improvement given a solution with fitness i.

Theorem 2.13 states that for the LEADINGONES benchmark function maximising the probability of success implies minimising the expected waiting times A_{n-i}. Doerr (2018a) proved recently that Theorem 2.13 holds for any unbiased (1+1) black box algorithm on LEADINGONES. The following results follow from Theorem 2.13.

Theorem 2.14 *The expected runtime of the (1+1) EA on* LEADINGONES *is* $\frac{e-1}{2}n^2 \pm o(n^2) \approx 0.859n^2$ *(Böttcher et al., 2010).*

Theorem 2.15 *The expected runtime of RLS on* LEADINGONES *is* $\frac{1}{2}n^2$ *(Buzdalov and Buzdalova, 2015).*

In Chapter 5, we will build upon these results to show that hyper-heuristics can be faster for the LEADINGONES problem.

Multiplicative Drift

If the drift is not constant throughout the optimisation process, the additive drift theorem may not provide tight runtime bounds. The *Multiplicative Drift* theorem allows us to bound the drift more carefully by considering how it changes throughout the process. If the drift depends on some function proportional to the current state of the solution, the Multiplicative Drift theorem may be applied. This technique was first introduced by Doerr et al. (2012) and extended by Doerr and Goldberg (2010). It is tailored towards applications where a logarithmic factor appears in the runtime bound (i.e., when the progress of an EA slows towards the optimum). The multiplicative drift theorem gives an upper bound on the expected runtime as well as a probability bound on the runtime.

Theorem 2.16 (Multiplicative Drift (Doerr et al., 2012)) *Let* $\{X_t\}_{t\geq 0}$ *be a sequence of random variables over a finite set of states* S *and let* T *be the random variable that denotes the first point in time for which* $X_t = 0$*. If there exists* $\delta > 0$ *such that*

$$\mathrm{E}[X_t - X_{t+1} \mid X_t] \geq \delta X_t,$$

then the expected optimisation time $\mathrm{E}(\mathrm{T})$ *satisfies*

$$\mathrm{E}(\mathrm{T}) \leq \frac{1}{\delta}\left(1 + \ln(X_0)\right),$$

and for every $c > 0$

$$\Pr\left(\mathrm{T} > \frac{1}{\delta}(\ln(X_0) + c)\right) \leq e^{-c}.$$

A bound on the drift that is proportional to the current state is often available when studying functions of unitation with unbiased mutation operators (i.e., those that do not favour flipping 0-bits over 1-bits and vice versa). An example of the use of the multiplicative drift theorem is in the following example, which tightly bounds the expected runtime of the (1+1) EA on ONEMAX.

Theorem 2.17 *The expected runtime of the (1+1) EA on* OneMax *is at most* $en(\ln(n)+1)$ *(Oliveto and Yao, 2011).*

Proof: Let $X_t = n-i$ denote the distance to the global optimum at the point the current solution has i 1-bits and $n-i$ 0-bits. To make an improvement in the OneMax function, at least one of the $n-i$ 0-bits must flip into a 1-bit (with probability $\frac{n-i}{n}$) while keeping the current i 1-bits unflipped (with probability $\left(1-\frac{1}{n}\right)^i$). Hence

$$\Delta(t) \geq \frac{n-i}{n} \cdot \left(1-\frac{1}{n}\right)^i \geq \frac{n-i}{n} \cdot \left(1-\frac{1}{n}\right)^{n-1} \geq \frac{n-i}{en} = \frac{X_t}{en}.$$

We take $\delta = \frac{1}{en}$ and use the multiplicative drift theorem with $X_{max} = n$ to derive the expected time T for the (1+1) EA to optimise OneMax:

$$\mathrm{E(T)} \leq en(1+\ln(n)).$$

□

Note that if the additive drift theorem was used to obtain the result, we would have had to bound $\delta_u \geq \frac{1}{en}$ (as in the last step such a drift is exact) which would have resulted in an upper bound of $\mathrm{E(T)} \leq en^2$. This is an asymptotically worse result than using multiplicative drift, implying that tighter bounds on the drift lead to tighter bounds on the expected runtime.

The multiplicative drift theorem has been used used to prove that $1/n$ is the best mutation rate for the (1+1) EA when optimising OneMax (Witt, 2013). This result improved on the result by Droste et al. (2002) which stated that the best mutation rate was of the form c/n for some constant c. Witt (2013) proved explicitly that the expected runtime of the (1+1) EA on OneMax with a mutation rate of c/n is $(1+o(1))\frac{e^c}{c}n\ln n$, which is clearly minimised for $c=1$.

Variable Drift

If the drift throughout the optimisation process depends on some monotone function of the state (X_t), then the *Variable Drift* theorem may be applied. Any monotone increasing function of X_t can be used to bound the drift. The following theorem was introduced by Johannsen (2010).

Theorem 2.18 (Variable Drift Theorem (Johannsen, 2010)) *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* $h(x)$ *be an in-*

tegrable, monotone increasing function on $[x_{\min}, x_{\max}]$ *such that* $\mathrm{E}(X_t - X_{t+1} \mid X_t) \geq h(X_t)$. *Then it holds for the first hitting time* $\mathrm{T} := \min\{t \mid X_t = 0\}$ *that*

$$\mathrm{E}(\mathrm{T} \mid X_0) \leq \frac{x_{\min}}{h(x_{\min})} + \int_{x_{\min}}^{X_0} \frac{1}{h(x)}\, dx.$$

As a simple example of the application of the variable drift theorem, we will consider Randomised Local Search on ONEMAX. While a similarly tight bound could have been proved with the multiplicative drift theorem, we present the use of the variable drift theorem for the purpose of understanding how it is applied.

Theorem 2.19 *The expected runtime of RLS on* ONEMAX *is at most* $n + n\ln n$ *(Lehre and Oliveto, 2018).*

Proof: Let $X_t = n - i$ denote the distance to the global optimum at the point the current solution has i 1-bits and $n - i$ 0-bits. To make an improvement in the ONEMAX function, we must flip one of the $n - i$ 0-bits into a 1-bit (with probability $\frac{n-i}{n}$). Hence

$$\Delta(t) = \frac{n-i}{n} = \frac{X_t}{n}.$$

We can bound X_t from below by 1. Hence $x_{\min} = 1$ and $h(x) = \frac{x}{n}$. We pessimistically assume we initialise with 0 1-bits (i.e., $X_0 = n$). Applying the variable drift theorem gives the result:

$$\mathrm{E}(\mathrm{T}) \leq \frac{1}{h(1)} + \int_1^n \frac{n}{x}\, dx = n + n\int_1^n \frac{1}{x}\, dx = n + n\ln n.$$

□

Negative Drift

If the drift is negative, the algorithm moves (in expectation) away from the global optimum. Given a stochastic process that has to cross an interval where the drift is negative, we can use the *Negative Drift* theorem to attempt to prove that the algorithm is inefficient for the problem at hand. The following theorem provides exponential lower bounds in the length of the drift interval on the runtime provided that the probability of performing jumps in the interval decreases exponentially with the size of the jump. We present the version that was introduced by Oliveto and Witt (2011), fixed in the erratum (Oliveto and Witt, 2012).

Theorem 2.20 (Negative Drift Theorem (Oliveto and Witt, 2011, 2012)) *Let X_t, $t \geq 0$, be the random variables describing a Markov process over some state space, and let $\delta_t(i) := (X_t - X_{t+1} \mid X_t = i)$ for $i \in S$ and $t \geq 0$. Suppose there exist an interval $[a,b]$ of the state space and two constants $\delta, \varepsilon > 0$ such that for all $t \geq 0$ the following two conditions hold:*

1. $\mathrm{E}(\delta_t(i)) \leq -\varepsilon$ *for $a < i < b$,*
2. $\Pr(|\delta_t(i)| \geq j) \leq 1/(1+\delta)^j$ *for $i > a$ and $j \geq 0$.*

Then there is a constant $c^* > 0$ such that for $\mathrm{T}^* := \min\{t \geq 0 : X_t \leq a \mid X_0 \geq b\}$ it holds $\Pr(\mathrm{T}^* \leq 2^{c^*(b-a)}) = 2^{-\Omega(b-a)}$.

The above theorem requires a region of negative drift $[a,b]$ and if the second condition also holds, the runtime will be exponential in the size of the region with a probability that increases exponentially with the size of the region. Further variants of the theorem include an extension by Rowe and Sudholt (2014) which allows for algorithms with self-loops (i.e., algorithms where the fitness of the current search point does not change with high probability). We now show a simple example of an application of the negative drift theorem, where a poor choice of the function $\alpha(n)$ (i.e., the function that decides whether worsening moves should be accepted) in the non-elitist METROPOLIS algorithm (Algorithm 8) can lead to exponential lower bounds in the runtime for ONEMAX. We will build upon this application of the negative drift theorem in Chapter 7 when comparing hyper-heuristics for multimodal optimisation problem classes against METROPOLIS.

Theorem 2.21 *Let $\alpha(n) = c'$, for some constant $c' > 0$. Then the runtime of the* METROPOLIS *algorithm on* ONEMAX *is at least $2^{c^\star\left(\frac{n}{\log n}-1\right)}$ for some constant $c^\star > 0$ with probability at least $1 - 2^{-\Omega\left(\frac{n}{\log n}\right)}$.*

Proof: Let $X_t = n - i$ represent the number of 0-bits in the bit-string at time t. Since the METROPOLIS algorithm is a local search algorithm, it will accept with probability 1 all fitness improvements, which occur with probability $\frac{X_t}{n}$, and accept any fitness worsenings with probability $\alpha(n)^{-1} = \frac{1}{c'}$, which occur with probability $\frac{n-X_t}{n}$. Hence, we can calculate the drift at the state X_t, by considering the forwards and backwards drift:

$$\Delta(t) = \frac{X_t}{n} \cdot 1 - \frac{n - X_t}{n} \cdot \frac{1}{C'} \cdot 1 = \frac{X_t}{n}\left(1 + \frac{1}{c'}\right) - \frac{1}{c'}.$$

To use the negative drift theorem, we need to define an interval $[a,b]$ and show that we satisfy conditions 1 and 2 from Theorem 2.20. Let us define the interval $[a,b]$ with $a = 1$ and

$b = \frac{n}{\log n}$. To satisfy condition 1, we must bound the drift from above by a negative constant. Clearly the drift will decrease as X_t decreases; hence it suffices to show that we can bound the drift at the point $X_t = b = \frac{n}{\log n}$:

$$\Delta(t) = -\left(\frac{1}{c'} - \frac{1}{\log n}\left(1 + \frac{1}{c'}\right)\right) \leq -C$$

for some $C > 0$, given a sufficiently large problem size n. Hence, condition 1 holds. Condition 2 holds simply because the probability of making jumps outside the Hamming neighbourhood is 0 due to the local search operator. Thus, the theorem statement holds. □

We also present the *Simplified Drift Theorem with Scaling*, which is a generalised version of the negative drift theorem that allows the negative drift ε to be sub-constant in magnitude.

Theorem 2.22 (Simplified Drift Theorem with Scaling (Oliveto and Witt, 2015)) *Let X_t, $t \geq 0$ be real-valued random variables describing a stochastic process over some state space. Suppose that there exist an interval $[a,b] \subseteq \mathbb{R}$ and, possibly depending on $\ell := b-a$, a drift bound $\varepsilon := \varepsilon(\ell) > 0$, as well as a scaling factor $r := r(\ell)$ such that for all $t \geq 0$ the following conditions hold:*

1. $\mathrm{E}(X_{t+1} - X_t \mid X_0, \ldots, X_t; a < X_t < b) \geq \varepsilon$,
2. $\Pr(|X_{t+1} - X_t| \geq jr \mid X_0, \ldots, X_t; a < X_t) \leq e^{-j}$ *for all* $j \in \mathbb{N}_0$,
3. $1 \leq r^2 \leq \varepsilon\ell/(132\log(r/\varepsilon))$.

Then for the first hitting time $\mathrm{T}^* := \min\{t \geq 0 : X_t \leq a \mid X_0 > b\}$ *it holds that* $\Pr(\mathrm{T}^* \leq e^{\varepsilon\ell/(132r^2)}) = O(e^{-\varepsilon\ell/(132r^2)})$.

We will use the Simplified Drift Theorem with Scaling in Chapter 7 to prove negative results on the performance of a hyper-heuristic which uses the ALLMOVES move-acceptance operator in each iteration with a sufficiently high probability.

2.5.3 Theory Driven Experimentation

In this thesis, we will often present supplementary experimental results to check whether our theoretical results hold for realistic problem sizes. Throughout this thesis (e.g., in Chapters 5, 8 and 9) we see experimental results for mutation-only algorithms for quite large problem sizes on the presented benchmark functions (i.e., on LEADINGONES).

In order to efficiently handle large problem dimensions experimentally, we do not simulate each individual mutation performed by the algorithm, but rather sample the waiting times for a fitness-improving mutation to occur using a geometric distribution (with the success probability p depending on the chosen mutation operator and fitness value of the current solution). The success probability p is found from a rigorous theoretical analysis and depends on the chosen algorithm and function.

Specifically, suppose $\mathrm{T_{IM}}$ is a random variable denoting the number of mutations required to achieve one fitness-improving mutation. Since $\mathrm{T_{IM}}$ counts the number of independent trials, each with probability p of success, we have $\Pr(\mathrm{T_{IM}} \leq k) = 1-(1-p)^k$ by the properties of the geometric distribution. Given access to a uniform $(0,1)$-distributed random variable U, $\mathrm{T_{IM}}$ can be sampled by computing $\left\lceil \frac{\log(1-U)}{\log(1-p)} \right\rceil$ (see e.g., Rizzo 2007).

2.6 Chapter Summary

In this chapter, we have introduced the field of randomised search heuristics (RSHs) as general-purpose optimisers. Their strength lies in their problem-independence, ease of application and stochasticity. Given a correct representation and a way of comparing candidate solutions, any optimisation problem can be formulated as a problem for a RSH to attempt to solve. We have introduced EAs as examples of sophisticated RSHs with many associated parameters, including population size and mutation rate, which must be correctly set for efficient performance. Given an optimisation problem at hand, the choice of which RSH to apply and which parameter values to use is not trivial. For example, when applying an EA, decisions must be made regarding the choice of population size, mutation rate, selection operator, and many other parameters. Rather than applying tedious trial-and-error type approaches, the field of hyper-heuristics (which will be discussed in detail in the next chapter) aims to solve this problem by automatically configuring the algorithm for the problem at hand. In Subsection 2.3.1 we introduced the heuristics that the hyper-heuristics considered in this thesis will select from and in Section 2.4 the benchmark functions that will be used for the analysis of their performance.

Throughout this thesis we will use runtime analysis techniques to answer the following important research question: how long does it take a hyper-heuristic to solve a given problem? Providing this theoretical understanding of hyper-heuristics will help practitioners understand how to apply hyper-heuristics in realistic settings. The techniques that we will use in the analysis, which include commonly used tools from probability theory (e.g., tail bounds) and techniques designed with the analysis of randomised search heuristics in mind (e.g., drift analysis), have been introduced in Section 2.5. For many of the techniques, we have presented example applications of their use that have provided runtime results for simplified algorithms on simple problem classes that will be used to compare the performance of hyper-heuristics in later chapters.

Chapter 3

Hyper-heuristics

In the previous chapter, we discussed how general-purpose randomised search heuristics (RSHs) such as Evolutionary Algorithms (EAs) can be applied in a variety of settings. However, deciding which particular search heuristic will be efficient for a particular problem is a difficult task. Furthermore, the performance of RSHs relies heavily on how the inherent parameters are set. For EAs in particular, one must specify parameters such as the *parent population size*, *offspring population size*, *mutation rate* and *crossover rate*, amongst others. Different parameter settings lead to different algorithmic configurations and different performances. Identifying high-performing configurations and parameter settings effectively for the problem at hand is a difficult task and much effort has been put into researching mechanisms and techniques to simplify or automate this task.

In this chapter, we introduce hyper-heuristics which aim to automate the problem of algorithm selection and configuration. Firstly, we will give an overview of the field of hyper-heuristics. In Section 3.1 we present a classification of the different types of hyper-heuristics. In Section 3.2 introduce the selection hyper-heuristics which are the main focus of this thesis. In Section 3.3, we give an overview of the existing theoretical results regarding the performance of hyper-heuristics that we will considerably extend upon in the remainder of the thesis. We give an overview of the related area of offline parameter setting and online parameter control in Section 3.4.

3.1 Hyper-heuristics

Hyper-heuristics (HHs) are general algorithm generation and selection methodologies. As opposed to EAs which operate on the solution space, HHs operate on the heuristic search space by evolving heuristics for the problem at hand. The goal is to either evolve heuristics (using a training set of instances) that will have good performance on a wide range of problem classes, rather than efficiently solving just one problem, or that the HH will make 'smart' online choices of which heuristics to apply to the optimisation problem at hand, thus finding the right method or sequence of heuristics, as opposed to directly solving the problem.

The main motivation of HH approaches is to automate the design and tuning of heuristic methods to solve hard computational search problems (Burke et al., 2013). Further motivations for employing HH methodologies include (but are not limited to) bypassing the issue of selecting or configuring heuristics prior to the run of the algorithm (which removes having to spend long computational time finding problem-specific algorithms, instead providing *good enough* general-purpose techniques) and that different heuristics may be effective in some parts of the search space and ineffective in others. As an example of the advantages of employing multiple heuristics, consider that local search methods are fast at hillclimbing yet unable to make large jumps in the search space (see Section 2.3).

In this section, we present a classification of the different types of HHs in the literature. The nomenclature used in this section is consistent with that of the HH literature. When there are exact parallels with the field of EAs, we point this out. For an extended survey on HHs, including extended discussions on the history of HHs and related areas, see e.g., surveys by Burke et al. (2013, 2010b); Epitropakis and Burke (2018); Özcan et al. (2008); Ross (2005).

3.1.1 Classification

Although the basic ideas of algorithm design and heuristic selection date back to the 1960s (see Burke et al. (2013) for a comprehensive history of approaches), hyper-heuristics were originally introduced by Cowling et al. (2001) as *'heuristics to choose heuristics'* in the context of combinatorial optimisation. They applied a variety of online heuristic selection mechanisms to the problem of scheduling a sales summit. Burke et al. (2010b) proposed the following update to this definition: *'A hyper-heuristic is an automated methodology for selecting or generating heuristics to solve hard computational search problems'*. Essentially, a hyper-heuristic is an algorithm acting on the domain of heuristics (as opposed to the domain of solutions) with the aim of evolving or combining heuristics effectively to solve complex optimisation problems. Chakhlevitch and Cowling (2008) defined a hyper-heuristic more elegantly with the following three criteria:

1. A hyper-heuristic is a *high-level heuristic* which manages a set of *low-level heuristics*, of cardinality greater than one.

2. A hyper-heuristic searches for a good *method* to solve a problem, rather than for a good *solution*.

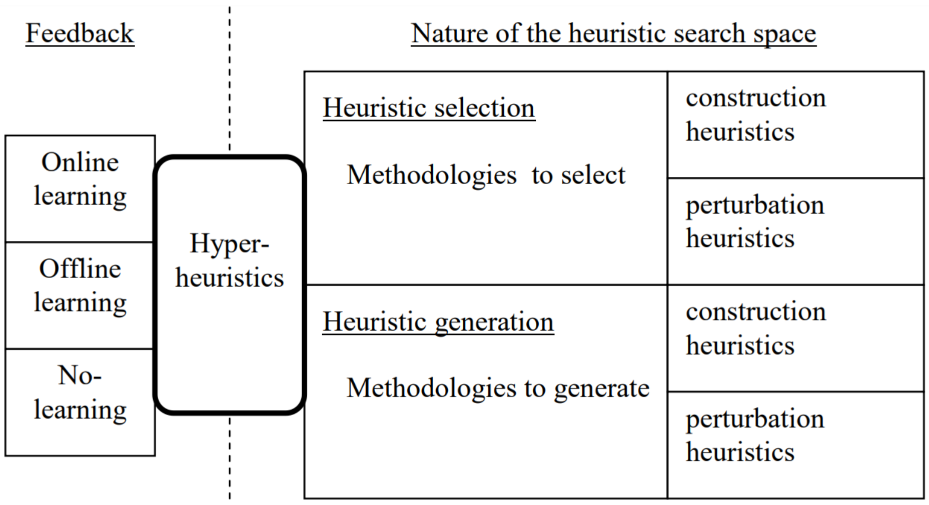

Fig. 3.1 A classification of hyper-heuristics approaches according to the nature of the heuristic search space and the source of feedback during learning (Burke et al., 2010b).

3. A hyper-heuristic uses at most limited problem-specific information.

The most common classification of hyper-heuristics was provided by Burke et al. (2010b) which classifies them based on their domains and the source of feedback. This classification is detailed in Figure 3.1 (which is reproduced here for completeness).

Hyper-heuristics are learning algorithms which use feedback from the search process. The source of feedback is classified into three subclasses: online learning (where feedback is provided while solving the problem instance and guides the hyper-heuristic), offline learning (where the hyper-heuristic gathers knowledge from a set of training instances with the aim of identifying an algorithm that is able to efficiently solve unseen instances), and no learning.

According to the nature of the heuristic search space, a distinction is made between *generation* and *selection* HHs. We discuss selection-perturbation HHs in detail in the next section.

Generation hyper-heuristics: Generation HHs search on the space of heuristics constructed from *components* of existing heuristics, rather than the space of complete, pre-defined heuristics. The goal of generation HHs is that as well as producing a solution (or a population of solutions) to an instance of an optimisation problem, the HH will also generate an efficient heuristic which can then be applied to different problem instances or similar problem classes. If the generated HH is is created for a single problem instance, it is said to be *disposable*. However, *reusable* heuristics, which are often trained offline, are intended to be successfully applied to other problem instances (Bader-El-Den and Poli, 2008). Genetic programming (Koza, 1992), which is an evolutionary computation technique that evolves a population of computer programs, is often used to generate the heuristics (see e.g., (Bader-El-Den et al., 2009) for an example for the application of timetabling).

There is a natural distinction between the low-level heuristics used within HH methodologies, as seen in Figure 3.1. *Construction heuristics* construct new solutions incrementally from partially constructed existing solutions. Starting from an initially 'empty' solution, the selection HH will intelligently select from a set of pre-existing constructive heuristics which to apply to gradually build up a complete solution. The pre-existing construction heuristics are usually problem-specific, so smart heuristic selection techniques must be applied in order to find the most suitable heuristic for the given partially constructed heuristic. As an example of proof of concept, Ahmadi et al. (2003) applied a selection-construction HH for the examination timetabling problem. The construction heuristics built up a timetable for an examination period by iteratively scheduling an appropriate exam at an appropriate time in an appropriate room, subject to the constraints imposed by the existing solution. The construction heuristics differed by their method of selection (firstly selecting an exam and then a room). The exam heuristics included scheduling the largest unscheduled exam, while the room heuristics included scheduling the exam in the smallest available room.

Since in this thesis we focus entirely on the analysis of selection-perturbation HHs, we refer the reader to the aforementioned surveys (e.g., Burke et al. 2013) for further information regarding generation HHs and construction heuristics, and we describe selection-perturbation hyper-heuristics precisely within the next section.

3.2 Selection-Perturbation Hyper-heuristics

Selection hyper-heuristics are methodologies which choose the next heuristic to be applied in the optimisation process from a set of available low-level heuristics. Whereas generation hyper-heuristics approaches are used with the aim of creating a generally good heuristic for the problem at hand, this might come at the cost of optimal performance for a particular instance. Selection hyper-heuristics, on the other hand, adapt to the instance at hand by selecting appropriate heuristics with the goal of providing a solution of the highest quality.

Throughout this thesis, the hyper-heuristics we will consider will be selection hyper-heuristic methodologies to select from sets of *perturbation heuristics*, which perturb complete candidate solutions. These approaches aim to improve a candidate solution through the repeated selection and application of low-level heuristics. Examples of such perturbation heuristics include Randomised Local Search, Evolutionary Algorithms and METROPOLIS (see Chapter 2). Hyper-heuristic approaches to selecting perturbation hyper-heuristics combine the decisions of two separate components: *heuristic selection* and *move acceptance*

Algorithm 9 Simple Selection-Perturbation Hyper-heuristic (Alanazi and Lehre, 2014; Cowling et al., 2001, 2002b; Lehre and Özcan, 2013)

```
1: Choose $s \in S$ uniformly at random
2: while stopping conditions not satisfied do
3:     Choose $h \in H$ according to the learning mechanism
4:     $s' \leftarrow h(s)$
5:     if $s'$ is accepted by the move acceptance operator then
6:         $s \leftarrow s'$
7:     end if
8: end while
```

(Bilgin et al., 2007; Özcan et al., 2008). Algorithm 9 shows the pseudocode for the application of a simple selection hyper-heuristic to optimise some function $f : S \to \mathbb{R}$ over the search space S. In the next two subsections, we discuss examples of the learning mechanisms for heuristic selection and the move acceptance operators used frequently in the literature.

3.2.1 Heuristic Selection Methodologies

Smart heuristic selection methodologies employ a *learning mechanism* to guide the heuristic selection process (see Algorithm 9, line 3). A learning mechanism is used to improve the decision making process. We now detail some of the most common learning mechanisms.

The majority of heuristic selection methods used within the literature give each heuristic $h \in H$ an online, constantly updating score based on how likely it is to lead to an improvement in the next step and the scores are used to influence the heuristic selection process (Burke et al., 2013). One of the commonly used methods within the field of hyper-heuristics is to use reinforcement learning mechanisms to update the scores of each heuristic (Kaelbling et al., 1996). When applied within a hyper-heuristic framework, a reinforcement learning technique will attempt to identify the appropriate low-level heuristic to apply in each iteration through trial-and-error applications of the heuristics. Each heuristic is usually assigned a *weight*, which is increased or decreased through some reinforcement reward or punishment scheme in each iteration. Nareyek (2004) discussed different approaches for selecting low-level heuristics using reinforcement learning techniques. The weights are updated due to a variety of reinforcement rules, depending on the success of the chosen heuristic:

- **(Escalating) Additive Adaptation** - the weights are updated by a small (possibly increasing) additive factor;

- **(Escalating) Multiplicative Adaptation** - the weights are updated by a small (possibly increasing) multiplicative factor;

- **Power Adaptation** - the weights are taken to the power of some small exponent (> 1).

Respective rules for negative reinforcement are also applied. Different heuristic selection mechanisms, such as a deterministic choice of the heuristic with the highest weight or a roulette wheel selection of the weights, are commonly used. The function used to determine which heuristic is to be selected is typically referred to as the *Choice Function*.

- **Choice Function** - A *choice function* F, which is used to measure how likely each heuristic is to be effective in the next iteration, is used to decide which low-level heuristic to apply in each iteration. The function value (F value) of each heuristic is updated in each iteration depending on its success in the previous iteration. Several choice function approaches exist. For example, Cowling et al. (2001) used the following approaches:
  - **Straight Choice** - The low-level heuristic $h = N_i$ ($1 \leq i \leq |H|$) with the largest value of F is chosen in each iteration;
  - **Ranked Choice** - The performance of a fixed-proportion of the heuristics with the highest values of F is evaluated and the heuristic which performs the best is selected;
  - **Roulette Choice** - A low-level heuristic, say N_i, is selected with probability $\frac{F(N_i)}{\sum_i F(N_i)}$;
  - **Decomp Choice** - The performance of the (up to four) low-level heuristics with the highest values of F and its three constituent components (f_1, f_2, f_3) is evaluated and the heuristic which performs the best is selected. The three components are: f_1, which evaluates the recent effectiveness of the heuristic; f_2, which evaluates the recent effectiveness of pairs of heuristics; f_3, which evaluates the amount of time since each heuristic was used.

The following learning mechanism uses a Tabu list of low-level heuristics.

- **Tabu Search (Burke and Soubeiga, 2003)** - The low-level heuristics compete against each other and their *score* is increased if their application leads to an improvement in the fitness (the score is decreased if not). The heuristic with the highest score is selected in each iteration. However, a *tabu* list is kept of the heuristics that did not lead to an improvement. If a heuristic is on the tabu list, it cannot be chosen. After some time (usually until an improvement is found), the heuristics on the tabu list are *released* and can be selected again.

Some very simple learning mechanisms have also been reported to be successful in the literature. Naturally, we will use these to begin our performance analysis of hyper-heuristics to see if good performance can be achieved without the need for an RL mechanism. The four most common ones are:

- **Simple Random (Cowling et al., 2001)** - A low-level heuristic $h \in H$ is randomly chosen (usually each with probability $1/|H|$) in each iteration;
- **Greedy (Cowling et al., 2001)** - The performance of each low-level heuristic is evaluated and the heuristic which gives the best performance is applied, given that this yields an improvement;
- **Random Gradient/Random Descent (Cowling et al., 2001)** - A low-level heuristic is randomly chosen and applied repeatedly until it does not result in an improvement;
- **Permutation/Random Permutation (Cowling et al., 2002b)** - A permutation of the low-level heuristics is created prior to the run of the algorithm and the heuristics are applied iteratively in this sequence.

3.2.2 Move Acceptance Operators

A move acceptance operator decides if the perturbed solution (resulting from the application of the chosen low-level heuristic) will replace the previous solution (or possibly some other solution if there is a population of solutions). A similar discussion regarding move acceptance operators (i.e., selection operators) within an Evolutionary Algorithm framework was presented in Section 2.3. Since hyper-heuristics are often applied to combinatorial optimisation problems (often containing many local optima), elitist variants (which always reject perturbations which decrease the fitness of the individual) do not necessarily achieve the best performance. We now describe some of the most common move acceptance operators (see, e.g., (Burke et al., 2013) for further examples). We will make a distinction between *deterministic* and *non-deterministic* move acceptance.

Deterministic move acceptance operators make the same decision of acceptance when comparing two solutions during the search process. The following deterministic move acceptance operators are often considered to be compared against when the performance of more sophisticated move acceptance operators is being assessed.

- **ALLMOVES (AM) (Cowling et al., 2001)** - All moves are accepted regardless of their success.

- **OnlyImproving (OI) (Cowling et al., 2001)** - Only moves which improve the current solution are accepted.
- **ImprovingandEqual (IE) (Bilgin et al., 2007)** - Only worsening moves are rejected.

Non-deterministic move acceptance operators will not necessarily make the same decision for the same solutions if they are encountered more than once during the search process. Hence, additional parameters are used to assist in the acceptance decision. Examples include GreatDeluge and MonteCarlo acceptance operators.

- **GreatDeluge (GD) (Kendall and Mohamad, 2004)** - All moves which exceed some *threshold* are accepted. The threshold is set initially to the value of the initial candidate solution and moves at a linear rate towards the expected value of the global optimum.
- **MonteCarlo (MC) (Ayob and Kendall, 2003)** - All improving moves are accepted, yet non-improving moves may also be accepted based on some dynamically changing probability p_i. The probability will depend on δ, the fitness difference between the perturbed offspring solution and the parent solution. If $\delta \leq 0$ (i.e., the offspring solution is equal or better to the parent) then the move is accepted. Different variants will calculate p_i using $\delta > 0$ in different ways.
  - **Linear Monte Carlo (LMC)** uses a negative linear ratio based on the fitness worsening. The move is accepted with probability $p_i = M - \delta$, for some constant $M > 0$.
  - **Exponential Monte Carlo (EMC)** uses a negative exponential ratio based on the fitness worsening. The move is accepted with probability $p_i = e^{-\delta}$.
  - **Exponential Monte Carlo with Counter (EMCQ)** is similar to EMC, but if no improvement is seen over a number of iterations, the probability p_i will start increasing. The move is accepted with probability $p_i = e^{\frac{-\delta \cdot t}{\tau}}$, where t is a measure of time and $\tau = \tau(Q)$ is some function of the number of consecutive non-improvements Q.

Naturally, the use of more sophisticated move acceptance operators (as opposed to simplified models) can substantially improve the performance of selection hyper-heuristics. Burke et al. (2013) claimed in their review of the state-of-the-art of hyper-heuristics that:

> "It appears that the choice of move acceptance component is slightly more important than the choice of heuristic selection."

Application Domain	**References**
Personnel Scheduling	Bai et al. (2012); Burke et al. (2003); Cowling and Chakhlevitch (2003); Cowling et al. (2002a, 2001, 2002b); Han and Kendall (2003); Mısır et al. (2010).
Educational Timetabling	Bai et al. (2012, 2007); Bilgin et al. (2007); Burke et al. (2005a, 2003); Chen et al. (2007); Cowling et al. (2001, 2002b); Demeester et al. (2012); Ozcan et al. (2009); Özcan et al. (2010).
Space Allocation	Bai et al. (2008); Bai and Kendall (2005); Burke et al. (2005b).
Cutting and Packing	Bai et al. (2012); Dowsland et al. (2007).
Vehicle Routing	Meignan et al. (2010); Mısır et al. (2011); Pisinger and Ropke (2007).
Sports Scheduling	Gibbs et al. (2010); Mısır et al. (2009).
Cross-domain (HyFlex)	Burke et al. (2010a, 2011); Chan et al. (2012); Di Gaspero and Urli (2012); Drake et al. (2012); Hsiao et al. (2012); Mısır et al. (2012); Ochoa et al. (2012a,b); Özcan and Kheiri (2012); Walker et al. (2012).

Table 3.1 Most studied application domains of methodologies to select perturbative heuristics, up to 2013 (Burke et al., 2013). For an exhaustive survey of more recent publications, see the webpage of (Mısır, 2019).

For the ease of analysis, throughout this thesis we analyse selection-perturbation hyper-heuristics with deterministic move acceptance operators, with the goal of building up techniques that will allow the analyses of more sophisticated deterministic and non-deterministic move acceptance techniques in the future.

3.2.3 Example Applications of Hyper-heuristics

Hyper-heuristics boast a large quantity of successful applications to combinatorial optimisation problems. Table 3.1, adapted from Burke et al. (2013), lists the most studied applications of selection-perturbation hyper-heuristics, including problems such as personnel scheduling, educational timetabling and cutting and packing.

To better understand how hyper-heuristics are applied in practice, we now discuss three specific proof of concept examples from Table 3.1 in detail for the applications of scheduling, timetabling and packing. For each example, we discuss the problem in detail, the high-level hyper-heuristic methodologies the low-level heuristics used by the authors and their presented results.

Scheduling: A lot of the initial studies on hyper-heuristics, such as the seminal work by Cowling et al. (2001), looked at the application of hyper-heuristics to scheduling problems. The goal is to successfully schedule an event (e.g., a sales summit) while minimising cost and constraint violations. Such constraints are classified as high-priority constraints (e.g., two presentations cannot overlap) and low-priority constraints (e.g., ensuring each presentation occur on a certain day). The hyper-heuristic methodologies employed include *greedy* and *choice function* approaches. Examples of the low-level heuristics used include 'increase priority of one meeting', 'move a presentation from one session to another' and 'replace one delegate in a session'. The results presented by Cowling et al. (2001) implied that the *choice function* approaches gave the best results for the studied problem instance.

Timetabling: Further studies have considered hyper-heuristics on timetabling problems, such as work by Bilgin et al. (2007). In the majority of cases, educational timetabling problems of universities and schools are considered with the goal of timetabling lectures and seminars for students throughout the week, subject to a number of constraints. These constraints include ensuring students are not scheduled to be in two places at once, not exceeding room capacity and giving students a sufficient break between lectures. The hyper-heuristic methodologies employed include *choice function* and *tabu search* approaches. Examples of the low-level heuristics used include 'move a random event from its current timeslot to a random one' and 'swap the timeslots of two random events'. The results presented by Bilgin et al. (2007) implied that, while no hyper-heuristic methodology is the best over all their presented benchmark functions, the *choice function* approach with a Monte Carlo move acceptance mechanism has slightly better performance on average than other variants.

Packing: The problem of packing has always been logistically troubling. The goal of the common combinatorial benchmark problem of bin packing is to fit goods with maximal value into a bin, subject to weight or size constraints. Dowsland et al. (2007) considered at a more complex version, in which goods have to be packed into a number of vans, where the goal is minimise the number of vans given that the use of space is optimised. Each of the goods has a measure for length, height, width and weight and the vans can only carry to a certain size. The hyper-heuristic methodologies employed include *random descent* and *tabu search* approaches. Examples of the low-level heuristics used include 'sample a new solution uniformly from the neighbourhood of possible solutions' and 'sample a new solution from the neighbourhood using roulette-wheel selection'. The results presented by Dowsland et al. (2007) suggest that a *tabu search* approach within a simulated annealing move acceptance framework gave the best results, yet more sophisticated configurations are required to match the state of the art approaches.

In order to present comparable results for hyper-heuristic methodologies on a range of benchmark instances of classical combinatorial optimisation problems, practitioners often use the *HyFlex* framework (Ochoa et al., 2012a). The HyFlex software framework contains various problem instances of six hard combinatorial problems: maximum satisfiability, one dimensional bin packing, permutation flowshop, personnel scheduling, traveling salesman and vehicle routing. Applying hyper-heuristics to the *HyFlex* problem instances allows practitioners to compare their hyper-heuristics with a wide range of results from general-purpose and problem-specific algorithms.

3.3 Theoretical Foundations of Hyper-heuristics

Although hyper-heuristics have found various successful applications (and many well-established methods are available for experimental analyses), they are not yet fully understood. In particular, it is not clear which hyper-heuristic methodology and which move acceptance operator should be applied to a given optimisation problem, and which set of low-level heuristics should be used. Hence, a rigorous theoretical foundational understanding is required in order to provide insights into which problem classes hyper-heuristics perform efficiently and inefficiently. Burke et al. (2013) stated the following:

> "It is often the case that a hyper-heuristic does not aim to outperform a custom-made solver for a given problem. In such an enviroment, applicability over a wide range of problem domains is more crucial. For this reason, comparison measures across different problems are of interest."

In order to assess the behaviour and performance of hyper-heuristics, some theoretical analyses of hyper-heuristics have appeared. Many works considering similar ideas about heuristic generation and selection were often presented under a different name. However, since we can find parallels with the field of hyper-heuristics, we discuss these existing results. Over the course of the remainder of the thesis, we will substantially build upon the existing theoretical results regarding selection hyper-heuristics. We aim to improve the understanding of when, how and why hyper-heuristics can be succesfully applied.

We first discuss the existing theoretical results on selection hyper-heuristics from the literature, which we partition into two areas based on the low-level heuristics employed: mutation-based heuristics and selection-based heuristics.

3.3.1 Mutation-Based Low-level Heuristics

He et al. (2012) looked at when a *mixed strategy* EA (where several mutation operators are employed within the same EA framework) can be beneficial over a *pure strategy* EA (where only one mutation operators is used). This provides insights into when the application of a hyper-heuristic methodology is preferable to a single-heuristic strategy, since the differing mutation operators act as the inherent low-level heuristics. The authors proved that the expected runtime of a mixed strategy (1+1) EA is not worse than the worst pure strategy (1+1) EA using one of the operators. He et al. (2013) looked further at when a mixed strategy EA can outperform a pure strategy EA. They introduced the *Complementary Strategy Theorem*, which states that if the drift (see Section 2.5) of one pure strategy PS_1 is better (in at least one state of the optimisation process) than the drift of another pure strategy PS_2, then PS_2 is said to be *complementary* to PS_1. Furthermore, there exists a mixed strategy of PS_1 and PS_2 such that the expected runtime of this mixed strategy is better than the expected runtime of the pure strategy PS_1. Hence, a hyper-heuristic framework selecting between two low-level heuristics is preferable to a single heuristic in the same context, assuming the low-level heuristics are applied correctly.

Theoretical results explicitly analysing hyper-heuristics have considered basic selection hyper-heuristics. The first rigorous theoretical analysis of hyper-heuristics was performed by Lehre and Özcan (2013). They considered a simple hyper-heuristic in the framework of Algorithm 9 (where S is a finite search space, H is a set of low-level heuristics and $f : S \to \mathbb{R}^+$ is a cost function). The hyper-heuristic selected from the set of heuristics $H = \{\mathrm{RLS}_1, \mathrm{RLS}_2\}$, which are local search heuristics mutating one or two distinct bits of the bit-string uniformly at random. Their hyper-heuristic selected RLS_1 in each iteration with probability p and RLS_2 with probability $1-p$. This can be thought of as a general variant of the simple random hyper-heuristic introduced by Cowling et al. (2001) (see Subsection 3.2.1). The authors considered the GAPPATH (GP) function:

$$\textsc{GapPath}(x) = \begin{cases} \textsc{ZeroMax}(x) & \text{if } \textsc{Ridge}(x) \equiv 1 \text{ (mod 3)}, \\ \textsc{ZeroMax} + 2n\textsc{Ridge}(x) & \text{otherwise.} \end{cases}$$

where $\textsc{ZeroMax}(x) := \sum_{i=1}^{n}(1-x_i)$ and

$$\textsc{Ridge}(x) = \begin{cases} i & \text{if } x = 1^i 0^{n-i} \text{ for } i \in [0 \dots n], \\ 0 & \text{otherwise.} \end{cases}$$

The GP function requires the hyper-heuristic to alternate between the two heuristics in order to make progress. Lehre and Özcan (2013) showed that by setting either $p = 0$ or $p = 1$ (i.e., only using one heuristic), the expected runtime of their simple hyper-heuristic to optimise the GAPPATH function was infinite. However, by setting $p \in (0,1)$, the hyper-heuristic optimised GP in expected finite time. In particular, their presented expected runtime was

$$\frac{n^2}{3(1-p)}\left(\frac{n-3}{2}+\frac{1}{p}\right).$$

A static parameter $p = \frac{1}{n}$ would thus give an expected runtime of $\frac{n^3}{2} + O(n^2)$ (the best value of p is $\approx \sqrt{\frac{6}{n}}$, giving expected runtime $\approx \frac{n^3}{18}$). The authors also considered a simple reinforcement learning based technique to allow the algorithm to choose between the parameters p_{low} and p_{high} (where $p_{\text{low}} = \frac{1}{n}$ and $p_{\text{high}} = 1 - \frac{1}{n}$). Due to the poor setup of this revised hyper-heuristic, this hyper-heuristic presented a worse expected runtime of $\frac{n^4}{6} + \frac{n^3}{6}$. We will build upon these results in Chapter 4 by extending the GAPPATH function such that it is necessary for a hyper-heuristic to learn to prefer a certain heuristic in order to make progress through the fitness landscape and prove that two simple hyper-heuristics can optimise the extended GP function with overwhelming probability while *simple random* fails with overwhelming probability.

Alanazi and Lehre (2014) extended on the results of Lehre and Özcan (2013) by analysing the expected runtimes of four simple hyper-heuristics on the LEADINGONES (LO) benchmark function (see Section 2.4). They considered the *simple random*, *random gradient*, *greedy* and *permutation* hyper-heuristics (see Subsection 3.2.1) as the learning mechanisms within the framework of Algorithm 9, and used $H = \{1\text{BITFLIP}, 2\text{BITFLIP}\}$ as the set of low-level heuristics (Algorithm 6). The authors found that all the four hyper-heuristics have the same asymptotic expected performance on LO, with expected runtimes of $\Theta(n^2)$. The constants in the leading terms for their lower and upper expected runtime bounds were also derived (which we present in Table 3.2). An experimental comparison showed that all four had practically the same performance on LO, implying that their attempts to learn were not working and all mechanisms essentially choose heuristics at random in each iteration. Due to their similar experimental performance, the authors wanted to improve their performance and suggested the *improved random gradient* mechanism, where the randomly chosen heuristic is applied for a fixed period of time regardless of success. An experimental comparison suggested such an alteration to the mechanism results in better performance on LEADINGONES. We will build upon these results in Chapters 5, 6, 8 and 9 by rigorously proving that all four learning mechanisms exhibit the same performance on LO and we will generalise the *greedy* and *random gradient* hyper-heuristics such that they are able to better appreciate the performance

Hyper-heuristic	**Lower Bound**	**Upper Bound**
Simple Random	$\frac{\ln(3)}{6} \cdot n^2 \approx 0.183n^2$	$\ln(3) \cdot n^2 + O(n) \approx 1.099n^2$
Random Gradient	$\frac{4+3\ln\left(\frac{10}{3}\right)}{9} \cdot n^2 \approx 0.846n^2$	$2\ln\left(\frac{5}{2}\right) \cdot n^2 + o(n^2) \approx 1.833n^2$
Greedy	$\frac{\ln(3)}{6} \cdot n^2 + o(n^2) \approx 0.183n^2$	$\ln(3) \cdot n^2 + o(n^2) \approx 1.099n^2$
Permutation	$\frac{\ln(3)}{6} \cdot n^2 + o(n^2) \approx 0.183n^2$	$\ln(3) \cdot n^2 + o(n^2) \approx 1.099n^2$

Table 3.2 : Lower and upper bounds on the expected runtimes of four simple hyper-heuristic mechanisms on LEADINGONES (Alanazi and Lehre, 2014). We believe the authors made a small mistake in their proof and the lower bound for the *random gradient* hyper-heuristic should have been $\frac{3\ln\left(\frac{10}{3}\right)}{9}n^2 + o(n^2) \approx 0.401n^2$

of the selected heuristics. We will show that the *generalised* hyper-heuristics are faster than the simple mechanisms and their inherent low-level heuristics on LO, even achieving the best expected runtime possible with the available low-level heuristics (up to lower order terms).

Considering more sophisticated hyper-heuristic methodologies, Alanazi and Lehre (2016) theoretically analysed reinforcement learning mechanisms in selection hyper-heuristics. The authors proved that if the probability of improving the candidate solution is less than $1/2$, which is unlikely in many realistic situations, then additive reinforcement learning mechanisms are equivalent to performing a random choice of heuristics. The authors suggested using alternative learning mechanisms than the considered additive reinforcement learning mechanism.

Further works analysing algorithms selecting between mutation-based heuristics of different neighbourhood sizes have been presented. Although they do not explicitly mention hyper-heuristics, parallels can be drawn between the presented works and the selection hyper-heuristic frameworks presented within this chapter. Doerr et al. (2016b) showed that a unary unbiased algorithm selecting the parameter k of the mutation operator $\mathrm{RLS_k}$ such that the drift is maximised at all times of the optimisation process has expected runtime on ONEMAX at most a small linear term (εn for an arbitrary small $\varepsilon > 0$) larger than the unary unbiased black-box complexity. Doerr et al. (2016a) showed that an algorithm with self adjusting parameter k of the mutation operator $\mathrm{RLS_k}$ is able to match this expected performance on ONEMAX up to a small order sublinear term and outperforms the (1+1) EA and RLS in expectation on LEADINGONES and the Minimum Spanning Tree problem.

3.3.2 Selection-Based Low-level Heuristics

The majority of theoretical results in the context of hyper-heuristics to date have concerned learning mechanisms to select between different mutation-based heuristics. Mutation operators are well-studied and thus well understood. However, hyper-heuristics often consider low-level heuristics that take different forms. Thus, analyses of hyper-heuristics considering different operators (or parameters), including selection operators as low-level heuristics have been considered. Analysing hyper-heuristics in different settings will bring a greater understanding of how to apply them to achieve good performance on a wide range of complex optimisation problems.

Lehre and Özcan (2013) presented a hyper-heuristic which is allowed to choose between different move acceptance operators. The authors considered a Simple Random hyper-heuristic in the framework of Algorithm 9. The hyper-heuristic uses RLS as a mutation operator and applies the ALLMOVES acceptance operator with probability p and the ONLY-IMPROVING acceptance operator with probability $1-p$. They presented an analysis on the ROYALROAD$_k$ benchmark function, with $k \geq 2$:

$$\text{ROYALROAD}_k(x) := \sum_{i=1}^{\lfloor n/k \rfloor} \prod_{j=1}^{k} x_{k(i-1)+j}.$$

In order to optimise this function, it is necessary to optimise k equal sized blocks concurrently. Lehre and Özcan (2013) theoretically proved that it is necessary to mix the acceptance operators on this function in order for the hyper-heuristic to be efficient, since by only accepting improvements (i.e., setting $p = 0$) the runtime is infinite due to the hyper-heuristic not being able to cross the plateaus of equal fitness, while by always accepting any move (i.e., setting $p = 1$), the algorithm simply performs a random search, giving exponential expected runtime. By setting the value of the parameter p appropriately, they provide an upper bound on the expected runtime of the hyper-heuristic of $O(n^3 \cdot k^{2k-3})$. However, it is still beaten by the $O(n\log(n) \cdot (2^k/k))$ expected time required by simply just using the IMPROVINGANDEQUAL acceptance operator throughout the run coupled with standard bit mutation (Doerr et al., 2013). Hence, the advantages of switching between selection operators rather than just using one all the time are not evident. We will build upon these results in Chapter 7 by presenting example benchmark problems where selection-based hyper-heuristics perform efficiently and inefficiently.

Qian et al. (2016) also demonstrated the effectiveness of selection hyper-heuristics for multi-objective optimisation problems. They considered three different components as low level heuristics: the parent selection mechanism, mutation operators and move acceptance strategies. On three pseudo-Boolean functions, it was rigorously shown that mixing selection hyper-heuristics in each of the three respective components can lead to exponential speedups in the expected runtime, as opposed to just applying the low-level heuristics individually. These results further motivate the use of hyper-heuristics as general-purpose problem solvers, since they can provably act effectively on the different components of stochastic search algorithms.

Most theoretical results to date (and the majority of the original results we will present throughout the remainder of this thesis) consider hyper-heuristic methodologies selecting between low-level heuristics that differ by a single parameter. Such simplified hyper-heuristics are easier to analyse and the rigorous results and proof techniques used will allow us to build up a foundation of techniques to analyse more sophisticated hyper-heuristics.

3.4 Related Areas

In Chapter 2, we discussed that an algorithm's performance may depend strongly on the way the parameters are set. It is known that there are no globally good parameter values (Wolpert and Macready, 1997). Hence, different optimisation problems may require vastly different settings. Even within a specific instance, the best values of the parameters might change as the state of the optimisation process changes. Furthermore, the performance may heavily depend on two or more parameters simultaneously and how these parameters depend on each other. *Parameter setting* (i.e., the task of identifying high-performing parameter configurations) is an important task. Configuring the parameters of a randomised search heuristics can be done prior to the run of the algorithm (i.e., offline) or during the run of the algorithm (i.e., online).

3.4.1 Offline Algorithm Configuration

By deciding on good configurations before the run, the aim of *parameter tuning* (i.e., offline algorithm configuration) is to derive parameters leading to good performance on vast classes of instances without having to adapt and update the parameters as the algorithm progresses through the search space. There are many experimental works for different types of parameter tuners (see, e.g. (López-Ibáñez et al., 2016) for an in-depth survey), yet so far only one theoretical work exists regarding offline tuning of the parameters of randomised search heuristics (Hall et al., 2019). Most strategies involve tuning the parameters on a training set of problem instances displaying similar characteristics to the problem that will be solved.

We now mention some of the most commonly used parameter tuning techniques used within the Evolutionary Computation community.

Hutter et al. (2009) introduced *ParamILS*, which is an iterated local search framework used for automatic algorithm configuration (i.e., parameter tuning). ParamILS essentially acts as a local search algorithm looking for better parameter values. The method works as follows. Firstly, some parameter configuration is selected and this solution is perturbed. Local search is then applied to the parameter set until another local optima is reached and this process repeats until some termination criteria. Some acceptance criteria are necessary to decide whether one solution is 'better' than another. The standard method is to apply both settings for a fixed number of iterations and compare their fitness afterwards.

One of the most widely used parameter tuning techniques is that of *racing*, where different parameter sets are compared on different instances of the desired problem. Birattari et al. (2002) brought this method, inspired by practices from machine learning, into the field of parameter tuning in evolutionary computation. The authors introduced *F-race*, where sets of generated parameters are tested on different problem instances and any that are performing significantly worse statistically are eliminated. Hence, computational effort is not wasted on bad parameter values. Balaprakash et al. (2007) sought to improve upon F-race, and introduced *I/F-race*, where high performing parameter sets are used to bias the probability distribution of newly generated parameter sets in their favour. Hence, the race can exploit good parameter values. Perhaps the most commonly used racing method is *irace* (which is an extension of I/F-race) introduced by López-Ibáñez et al. (2016). Parameter sets are sampled according to a particular distribution and the best configurations are selected by racing (on different instances). Finally, the distribution is updated to bias towards the best parameter sets found from the race. Each parameter will have their own distribution.

While irace and other parameter tuners have seen many successful applications, an open and promising area of research is the theoretical analysis of offline parameter tuners. Hall et al. (2019) rigorously proved that the cutoff time (i.e., how long each configuration is run for in each comparison) of a tuner must be carefully chosen if using optimisation time as a measure of a configuration's performance, whereas if fitness achieved within the cutoff time is used as this performance measure, then the tuner is robust under the choice of cutoff time.

3.4.2 Online Parameter Control

Parameter control refers to techniques which update the initial parameter values of an algorithm throughout the run. Thus, the parameters are non-static during the optimisation process. As opposed to parameter tuning, algorithms using parameter control techniques have many advantages. Firstly, the time taken tuning parameters prior to the run of the algorithm can often be costly and consuming. Whereas many parameter tuners are relatively efficient and provide good parameter values, finding parameter values on the fly (i.e., during the run of the algorithm) can save lots of expensive computation. Secondly, and perhaps most importantly, is the notion that the best parameter settings for an algorithm are not necessarily fixed throughout the search space and may vary throughout the run (Doerr, 2018b). Non-static parameter algorithms aim to 'learn' (or find) the best parameters for different parts of the optimisation process. Thus, if successful, they can vastly outperform algorithms with static parameters. Finally, learning or finding the best parameter values throughout the run does not limit an algorithm to a specific class of problems or problem instances. Parameter tuning will hopefully lead to parameter values that perform well for a specific problem, but it is not necessarily true that these, or similar values, will perform well for a similar problem.

As a result of their advantages over parameter tuning techniques, parameter control has been one of the main focal points of theoretical EA research in recent years. As a simple example to motivate the use of parameter control techniques, we now present an important theorem, adapted from the work of Böttcher et al. (2010). The result gives an expected runtime for the (1+1) EA on the benchmark LEADINGONES function with three parameter settings for the mutation rate: the standard mutation rate of $\frac{1}{n}$, the best static mutation rate and a fitness-dependent non-static mutation rate. These results were proved using Theorem 2.13 (see Section 2.5).

Theorem 3.1 (Böttcher et al. 2010) *Let p be the mutation rate of the (1+1) EA when optimising the* LEADINGONES *benchmark function. Then,*

1. *The expected runtime with $p = \frac{1}{n}$ (i.e., the standard mutation rate) is $\frac{e-1}{2}n^2 \pm o(n^2) \approx 0.859n^2$.*
2. *The expected runtime with $p \approx \frac{1.5936}{n}$ (i.e., the best static mutation rate) is $\approx 0.772n^2$.*
3. *The expected runtime with $p = \frac{1}{\mathrm{LO}(x)+1}$ (i.e., the best non-static mutation rate) is $\frac{e}{4}n^2 + o(n) \approx 0.680n^2$.*

We see from Theorem 3.1 that the best static mutation rate improves upon the performance with the commonly used standard mutation rate and further that a non-static mutation rate outperforms any static mutation rate. Even in this simple example, the best mutation rate changes throughout the run. Hence, it may be worth evolving non-static mutation rates and it may have advantages over static parameter tuning techniques. In Chapter 5, we will show that hyper-heuristics can run in the best expected runtime achievable with the available mutation operators.

In terms of classifying parameter control techniques, Doerr and Doerr (2018) recently proposed the classification seen in Figure 3.2 which explicitly separates parameter control techniques into five subclasses: state-dependent, success-based, learning-inspired, self-adaptive and hyper-heuristic techniques. The work by Doerr and Doerr (2018) also provides an extensive survey of theoretical results for a variety of parameter control mechanisms.

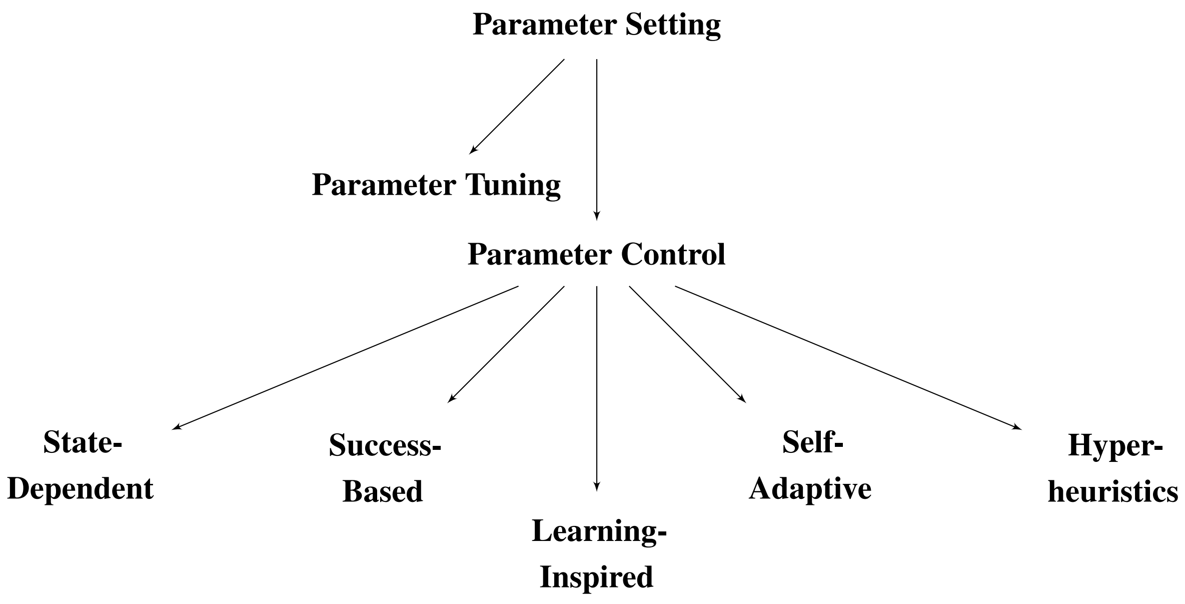

Fig. 3.2 A more recent classification of parameter setting techniques proposed by Doerr and Doerr (2018), which is an extension of the classification of (Eiben et al., 1999).

While selection HHs may be used for the purpose by including only heuristics that differ by a parameter value in the low-level set, HH have more general algorithm configuration capabilities which have been discussed in detail earlier within this chapter.

3.5 Chapter Summary

Hyper-heuristics are general algorithm selection and generation methodologies. The overall goal of the field is to automate the algorithm selection and configuration process in order to find heuristics that efficiently solve optimisation problems. We discussed a very general classification of the different types of hyper-heuristic and described in greater detail the class of selection-perturbation hyper-heuristics which will be the focus of the rest of the thesis. The limited number of theoretical works on the performance of hyper-heuristics were reviewed. We will considerably build upon these in the upcoming chapters. Connections to the related areas of offline and online parameter configuration have also been discussed.

To date, the majority of theoretical results regarding hyper-heuristics have analysed heuristic selection mechanisms for selecting certain variation or selection operators (that differ by a single parameter), yet it is only natural to begin the analysis of a new paradigm in simple settings in order to build up a variety of general analysis techniques. Eventually, given a greater foundational understanding of simple hyper-heuristics, the natural goal of the field (of theoretical hyper-heuristic research) will be to provide rigorous theoretical analyses of sophisticated state-of-the-art heuristic selection and generation methodologies used successfully in the literature to bring a greater understanding of the field.

Part II

Runtime Analysis of Hyper-Heuristics

Chapter 4

An Example Where Learning is Necessary

In this chapter, we motivate the use of hyper-heuristics by considering a simple benchmark problem designed such that it is necessary to use a combination of different heuristics to solve the problem, because the use of only one of the available heuristics does not suffice. For the purpose, we consider the simple hyper-heuristic framework introduced in Section 3.2 and equip it with the four most simple learning mechanisms from the literature (see Subsection 3.2.1): *Simple Random*, *Permutation*, *Greedy* and *Random Gradient*. The idea behind the learning mechanisms is to continue to exploit the currently selected heuristic as long as it is successful. However, the probability that a promising heuristic is successful in the next step is relatively low when perturbing a reasonable solution to a combinatorial optimisation problem. We build on the work of Alanazi and Lehre (2014) who analysed the Simple Random hyper-heuristic mechanism on the simple pseudo-Boolean function GAPPATH. We generalise the GAPPATH function to consider GENERALISEDGAPPATHWITHTRAPS (GGPT), a class of functions where it is necessary to learn to prefer a certain heuristic, and show that the four *simple* mechanisms fail to find the global optimum with overwhelming probability. Since this result implies they are not learning, and an experimental analysis by Alanazi and Lehre (2014) showed that they are essentially choosing low-level heuristics at random, we generalise the simple mechanisms to allow a longer time period (we refer to τ as the length of this *learning period*) to decide whether the currently chosen low-level heuristic is successful or not. The newly introduced *Generalised Greedy* and *Generalised Random Gradient* hyper-heuristics are the focus of the analysis. We show that the generalised hyper-heuristics are efficient on GAPPATH and find the global optimum on GGPT with overwhelming probability given an appropriate choice of the learning period τ.

4.1 Four Simple Learning Mechanisms that Fail to Learn

Selection hyper-heuristics equipped with the following heuristic selection learning mechanisms (which were discussed in Chapter 3 and applied within the framework of Algorithm 9) have been commonly used in the literature to solve combinatorial optimisation problems (Cowling et al., 2001, 2002b):

- **Simple Random**, which selects a low-level heuristic independently with probability p_h in each iteration (usually $p_h = 1/|H|$, where $|H|$ is the size of the set of low-level heuristics, i.e., uniformly at random);
- **Permutation**, which generates a random ordering of low-level heuristics and returns them in that sequence when called by the hyper-heuristic;
- **Greedy**, which applies all available low-level heuristics in parallel and returns the best found solution;
- **Random Gradient**, which randomly selects a low-level heuristic and keeps using it as long as it obtains improvements.

Although these mechanisms seem simple, they have formed the basis of theoretical hyper-heuristic research for a number of years (Alanazi and Lehre, 2014; Lehre and Özcan, 2013). Alanazi and Lehre (2014) showed experimentally that all four mechanisms have essentially the same performance as just choosing low-level heuristics at random on the LEADINGONES benchmark function.

In this section we introduce a benchmark problem where it is necessary to 'learn' which operator to apply and use the problem to rigorously show that the mechanisms are not capable of learning. Lehre and Özcan (2013) motivated the use of hyper-heuristics by designing a benchmark function called GAPPATH as an example problem class where two mutation operators with different neighbourhood size have to be applied to solve the problem (i.e., the use of either operator alone would not suffice). The function is defined as follows:

$$\textsc{GapPath}(x) = \begin{cases} \textsc{ZeroMax}(x) & \text{if } \textsc{Ridge}(x) \equiv 1 \pmod 3, \\ \textsc{ZeroMax} + 2n\textsc{Ridge}(x) & \text{otherwise.} \end{cases}$$

where $\textsc{ZeroMax}(x) := \sum_{i=1}^{n}(1 - x_i)$ and

$$\textsc{Ridge}(x) = \begin{cases} i & \text{if } x = 1^i 0^{n-i} \text{ for } i \in [0 \dots n], \\ 0 & \text{otherwise.} \end{cases}$$

The GAPPATH function consists of a short path which corresponds to all search points of the form 1^i0^{n-i}. The path contains gaps where the function values are inferior to the rest of the path. Such gaps consist of all path points where $i \equiv 1 \pmod 3$. As a result of these gaps, an algorithm that only uses either the 1BITFLIP or the 2BITFLIP operator will have infinite expected runtime, since it is necessary to alternate the operators in order to make progress on the path. Lehre and Özcan (2013) proved that the expected runtime of the Simple Random mechanism, initialised on the 0^n bit string and using only the 1BITFLIP and 2BITFLIP operators with probability p and $1-p$ respectively, is $\frac{n^3}{6(1-p)} + \frac{n^2}{3p}$ for any probability $p \in (0,1)$. In particular, if $p = 0.5$ (as with the standard mechanism), the expected runtime will be $\frac{n^3+2n^2}{3}$.

While the GAPPATH function was designed to showcase the necessity of hyper-heuristics, it does not explore their learning capabilities. In particular, even the non-learning Simple Random hyper-heuristic has efficient performance. We now modify the function class such that it is necessary for the hyper-heuristics to learn which heuristic to use if they are to be efficient. This function class is called GENERALISEDGAPPATH (GGP$_k$). It is similar to the GAPPATH function with the difference that an algorithm will require k consecutive improvements with 2BITFLIP after each improvement with 1BITFLIP. Hence, learning to use 2BITFLIP more often that 1BITFLIP flips leads to improved performance. We formally define GGP$_k$ as such. Let n be of the form $n = d(2k+1)$ for some $d, k \in \mathbb{N}$:

$$\mathrm{GGP}_k(x) := \begin{cases} \textsc{ZeroMax}(x) & \text{if } \textsc{Ridge}(x) \in S_k \\ \textsc{ZeroMax}(x) + 2n\textsc{Ridge}(x) & \text{otherwise} \end{cases}$$

where $\textsc{ZeroMax}(x) := \sum_{i=1}^{n}(1-x_i)$, $\textsc{Ridge}(x) := i$ if $x = 1^i0^{n-i}$, and 0 otherwise, and $S_k = \{c(2k+1)+1-2\beta \mid c, \beta \in \mathbb{Z}^+, c \le d, \beta \le k\}$.

In order to further penalise hyper-heuristics that are not learning, we modify the GGP$_k$ function by inserting a *trap* point (i.e., a point of high non-optimal fitness) at a random position within each group of k 2-bit flips. These trap points can be reached from the path by a 1-bit mutation (with the 1BITFLIP heuristic), while the next path point remains reachable only by a 2-bit mutation (with the 2BITFLIP heuristic), and are local optima that cannot be escaped from by either mutation operator. We call the function class GENERALISEDGAP-PATHWITHTRAPS (GGPT$_k$):

$$\mathrm{GGPT}_k(x) := \begin{cases} 2n^2 - 1 & \text{if } \textsc{Ridge}(x) \in S_{\textsc{Trap}} \\ \textsc{ZeroMax}(x) & \text{if } \textsc{Ridge}(x) \in S_k \setminus S_{\textsc{Trap}} \\ \textsc{ZeroMax}(x) + 2n\textsc{Ridge}(x) & \text{otherwise} \end{cases}$$

where $S_{\text{TRAP}} = \{c(2k+1)+1-2\gamma \mid c \in \mathbb{Z}^+, c \leq d\}$ for some constant $0 < \gamma < k$ and S_k as before. If a 'trap' point is ever constructed then the algorithm will not be able to reach the global optimum. Since there is a finite probability for any mechanism to construct a 'trap' point, their expected runtimes will all be infinite by the law of total expectation. We derive the success probabilities for the 'simple' mechanisms and generalised hyper-heuristics on this modified function (denoted GGPT_k) using $H = \{1\text{BITFLIP}, 2\text{BITFLIP}\}$.

Theorem 4.1 *Initialised at 0^n, the Greedy, Permutation, Simple Random and Random Gradient hyper-heuristic mechanisms will fail to find the global optimum on GGPT_k in finite time, with probability at least $1 - n^{-\Omega(n)}$.*

Proof: We will show that when each mechanism is at a point from which a 'trap' point is accessible, it will construct the 'trap' point before the next path point with probability $1 - O(1/n)$. If this holds, as there are at least $n/(2k+1) = \Theta(n)$ points from which a 'trap' point is accessible, the 'simple' mechanisms will construct a 'trap' point before reaching the global optimum with probability $1 - O(1/n)^{\Theta(n)} = 1 - n^{-\Theta(n)}$.

The Greedy mechanism applies both mutation operators in one iteration. The probability that a 'trap' point is constructed via a 1-bit mutation is $1/n$, while the probability that the next path point is constructed is $2/n^2$. If both are constructed, the mechanism accepts the 'trap' point as it has higher fitness. The probability that a 'trap' point is constructed before the next path point is therefore at least $\frac{1/n}{1/n+2/n^2} = 1 - \frac{2}{n+2} = 1 - O(1/n)$ by considering the first hit probability (Theorem 2.3 in Chapter 2).

The Permutation mechanism, following a two-bit success leading to a point from which a 'trap' point is accessible, will perform 1-bit and 2-bit mutations successively in this order. The probability that a 'trap' point is constructed over two iterations is thus $1/n$, while the probability that the next path point is constructed over two iterations is $(1-1/n) \cdot 2/n^2$. The probability that a 'trap' point is constructed before the next path point is therefore $\frac{1/n}{1/n+(1-1/n)2/n^2} = \frac{n^2}{n^2+2(n-1)} = 1 - O(1/n)$.

The Simple Random mechanism constructs a 'trap' point with probability $1/(2n)$, and the next path point with probability $1/n^2$. The probability that a 'trap' point is constructed before the next path point is therefore at least $\frac{1/(2n)}{1/(2n)+1/n^2} = 1 - \frac{2}{n+2} = 1 - O(1/n)$.

The Random Gradient mechanism behaves equivalently to the Simple Random mechanism, unless it manages to construct an improvement by a two-bit mutation immediately following another two-bit improvement. The probability that an individual 'trap' point is skipped in this manner is $2/n^2 = O(1/n)$, and thus the probability that the Random Gradient constructs the 'trap' point before the next path point is $(1 - O(1/n))^2 = 1 - O(1/n)$. □

Thus, all the 'simple' mechanisms will construct a trap point and fail to find the global optimum of GGPT with overwhelming probability, since they do not learn to prefer the 2BITFLIP heuristic.

4.2 Simple Hyper-heuristics with Extended Learning Periods can Learn

In the previous section we showed that by making a heuristic selection decision in every iteration without taking past performance into account, the 'simple' mechanisms do not have enough time to learn which operator is preferable at the current optimisation stage. In this section, we generalise the simple mechanisms to allow a longer time period to decide whether a low-level heuristic is currently successful or not, aiming to maintain the intrinsic ideas of the 'simple' learning mechanisms while generalising sufficiently to allow learning to take place. Since the Simple Random and Permutation learning mechanisms make no attempt to learn, we will consider:

- **Generalised Greedy** (presented in Algorithm 10), where all low-level heuristics are tested on the same candidate solution until an improvement is found (*Decision Stage*)[1]. The improvement (chosen uniformly at random if there are multiple) is then accepted and the corresponding operator is run for a fixed period of time τ (*Exploitation Stage*).

The idea of this methodology is to make a 'smart' choice of low-level heuristic by testing which heuristic is more suitable in the current area of the search space. This smart choice is then exploited for a fixed number of iterations before another decision is made, thus ensuring that many smart choices can be made. We also consider:

- **Generalised Random Gradient** (presented in Algorithm 11), where a low-level heuristic is chosen uniformly at random (*Decision Stage*) and run for a period of fixed time τ. If an improvement is found before the end of the period, then a new period of time τ is immediately initialised (*Exploitation Stage*). If the chosen operator fails to provide an improvement in τ iterations, a new operator is chosen at random.

The idea of this methodology is to exploit a good heuristic if it is chosen by the random decision. A bad heuristic will fail quickly and not be exploited, whereas a good choice of heuristic will keep being used as long as it is being successful.

[1]The fitness function evaluations of all heuristics in the Decision Stage contribute to the runtime.

Algorithm 10 Generalised Greedy Hyper-heuristic

```
1: Choose $x \in S$ uniformly at random
2: while stopping conditions not satisfied do
3:     $\forall h \in H$, apply $h(x)$ repeatedly until some $h'$ has $f(h'(x)) > f(x)$
4:     $c_t \leftarrow 0$
5:     while $c_t < \tau$ do
6:         $c_t \leftarrow c_t + 1$; $x' \leftarrow h'(x)$
7:         if $f(x') > f(x)$ then
8:             $x \leftarrow x'$
9:         end if
10:    end while
11: end while
```

Algorithm 11 Generalised Random Gradient Hyper-heuristic

```
1: Choose $x \in S$ uniformly at random
2: while stopping conditions not satisfied do
3:     Choose $h \in H$ uniformly at random
4:     $c_t \leftarrow 0$
5:     while $c_t < \tau$ do
6:         $c_t \leftarrow c_t + 1$; $x' \leftarrow h(x)$
7:         if $f(x') > f(x)$ then
8:             $c_t \leftarrow 0$; $x \leftarrow x'$
9:         end if
10:    end while
11: end while
```

In the following two theorems, we show that the generalised hyper-heuristics have the same efficient performance on GAPPATH as the simple mechanisms when selecting between the same two operators, with a lower order additive term (if $\tau = O(n)$).

Theorem 4.2 *Initialised at* 0^n*, the expected runtime of the Generalised Greedy hyper-heuristic on* GAPPATH *is* $\frac{1}{3}\left(n^3 + 2n^2 + 2\tau n\right) - \tau$.

Proof: The GAPPATH function is optimised by requiring multiple repetitions of 1-bit and 2-bit consecutive improvements. There is only one possible successful move at each point on the path and only the use of the correct operator will lead to an improvement.

If a 1-bit improvement is necessary, the success probability of the 1BITFLIP operator is $\frac{1}{n}$ and, by the Waiting Time Argument (see Theorem 2.5), n expected 1-bit mutations will occur before an improvement is found. During this time, n 2-bit mutations will also be performed, as the Generalised Greedy hyper-heuristic applies both operators during the Decision Stage.

When a 2-bit improvement is necessary, the success probability of the 2BITFLIP operator is $\frac{1}{n} \cdot \frac{1}{n} \cdot 2$ (since two specific bits must be flipped, but can be chosen in either order) implying $\frac{n^2}{2}$ expected 2-bit mutations before an improvement is found. The same number of 1-bit mutations will be evaluated in parallel.

After an operator has succeeded, the Generalised Greedy hyper-heuristic will deterministically apply the successful operator (on its own) for another τ iterations (the Exploitation Stage), during which no improvement can occur (due to the nature of the GAPPATH function), before starting the next Decision Stage.

Initially, a 2-bit improvement is required. The expected number of fitness function evaluations needed to reach the first point on the path 110^{n-2} from the 0^n bit string is $2 \cdot \frac{n^2}{2} = n^2$, which will be followed by τ non-improving steps. The next improvement will then occur by the 1BITFLIP operator after an expected $2n$ steps, followed by τ unsuccessful 1-bit flips. Hence, the expected time to optimise the first three bits is $n^2 + \tau + 2n + \tau = n^2 + 2n + 2\tau$.

This three bit pattern occurs a total of $n/3$ times before the optimum is reached, and since the global optimum is reached just before the final Exploitation Stage starts, the total expected runtime is $\frac{n}{3} \cdot (n^2 + 2n + 2\tau) - \tau = \frac{1}{3} \cdot (n^3 + 2n^2 + 2\tau n) - \tau$. □

Theorem 4.3 *Initialised at 0^n, the expected runtime of the Generalised Random Gradient hyper-heuristic on* GAPPATH *is at most $\frac{1}{3}(n^3 + 2n^2 + 4\tau n) - \tau$.*

Proof: The Generalised Random Gradient hyper-heuristic chooses a mutation operator uniformly at random, and applies it for a period of τ iterations. If an improvement is produced, a new period of τ iterations using the same operator is started immediately.

Let T_1 and T_2 respectively be the number of iterations before a 1-bit or a 2-bit improvement is constructed when it is possible to do so. If the appropriate operator i is chosen (i.e., with probability $\frac{1}{2}$), it may succeed with probability p_i in each iteration (with $p_1 = \frac{1}{n}$ and $p_2 = \frac{2}{n^2}$), or it may fail during all τ iterations with probability $(1 - p_i)^\tau$ in which case the random choice is repeated. If the inappropriate operator is chosen, τ iterations are wasted before the random choice is repeated. Combined, using $q_i := 1 - p_i$ for brevity:

$$\begin{aligned}
\mathrm{E}(\mathrm{T}_i) &= \frac{1}{2}\left(\left(\sum_{k=1}^{\tau} k q_i^{k-1} p_i\right) + q_i^\tau(\tau + \mathrm{E}(\mathrm{T}_i))\right) + \frac{1}{2}(\tau + \mathrm{E}(\mathrm{T}_i)) \\
&= \left(\sum_{k=1}^{\tau} k q_i^{k-1} p_i\right) + q_i^\tau \mathrm{E}(\mathrm{T}_i + \tau) + \tau \qquad (4.1) \\
&= \left(\sum_{k=1}^{\tau} k q_i^{k-1} p_i\right) + q_i^\tau \left(\left(\sum_{k=1}^{\tau} (k+\tau) q_i^{\tau+k-1} p_i\right) + q_i^\tau \left(\sum_{k=1}^{\tau} (k+2\tau) \ldots\right) + \tau\right) + \tau
\end{aligned}$$

$$= \sum_{j=0}^{\infty} \left(\left(\sum_{k=1}^{\tau} (k+j\tau) q_i^{j\tau+k-1} p_i \right) + \tau q_i^{j\tau} \right) \tag{4.2}$$

$$= \left(\sum_{k=1}^{\infty} k q_i^{k-1} p_i \right) + \tau + \frac{\tau q_i^{\tau}}{1-q_i^{\tau}} = \frac{1}{p_i} + \tau + \frac{\tau q_i^{\tau}}{1-q_i^{\tau}}. \tag{4.3}$$

Rearranging the original equation yields (4.1), where we have $\tau + \mathrm{E}(\mathrm{T}_i) = \mathrm{E}(\mathrm{T}_i + \tau)$. Recursively expanding the $q_i^{j\tau}\,\mathrm{E}(\mathrm{T}_i + j\tau)$ terms and simplifying to be included in one infinite sum yields (4.2). By separating terms, we can rearrange into an infinite sum for the expectation of a geometrically-distributed random variable and a sum of an infinite geometric series in (4.3). The expected number of periods of time τ where the correct operator is applied but does not produce an improvement is $\frac{q_i^{\tau}}{1-q_i^{\tau}}$. Thus, $\frac{\tau q_i^{\tau}}{1-q_i^{\tau}}$ is a lower bound on the number of iterations where a correct operator is applied before a success (i.e., $\frac{1}{p_i}$) and $\tau + \frac{\tau q_i^{\tau}}{1-q_i^{\tau}}$ is an upper bound on $\frac{1}{p_i}$. Substituting these bounds into Eq. (4.3) yields $\frac{2}{p_i} \le \mathrm{E}(\mathrm{T}_i) \le \frac{2}{p_i} + \tau$.

After an improvement is constructed, the successful operator will run for an additional τ iterations while not being able to construct the next improvement. To reach the optimum from 0^n, improvements must be constructed by alternating 2-bit ($\mathrm{E}(\mathrm{T}_2) \le n^2 + \tau$) and 1-bit mutations ($\mathrm{E}(\mathrm{T}_1) \le 2n + \tau$), $n/3$ times, while the final τ-iteration Exploitation Stage does not occur. The total expected optimisation time is therefore:

$$\frac{n}{3}(\mathrm{E}(\mathrm{T}_1) + \mathrm{E}(\mathrm{T}_2) + 2\tau) - \tau \le \frac{1}{3}(n^3 + 2n^2 + 4\tau n) - \tau.$$

□

Although the GAPPATH function is the worst case scenario for the generalised hyper-heuristics (since the best low-level heuristic alternates with every success and a smart choice of heuristic cannot be exploited), they still show efficient performance. In order to see that the generalised hyper-heuristics are able to learn to prefer certain heuristics and exploit smart heuristic choices, we now analyse their performance on the GENERALISEDGAPPATHWITHTRAPS function.

The following theorem considers two asymptotic ranges of values for τ. For $\tau = O(n^2)$, we show that Generalised Greedy will construct a trap point with overwhelming probability, while if $\tau = \Omega(n^3)$, it avoids all trap points with overwhelming probability.

Theorem 4.4 *Initialised at 0^n for $\tau = O(n^2)$, the Generalised Greedy hyper-heuristic will fail to reach the global optimum of GGPT_k in finite time with probability at least $1 - 2^{-\Omega(n)}$. However, for $\tau = \Omega(n^3)$, the Generalised Greedy hyper-heuristic will find the global optimum* *of* GGPT$_k$ *($k = \Theta(1)$) in at most* $\frac{2\tau n}{2k+1} + \frac{n^3}{2k+1} + O(n^2)$ *expected steps with probability at least* $1 - 2^{-\Omega(n)}$.

Proof: The Generalised Greedy hyper-heuristic avoids making a decision inside the two-bit region if and only if it can produce $k-1$ two-bit improvements within τ iterations following the first two-bit improvement. Let X be a binomially-distributed random variable with parameters $p = 2/n^2$ (the success probability of the two-bit mutation operator) and τ trials; we have $\mathrm{E}(X) = 2\tau/n^2$. We note that for $0 \leq X < k-1$, the probability distribution of X is identical to that of the number of two-bit improvements constructed during the Decision Stage. We also note that $X \geq k-1$ corresponds to constructing all remaining $k-1$ improvements.

For $\tau = cn^2$, we note that the probability of constructing exactly zero improvements during the τ iterations is at least $(1-2/n^2)^{cn^2} \geq e^{-2c} - O(1/n^2) = \Omega(1)$. If another Decision Stage is performed inside the region of two-bit improvements, a 'trap' point is accessible via a 1-bit mutation from the point at which the decision occurs with probability at least $1/(k-1) = \Theta(1)$ (as traps are placed uniformly at random inside these regions), and is found by the Greedy mechanism with probability at least $\frac{1/n}{1/n+2/n^2} \geq 1/3$. Thus, the probability that the Generalised Greedy hyper-heuristic does not construct a 'trap' point in a single region of k two-bit improvements is at most $2/3$, which means that over the $n/(2k+1)$ such regions, the probability that a 'trap' point is constructed is at least $1-2^{-\Omega(n)}$.

For $\tau = cn^3$, where c is a constant, $\mathrm{E}(X) = 2cn$. Applying a Chernoff bound (see Theorem 2.10), we note that the probability of observing fewer than cn successes in τ steps is at most $2^{-\Omega(n)}$, and if n is sufficiently large, $cn > k$. Thus, with overwhelming probability $1-2^{-\Omega(n)}$, a decision step does not occur inside a single two-bit region, and by a union bound (Theorem 2.2 from Chapter 2; $(1-2^{-\Omega(n)})^{n/(2k+1)} = 1-2^{-\Omega(n)}$), does not occur inside any of the $n/(2k+1)$ two-bit regions in the bit string. Thus, with overwhelming probability, the algorithm is not trapped. Hence, for each region of $2k+1$ bit-flips, one 1-bit flip (found in the Decision Stage, taking expected time $2n$) and k 2-bit flips (the first is found in the Decision Stage, taking expected time n^2, and the rest are found within the τ steps of the Exploitation Stage) have to be performed, giving an expected optimisation time of at most

$$\frac{n}{2k+1}\left(2n+\tau+n^2+\tau\right) = \frac{2\tau n}{2k+1} + \frac{n^3}{2k+1} + O(n^2).$$

□

The following theorem for the Generalised Random Gradient hyper-heuristic can be proved using a similar approach.

Theorem 4.5 *Initialised at* 0^n *for* $\tau = O(n^2)$*, the Generalised Random Gradient hyper-heuristic will fail to reach the global optimum of* GGPT_k *in finite time with probability at least* $1-2^{-\Omega(n)}$*. However, for* $\tau = \Omega(n^3)$*, the Generalised Random Gradient hyper-heuristic will find the global optimum of* GGPT_k *(*$k=\Theta(1)$*) in at most* $\frac{4\tau n}{2k+1}+\frac{kn^3}{2(2k+1)}+\Theta(n^2)$ *expected steps with probability at least* $1-2^{-\Omega(n)}$*.*

Proof: The Generalised Random Gradient hyper-heuristic avoids making a decision inside the region of 2-bit successes if it is able to construct $k-1$ two-bit improvements within τ iterations each. The probability of success for a 2-bit mutation is $2/n^2$, the probability of no successes in τ steps is $(1-2/n^2)^\tau$ and the probability of (at least) one success in a period of τ steps is $1-(1-2/n^2)^\tau$. Thus, the probability that all $k-1$ two-bit improvements are found before a trap is constructed is $\left(1-(1-2/n^2)^\tau\right)^{k-1}$.

For $\tau = cn^2$, the probability of the required $k-1$ successful periods is $\left(1-\left(1-2/n^2\right)^{cn^2}\right)^{k-1} = (1-e^{-2c}-o(1))^{k-1} = \Theta(1)$, and so the probability of failing to do so is constant. If another random Decision Stage is performed inside the region of two-bit improvements, a 'trap' point is accessible via a 1-bit mutation from the point at which the decision occurs with probability at least $1/(k-1) = \Theta(1)$ (as traps are placed uniformly at random inside these regions) and is found by the Generalised Random Gradient hyper-heuristic with probability at least $1/2\cdot(1-(1-1/n)^{cn^2}) \geq 1/4$ for $n \geq \ln(2)/c$. Thus, the probability that the no 'trap' point is constructed in a single region of k two-bit improvements is at most $3/4$, which means that over the $n/(2k+1)$ such regions, the probability that a 'trap' point is constructed is at least $1-2^{-\Omega(n)}$.

For $\tau = cn^3$, the probability that all $T = (k-1)\cdot n/(2k+1) = \Theta(n)$ two-bit successes are constructed within τ iterations is at least $\left(1-\left(1-2/n^2\right)^{cn^3}\right)^T \geq (1-e^{-2cn})^T \geq 1-Te^{-2cn} = 1-2^{-\Omega(n)}$ by applying a union bound (Theorem 2.2 from Chapter 2, and the fact that $\left(1-\frac{1}{n}\right)^n \leq \frac{1}{e}$). Thus, with overwhelming probability, the algorithm is not trapped. Hence, for each region of $2k+1$ bit-flips, there must be one 1-bit flip (taking expected time n) and k 2-bit flips (each taking expected time $n^2/2$) and the expected optimisation time is at most

$$\frac{n}{2k+1}\left(n+2\tau+\frac{k}{2}n^2+2\tau\right) = \frac{4\tau n}{2k+1}+\frac{kn^3}{2(2k+1)}+\Theta(n^2),$$

with the 2τ terms corresponding to using the wrong operator for, in expectation, one period of τ iterations at the start of the phase, and using the successful operator for τ iterations while the next improvement can only be constructed by the other operator. □

We point out that, if the function values were inverted such that the 'trap' points had optimal fitness and the 1^n bit string did not, then the generalised hyper-heuristics still find the 1^n bit-string and fail to find any of the global optima (w.o.p), while the 'simple' mechanisms would still construct a 'trap' point, hence finding the global optimum (w.o.p). It is not surprising that learning mechanisms fail on a function especially designed to deceive them.

We have seen that the Generalised Greedy hyper-heuristic is able to make a smart choice of heuristic and the Generalised Random Gradient hyper-heuristic is able to exploit a good heuristic choice. In the next chapters, we will analyse the hyper-heuristics on the unimodal benchmark function LEADINGONES to show that they have good performance on functions where the low-level heuristics are also efficient.

4.3 Chapter Summary and Discussion

Previously, Lehre and Özcan (2013) analysed the performance of the Simple Random hyper-heuristic mechanism on the pseudo-Boolean function GAPPATH, whereby it is necessary to alternate two mutation operators. In this chapter, we have generalised the function such that it is necessary for an algorithm to learn to favour a specific mutation operator. We proved that four 'simple' mechanisms (namely Simple Random, Permutation, Random Gradient and Greedy) failed to find the global optimum of the function with overwhelming probability. In order to be able to exploit a good heuristic choice, we generalised the Random Gradient and Greedy algorithm such that success is measured over a learning period τ. We showed that the new Generalised Random Gradient and Generalised Greedy hyper-heuristics, whilst also being efficient on GAPPATH, were able to find the global optimum of the GENERALISEDGAPPATHWITHTRAPS (GGPT) function with overwhelming probability.

We will see in the next chapter that the Generalised Greedy and Generalised Random Gradient hyper-heuristics can be faster than standard evolutionary and local search algorithms, even when the former are efficient on the problem at hand. We will analyse their performance on the unimodal benchmark function LEADINGONES. Furthermore, we will rigorously prove on the same function (up to tight bounds on the leading constants) that the 'simple' hyper-heuristic mechanisms are not learning, but rather essentially choosing operators at random, as suggested by the experimental analysis of Alanazi and Lehre (2014). Hence, we provide evidence that the generalisations of the hyper-heuristics are indeed useful.

Chapter 4 is based on the following publication:

1. Lissovoi, A., Oliveto, P. S., and Warwicker, J. A. (2017). On the Runtime Analysis of Selection Hyper-heuristics for Pseudo-Boolean Optimisation. *Proceedings of the*

Annual Conference on Genetic and Evolutionary Computation (GECCO '17), pages 849-856. ACM.

Chapter 5

An Example Where Mutation-based Hyper-heuristics Are Faster

In Chapter 4 we motivated the use of hyper-heuristics (HHs) by showing that in some cases it is necessary to employ a HH methodology. In particular, we presented example problem classes where it is necessary to use multiple heuristics and where it is necessary for a hyper-heuristic to 'learn' which heuristic is preferable for efficient optimisation to occur with the available low-level heuristics.

In this chapter, we address the question of whether a HH may be faster than its low-level heuristics used by themselves. In particular, we investigate whether by applying different low-level heuristics at different stages of the optimisation process, a HH can outperform its best-performing low-level heuristic for the problem. By considering the LEADINGONES (LO) benchmark problem and a low-level heuristic set $H = \{1\text{BITFLIP}, 2\text{BITFLIP}\}$, we prove that this is the case for the generalised HHs introduced in the previous chapter. Concerning Generalised Random Gradient, we prove an even stronger result: the Generalised Random Gradient HH with an appropriate value for the learning period runs in the best possible expected runtime achievable by any algorithm using these two heuristics (up to lower order terms). We begin the chapter by deriving the best possible runtime achievable by any combination of the operators for LO (i.e., $\frac{1+\ln 2}{4}n^2 + o(n^2) \approx 0.423n^2$) in Section 5.1. In Section 5.2 we prove a conjecture derived from an experimental analysis of Alanazi and Lehre (2014): that the four simple hyper-heuristic mechanisms that do not use a learning period (i.e., Simple Random, Permutation, Greedy and Random Gradient) all have the same expected runtime (i.e., $\frac{\ln 3}{2}n^2 + o(n^2) \approx 0.549n^2$) on LO. The result implies that they all essentially choose heuristics at random. We then analyse the generalised HHs (Generalised Greedy and Generalised Random Gradient) in Section 5.3. We rigorously prove that both generalised HHs outperform Randomised Local Search up to lower order terms for appropriately chosen values of the learning period τ and that the Generalised Random Gradient HH achieves the best possible leading constant in the expected runtime for learning periods τ that satisfy both $\tau = \omega(n)$ and $\tau \leq (\frac{1}{2} - \varepsilon)n \ln n$, for some constant $\varepsilon > 0$. On the other hand, the best expected runtime that the Generalised Greedy HH can achieve is $\left(\frac{\ln 5}{8} + \frac{\arctan 2}{4}\right) n^2 + o(n^2) \approx 0.478n^2$ for $\tau = \omega(n) \cap o(n^2)$. This bound is tight since we also prove a matching expected runtime for an ideal Greedy algorithm which performs a Decision Stage 'for free' in every iteration. We supplement the theoretical results with an experimental analysis for different problem sizes (up to $n = 10^8$) in Section 5.4.

5.1 Best Performance Achievable on LEADINGONES with Combinations of 1BITFLIP and 2BITFLIP

We begin this section by presenting some existing theoretical results regarding the LEADINGONES benchmark function. Recall that the LEADINGONES function counts the number of consecutive one-bits in a bit string before the first zero-bit.

$$\mathrm{LO}(x) := \sum_{i=1}^{n} \prod_{j-1}^{i} x_j$$

Recall from Chapter 2 that Randomised Local Search has an expected runtime of $0.5n^2$ (Buzdalov and Buzdalova, 2015) on LEADINGONES while the standard (1+1) EA (with mutation rate $\frac{1}{n}$) has an expected runtime of $\frac{e-1}{2}n^2 \pm o(n^2) \approx 0.859n^2$ (Böttcher et al., 2010). In Section 3.4, we saw that Böttcher et al. (2010) proved that an appropriately chosen dynamic mutation rate can outperform any static choice of mutation rate for the (1+1) EA on LEADINGONES (i.e., $\approx 0.680n^2$).

We now introduce a lower bound on the expected runtime of all algorithms which use only the 1BITFLIP and 2BITFLIP operators. The expected runtime of such an algorithm can be proved following the approach used for the (1+1)-EA by Böttcher et al. (2010) (using Theorem 2.13 from Chapter 2), which was shown to work for Randomised Local Search (RLS) by Buzdalov and Buzdalova (2015). Since the algorithm we consider is an unbiased (1+1) black box algorithm, we can use the same theorem. Furthermore, we know from this that any successful mutation will lead to the same expected increase in the fitness. As the result of Theorem 2.13 suggests, using the operator that maximises the probability of a successful mutation will minimise the expected waiting time for each fitness increase on LEADINGONES (note that this is not necessarily the case for other functions). Hence, using an operator with the maximum success probability at each point will lead to the fastest possible algorithm using such operators. We present this result for the set of heuristics $H = \{1\text{BITFLIP}, 2\text{BITFLIP}\}$ in Theorem 5.1.

Since the 1BITFLIP operator has a success probability (the probability of a fitness improvement after the use of this operator) of $\frac{1}{n}$, while the 2BITFLIP operator has a success probability of $\frac{1}{n} \cdot \frac{n-i-1}{n} \cdot 2 = \frac{2n-2i-2}{n^2}$ where $i = \text{LO}(x)$, for $i \leq \frac{n-1}{2}$ the 2BITFLIP operator is more effective (i.e., has a lower expected waiting time for each improvement), while the 1BITFLIP operator is preferable afterwards. Hence, the expected runtime of an algorithm using these operators in such a way gives a lower bound on all stochastic unbiased algorithms using only the same two operators.

Theorem 5.1 *The best-possible expected runtime on* LEADINGONES *for an algorithm using only the* 1BITFLIP *and* 2BITFLIP *operators is* $\frac{1+\ln(2)}{4}n^2 + O(n) \approx 0.42329n^2$.

Proof: We partition the analysis into two phases: Phase 1, where the 2BITFLIP operator is used for the first half of the search ($0 \leq i \leq \frac{n}{2} - 1$), and Phase 2, where the 1BITFLIP operator is used for the second half of the search ($\frac{n}{2} \leq i \leq n-1$). We consider their expected runtimes ($\text{E}(\text{T}_1)$ and $\text{E}(\text{T}_2)$) separately and sum them to find the final expected runtime.

The 2BITFLIP operator can be considered as an unbiased (1+1) black box algorithm, and hence Theorem 2.13 holds for Phase 1. To find the expected runtime for this phase, we can consider every point $\text{LO}(x) = i \geq \frac{n}{2}$ to be optimal and we seek the expected time until a solution with such fitness is constructed. Since the probability of improvement of the 2BITFLIP operator is $\frac{2n-2i-2}{n^2}$, the expected time to make an improvement for $0 \leq i < \frac{n}{2}$ is $A_{n-i} = \frac{n^2}{2n-2i-2}$. We also consider for $i \geq \frac{n}{2}$, $A_{n-i} = 0$. Hence, we note that for Phase 1,

$$\begin{aligned}\text{E}(\text{T}_1) &= \frac{1}{2}\sum_{i=0}^{n-1} A_{n-i} = \frac{1}{2}\sum_{i=0}^{n/2-1} \frac{n^2}{2n-2i-2} = \frac{1}{2}\sum_{k=n/2}^{n-1} \frac{n^2}{2k} \\ &= \frac{n^2}{4}\sum_{k=n/2}^{n-1} \frac{1}{k} = \frac{n^2}{4}\Big(H_{n-1} - H_{n/2-1}\Big),\end{aligned}$$

where H_x is the x^{th} Harmonic number, which can be bounded by

$$\ln(n+1) + \gamma - \frac{1}{n+1} \leq H_n \leq \ln(n+1) + \gamma - \frac{1}{2(n+1)},$$

and γ is the Euler-Mascheroni constant ($\gamma \approx 0.57722\ldots$). Hence, we can bound the term $H_{n-1} - H_{n/2-1}$ from below by $\ln(2)$, and from above by $\ln(2) + \frac{3}{2n}$, which gives the following bounds on $\mathrm{E}(\mathrm{T}_1)$:

$$\ln(2) \cdot \frac{n^2}{4} \leq \mathrm{E}(\mathrm{T}_1) \leq \left(\ln(2) + \frac{3}{2n}\right) \cdot \frac{n^2}{4}.$$

Since $\mathrm{E}(\mathrm{T}_1)$ is at most $\frac{3n}{8} = O(n)$ greater than the lower bound of $\ln(2) \cdot \frac{n^2}{4}$, we have:

$$\mathrm{E}(\mathrm{T}_1) = \frac{\ln(2)}{4} n^2 + O(n).$$

The expected runtime for Phase 2 can be calculated using Theorem 2.13 with $A_{n-i} = n$ for $i \geq \frac{n}{2}$ (as Buzdalov and Buzdalova (2015) did with RLS). Similarly to Phase 1, we state $A_{n-i} = 0$ for $i < \frac{n}{2}$.

$$E(\mathrm{T}_2) = \frac{1}{2} \sum_{i=0}^{n-1} A_{n-i} = \frac{1}{2} \sum_{j=1}^{n/2} n = \frac{1}{2} \cdot n \cdot \frac{n}{2} = \frac{n^2}{4}.$$

Hence, by the linearity of expectation, the total expected runtime is given by the sum of the two phases:

$$\frac{1+\ln(2)}{4} n^2 \leq E(T_1) + E(T_2) \leq \frac{1+\ln(2)}{4} n^2 + \frac{3n}{8}.$$

□

We have proved a theoretical lower bound of $\frac{1+\ln(2)}{4} n^2 + O(n) \approx 0.42329n^2$ for any algorithm using the 1BITFLIP and 2BITFLIP operator. Note specifically that this performance is faster than the RLS (or 1BITFLIP) algorithm, which has expected runtime $0.5n^2$.

5.2 Simple Hyper-heuristics are Slow

In this section we show that the standard 'simple' learning mechanisms (heuristic selection mechanisms) that do not use a learning period (i.e., Simple Random, Permutation, Greedy and Random Gradient) all have the same expected runtime on LEADINGONES, up to lower order terms, as suggested by the experimental analysis of Alanazi and Lehre (2014). This implies that these four mechanisms do not learn and all essentially choose low-level heuristics at random in each iteration, similarly to the Simple Random mechanism. Theorem 5.2 derives the expected runtime for the Simple Random mechanism. Corollary 5.3 extends the result to all the other 'simple' mechanisms.

Theorem 5.2 *Let the Simple Random mechanism apply the* 1BITFLIP *mutation operator independently in each iteration with probability p (and* $1-p$ *for the* 2BITFLIP *operator). The expected runtime of the Simple Random mechanism on* LEADINGONES *for* $p \in (0,1)$ *is* $\frac{1}{4(1-p)} \ln\left(\frac{2-p}{p}\right) n^2 + o(n^2)$. *If* $p = 0$ *the expected runtime is infinite. If* $p = 1$, *the expected runtime is* $0.5n^2$.

Proof: If $p = 0$, only the 2BITFLIP operator is used. There is a non-zero probability of reaching the point $1^{n-1}0$, which cannot be improved by using the 2BITFLIP operator. Hence, by the law of total probability, the expected runtime is infinite.

If $p = 1$, only the 1BITFLIP operator is used, resulting in exactly RLS, which has expected runtime $0.5n^2$.

Since the 1BITFLIP and 2BITFLIP operators are unbiased (1+1) black box algorithms, we can use Theorem 2.13. If $p \in (0,1)$, we have an expected runtime of $\mathrm{E(T)} = \frac{1}{2}\sum_{i=1}^{n} A_{n-i}$, where A_{n-i} is the expected time needed to find an improvement given a solution with fitness i and the algorithm is initialised with a search point chosen uniformly at random. In each iteration, the 1BITFLIP operator is chosen with probability p and leads to a fitness improvement with probability $\frac{1}{n}$; the 2BITFLIP operator is chosen with probability $1-p$ and leads to a fitness improvement with probability $\frac{2n-2i-2}{n^2}$. Hence, $(A_{n-i})^{-1} = p \cdot \frac{1}{n} + (1-p) \cdot \frac{2n-2i-2}{n^2}$, and

$$A_{n-i} = \frac{1}{\frac{p}{n} + \frac{(1-p)(2n-2i-2)}{n^2}} = \frac{n^2}{2(1-p)(n-i-1)+np}.$$

Hence, by Theorem 2.13, the total expected runtime is

$$\frac{1}{2}\sum_{i=0}^{n-1} \frac{n^2}{2(1-p)(n-i-1)+np} = \frac{n^2}{2}\sum_{k=1}^{n} \frac{1}{(2p-2)k+(2-p)n} \tag{5.1}$$

$$= \frac{n^2}{2(2p-2)}\left(\ln\left(\frac{n+\frac{(2-p)n}{2p-2}}{\frac{(2-p)n}{2p-2}}\right) + o(1)\right) = \frac{1}{4(1-p)} \ln\left(\frac{2-p}{p}\right) n^2 + o(n^2), \tag{5.2}$$

where Equation 5.1 becomes Equation 5.2 via the following simplification:

$$\sum_{k=1}^{n} \frac{1}{a \cdot k + b \cdot n} = \frac{1}{a} \cdot \sum_{k=1}^{n} \frac{1}{k + \frac{b}{a} \cdot n} = \frac{1}{a}\left(\sum_{k=1}^{(1+\frac{b}{a})n} \frac{1}{k} - \sum_{k=1}^{\frac{b}{a}n} \frac{1}{k}\right)$$

$$= \frac{1}{a}\left(\ln\left(\frac{1+\frac{b}{a}}{\frac{b}{a}}\right) + o(1)\right),$$

with $a = 2p - 2$ and $b = 2 - p$. □

When $p = 0.5$ (i.e., there is an equal chance of choosing each operator in each iteration), the standard Simple Random mechanism has $A_{n-i} = \frac{2n^2}{3n-2i-2}$ and has expected runtime $\frac{\ln(3)}{2}n^2 + o(n^2) \approx 0.54931n^2$. This is worse than the expected runtime of the single operator algorithm RLS, which has expected runtime $0.5n^2$. The expected runtime improves with increasing p, hence the optimal choice is $p = 1$.

Corollary 5.3 *The expected runtime of the Permutation, Greedy and Random Gradient mechanisms on* LEADINGONES *is* $\frac{\ln(3)}{2}n^2 + o(n^2) \approx 0.54931n^2$.

Proof: Let p_i be the probability that at least one improvement is constructed within two fitness function evaluations. For the Greedy and Permutation mechanisms, we have $p_i = \frac{1}{n} + \left(1 - \frac{1}{n}\right) \cdot \frac{2n-2i-2}{n^2}$ as either the 1BITFLIP operator can succeed in one iteration or it can fail and then the 2BITFLIP operator can succeed in the next iteration. To upper bound the expected optimisation time, we note that the difference between $2/p_i = \frac{2n^2}{3n-2i-2} + O(1)$ (i.e., the expected waiting time for an improvement to be constructed in terms of fitness evaluations) and the A_{n-i} waiting times for the standard Simple Random mechanism with $p = 0.5$ ($A_{n-i} = \frac{2n^2}{3n-2i-2}$) is limited to a lower order constant term and thus the difference between the expected runtimes of these mechanisms and Simple Random is limited to lower-order terms (through the application of Theorem 2.13). We note that with probability at most $\frac{2}{n^2}$, both mutations (the two mutations performed in parallel by the Greedy mechanism, or the two mutations performed sequentially by the Permutation mechanism) considered are improvements. An upper bound on the mechanism is given by assuming each random choice of the 2BITFLIP operator fails and only the 1BITFLIP operator succeeds, giving an upper bound of $2 \cdot \frac{n^2}{2} = n^2 = O(n^2)$. As two sequential improvements occurs at most a constant number of times in expectation (i.e., $O(n^2) \cdot \frac{2}{n^2} = O(1)$) and the maximum expected waiting time for any improving step is $O(n)$, the lower bound differs from the upper bound by at most an $O(1) \cdot O(n) = O(n)$ term. Thus, the expected runtime of these mechanisms is also $\frac{\ln(3)}{2} \cdot n^2 + o(n^2)$.

For the Random Gradient mechanism, we note that the probability that an operator, when repeated following a success, is successful again is at most $\frac{2}{n}$ (an upper bound on the success probability of the 2BITFLIP operator). Since there are at most n improvements to be made throughout the search space, the expected number of repeats which produce an improvement is at most 2. If the chosen operator is not successful, the Random Gradient mechanism behaves identically to the Simple Random mechanism. Its expected runtime is therefore at least the expected runtime of the Simple Random mechanism less an $O(n)$ term and at most the expected runtime of the Simple Random mechanism plus n (as there are at most n iterations where the mechanisms differ in operator selection). Thus, its expected runtime is also $\frac{\ln(3)}{2} \cdot n^2 + o(n^2)$. □

We point out that the expected runtime for the Random Gradient hyper-heuristic on LEADINGONES given by Corollary 5.3 contradicts the lower bound of $\frac{n^2}{9} \cdot \left(4 + 3\ln\left(\frac{10}{3}\right)\right) \approx 0.846n^2$ found by Alanazi and Lehre (2014). However, their bound results from a small mistake in their proof and they should have found a lower bound of $\frac{n^2}{9} \cdot \left(3\ln\left(\frac{10}{3}\right)\right) + o(n^2) \approx 0.401n^2$, which agrees with our result of $\frac{\ln 3}{2} n^2 + o(n^2) \approx 0.549n^2$.

Theorem 5.2 and Corollary 5.3 provide matching expected runtimes (up to lower order terms), implying that similarly to the Simple Random mechanism, the Permutation, Greedy and Random Gradient mechanisms essentially choose heuristics at random rather than learning which one is preferable. In the next section, we show that the generalised hyper-heuristics introduced in Chapter 4 are able to achieve faster expected runtimes on LEADINGONES than the single-operator mechanism RLS.

5.3 Generalised Hyper-heuristics Achieve Optimal Expected Runtime

In this section we present a rigorous theoretical analysis of the generalised hyper-heuristics introduced in Chapter 4 on the LEADINGONES benchmark function. We start by introducing an important mathematical result which will be used in the analyses.

Theorem 5.4 ***Wald's Inequality (Wald, 1944)*** *Let T be a random variable with bounded expectation and let $X_1, X_2, \ldots$ be non-negative random variables with $\mathrm{E}(X_i | T \geq i) \leq C$. Then*

$$\mathrm{E}\left(\sum_{i=1}^{T} X_i\right) \leq \mathrm{E(T)} \cdot C.$$

In Subsection 5.3.1, we analyse the performance of the Generalised Greedy hyper-heuristic, while the Generalised Random Gradient hyper-heuristic will be analysed in Subsection 5.3.2.

5.3.1 Generalised Greedy Hyper-Heuristic

Recall that the Generalised Greedy hyper-heuristic initially runs all low-level heuristics in a Decision Stage, before applying, for τ iterations, the heuristic that is the first to make a fitness improvement in an Exploitation Stage. Then, another Decision Stage begins and the process is repeated until some termination condition is satisfied.

We first present the expected runtime of an 'ideal' Greedy algorithm on LEADINGONES. This ideal algorithm is sped-up by getting the result of the Decision Stages of the Generalised Greedy hyper-heuristic for 'free' in every iteration and uses this result to decide which operator to apply. In particular, if the algorithm is at $\mathrm{LO}(x) = i$, the 1BITFLIP operator is chosen with probability $\frac{n}{3n-2i-2}$ and the 2BITFLIP operator is chosen with probability $\frac{2n-2i-2}{3n-2i-2}$. This gives the same probabilities as the Generalised Greedy hyper-heuristic, yet saves time by not running any Decision Stages. This algorithm is equivalent to running the Exploitation Stages of the Generalised Greedy hyper-heuristic without the Decision Stages. Its expected runtime provides a lower bound on the expected runtime of the Generalised Greedy hyper-heuristic.

Theorem 5.5 *The 'ideal' greedy algorithm which, in each iteration, independently at random applies the* 1BITFLIP *operator with probability* $\frac{n}{3n-2i-2}$ *and the* 2BITFLIP *operator with probability* $\frac{2n-2i-2}{3n-2i-2}$*, where* $i = \mathrm{LO}(x)$ *is the fitness of the current solution, on* LEADINGONES*, has expected runtime* $\left(\frac{\ln(5)}{8} + \frac{\arctan(2)}{4}\right) n^2 + o(n^2) \approx 0.47797n^2$.

Proof: The probability of the Decision Stage choosing each operator is $\frac{1/n}{1/n+(2n-2i-2)/n^2} = \frac{n}{3n-2i-2}$ for the 1BITFLIP operator and $\frac{(2n-2i-2)/n^2}{1/n+(2n-2i-2)/n^2} = \frac{2n-2i-2}{3n-2i-2}$ for the 2BITFLIP operator. Using Theorem 2.13, we have:

$$\begin{aligned}
\mathrm{E(T)} &= \frac{1}{2} \cdot \sum_{i=0}^{n-1} \left(\frac{n}{3n-2i-2} \cdot \frac{1}{n} + \frac{2n-2i-2}{3n-2i-2} \cdot \frac{2n-2i-2}{n^2} \right)^{-1} \\
&= \frac{1}{2} \cdot \sum_{i=0}^{n-1} \frac{(3n-2i-2)n^2}{(2n-2i-2)^2+n^2} = \left(\frac{\ln(5)}{8} + \frac{\arctan(2)}{4} \right) n^2 + o(n^2).
\end{aligned}$$

□

The main result of this subsection is that the Generalised Greedy hyper-heuristic is able to match, up to lower order terms, the performance of the 'ideal' Greedy algorithm when both algorithms select between the 1BITFLIP and 2BITFLIP low-level heuristics. We present the result in Corollary 5.6 now, with the proof at the end of the subsection.

Corollary 5.6 *[Of Theorem 5.8] The expected runtime of the Generalised Greedy hyper-heuristic on* LEADINGONES *with* $\tau = \omega(n) \cap o(n^2)$ *is at most* $\left(\frac{\ln(5)}{8} + \frac{\arctan(2)}{4}\right) n^2 + o(n^2) \approx 0.47797n^2$.

We first present some prerequisite results for the analysis. To simplify the presentation of the proofs in this subsection, we will use 2 as a bound on the expected increase in fitness value following a successful mutation throughout the process. As the bits following the first zero-bit in any individual remain uniformly distributed throughout the process, this expectation would be correct if the bit string was of infinite length. We note that the law of total expectation can be used to prove that this over-estimation will only affect lower-order terms in the derived expected runtimes. For example, the free-riders (1-bits after the first 0-bit) are overwhelmingly unlikely to span all the way to the end of the bit string while the parent individual has fitness at most $n-\log^2 n$ and an upper bound for Generalised Greedy on the last $k \leq n$ bits is $k(2n+\tau) = o(n^2)$, since each (of at most k) Decision Stage is expected to last $2n$ steps (since the 1BITFLIP operator will always find a success in an expected n iterations and the 2BITFLIP operator will be run in parallel). In the worst case, the τ iterations are wasted with no improvements, and hence the law of total expectation implies that the runtime will be affected by at most $2^{-\log^2 n} \cdot \tau \log^2 n = o(n^2)$.

We begin by stating a lemma on the expected time and progress for a combined Decision and Exploitation stage (i.e., the expected time and progress when making a greedy choice of low-level heuristic and applying it for τ iterations).

Lemma 5.7 *Let $i = \text{LEADINGONES}(x^{(t)}) < n-1$, and $Y^{(t)} = n-i$ be the distance of the Generalised Greedy hyper-heuristic's current solution at the start of the t^{th} Decision Stage to the* LEADINGONES *optimum, and let $\tau = o(n^2)$. Then,*

$$E(Y^{(t)} - Y^{(t+1)} \mid Y^{(t)} = n-i \geq 2) = \frac{2\cdot\tau\cdot(5n^2-(8i+8)n+4(1+i)^2)}{n^2(3n-2i-2)} + 2 - o(1),$$

and in expectation, the next Decision Stage begins after

$$\tau + \frac{2n^2}{3n-2i-\left(4-\frac{2(i+1)}{n}\right)}$$

fitness evaluations.

Proof: We look at the expected number of fitness evaluations for the Decision and Exploitation stages, given the starting position $\text{LO}(x) = i$.

During the Decision Stage, the hyper-heuristic will run both operators until one of them is successful (i.e., makes an improvement in the fitness). The expected number of fitness evaluations for this to happen is given by:

$$\begin{aligned}\mathrm{E}(\mathrm{T}_{\mathrm{DS}_i}) &= 2\cdot\left(1-\left(1-\frac{1}{n}\right)\left(1-\frac{2n-2i-2}{n^2}\right)\right)^{-1}\\ &= \frac{2n^3}{3n^2-2in+2i-4n+2} = \frac{2n^2}{3n-2i-(4-\frac{2(i+1)}{n})}.\end{aligned}$$

The probability of the Decision Stage choosing each operator, given by the first hit probability (Theorem 2.3 in Chapter 2) is $\frac{1/n}{1/n+(2n-2i-2)/n^2} = \frac{n}{3n-2i-2}$ for the 1BITFLIP operator and $\frac{(2n-2i-2)/n^2}{1/n+(2n-2i-2)/n^2} = \frac{2n-2i-2}{3n-2i-2}$ for the 2BITFLIP operator. It is possible for both operators to succeed in one iteration. For our analysis, we note that this happens an expected $O(n)\cdot\frac{1}{n}\cdot\frac{2n-2i-2}{n^2} = O\left(\frac{1}{n}\right)$ times throughout the optimisation process and opt to ignore any progress made by such Decision and Exploitation stages, accounting for this by adding a lower-order $O(\tau+n)$ term to the expected runtime.

In the Exploitation Stage, the operator which succeeded during the Decision Stage runs for τ steps. Throughout the two stages, the function value will increase. We can lower bound the total increase by keeping $\mathrm{LO}(x) = i$ constant throughout (i.e., using the value from the Decision Stage) and upper bound it by applying a high-probability bound on the number of observed LEADINGONES improvements. The highest success probability occurs when using the 2BITFLIP operator at $i = 0$; i.e $\frac{2n-2}{n^2} < \frac{2}{n}$. Since $\tau = o(n^2)$, we can use Chernoff bounds (Theorem 2.10) to bound the number of successes in τ steps as $o(n)$ with probability $1-2^{-\Omega(n)}$ (i.e., the probability of $\Omega(n)$ successes in τ steps is at most $2^{-\Omega(n)}$). Hence, we can bound the number of LEADINGONES improvements from above by considering $\mathrm{LO}(x) = i + o(n)$. The probability of each operator being chosen will remain the same, since this is based on the original value of i at the Decision Stage.

Hence, the expected progress of the Exploitation Stage in τ iterations is at most

$$\begin{aligned}&2\cdot\tau\cdot\left(\frac{n}{3n-2i-2}\cdot\frac{1}{n}+\frac{2n-2i-2}{3n-2i-2}\cdot\frac{2n-2i-2}{n^2}\right)\\ &= \frac{2\cdot\tau\cdot(5n^2-(8i+8)n+4(1+i)^2)}{n^2(3n-2i-2)}\end{aligned}$$

and at least

$$\begin{aligned}&2\cdot\tau\cdot\left(\frac{n}{3n-2i-2}\cdot\frac{1}{n}+\frac{2n-2i-2}{3n-2i-2}\cdot\frac{2n-2(i+o(n))-2}{n^2}\right)\\ &= \frac{2\cdot\tau\cdot(5n^2-(8i+8)n+4(1+i)^2)}{n^2(3n-2i-2)}-o(1).\end{aligned}$$

We further note that for $\mathrm{LO}(x) = n-1$, the decision step takes an expected $2n$ iterations and will conclude the optimisation process.

Summing the expected progress and expected number of fitness evaluations performed in both stages yields the Lemma statement. $\square$

As the number of iterations spent in the decision step is not deterministic and not identically distributed over the optimisation process, we cannot directly apply the Variable Drift Theorem (Theorem 2.18) to bound the expected runtime for the Generalised Greedy hyper-heuristic. However, we can upper bound the expected runtime through an application of Wald's Inequality (Theorem 5.4). By splitting the analysis into w chunks, we can upper bound the expected times for a Decision and Exploitation Stage within each chunk. We will then use the Variable Drift Theorem to bound the expected number of Decision and Exploitation Stages required to optimise each segment and combine the two results to upper bound the total expected runtime.

The following general theorem bounds the expected runtime of the Generalised Greedy hyper-heuristic for any value of $\tau = o(n^2)$. The best achievable expected runtime (approximately $0.47797n^2 + o(n^2)$) is achieved for $\tau = \omega(n) \cap o(n^2)$ as highlighted in Corollary 5.6, and depicted in Figure 5.1. Theorem 5.5 will prove that the resulting bound is tight. Corollary 5.9 shows that, by setting $\tau = 0$, Theorem 5.8 provides a tight upper bound on the expected runtime of the simple Greedy mechanism, up to the leading constant.

Theorem 5.8 *The expected runtime of the Generalised Greedy hyper-heuristic on* LEADINGONES *with* $\tau = o(n^2)$ *is at most*

$$\sum_{k=1}^{w}\left(\left(\frac{2w}{3w-2k}+\frac{\tau}{n}\right)n\int_{\frac{(k-1)n}{w}}^{\frac{kn}{w}}\frac{1}{\frac{2\tau\cdot(5n^2-(8i+8)n+4(1+i)^2)}{n^2(3n-2i-2)}+2}\,di\right)+o(n^2)$$

for any $w = o\left(\min\left\{n, \frac{n^2}{\tau}\right\}\right)$.

Proof: Let $\mathrm{LO}(x) = i$ denote the current state. We divide the analysis into w chunks of length $\frac{n}{w}$ (for $w = o\left(\min\left\{n, \frac{n^2}{\tau}\right\}\right)$). During chunk k, the LO value of the current solution is at least $\frac{(k-1)n}{w}$ and less than $\frac{k\cdot n}{w}$. We can bound the expected time for a Decision and Exploitation stage (using the result from Lemma 5.7) in the k^{th} chunk, where $\frac{(k-1)n}{w} \le i \le \frac{kn}{w} - 1$, by

$$\frac{2n^2}{3n-2i-\left(4-\frac{2(i+1)}{n}\right)}+\tau \le \left(\frac{2w}{3w-2k}+\frac{\tau}{n}\right)\cdot n.$$

We can make this claim as $w = o(n)$.

Within each chunk, we will use the Variable Drift Theorem to bound the number of expected Decision and Exploitation Stages by looking at the expected LO progress that happens in a 'D+E' Stage, i.e., in expectation at most $\left(\frac{2w}{3w-2k} + \frac{\tau}{n}\right) \cdot n$ steps. We note that there may be some minor overlap between chunks (i.e., if the LO value increases into the next chunk) although, at most, this will result in the addition of $\tau \cdot w = o(n^2)$ iterations throughout the search.

We have seen in Lemma 5.7 that

$$\mathrm{E}(Y^{(t)} - Y^{(t+1)} \mid Y^{(t)}) = \frac{2 \cdot \tau \cdot (5n^2 - (8i+8)n + 4(1+i)^2)}{n^2(3n-2i-2)} + 2 - o(1) = h(n-i) = h(x).$$

We consider $Y^{(t)} = 0$ (pessimistically) to be beyond the end of the chunk. Let $x_{\min} = 1$ represent the point $\frac{kn}{w} - 1$, and we note that $\frac{x_{\min}}{h(x_{\min})} = O(\frac{n}{\tau})$.

Hence, by Wald's Inequality and the Variable Drift Theorem, the expected runtime while the Generalised Greedy hyper-heuristic is in the k^{th} chunk is

$$\begin{aligned}\mathrm{E}(\mathrm{T}_k) \le & \left(\frac{2w}{3w-2k} + \frac{\tau}{n}\right) \cdot n \cdot \\ & \left(\int_{\frac{(k-1)n}{w}}^{\frac{kn}{w}-1} \left(\frac{1}{\frac{2\tau \cdot (5n^2-(8i+8)n+4(1+i)^2)}{n^2(3n-2i-2)} + 2} + o(1)\right) di + \frac{x_{\min}}{h(x_{\min})}\right) \\ \le & \left(\frac{2w}{3w-2k} + \frac{\tau}{n}\right) n \int_{\frac{(k-1)n}{w}}^{\frac{kn}{w}} \frac{1}{\frac{2\tau \cdot (5n^2-(8i+8)n+4(1+i)^2)}{n^2(3n-2i-2)} + 2} di + O(\max\{\tau, n\}).\end{aligned}$$

The total expected runtime by Wald's Inequality will be at most equal to the sum of the expected runtimes of each chunk, plus some lower-order terms amounting to at most $O(\max\{\tau,n\}) \cdot o\left(\min\left\{n, \frac{n^2}{\tau}\right)\right\} = o(n^2)$. Thus,

$$\mathrm{E}(\mathrm{T}) \le \left(\sum_{k=1}^{w} \mathrm{E}(\mathrm{T}_k)\right) + o(n^2).$$

□

The upper bound delivered by the theorem holds for any subquadratic value of τ. This indicates that the Generalised Greedy hyper-heuristic is not very sensitive to the parameter τ for LeadingOnes (as seen in Corollary 5.6). As a result, the parameter should be easy to set in practice, as confirmed by experiments in Section 5.4.

By building upon this result, we now provide a tight upper bound on the expected runtime of the simple Greedy mechanism (when $\tau = 0$, the two mechanisms act equivalently). We know this bound is tight by Corollary 5.3.

Corollary 5.9 *The expected runtime of the Generalised Greedy hyper-heuristic on* LEADINGONES*, with $\tau = 0$, is at most $\frac{\ln(3)}{2}n^2 + o(n^2)$.*

Proof: We set $w = \log n$. Inserting the relevant parameters into the result from Theorem 5.8 will give:

$$\begin{aligned}
\mathrm{E(T)} &\le \sum_{k=1}^{w}\left(\left(\frac{2w}{3w-2k}+\frac{\tau}{n}\right)\cdot n\cdot\int_{\frac{(k-1)n}{w}}^{\frac{kn}{w}}\frac{1}{\frac{2\tau\cdot(5n^2-(8i+8)n+4(1+i)^2)}{n^2(3n-2i-2)}+2}\,di\right)+o(n^2)\\
&\le \sum_{k=1}^{w}\left(\left(\frac{2w}{3w-2k}\right)\cdot n\cdot\int_{\frac{(k-1)n}{w}}^{\frac{kn}{w}}\frac{1}{2}\,di\right)+o(n^2)\\
&= n^2\cdot\sum_{k=1}^{w}\left(\frac{1}{3w-2k}\right)+o(n^2)=\frac{\ln(3)}{2}n^2+o(n^2).
\end{aligned}$$

□

We can now prove the main result of this subsection (Corollary 5.6) which states that the Generalised Greedy with a suitably selected learning period has expected runtime $\approx 0.47797n^2$ on LEADINGONES.

Proof: [Of Corollary 5.6]

We constrain $w = o(n)$. For $\tau = \omega(n) \cap o(n^2)$, we have:

$$\frac{2w}{3w-2k}+\frac{\tau}{n}=\frac{\tau}{n}+o\left(\frac{\tau}{n}\right).$$

$$\frac{2\tau\cdot(5n^2-(8i+8)n+4(1+i)^2)}{n^2(3n-2i-2)}+2=\frac{2\tau(5n^2-8in+4i^2)}{n^2(3n-2i-2)}+o\left(\frac{\tau}{n}\right).$$

Hence, we can simplify the expected runtime of the hyper-heuristic:

$$\begin{aligned}
\mathrm{E(T)} &\le \sum_{k=1}^{w}\left(\left(\frac{2w}{3w-2k}+\frac{\tau}{n}\right)\cdot n\cdot\int_{\frac{(k-1)n}{w}}^{\frac{kn}{w}}\frac{1}{\frac{2\tau\cdot(5n^2-(8i+8)n+4(1+i)^2)}{n^2(3n-2i-2)}+2}\,di\right)+o(n^2)\\
&\le \sum_{k=1}^{w}\left(\tau\cdot\int_{\frac{(k-1)n}{w}}^{\frac{kn}{w}}\frac{n^2(3n-2i-2)}{2\tau(5n^2-8in+4i^2)}\,di\right)+o(n^2)
\end{aligned}$$

$$= \frac{n^2}{2} \cdot \sum_{k=1}^{w} \left(\int_{\frac{(k-1)n}{w}}^{\frac{kn}{w}} \frac{3n-2i-2}{5n^2-8in+4i^2} di \right) + o(n^2) \leq \frac{n^2}{2} \left(\frac{1}{4} \ln(5) + \frac{1}{2} \arctan(2) \right) + o(n^2)$$
$$= \left(\frac{\ln(5)}{8} + \frac{\arctan(2)}{4} \right) n^2 + o(n^2).$$

□

The lower bound of Theorem 5.5 matches the upper bound of Corollary 5.6. Since this 'ideal' Greedy algorithm is faster than the Generalised Greedy hyper-heuristic, the upper bound provided in Corollary 5.6 is tight. Although the Generalised Greedy hyper-heuristic cannot achieve optimal expected runtime, we see that it still outperforms the single operator algorithm RLS.

5.3.2 Generalised Random Gradient Hyper-heuristic

We now present the analysis of the expected runtime of the Generalised Random Gradient hyper-heuristic on LEADINGONES. Recall that the Generalised Random Gradient hyper-heuristic begins by selecting a low-level heuristic uniformly at random from the set of heuristics. The chosen heuristic is applied for τ iterations and, if a fitness improvement occurs within these τ iterations, another period of τ iterations immediately begins. When the heuristic has τ iterations without a successful mutation, another random choice of heuristic occurs and the process restarts. This continues until some termination condition occurs.

The following results regarding the expected runtime of the Generalised Random Gradient hyper-heuristic can be proved using similar techniques and calculations to those used in the analysis of the Generalised Greedy hyper-heuristic, but applying the Additive Drift Theorem instead of the Variable Drift Theorem.

The main result of this subsection is that the Generalised Random Gradient hyper-heuristic is able to match, up to lower order terms, the best-possible performance of any algorithm using the 1BITFLIP and 2BITFLIP operators on the LEADINGONES benchmark function, as presented in Theorem 5.1. We present the main result in Corollary 5.10 now, with the proof at the end of the subsection.

Corollary 5.10 *[Of Theorem 5.11] The expected runtime of the Generalised Random Gradient hyper-heuristic on* LEADINGONES *with τ that satisfies both $\tau = \omega(n)$ and $\tau \leq \left(\frac{1}{2} - \varepsilon\right) n \ln(n)$, for some constant $\varepsilon > 0$, is at most $\frac{1+\ln(2)}{4} n^2 + o(n^2) \approx 0.42329n^2$.*

We first present some necessary prerequisite results. The following general theorem bounds the expected runtime of the Generalised Random Gradient hyper-heuristic for any value of τ smaller than $\frac{1}{2}n\ln n$. The theorem allows us to identify values of τ for which the expected runtime of the hyper-heuristic for the function is the optimal expected runtime that may be achieved by using two operators. This result is highlighted in Corollary 5.10 for the appropriate values of the learning period τ and depicted in Figure 5.1. Our proof technique involves partitioning the search space into w chunks, each representing an equal range of fitness.

Theorem 5.11 *The expected runtime of the Generalised Random Gradient hyper-heuristic on* LEADINGONES *with $\tau \leq \left(\frac{1}{2} - \varepsilon\right) n \ln(n)$, for some constant $\varepsilon > 0$, is at most*

$$\frac{n^2}{2}\left(\sum_{j=1}^{w} \frac{\frac{2\tau}{n} + e^{\frac{\tau}{n}} M_1(1) + e^{2\frac{\tau}{n}(1-(j-1)/w)} M_2(j,w)}{(e^{\frac{\tau}{n}} + e^{2\frac{\tau}{n}(1-j/w)} - 2)w}\right) + o(n^2)$$

where

$$M_2(j,w) := \begin{cases} \frac{\tau}{n} & \text{if } j = w \text{ or } \frac{0.5}{1-j/w} > \frac{\tau}{n} \\ \frac{0.5}{1-j/w} & \text{otherwise} \end{cases}$$

and $M_1(x) := \min\{\frac{\tau}{n}, x\}$, for any integer $w = \Omega(1) \cap o\left(\frac{n}{\exp(2\tau/n)}\right)$.

Theorem 5.11 is a general theorem for any $\tau \leq \left(\frac{1}{2} - \varepsilon\right) n \ln(n)$, for some constant $\varepsilon > 0$. Figure 5.1 presents the theoretical upper bounds from Theorem 5.8 and Theorem 5.11 for a range of linear τ values for the Generalised Greedy and Generalised Random Gradient hyper-heuristics. The Generalised Greedy hyper-heuristic quickly tends towards the leading constant provided by Corollary 5.6. For $\tau = 5n$ already, the Generalised Random Gradient hyper-heuristic outperforms RLS, giving an expected runtime of $0.46493n^2$. For $\tau = 100n$, the performance improves to $0.42368n^2$, matching the best possible expected performance (i.e., 2_{opt}) from Theorem 5.1 up to 3 decimal places. We have seen that Generalised Random Gradient is able to exactly match this best possible performance for $\tau = \omega(n)$ in Corollary 5.10 (up to lower order terms).

Proof: [Of Theorem 5.11]

For the purpose of this proof, we partition the optimisation process into w chunks based on the value of the LEADINGONES fitness function: during chunk j, the LO value of the current solution is at least $\frac{(j-1)\cdot n}{w}$ and less than $\frac{j\cdot n}{w}$. After all the w chunks have been completed, the global optimum, with a LO value of n, will have been found.

We provide an upper bound on the expectation of the runtime T of the Generalised Random Gradient hyper-heuristic on LO by the sum of the expected values of T_j, the runtimes on each chunk of the optimisation process, plus the times taken to begin a chunk.

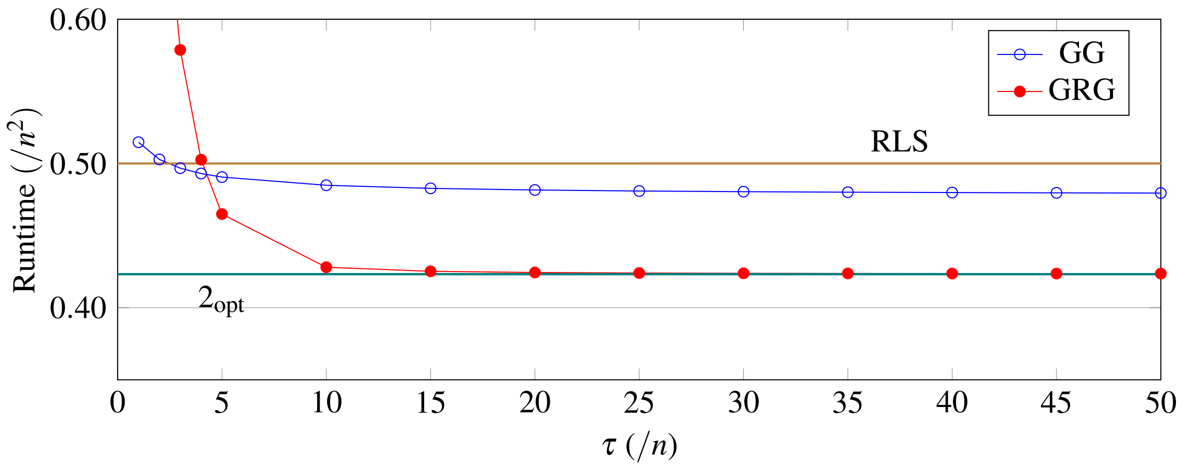

Fig. 5.1 The leading constants in the expected theoretical upper bounds on the expected number of fitness function evaluations required by the Generalised Greedy and Generalised Random Gradient hyper-heuristics to find the LEADINGONES optimum ($w = 100{,}000$).

As our analysis of $\mathrm{E}(\mathrm{T}_j)$ requires the chunk to start with a random choice of mutation operator, we bound $\mathrm{E}(\mathrm{T}) \leq \sum_{j=1}^{w} \left(\mathrm{E}(\mathrm{T}_j) + \mathrm{E}(S_{j+1})\right)$ where S_{j+1} is a random variable denoting the expected number of iterations between the first solution in chunk $j+1$ being constructed and the following random operator choice. We will later show that with proper parameter choices, the contribution of the S_{j+1} terms can be bounded by $o(n^2)$ and therefore does not affect the leading constant in the overall bound.

Let us now consider T_j, the number of iterations the hyper-heuristic spends in chunk j. Recall that a mutation operator is selected uniformly at random and is allowed to run until it fails to produce an improvement for τ sequential iterations. Let N_j be a random variable denoting the number of random operator choices the hyper-heuristic performs during chunk j and $X_{j,1}, \ldots, X_{j,N_j}$ be the number of iterations the hyper-heuristic runs each chosen operator for. In which case, $\mathrm{T}_j = \sum_{k=1}^{N_j} X_{j,k}$ and by applying Wald's Inequality (Theorem 5.4), using $\mathrm{E}(X_j)$ to denote an upper bound on all $\mathrm{E}(X_{j,k})$ in chunk j:

$$\mathrm{E}(\mathrm{T}_j) = \textstyle\sum_{k=1}^{N_j} \mathrm{E}(X_{j,k}) \leq \mathrm{E}(N_j)\,\mathrm{E}(X_j). \tag{5.3}$$

To bound $\mathrm{E}(N_j)$, the expected number of times the random operator selection is performed during chunk j, we lower bound the expected number of improvements found following the operator selection and apply the Additive Drift Theorem (Theorem 2.11) to find the expected number of random operator selections occurring before a sufficient number of improvements have been found to enter the next chunk.

Let F_1 and F_2 denote the events that the 1BITFLIP and 2BITFLIP operators fail to find an improvement during τ iterations. For the 1BITFLIP operator (with probability of fitness improvement in one iteration $\frac{1}{n}$), this event occurs with probability $\Pr(F_1) = \left(1 - \frac{1}{n}\right)^{\tau}$ throughout the process, which is within $[e^{-\frac{\tau}{n}} - \frac{1}{n}, e^{-\frac{\tau}{n}}]$ (since $\left(1-\frac{1}{n}\right)^n \leq \frac{1}{e} \leq \left(1-\frac{1}{n}\right)^{n-1}$). For the 2BITFLIP operator (with probability of fitness improvement in one iteration $\frac{2n-2i-2}{n^2}$), recall that during chunk j, the ancestor individual has at most $i = \frac{jn}{w} - 1$ and at least $i = \frac{(j-1)n}{w}$, one bits. Thus:

$$\begin{aligned}
\Pr(F_2) &\leq (1 - 2 \cdot (1/n) \cdot (n - (jn/w - 1) - 1)/n)^{\tau} \leq e^{-2\frac{\tau}{n}(1-j/w)}, \\
\Pr(F_2) &\geq (1 - 2 \cdot (1/n) \cdot (n - (j-1)n/w - 1)/n)^{\tau} \\
&> \left(1 - \frac{2(1-(j-1)/w)}{n}\right)^{\tau} \geq e^{-\frac{2\tau}{n}(1-(j-1)/w)} - 1/n.
\end{aligned}$$

We note that a geometric distribution with parameter $p = \Pr(F_1)$ can be used to model the number of improvements that the 1BITFLIP operator finds prior to failing to find an improvement for τ iterations; the expectation of this distribution is $\frac{1-p}{p} = \frac{1}{p} - 1$. Combined over both operators, the expected number of improvements (D_j) produced following a single random operator selection during chunk j, is:

$$\mathrm{E}(D_j) = \frac{1}{2}\left(\frac{1}{\Pr(F_1)} - 1\right) + \frac{1}{2}\left(\frac{1}{\Pr(F_2)} - 1\right) \geq e^{\frac{\tau}{n}}/2 + e^{\frac{2\tau}{n}(1-j/w)}/2 - 1,$$

by inserting the upper bounds on $\Pr(F_1)$ and $\Pr(F_2)$. We use this expectation as the drift on the progress of the randomly chosen operator in the Additive Drift Theorem to upper bound $\mathrm{E}(N_j)$. Recall that each chunk consists of advancing through at most $\frac{n}{w}$ fitness values; as bits beyond the leading ones prefix and the first zero bit remain uniformly distributed, at most $\frac{n}{2w}$ improvements by mutation are required in expectation. If each step of a random process in expectation contributes $\mathrm{E}(D_j)$ improvements by mutation, then the expected number of steps required to complete chunk j is at most:

$$\mathrm{E}(N_j) \leq \frac{n/(2w)}{\mathrm{E}(D_j)} \leq \frac{n}{(e^{\frac{\tau}{n}} + e^{\frac{2\tau}{n}(1-j/w)} - 2)w} \tag{5.4}$$

by the Additive Drift Theorem.

To bound $\mathrm{E}(X_j)$, the expected number of iterations before a selected mutation operator fails to produce an improvement for τ iterations, we apply Wald's Inequality: let S be the number of improvements found by the operator before it fails and $W_1, \ldots, W_S$ be the number of iterations it took to find each of those improvements; then, once selected, the 1BITFLIP operator runs for:

$$\mathrm{E}(X_j \mid \text{1-bit}) = \tau + \sum_{k=1}^{S} \mathrm{E}(W_k) = \tau + \mathrm{E}(S) \cdot \mathrm{E}(W_1 \mid S \geq 1, \text{1-bit})$$

where τ accounts for the iterations immediately before failure, and the sum for the iterations preceding each constructed improvement.

Recall that $\mathrm{E}(S) = 1/\Pr(F_1) - 1$ by the properties of the geometric distribution, and observe that $\mathrm{E}(W_1 \mid S \geq 1) = \mathrm{E}(W_1 \mid W_1 \leq \tau) \leq \min\{\tau, \mathrm{E}(W_1)\}$. Using a waiting time argument, gives $\mathrm{E}(W_1) = 1/\Pr(F_1)$, where F_1 is the event that the 1BITFLIP operator fails to find an improvement during τ iterations, and, we get (with a similar argument for the two-bit mutation operator, with $\mathrm{E}(W_2) \leq \frac{n^2}{2(n-(jn/w-1)-1)} = \frac{n}{2-2j/w}$):

$$\begin{aligned}
\mathrm{E}(X_j \mid \text{1-bit}) &\leq \tau + (1/\Pr(F_1) - 1)\min\{\tau, n\}, \\
\mathrm{E}(X_j \mid \text{2-bit}) &\leq \tau + (1/\Pr(F_2) - 1)\min\left\{\tau, \frac{n}{2-2j/w}\right\}.
\end{aligned}$$

Combining these conditional expectations with lower bounds on $\Pr(F_1)$ and $\Pr(F_2)$ yields

$$\begin{aligned}
\mathrm{E}(X_j) &\leq \frac{1}{2} \cdot \mathrm{E}(X_j \mid \text{1-bit}) + \frac{1}{2} \cdot \mathrm{E}(X_j \mid \text{2-bit}) \\
&\leq \tau + \frac{\min\{\tau, n\}}{2(e^{-\frac{\tau}{n}} - 1/n)} + \frac{\min\{\tau, n/(2-2j/w)\}}{2(e^{-\frac{2\tau}{n}(1-(j-1)/w)} - 1/n)},
\end{aligned} \tag{5.5}$$

which can be simplified to:

$$\begin{aligned}
\mathrm{E}(X_j) &\leq \frac{n}{2}\left(\frac{2\tau}{n} + \frac{\min\{\frac{\tau}{n}, 1\}}{e^{-\frac{\tau}{n}} - 1/n} + \frac{\min\{\frac{\tau}{n}, 1/(2-2j/w)\}}{e^{-\frac{2\tau}{n}(1-(j-1)/w)} - 1/n}\right) \\
&\leq \frac{n}{2}\left(\frac{2\tau}{n} + e^{\frac{\tau}{n}}\min\left\{\frac{\tau}{n}, 1\right\} + e^{\frac{2\tau}{n}(1-(j-1)/w)}\min\left\{\frac{\tau}{n}, \frac{1}{2-2j/w}\right\}\right) + O(1)
\end{aligned}$$

using $a/(b-c/n) = a/b + ac/(b^2n - bc) = a/b + O(1/n)$, where a, b, and c are constants with respect to n, to limit the contributions of the $-1/n$ terms in denominators to lower-order terms.

If $j = w$, then $\min\left\{\frac{\tau}{n}, \frac{1}{2-2j/w}\right\}$ is undefined. However, we know that the chosen operator would only be applied for a maximum of τ steps. We thus refer to the functions $M_1(x) =$

$\min\left\{\frac{\tau}{n}, x\right\}$ and $M_2(j,w)$ from now on, where

$$M_2(j,w) := \begin{cases} \frac{\tau}{n} & \text{if } j = w \text{ or } \frac{0.5}{1-j/w} > \frac{\tau}{n} \\ \frac{0.5}{1-j/w} & \text{otherwise.} \end{cases}$$

Finally, we can bound the overall expected runtime of the hyper-heuristic. Recall that S_j denotes the number of iterations the hyper-heuristic spends in chunk j while using the random operator chosen during chunk $j-1$, and as $\mathrm{E}(S_j) \le \max\left(\mathrm{E}(X_j \mid \text{1-bit}), \mathrm{E}(X_j \mid \text{2-bit})\right) < 2 \cdot \mathrm{E}(X_j) = O(n \cdot \exp(2\tau/n))$, and since $w = o\left(\frac{n}{\exp(2\tau/n)}\right)$, $\sum_{j=1}^{w} \mathrm{E}(S_{j+1}) = o(n^2)$. Substituting the bounds (5.4) and (5.5) into (5.3) yields the theorem statement:

$$\begin{aligned} \mathrm{E}(\mathrm{T}) &\le \sum_{j=1}^{w} \left(\mathrm{E}(\mathrm{T}_j) + \mathrm{E}(S_{j+1})\right) \le \left(\sum_{j=1}^{w} \mathrm{E}(N_j)\,\mathrm{E}(X_j)\right) + o(n^2) \\ &\le o(n^2) + \frac{n^2}{2} \times \sum_{j=1}^{w} \frac{\frac{2\tau}{n} + e^{\frac{\tau}{n}} M_1(1) + e^{\frac{2\tau}{n}(1-(j-1)/w)} M_2(j,w)}{(e^{\frac{\tau}{n}} + e^{\frac{2\tau}{n}(1-j/w)} - 2) \cdot w}. \end{aligned}$$

$\square$

We can now prove the main result of this subsection (Corollary 5.10), which states that Generalised Random Gradient achieves optimal runtime for LEADINGONES up to lower order terms (i.e., no algorithm using the same components can be faster).

Proof: [Of Corollary 5.10]

We set $w = \log^2 n$.

Consider first the terms $M_1(1)$ and $M_2(j,w)$. Since $\tau = \omega(n)$, we have $M_1(1) = 1$. For $M_2(j,w)$, it is important to note that for the first half of the search, $M_2(j,w) = \frac{0.5}{1-j/w}$, and in the second half of the search, $M_2(j,w) = O(w) = n^{o(1)}$ (excluding the case when $j = w$ in which $M_2(j,w) = \frac{\tau}{n} = n^{o(1)}$). Note that $\frac{\tau}{n} = \omega(1)$ and $e^{2\tau/n \cdot 1/w} = o(n^{1/w}) = 1 + o(1)$. Hence, we can simplify the sum from Theorem 5.11:

$$\begin{aligned} \mathrm{E}(\mathrm{T}) &\le \frac{n^2}{2} \cdot \left(\sum_{j=1}^{w} \frac{\frac{2\tau}{n} + e^{\frac{\tau}{n}} M_1(1) + e^{\frac{2\tau}{n}(1-(j-1)/w)} M_2(j,w)}{(e^{\frac{\tau}{n}} + e^{2\frac{\tau}{n}(1-j/w)} - 2)w}\right) + o(n^2) \\ &= \frac{n^2}{2} \cdot \left(\sum_{j=1}^{w} \frac{\frac{2\tau}{n} + e^{\frac{\tau}{n}} + e^{2\tau/n \cdot 1/w} \cdot e^{\frac{2\tau}{n}(1-j/w)} M_2(j,w)}{(e^{\frac{\tau}{n}} + e^{2\frac{\tau}{n}(1-j/w)} - 2)w}\right) + o(n^2) \\ &= \frac{n^2}{2} \cdot \left(\sum_{j=1}^{w} \frac{\frac{2\tau}{n} + e^{\frac{\tau}{n}} + e^{\frac{2\tau}{n}(1-j/w)} M_2(j,w)}{(e^{\frac{\tau}{n}} + e^{2\frac{\tau}{n}(1-j/w)} - 2)w}\right) + o(n^2) \end{aligned}$$

$$
\begin{aligned}
&= \frac{n^2}{2}\cdot\left(\sum_{j=1}^{w}\frac{1}{w}\left(\frac{\frac{2\tau}{n}+2}{e^{\frac{\tau}{n}}+e^{2\frac{\tau}{n}(1-j/w)}-2}+\frac{e^{\frac{\tau}{n}}+e^{\frac{2\tau}{n}(1-j/w)}M_2(j,w)-2}{e^{\frac{\tau}{n}}+e^{2\frac{\tau}{n}(1-j/w)}-2}\right)\right)\\
&\quad+o(n^2)\\
&= \frac{n^2}{2}\cdot\left(\sum_{j=1}^{w}\frac{1}{w}\left(\frac{e^{\frac{\tau}{n}}+e^{\frac{2\tau}{n}(1-j/w)}M_2(j,w)-2}{e^{\frac{\tau}{n}}+e^{2\frac{\tau}{n}(1-j/w)}-2}\right)\right)+o(n^2).
\end{aligned}
$$

Since $\tau/n=\omega(1)$, the terms with the larger exponents will asymptotically dominate each summand. When $j>\frac{w}{2}$, we note that $\exp(\tau/n)>\exp(\frac{2\tau}{n}(1-\frac{j}{w}))$ and the first term will asymptotically dominate the second term in the numerator (and similarly in the denominator); vice versa for $j<\frac{w}{2}$. At $j=\frac{w}{2}$ the two values are equal and we consider this separately. Hence, we can split the sum into two sections:

$$
\begin{aligned}
\mathrm{E(T)} &\le \frac{n^2}{2}\cdot\left(\sum_{j=1}^{w}\frac{1}{w}\left(\frac{e^{\frac{\tau}{n}}+e^{\frac{2\tau}{n}(1-j/w)}M_2(j,w)-2}{e^{\frac{\tau}{n}}+e^{2\frac{\tau}{n}(1-j/w)}-2}\right)\right)+o(n^2)\\
&= \frac{n^2}{2}\cdot\left(\sum_{j=1}^{w/2-1}\frac{1}{w}\left(\frac{e^{\frac{2\tau}{n}(1-j/w)}\cdot\frac{0.5}{1-j/w}}{e^{\frac{2\tau}{n}(1-j/w)}}\right)\right)+\frac{n^2}{2w}+\frac{n^2}{2}\cdot\left(\sum_{j=w/2+1}^{w}\frac{1}{w}\left(\frac{e^{\frac{\tau}{n}}}{e^{\frac{\tau}{n}}}\right)\right)\\
&\quad+o(n^2)\\
&= \frac{n^2}{4}\cdot\left(\sum_{j=1}^{w/2}\frac{1}{w-j}\right)+\frac{n^2}{4}+o(n^2)=\left(\frac{1+\ln(2)}{4}\right)n^2+o(n^2).
\end{aligned}
$$

Note that at $j=w$, the respective 'dominating terms' are $\exp\left(\frac{\tau}{n}\right)$ and $\exp\left(\frac{2\tau}{n}\left(1-\frac{w}{w}\right)\right)=1$, and $M(j,w)=\frac{\tau}{n}<\frac{1}{2}\ln(n)$. Hence, $\exp\left(\frac{\tau}{n}\right)$ will dominate. □

5.4 Experimental Supplements

In the previous sections, we proved that the generalised hyper-heuristics perform efficiently on the LeadingOnes benchmark function for sufficiently large problem sizes n. In this section, we present some experimental results to shed light on their performance for problem sizes up to $n=10^8$ (see Subsection 2.5.3 in Chapter 2 for details on how we are able to give results up to such large problem sizes). All parameter combinations have been simulated 10,000 times.

Figure 5.2 shows the effects of increasing the problem size n for a variety of fixed τ values, for the Generalised Random Gradient hyper-heuristic. We can see that an increased problem size leads to faster runtimes. The performance difference between the τ values

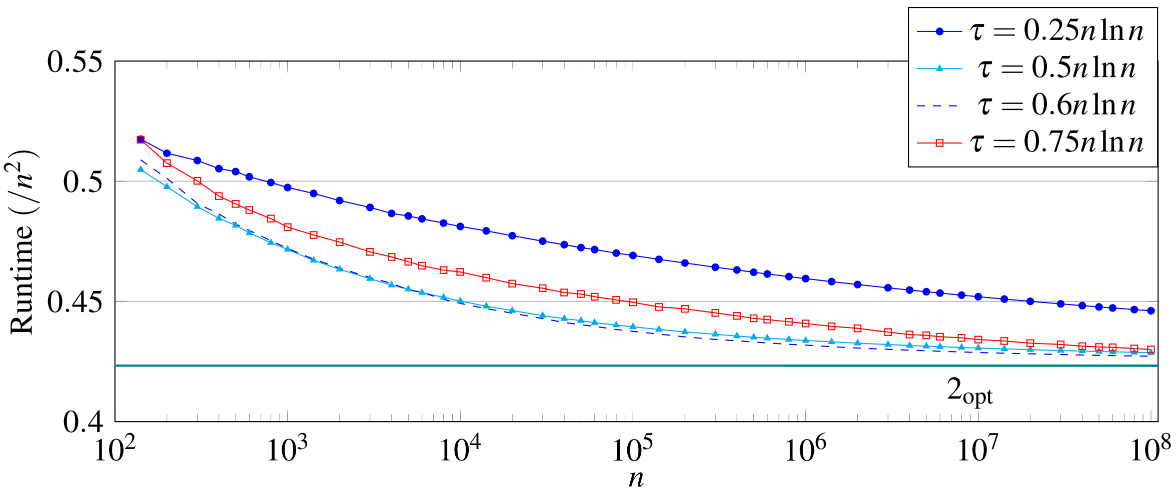

Fig. 5.2 Average number of fitness function evaluations required for the Generalised Random Gradient hyper-heuristic with $k = 2$ operators to find the LEADINGONES optimum as the problem size n increases.

decreases with increased n, implying that further increasing n would lead to similar, optimal performance for a large range of values, as implied by Corollary 5.10. For $n = 10^8$, the runtime for $\tau = 0.6n\ln n$ is $\approx 0.427n^2$, only slightly deviated from the optimal value of $\approx 0.423n^2$. These results suggest that for a problem size larger than $\approx 10^3$, any sensible choice of the learning period τ with the Generalised Random Gradient hyper-heuristic will outperform RLS.

5.5 Chapter Summary and Discussion

In Chapter 4 we saw that four ‘simple’ hyper-heuristics were not able to ‘learn’ to prefer certain heuristics at different places of the search space. However, two generalised hyper-heuristics (Generalised Greedy and Generalised Random Gradient) were able to learn.

In this chapter, we have rigorously proved that the same four simple mechanisms have equivalent performance on the LEADINGONES benchmark function, up to lower order terms, as suggested by the experimental analysis of Alanazi and Lehre (2014). That is, they essentially choose heuristics at random and do not learn which are preferable. Furthermore, on the same function we have proved that while the Generalised Greedy hyper-heuristic is able to match the performance of an ideal Greedy algorithm which performs a Decision Stage for free in each iteration, the Generalised Random Gradient hyper-heuristic is able to match the best possible performance possible with the low-level heuristics available.

Concerning the Generalised Random Gradient hyper-heuristic, the learning period τ must be large enough to have at least a constant expected number of successes within τ steps, if the hyper-heuristic has to learn about the operator performance. Naturally, setting large values of τ may lead to large expected runtimes, since switching operators requires $\Omega(\tau)$ steps. Similar considerations may also be made for the Generalised Greedy hyper-heuristic, although our results indicate it is even more robust to the choice of τ.

In the next chapter, we introduce a new hyper-heuristic based on the theoretical insights of this chapter. The new hyper-heuristic, Generalised Greedy Gradient, will combine the informed Decision Stage of the Generalised Greedy hyper-heuristic with the smart Exploitation Stage of the Generalised Random Gradient hyper-heuristic, such that it is able to make a smart choice of heuristic and then exploit this choice so long as it is successful.

Chapter 5 is based on the following publications:

1. Lissovoi, A., Oliveto, P. S., and Warwicker, J. A. (2017). On the Runtime Analysis of Selection Hyper-heuristics for Pseudo-Boolean Optimisation. *Proceedings of the Annual Conference on Genetic and Evolutionary Computation (GECCO '17)*, pages 849-856. ACM.

2. Lissovoi, A., Oliveto, P. S., and Warwicker, J. A. (2019). Simple Hyper-heuristics Control the Neighbourhood Size of Randomised Local Search Optimally for LeadingOnes. *Evolutionary Computation* (to appear).

Chapter 6

A New Hyper-heuristic Based on Theoretical Insights

In the previous two chapters, we have seen that the Generalised Greedy and Generalised Random Gradient hyper-heuristics are able to outperform their low-level heuristics, even when the heuristics are efficient for the problem at hand. In particular, the two learning mechanisms lead to 'smart' choices of heuristics that are exploited effectively, for the LEADINGONES problem.

In this chapter, we present a new hyper-heuristic that combines the Greedy and Random Gradient learning mechanisms into a new hyper-heuristic. We will show that this new hyper-heuristic, named *Generalised Greedy Gradient*, performs efficiently on the GAPPATH and the GGPT pseudo-Boolean functions, implying that it is able to make smart choices of heuristics. Furthermore, we will prove for the LEADINGONES benchmark function that Generalised Greedy Gradient outperforms its low-level heuristics and matches the best-possible performance achievable with the available low-level heuristics, up to lower order terms. In particular, for each learning period τ we provide an upper bound on the expected runtime of Generalised Greedy Gradient that is lower than the bound provided for Generalised Random Gradient in Chapter 5. An experimental analysis will confirm that Generalised Greedy Gradient outperforms Generalised Random Gradient on the same function in practice, for the best choices of τ.

6.1 A New Hyper-heuristic: Generalised Greedy Gradient

In this section, we introduce a new hyper-heuristic, called Generalised Greedy Gradient, which combines the Decision Stage idea of the Generalised Greedy hyper-heuristic (i.e.,

Algorithm 12 Generalised Greedy Gradient Hyper-heuristic

```
1: Choose $x \in S$ uniformly at random
2: while stopping conditions not satisfied do
3:     $\forall h \in H$, apply $h(x)$ repeatedly until some $h'$ has $f(h'(x)) > f(x)$.
4:     $c_t \leftarrow 0$
5:     while $c_t < \tau$ do
6:         $c_t \leftarrow c_t + 1; x' \leftarrow h'(x)$
7:         if $f(x') > f(x)$ then
8:             $c_t \leftarrow 0, x \leftarrow x'$
9:         end if
10:    end while
11: end while
```

where all operators are run in parallel and the first to make a fitness improvement is chosen) and applies this operator to the gradient-based Exploitation Stage of the Generalised Random Gradient hyper-heuristic (i.e., the chosen operator is run for a period of τ steps and each fitness improvement resets the period of τ). The idea is to combine the learning aspects of the two hyper-heuristics to create a hyper-heuristic that chooses the best operator and exploits this choice. The pseudocode for this hyper-heuristic is shown in Algorithm 12. Note that if no heuristic can find an improving move, the hyper-heuristic will run for infinite time (in realistic applications, some stopping criteria will be implemented).

In the next subsection, we will prove that the Generalised Greedy Gradient hyper-heuristic can solve the GAPPATH and GGPT$_k$ efficiently in a similar way to the Generalised Random Gradient and Generalised Greedy hyper-heuristics introduced in Chapter 4.

6.1.1 GAPPATH and GGPT

In this subsection we show that the hyper-heuristic can learn to use a given operator if it is useful often enough, in a similar way as shown for the other generalised hyper-heuristics in Chapter 4. The results we present in this subsection will refer to several theorems presented in Chapter 4.

Theorem 6.1 *Starting at 0^n, the expected runtime of the Generalised Greedy Gradient hyper-heuristic on* GAPPATH *is* $\frac{1}{3}\left(n^3 + 2n^2 + 2\tau n\right) - \tau$.

Proof: The Generalised Greedy Gradient hyper-heuristic will act equivalently to Generalised Greedy on GAPPATH. That is, it will run the Greedy operator Decision Stage and deterministically choose the unique operator which will give a success. Then, the mechanism will waste τ iterations, since it cannot enact any improvements in this Exploitation Stage with the selected operator. Hence, the expected runtime is equivalent to the expected runtime presented in Theorem 4.2. □

Theorem 6.2 *Starting at* 0^n *for* $\tau = O(n^2)$*, the Generalised Greedy Gradient hyper-heuristic will fail to reach the global optimum of* GGPT *in finite time with probability at least* $1-2^{-\Theta(n)}$*. However, for* $\tau = \Omega(n^3)$*, the Generalised Greedy Gradient hyper-heuristic will find the global optimum of* GGPT_k *in at most* $\frac{2\tau n}{2k+1} + \frac{(k-1)n^3}{2(2k+1)} - \tau + O(n^2)$ *steps with probability at least* $1-2^{-\Theta(n)}$*.*

Proof: If $\tau \in O(n^2)$, a similar argument to the proof of Theorem 4.5 can be made. That is, for $\tau = cn^2$, where c is a constant, the probability of achieving $k-1$ consecutive successful periods is $\left(1-\left(1-2/n^2\right)^{cn^2}\right)^{k-1} = (1-e^{-2c}-o(1))^{k-1} = \Theta(1)$ and so the probability of not achieving $k-1$ consecutive successful periods is at least constant. If another random Decision Stage is performed inside the region of two-bit improvements, a 'trap' is accessible via a 1-bit mutation from the point at which the decision occurs with probability at least $\frac{1}{k-1} = \Theta(1)$ (as traps are placed uniformly at random inside these regions), and is found by the Random Gradient mechanism with probability at least $1-\frac{1}{2}\cdot(1-\frac{1}{n})^{cn^2} \geq \frac{1}{4}$ for $n > 1$. Thus, the probability that no 'trap' point is constructed in a single region of k two-bit improvements is at most $3/4$, which means that over $\frac{n}{2k+1}$ such regions, the probability that a 'trap' point is constructed is at least $1-2^{-\Theta(n)}$.

Initially, a 1BitFlip improvement is necessary before the first 2BitFlip flip. For $\tau = \Omega(n^3)$, similarly to Theorem 4.4, this will take expected time $2n+\tau$ to account for the wasted iterations. The Generalised Greedy Gradient hyper-heuristic will then run the Greedy Decision Stage to choose the 2BitFlip operator in time $2\cdot\frac{n^2}{2} = n^2$. The Exploitation Stage will continue with this operator to find the remaining necessary 2BitFlip improvements and avoid the 'trap' point with probability $1-2^{-\Theta(n)}$, as per the proof of Theorem 4.5. The expected time for the remaining $k-1$ improvements will be $\frac{(k-1)n^2}{2}+\tau$, to account for the wasted iterations at the end. This process must be repeated $\frac{n}{2k+1}$ times to account for each run of 2-bit flips necessary to find the global optimum. Hence, the expected running time will be

$$\frac{n}{2k+1}\left(2n+\tau+n^2+\frac{k-1}{2}n^2+\tau\right)-\tau$$

which gives the result in the theorem statement. □

The Generalised Greedy Gradient hyper-heuristic is able to make a smart choice of heuristic and exploit this choice. We see in the next subsection that Generalised Greedy

Gradient runs in the best possible expected runtime achievable with the available low-level heuristics, up to lower order terms, for the LEADINGONES benchmark function.

6.1.2 LEADINGONES

The following theorems regarding the performance of Generalised Greedy Gradient on LEADINGONES can be proved using similar techniques and calculations as those used in the previous analyses for the Generalised Greedy and Generalised Random Gradient hyper-heuristics in Chapter 5, since Generalised Greedy Gradient combines the learning aspects of both hyper-heuristics.

The main result of this chapter is that the Generalised Greedy Gradient hyper-heuristic is able to match, up to lower order terms, the best possible performance for any algorithm using the 1BITFLIP and 2BITFLIP operators on the LEADINGONES benchmark function. We present the main result in Corollary 6.3 now, with the proof at the end of this section.

Corollary 6.3 *[Of Theorem 6.4] The expected runtime of the Generalised Greedy Gradient hyper-heuristic on* LEADINGONES*, with τ that satisfies both $\tau = \omega(n)$ and $\tau \le (\frac{1}{2} - \varepsilon)n\ln(n)$, for some constant $\varepsilon > 0$, is at most $\frac{1+\ln(2)}{4}n^2 + o(n^2) \approx 0.42329n^2$.*

In order to derive this result, we first present the necessary prerequisite results. The following general theorem bounds the expected runtime of the Generalised Greedy Gradient hyper-heuristic for any value of τ smaller than $\frac{1}{2}n\ln n$. Firstly, we show that the Generalised Greedy Gradient hyper-heuristic can lead to improved performance on LEADINGONES than the Generalised Random Gradient hyper-heuristic. For each value of τ, the upper bound achieved by Generalised Greedy Gradient exceeds the upper bound achieved by Generalised Random Gradient by at most $o(n^2)$ (in fact, we see empirically that Generalised Greedy Gradient is faster in Figure 6.1). Secondly, the theorem allows us to identify values of τ for which the expected runtime of the hyper-heuristic for the function is the optimal expected runtime that may be achieved by using two operators. This result is highlighted in Corollary 6.3 and depicted experimentally in Figure 6.1. Our proof technique partitions the search space into w chunks, each representing an equal range of fitness and is similar in structure to the proof of Theorem 5.11.

Theorem 6.4 *The expected runtime of the Generalised Greedy Gradient hyper-heuristic on* LEADINGONES*, with $\tau \le \left(\frac{1}{2} - \varepsilon\right) n\ln(n)$, for some constant $\varepsilon > 0$, is at most*

$$\mathrm{E(T)} \le o(n^2) + \frac{n^2}{2} \cdot \left[\sum_{j=1}^{w} \frac{3w-2j+2}{3w-2j} \right.$$

$$\left(\frac{2w+\frac{\tau}{n}\cdot(3w-2j)+w\cdot e^{\frac{\tau}{n}}\cdot M_1(1)+(2(w-j+1))\cdot e^{\frac{2\tau}{n}(1-(j-1)/w)}\cdot M_2(j,w)}{w\cdot\left(w\cdot e^{\frac{\tau}{n}}+2(w-j)\cdot e^{\frac{2\tau}{n}(1-j/w)}\right)}\right)\Bigg]$$

for any integer $w = o\left(\frac{n}{\exp(2\tau/n)}\right)$, *where* $M_1(x) = \min\{\frac{\tau}{n}, x\}$ *and*

$$M_2(j,w) := \begin{cases} \frac{\tau}{n} & \textit{if } j = w \textit{ or } \frac{0.5}{1-j/w} > \frac{\tau}{n} \\ \frac{0.5}{1-j/w} & \textit{otherwise.} \end{cases}$$

Proof: We partition the optimisation process into w chunks based on the value of the LEADINGONES fitness function: during chunk j, the LO value of the current solution is at least $\frac{(j-1)n}{w}$ and less than $\frac{j\cdot n}{w}$.

We follow the proof of Theorem 5.11 in which we upper bounded the final expected runtime by the sum of the expected values of T_j, the expected runtime of each chunk of the process, plus the expected times taken to reach the first operator decision within each chunk. Since the analysis of $\mathrm{E}(\mathrm{T}_j)$ requires an operator choice, we bound the expected runtime $\mathrm{E}(\mathrm{T}) \leq \sum_{j=1}^{w}(\mathrm{E}(\mathrm{T}_j)+\mathrm{E}(S_{j+1}))$, where S_{j+1} is a random variable denoting the expected number of fitness evaluations between the first solution in chunk $j+1$ being constructed and the first Decision Stage of chunk j beginning. We note as with the proof of Theorem 5.11 (with the same parameter combinations i.e., $\tau \leq \left(\frac{1}{2}-\varepsilon\right)n\ln(n)$, for some constant $\varepsilon > 0$, $w = o\left(\frac{n}{\exp(2\tau/n)}\right)$), that the contribution of the S_{j+1} terms can be bounded by $o(n^2)$.

We let N_j denote the number of Decision Stages within chunk j, and $X_{j,1},\ldots,X_{j,N_J}$ the number of fitness function evaluations within the Decision Stage and the Exploitation Stage of the chosen operator. Then $\mathrm{T}_j = \sum_{k=1}^{N_j} X_{j,k}$ and, by Wald's Inequality, using $\mathrm{E}(X_j)$ to denote an upper bound on all $\mathrm{E}(X_{j,k})$ in chunk j,

$$\mathrm{E}(\mathrm{T}_j) \leq \mathrm{E}(N_j)\cdot\mathrm{E}(X_j).$$

We can upper bound $\mathrm{E}(N_j)$, the number of Decision Stages in chunk j, by lower bounding the expected number of improvements found from the Decision and Exploitation Stage of the chosen operator, and using the Additive Drift Theorem (Theorem 2.11) to find the expected number of operator selections required to enter the next chunk.

Recall from the proof of Theorem 5.11 that, denoting F_1 and F_2 as the events that the 1BITFLIP and 2BITFLIP operators fail to find an improvement during τ iterations:

$$e^{-\frac{\tau}{n}}-\frac{1}{n} \leq \Pr(F_1) \leq e^{-\frac{\tau}{n}}; \quad e^{-\frac{2\tau}{n}(1-\frac{j-1}{w})}-\frac{1}{n} \leq \Pr(F_2) \leq e^{-\frac{2\tau}{n}(1-\frac{j}{w})}.$$

After each Decision Stage, an operator will be chosen. It is possible for both operators to succeed in one iteration; for our analysis, we note that this happens an expected $O(n) \cdot \frac{1}{n} \cdot \frac{2n-2i-2}{n^2} = O(1/n)$ times throughout the optimisation process, and opt to ignore any progress made by such Decision and Exploitation Stages, accounting for this by adding a lower-order $O(\tau + n)$ term to the expected runtime. With this in mind, we use the first hit probability (Theorem 2.3 in Chapter 2) to look at the probability of each operator being chosen:

$$\Pr(C_1) = \frac{\frac{1}{n}}{\frac{1}{n} + \frac{2(n-i-1)}{n^2}} = \frac{n}{3n-2i-2}; \qquad \Pr(C_2) = \frac{2n-2i-2}{3n-2i-2}.$$

Note that in chunk j, with $\frac{(j-1)n}{w} - 1 \leq i \leq \frac{jn}{w} - 1$, we can bound the two probabilities:

$$\frac{w}{3w-2j+2} \leq \Pr(C_1) \leq \frac{w}{3w-2j}; \qquad \frac{2w-2j}{3w-2j} \leq \Pr(C_2) \leq \frac{2w-2j+2}{3w-2j+2}.$$

We look at finding a lower bound for $\mathrm{E}(D_j)$, the expected number of improvements D_j produced following an operator Decision and Exploitation Stage in chunk j. Since we are looking for a lower bound, we use the lower bound for the probability of choosing each operator, and an upper bound on the probability of each operator failing to find an improvement during τ iterations. Note there is one improvement from the Decision Stage.

$$\begin{aligned}
\mathrm{E}(D_j) &\geq 1 + \Pr(C_1) \cdot \left(\frac{1}{\Pr(F_1)} - 1 \right) + \Pr(C_2) \cdot \left(\frac{1}{\Pr(F_2)} - 1 \right) \\
&\geq 1 + \frac{w}{3w-2j+2} \cdot \left(\frac{1}{\exp(-\frac{\tau}{n})} - 1 \right) + \frac{2w-2j}{3w-2j} \left(\frac{1}{\exp(-\frac{2\tau}{n}(1-\frac{j}{w}))} - 1 \right) \\
&\geq \frac{w}{3w-2j+2} \cdot e^{\frac{\tau}{n}} + \frac{2(w-j)}{3w-2j} \cdot e^{\frac{2\tau}{n}(1-\frac{j}{w})} \\
&\geq \frac{w}{3w-2j+2} \cdot e^{\frac{\tau}{n}} + \frac{2(w-j)}{3w-2j+2} \cdot e^{\frac{2\tau}{n}(1-\frac{j}{w})}.
\end{aligned}$$

In each chunk, $\frac{n}{2w}$ fitness improvements are needed to exit the chunk in expectation. If each random operator decision leads to more $\mathrm{E}(D_j)$ improvements, by the Additive Drift Theorem, the expected number of 'D+E' Stages required to complete chunk j, $\mathrm{E}(N_j)$ satisfies:

$$\begin{aligned}
\mathrm{E}(N_j) &\leq \frac{n/(2w)}{\mathrm{E}(D_j)} \leq \frac{n}{2w} \cdot \frac{1}{\frac{w}{3w-2j+2} \cdot e^{\frac{\tau}{n}} + \frac{2(w-j)}{3w-2j+2} \cdot e^{\frac{2\tau}{n}(1-\frac{j}{w})}} \\
&\leq \frac{n(3w-2j+2)}{2w} \cdot \left(\frac{1}{w \cdot e^{\frac{\tau}{n}} + 2(w-j) \cdot e^{\frac{2\tau}{n}(1-\frac{j}{w})}} \right).
\end{aligned}$$

We now look at bounding $\mathrm{E}(X_j)$, an upper bound on the number of fitness function evaluations for a Decision and Exploitation Stage within chunk j. The worst case expected time for the duration of the Decision Stage in chunk j is

$$\begin{aligned}\mathrm{E}(\mathrm{T}_{DS_j}) \leq 2\cdot\left(1-\left(1-\frac{1}{n}\right)\left(1-\frac{2(w-j)}{nw}\right)\right)^{-1} &= \frac{2n^2w}{(3w-2j)n-2(w-j)}\\ &= \frac{2nw}{3w-2j}+o(n).\end{aligned}$$

The $o(n)$ term will be omitted for brevity, but note that it makes no difference to the leading constant on the asymptotically significant term.

We can also upper bound $\mathrm{E}(X_j|C)$, the expected iterations before a selected operator fails to produce an improvement within τ iterations:

$$\begin{aligned}\mathrm{E}(X_j|C_1) &\leq \tau+(1/\Pr(F_1)-1)\cdot\min\{\tau,n\}\\ \mathrm{E}(X_j|C_2) &\leq \tau+(1/\Pr(F_2)-1)\cdot\min\{\tau,0.5n/(1-j/w)\}.\end{aligned}$$

Hence:

$$\begin{aligned}\mathrm{E}(X_j) &\leq \mathrm{E}(\mathrm{T}_{DS_j})+\Pr(C_1)\cdot\mathrm{E}(X_j|C_1)+\Pr(C_2)\cdot\mathrm{E}(X_j|C_2)\\ &\leq \frac{2nw}{3w-2j}+\tau+\frac{w}{3w-2j}\cdot\frac{\min\{\tau,n\}}{e^{-\frac{\tau}{n}}-1/n}+\frac{2w-2j+2}{3w-2j+2}\cdot\frac{\min\{\tau,n/(2-2j/w)\}}{e^{-\frac{2\tau}{n}(1-\frac{j-1}{w})}-1/n}\\ &\leq \frac{2nw}{3w-2j}+\tau+\frac{w}{3w-2j}\cdot\frac{\min\{\tau,n\}}{e^{-\frac{\tau}{n}}-1/n}+\frac{2w-2j+2}{3w-2j}\cdot\frac{\min\{\tau,n/(2-2j/w)\}}{e^{-\frac{2\tau}{n}(1-\frac{j-1}{w})}-1/n}.\end{aligned}$$

If $j=w$, then $\min\left\{\frac{\tau}{n},\frac{1}{2-2j/w}\right\}$ is undefined. However, we know that the chosen operator would only be applied for a maximum of τ steps. We thus refer to the functions $M_1(x)=\min\left\{\frac{\tau}{n},x\right\}$ an $M_2(j,w)$ from now on, where

$$M_2(j,w):=\begin{cases}\frac{\tau}{n} & \text{if } j=w \text{ or } \frac{0.5}{1-j/w}>\frac{\tau}{n}\\ \frac{0.5}{1-j/w} & \text{otherwise.}\end{cases}$$

Now, $\mathrm{E}(X_j)$ can be simplified to

$$\mathrm{E}(X_j)\leq o(n)+\frac{n}{3w-2j}.$$

$$\left(2w + \frac{\tau}{n}(3w-2j) + we^{\frac{\tau}{n}}M_1(1) + (2w-2j+2)e^{\frac{2\tau}{n}(1-\frac{j-1}{w})}M_2(j,w)\right).$$

Finally, we bound the overall expected runtime of the hyper-heuristic. We have:

$$\mathrm{E(T)} \le \sum_{j=1}^{w} (\mathrm{E(T}_j) + \mathrm{E}(S_{j+1})) \le \left(\sum_{j=1}^{w} \mathrm{E}(N_j)\,\mathrm{E}(X_j)\right) + o(n^2)$$

and hence

$$\mathrm{E(T)} \le o(n^2) + \frac{n^2}{2} \cdot \left[\sum_{j=1}^{w} \frac{3w-2j+2}{3w-2j} \left(\frac{2w + \frac{\tau}{n}(3w-2j) + we^{\frac{\tau}{n}}M_1(1) + (2(w-j+1))e^{\frac{2\tau}{n}(1-\frac{j-1}{w})}M_2(j,w)}{w \cdot \left(w \cdot e^{\frac{\tau}{n}} + 2(w-j) \cdot e^{\frac{2\tau}{n}(1-\frac{j}{w})}\right)}\right)\right].$$

□

We can now prove the main result of this chapter, i.e., Corollary 6.3, which states that Generalised Greedy Gradient finds the best possible expected runtime achievable for LEADINGONES with the available heuristics, up to lower order terms.

Proof: [Of Corollary 6.3]

We set $w = \log^2 n$.

Consider first the terms $M_1(1)$ and $M_2(j,w)$. Since we have $\tau = \omega(n)$, we have $\frac{\tau}{n} = \omega(1)$; it is easy to see that $M_1(1) = 1$. For $M_2(j,w)$, it is important to note that for the first half of the search, $M_2(j,w) = \frac{0.5}{1-j/w}$, and in the second half of the search, $M_2(j,w) = O(\frac{\tau}{n}) = O(\ln n)$ for any constant $c > 0$ (there is a special case when $j = w$ in which $M_2(j,w) = \frac{\tau}{n} = O(\ln n)$). Note also that $\frac{\tau}{n} = \omega(1)$, $e^{2\tau/n \cdot 1/w} = o(n^{1/w}) = 1 + o(1)$, $\frac{3w-2j+2}{3w-2j} = 1 + o(1)$.

Hence, by Theorem 6.4, the expected runtime will become:

$$\mathrm{E(T)} \le o(n^2) + \frac{n^2}{2} \cdot \sum_{j=1}^{w} \left[(1+o(1)) \cdot \left(\frac{2w + \frac{\tau}{n} \cdot (3w-2j) + w \cdot e^{\frac{\tau}{n}} + (2(w-j+1)) \cdot e^{\frac{2\tau}{n} \cdot \frac{1}{w}} e^{\frac{2\tau}{n}(1-\frac{j}{w})} \cdot M_2(j,w)}{w \cdot \left(w \cdot e^{\frac{\tau}{n}} + 2(w-j) \cdot e^{\frac{2\tau}{n}(1-\frac{j}{w})}\right)}\right)\right]$$

$$= o(n^2) + \frac{n^2}{2} \cdot \sum_{j=1}^{w} \left(\frac{2w + \frac{\tau}{n} \cdot (3w-2j) + w \cdot e^{\frac{\tau}{n}} + (2(w-j+1)) \cdot e^{\frac{2\tau}{n}(1-\frac{j}{w})} \cdot M_2(j,w)}{w \cdot \left(w \cdot e^{\frac{\tau}{n}} + 2(w-j) \cdot e^{\frac{2\tau}{n}(1-\frac{j}{w})}\right)}\right)$$

$$= o(n^2) + \frac{n^2}{2} \cdot \sum_{j=1}^{w} \left[\frac{1}{w} \cdot \right.$$

$$\left. \left(\frac{2w + \frac{\tau}{n} \cdot (3w - 2j)}{w \cdot e^{\frac{\tau}{n}} + 2(w-j) \cdot e^{\frac{2\tau}{n}(1-\frac{j}{w})}} + \frac{w \cdot e^{\frac{\tau}{n}} + 2(w-j+1) \cdot e^{\frac{2\tau}{n}(1-\frac{j}{w})} \cdot M_2(j,w)}{w \cdot e^{\frac{\tau}{n}} + 2(w-j) \cdot e^{\frac{2\tau}{n}(1-\frac{j}{w})}} \right) \right]$$

$$= o(n^2) + \frac{n^2}{2} \cdot \sum_{j=1}^{w} \left[\frac{1}{w} \cdot \left(\frac{w \cdot e^{\frac{\tau}{n}} + 2(w-j+1) \cdot e^{\frac{2\tau}{n}(1-\frac{j}{w})} \cdot M_2(j,w)}{w \cdot e^{\frac{\tau}{n}} + 2(w-j) \cdot e^{\frac{2\tau}{n}(1-\frac{j}{w})}} \right) \right].$$

Since $\tau/n = \omega(1)$, the terms with the larger exponents will asymptotically dominate each summand. When $j > \frac{w}{2}$, we note that $w \cdot \exp(\frac{\tau}{n}) = \omega(2(w-j+1) \cdot \exp(\frac{2\tau}{n}(1-\frac{j}{w})))$ and the first term will asymptotically dominate the second term in the numerator (and similarly in the denominator); vice versa for $j < \frac{w}{2}$. We consider $j = \frac{w}{2}$ as a special case.

$$\mathrm{E(T)} \leq o(n^2) + \frac{n^2}{2} \cdot \sum_{j=1}^{w} \left[\frac{1}{w} \cdot \left(\frac{w \cdot e^{\frac{\tau}{n}} + 2(w-j+1) \cdot e^{\frac{2\tau}{n}(1-\frac{j}{w})} \cdot M_2(j,w)}{w \cdot e^{\frac{\tau}{n}} + 2(w-j) \cdot e^{\frac{2\tau}{n}(1-\frac{j}{w})}} \right) \right]$$

$$= o(n^2) + \frac{n^2}{2} \cdot \left(\sum_{j=1}^{w/2-1} \frac{1}{w} \cdot \frac{2(w-j+1)}{2(w-j)} \cdot \frac{e^{\frac{2\tau}{n}(1-\frac{j}{w})} \cdot \frac{0.5}{1-j/w}}{e^{\frac{2\tau}{n}(1-\frac{j}{w})}} \right)$$

$$+ \frac{n^2}{2w} \cdot \frac{(2w+2)e^{\frac{2\tau}{n}}}{2w \cdot e^{\frac{2\tau}{n}}} + \frac{n^2}{2} \cdot \left(\sum_{j=w/2+1}^{w} \frac{1}{w} \cdot \frac{e^{\frac{\tau}{n}}}{e^{\frac{\tau}{n}}} \right)$$

$$= o(n^2) + \frac{n^2}{2} \cdot \left(\sum_{j=1}^{w/2-1} \frac{1}{w} \cdot \frac{0.5}{1 - \frac{j}{w}} \right) + \frac{n^2}{2w} + \frac{n^2}{2} \cdot \left(\sum_{j=w/2+1}^{w} \frac{1}{w} \right)$$

$$= \frac{n^2}{4} \cdot \sum_{j=1}^{w/2} \left(\frac{1}{w-j} \right) + \frac{n^2}{4} + o(n^2) = \left(\frac{1+\ln(2)}{4} \right) n^2 + o(n^2).$$

since $\frac{2(w-(\frac{w}{2})+1)}{2(w-(\frac{w}{2}))} = \frac{2w+2}{2w} = 1 + o(1)$. □

We have shown that the Generalised Greedy Gradient hyper-heuristic is able to achieve the same optimal runtime as the Generalised Random Gradient hyper-heuristic. We will see in the next section that the Generalised Greedy Gradient hyper-heuristic is faster in practice.

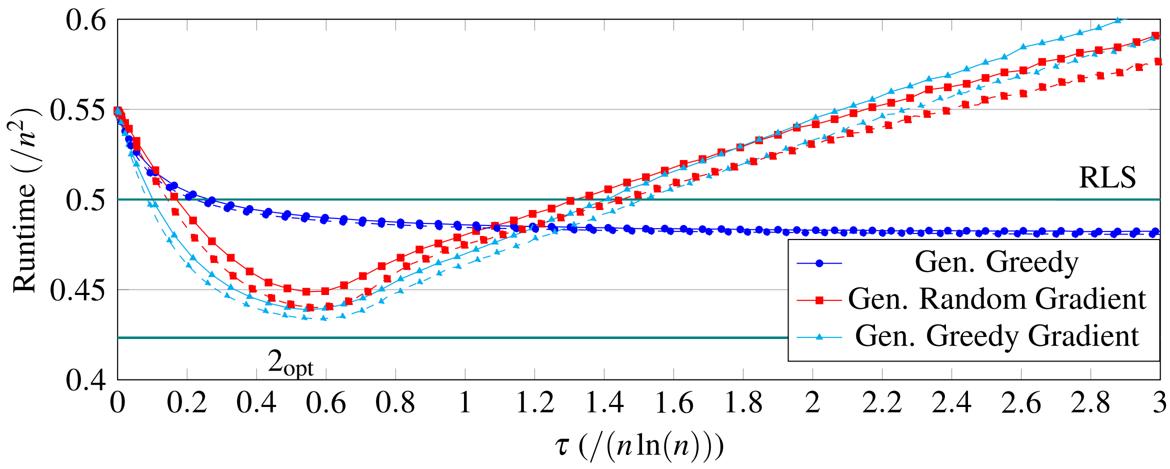

Fig. 6.1 Average number of fitness function evaluations required by the hyper-heuristics with $k = 2$ operators to find the LEADINGONES optimum for $n = 10{,}000$ (solid), $n = 50{,}000$ (dashed).

6.2 Experimental Supplements

In the previous section, we proved that the Generalised Greedy Gradient hyper-heuristic performs efficiently for the LEADINGONES benchmark function for sufficiently large problem sizes n. In this section, we present some experimental results comparing the Generalised Greedy Gradient hyper-heuristic with the Generalised Greedy and Generalised Random Gradient hyper-heuristic on LEADINGONES in order to shed light on their performance for different problem sizes ($n = 10,000$ and $n = 50,000$) (see Subsection 2.5.3 in Chapter 2 for details on how we are able to give results up to such large problem sizes). All parameter combinations have been simulated 10,000 times. We consider the generalised hyper-heuristics using two stochastic mutation operators (i.e., 1BITFLIP and 2BITFLIP).

We consider the impact of the parameter τ (i.e., the learning period) on the Generalised Greedy, Generalised Random Gradient and Generalised Greedy Gradient hyper-heuristics.

Figure 6.1 shows the runtimes for the three generalised hyper-heuristics on LEADINGONES for $n = 10{,}000$ and $n = 50{,}000$, illustrating the effect τ has on the runtime.

For the Generalised Greedy hyper-heuristic, we see that for the larger problem size, the relative runtimes improve slightly. The figure implies that for a sufficiently large τ the hyper-heuristic outperforms RLS. In particular, for every value of τ greater than approximately $0.3n\ln n$ (for $n = 50{,}000$ this value is 162,297). Also, for every $\tau \geq n\ln n$ the runtime of the GG hyper-heuristic does not change much confirming the robustness of the hyper-heuristic to the choice of τ, indicated by Corollary 5.6.

The performance of the Generalised Random Gradient hyper-heuristic seems more dependent on the choice of τ. It is worth noting that as the problem size increases, for $\tau \approx 0.6n\ln(n)$, the runtime seems to be approaching the optimal performance proved in Corollary 5.10 (i.e., $\frac{1+\ln(2)}{4}n^2 \approx 0.42329n^2$). For well chosen τ values, the hyper-heuristic beats the expected runtime of RLS, and also the experimental runtime for the recently presented reinforcement learning hyper-heuristic that chooses between different neighbourhood sizes for RLS$_k$ ($0.450n^2$) for LEADINGONES with the parameter choices used and presented by Doerr et al. (2016a). We did not perform an experimental comparison with this algorithm as it is unclear how to set the various parameters for a fair comparison (note also that theoretical performance guarantees of the reinforcement learning hyper-heuristic were only proven for ONEMAX (Doerr et al., 2016a)) . As τ increases past $0.6n^2$, we see a detriment in the performance of the hyper-heuristic. It is worth noting, however, that for $n = 50{,}000$, it is required that $\tau > 1.5n\ln(n) = 811{,}483$ to be worse than the RLS expected runtime of $0.5n^2$, indicating that the parameter is very robust as suggested by Corollary 5.6.

A similar consideration regarding the parameter τ can be made for the Generalised Greedy Gradient hyper-heuristic. We see that Generalised Greedy Gradient's performance is closer to the best possible one (i.e., from Corollary 5.10 of $\approx 0.42329n^2$) than the Generalised Random Gradient. Although theoretically the two hyper-heuristics both achieve this best performance up to lower order terms, the experimental results suggest that Generalised Greedy Gradient is faster in practice.

We will see further experimental results for the hyper-heuristics in Chapter 8, where we analyse their performance when selecting from sets of low-level heuristics of size greater than two.

6.3 Chapter Summary and Discussion

In this chapter, we have introduced a new hyper-heuristic based on the theoretical insights of Chapter 5 which combines the learning ideas from the Generalised Greedy and Generalised Random Gradient hyper-heuristic. We have proven that the new hyper-heuristic, Generalised Greedy Gradient, is able to learn efficiently which heuristic is preferable (e.g., on GENERALISEDGAPPATHWITHTRAPS) and utilise this knowledge to achieve the best-possible runtime achievable with the available low-level heuristics, up to lower order terms, on LEADINGONES (for smart choices of the learning period τ). Furthermore, for each value of the learning period τ, Generalised Greedy Gradient outperforms Generalised Random Gradient in expectation (giving a smaller upper bound in the expected runtime). An experimental analysis confirmed that Generalised Greedy Gradient outperforms Generalised

Random Gradient also in practice for the best choices of τ. Although Generalised Random Gradient is simpler to setup and makes a quicker heuristic decision, we believe Generalised Greedy Gradient is preferable in practice.

Chapter 7

Examples Where Selection-based Hyper-heuristics are Faster

Thus far, the majority of theoretical work on hyper-heuristics has considered low-level heuristic sets consisting of different mutation operators. However, one of the many strengths of hyper-heuristics is their ability to choose between low-level heuristics of various types. In realistic situations, the heuristics will not necessarily be operators differing by a single parameter. In this chapter, we extend the understanding of the behaviour and performance of hyper-heuristics by evaluating their capability at escaping local optima when equipped with low-level elitist and non-elitist selection heuristics.

In this chapter, we will present a systematic analysis of the move acceptance hyper-heuristic (MAHH) (introduced by Lehre and Özcan 2013) for multimodal optimisation problems, where the considerable advantages of changing the mutation operator during the run will be highlighted. We will provide examples of instance classes where MAHH is efficient at escaping local optima and examples where it is not. We first perform an analysis on the standard unimodal ONEMAX benchmark function to identify the range of parameter values for p that allow MAHH to hillclimb, hence to locate local optima efficiently. In particular, we prove that for any $p = \frac{1}{(1+\varepsilon)n}$, for any positive constant ε, MAHH is asymptotically as efficient as the best mutation-based unbiased black box algorithm (Lehre and Witt, 2012), even though it does not rely on elitism. Afterwards, we highlight the power of the hyper-heuristic by showing how it efficiently escapes the local optima of the standard CLIFF_d multimodal benchmark class of functions for which it even achieves the best known expected runtime of $O(n \log n)$ (for problem-independent search heuristics) on the hardest instances. Thus, we prove that it considerably outperforms established elitist and non-elitist evolutionary algorithms and the popular METROPOLIS algorithm (Algorithm 8 in Chapter 2 (Metropolis et al., 1953)). We complete the picture by using the standard JUMP_k multimodal

Algorithm 13 Move Acceptance Hyper-heuristic - OI Variant (MAHH$_{\text{OI}}$) (Lehre and Özcan, 2013)

```
1: Choose $x \in \{0,1\}^n$ uniformly at random
2: while termination criteria not satisfied do
3:     $x' \leftarrow$ FLIPRANDOMBIT($x$)
4:     $\text{ACC} \leftarrow \begin{cases} \text{ALLMOVES} & \text{with probability p} \\ \text{ONLYIMPROVING} & \text{otherwise} \end{cases}$
5:     if ACC($x,x'$) then
6:         $x \leftarrow x'$
7:     end if
8: end while
```

instance class to provide an example of problem characteristics where MAHH is not efficient. Nevertheless, it is efficient for instances of moderate jump size (i.e., constant) where it still considerably outperforms METROPOLIS.

7.1 Selection-based Hyper-heuristics

In this section, we introduce the move acceptance hyper-heuristic (MAHH) previously considered by Lehre and Özcan (2013). In their work, the advantages of switching between selection operators rather than just using one all the time were not evident. Recall that the ALLMOVES move acceptance operator accepts any move regardless of fitness and ONLYIMPROVING only accepts moves that increase the fitness of the candidate solution. In each iteration, the algorithm flips one bit chosen uniformly at random and with probability p, the ALLMOVES (AM) acceptance operator is used, while with probability $1-p$ the ONLYIMPROVING (OI) acceptance operator is used (see Section 3.1). Algorithm 13 shows its pseudocode.

We also consider a variant of Algorithm 13 whereby the IMPROVINGANDEQUAL (IE) (moves which lead to increased or equal fitness are accepted) acceptance operator is used instead of the OI operator. Thus, in Algorithm 14 the AM acceptance operator is used with probability p and the IE acceptance operator is used with probability $1-p$.

All the functions we consider in this chapter are functions of unitation (i.e., changing the number of 1-bits in the bit-string by 1 will always change the fitness value), since no plateaus of constant fitness are present. We point out that all the statements made in the chapter for the MAHH$_{\text{OI}}$ hyper-heuristic will also hold for the MAHH$_{\text{IE}}$ hyper-heuristic. In the rest of this chapter, we will only consider the analysis of the MAHH$_{\text{OI}}$ hyper-heuristic.

Algorithm 14 Move Acceptance Hyper-heuristic - IE Variant (MAHH$_{\text{IE}}$)

```
1: Choose $x \in \{0,1\}^n$ uniformly at random
2: while termination criteria not satisfied do
3:     $x' \leftarrow$ FLIPRANDOMBIT$(x)$
4:     ACC $\leftarrow \begin{cases} \text{ALLMOVES} & \text{with probability p} \\ \text{IMPROVINGANDEQUAL} & \text{otherwise} \end{cases}$
5:     if ACC$(x,x')$ then
6:         $x \leftarrow x'$
7:     end if
8: end while
```

Let $\Delta(i) = \mathrm{E}(f(x^{t+1}) - f(x^t) \mid f(x^t) = i)$ be the drift on the fitness (i.e., the expected change in the fitness in one iteration) of MAHH$_{\text{OI}}$ in the t-th iteration when the fitness function has value $f(x^t) = i$. We further define $\Delta^+(i) = \{\Delta(i) \mid \mathbb{1}\{\Delta(i) > 0\}\}$ (respectively $\Delta^-(i)$) to be the positive (and negative) contributions to the fitness in one iteration respectively (note that the positive and negative contributions are not conditional on the acceptance criteria). Hence, for MAHH$_{\text{OI}}$ we have

$$\begin{aligned}\Delta(i) &= (1-p)\cdot\Delta^+(i) + p\cdot(\Delta^+(i) - \Delta^-(i)) \\ &= \Delta^+(i) - p\cdot\Delta^-(i). \qquad (7.1)\end{aligned}$$

For MAHH$_{\text{OI}}$ to have a polynomial runtime, it is necessary that there are no large regions of negative drift (i.e., the fitness does not decrease in expectation). That is, we would like $\Delta(i) > 0$ to occur most of the time. Equation 7.1 does not necessarily say anything about the fitness landscape. Naturally, setting $p = 0$ will maximise $\Delta(i)$, but when there are no improving moves to be made, the AM operator must be used and $p > 0$ is necessary. However, for unimodal functions the best parameter value is $p = 0$ since negative contributions to the drift are never necessary as there are no local optima to be escaped from. We will identify in the next section ranges of p values that lead to efficient and inefficient performance for MAHH$_{\text{OI}}$ on the unimodal benchmark function ONEMAX.

7.2 Unimodal Functions

We begin the study of MAHH$_{\text{OI}}$ by analysing its performance on unimodal functions (i.e., functions with exactly one local optimum), which were introduced in Section 2.2. Obviously, for functions where accepting worsening moves is never required, we cannot expect the hyper-heuristic to perform better than elitist algorithms. Nevertheless, Theorem 7.3 shows that MAHH$_{\text{OI}}$ can still be very efficient on ONEMAX$(x) := \sum_{i=0}^{n} x_i$, even with $p > 0$. We first introduce Theorem 7.1 and Theorem 7.2, which show that the parameter p should not be too large.

Theorem 7.1 *The runtime of* MAHH$_{\text{OI}}$ *on* ONEMAX*, with* $p \geq \frac{\sqrt{n}\log^2 n}{n}$*, is at least* $n^{\Omega(\log n)}$ *with probability at least* $1 - n^{-\Omega(\log n)}$.

Proof: To prove this result, we will apply the Simplified Drift Theorem with Scaling (Theorem 2.22), which allows us to handle regions of sub-constant negative drift.

Let i denote the number of 1-bits in the current bit-string and consider the expected change in the Hamming distance to the optimum (which for ONEMAX is equivalent to the expected change in solution fitness) in one iteration of MAHH$_{\text{OI}}$:

$$\Delta(i) = \frac{n-i}{n} - p \cdot \frac{i}{n} = -\frac{i + pi - n}{n}. \tag{7.2}$$

Improving mutations (which require flipping one of the $n-i$ remaining 0-bits) are accepted by both acceptance operators, while worsening mutations (flipping one of the i 1-bits) are accepted only by the AM operator, chosen with probability p.

We will consider the drift in the region of $n - c\sqrt{n} \leq i < n$ of length $\ell = c\sqrt{n}$, where $c > 0$ is a constant. As $\Delta(i)$ decreases with i, the negative drift is weakest at $i = n - \ell$:

$$\begin{aligned}
\Delta(n-\ell) &= -\frac{(n-\ell) + (n-\ell)p - n}{n} = -\frac{np - \ell p - \ell}{n} \\
&= -\left(p - \frac{c\sqrt{n}p}{n} - \frac{c\sqrt{n}}{n}\right) \\
&\leq -\left(\frac{\sqrt{n}\log^2 n}{n} - \frac{c\sqrt{n}}{n} - \frac{c\log^2 n}{n}\right) \\
&\leq -c^* \frac{\sqrt{n}\log^2 n}{n},
\end{aligned}$$

for sufficiently large n, where $0 < c^* < 1$ is a constant. Choosing $\varepsilon = c^* \frac{\sqrt{n}\log^2 n}{n}$ thus satisfies the drift condition of the Simplified Drift Theorem with Scaling.

The second condition, forbidding large jumps, is satisfied by choosing $r = 2$ and verifying that for all $j \in \mathbb{N}_0$,

$$\Pr(|\delta_t(i)| \geq jr) \leq e^{-j},$$

which is trivially true: for $j = 0$, $e^{-j} = 1$, and for $j \geq 1$, the probability that the distance to the optimum changes by at least $jr \geq 2$ is 0, as the mutation operator used by MAHH$_{\text{OI}}$ only flips one bit per iteration.

Finally, we need to verify the following condition:

$$1 \le r^2 \le \varepsilon\ell/(132\log(r/\varepsilon)).$$

To this end, we note that

$$\varepsilon\ell = c^* \frac{\sqrt{n}\log^2 n}{n} \cdot c\sqrt{n} = c^* c \log^2 n,$$

and

$$\begin{aligned} 132\log(r/\varepsilon) &= 132\log\left(\frac{2\sqrt{n}}{c^*\log^2 n}\right) \\ &\le 132\log(2\sqrt{n}) \le 66\log(n) + O(1). \end{aligned}$$

Thus $\varepsilon\ell/(132\log(r/\varepsilon)) = \Omega(\log n)$ is greater than $r^2 = 4$ for sufficiently large n.

Having verified that all the conditions of the Simplified Drift Theorem with Scaling are satisfied, we apply the theorem, concluding that the probability that the optimum is found within $e^{\varepsilon\ell/(132r^2)} = n^{\Omega(\log n)}$ iterations is at most $O(e^{\varepsilon\ell/(132r^2)}) = n^{-\Omega(\log n)}$. □

For ONEMAX, the larger the value of p, the greater the probability of accepting a move away from the global optimum and the greater the expected runtime of the hyper-heuristic. For instance, an application of the standard Negative Drift Theorem (Theorem 2.20) for MAHH$_{\text{OI}}$ with any constant value of p gives the following result.

Theorem 7.2 *The runtime of* MAHH$_{\text{OI}}$ *on* ONEMAX*, with* $p = \Theta(1)$*, is at least* $2^{\Omega(n)}$ *with probability at least* $1 - 2^{-\Omega(n)}$.

Proof: Recall Equation 7.2 from the proof of Theorem 7.1: the drift in one iteration for MAHH$_{\text{OI}}$ on ONEMAX, when the current solution contains i 1-bits, is

$$\Delta(i) = \frac{n-i}{n} - p \cdot \frac{i}{n} = -\frac{i + pi - n}{n}.$$

Let $p > 0$ be a constant. We have that for $i \ge \frac{1+C}{1+p} \cdot n$, $\Delta(i) \le -C$, for some constant C, with $p > C > 0$. Hence, for a region of size $(n-1) - \frac{1+C}{1+p} \cdot n = \Omega(n)$, the drift is negative (at most $-C$). Since MAHH$_{\text{OI}}$ is a local search algorithm, this region of negative drift cannot be escaped by a large jump. By the Negative Drift Theorem (Theorem 2.20), the runtime in this case will be at least $2^{\Omega(n)}$ with probability at least $1 - 2^{-\Omega(n)}$. □

We now provide an upper bound on p for which the hyper-heuristic is efficient.

Theorem 7.3 *The expected runtime of* MAHH$_{\text{OI}}$ *on* ONEMAX*, with* $p < \frac{1}{n-1}$*, is at most*

$$\mathrm{E}(\mathrm{T}_p) \leq \frac{n}{1-p(n-1)} + \frac{n}{1+p} \cdot \ln\left(\frac{n}{1-p(n-1)}\right).$$

Proof: Recall Equation 7.2 from the proof of Theorem 7.1: the drift in one iteration for MAHH$_{\text{OI}}$ on ONEMAX, when the current solution contains i 1-bits, is

$$\Delta(i) = \frac{n-i}{n} - p \cdot \frac{i}{n} = \frac{n-(1+p)i}{n}.$$

In order to bound the expected runtime, we require $\Delta(i) > 0$ throughout the fitness landscape to apply the Variable Drift Theorem (Theorem 2.18). Clearly the drift decreases as the fitness value increases, and the drift will be smallest at $i = n-1$, and we note that for $p < \frac{1}{n-1}$, $\Delta(n-1) = \frac{n-(1+p)(n-1)}{n} > 0$.

The expected runtime of MAHH$_{\text{OI}}$ on ONEMAX is then, by the Variable Drift Theorem:

$$\begin{aligned}\mathrm{E}(\mathrm{T}_p) &\leq \frac{n}{1-p(n-1)} + \int_{i=0}^{n-1} \frac{n}{n-(1+p)i}\,di \\ &= \frac{n}{1-p(n-1)} + \frac{n}{1+p} \cdot \ln\left(\frac{n}{1-p(n-1)}\right).\end{aligned}$$

□

If $p = 0$, we have the well-studied Randomised Local Search (RLS) algorithm, and Theorem 7.3 gives an expected runtime of $\mathrm{E}(\mathrm{T}_0) \leq n + n\ln n = O(n\log n)$. For $p = \frac{1}{n}$, combining Theorem 7.3 with a separate argument regarding the time needed to reach a fitness of n from a fitness of $n-1$ (using e.g., Lemma 7.5) yields a $O(n\log n)$ bound on the expected runtime. For $p = \frac{1}{(1+\varepsilon)n}$ with any constant $\varepsilon > 0$, applying Theorem 7.3 directly shows that any ONEMAX style hillclimbs can be completed in expected time $O(n\log n)$[1].

Corollary 7.4 *The expected runtime of* MAHH$_{\text{OI}}$ *on* ONEMAX*, with* $p = \frac{1}{(1+\varepsilon)n}$*, for any constant* $\varepsilon > 0$*, is* $O(n\log n)$.

[1]Note that this parameter ε is not related to the parameter ε used in the previous (and future) chapters on mutation-based hyper-heuristics.

Proof: We substitute $p = \frac{1}{(1+\varepsilon)n}$ into the result from Theorem 7.3:

$$\begin{aligned} \mathrm{E}\left(\mathrm{T}_{\frac{1}{(1+\varepsilon)n}}\right) &\leq \frac{n^2(1+\varepsilon)}{\varepsilon n+1} + \frac{n^2(1+\varepsilon)\ln\left(\frac{n^2(1+\varepsilon)}{\varepsilon n+1}\right)}{(1+\varepsilon)n+1} \\ &= O(n\log n). \end{aligned}$$

□

Since with higher values of p the hyper-heuristic cannot hillclimb in $O(n\log n)$ expected time (i.e., the best possible expected time for unbiased (1+1) black box heuristics), for the remainder of the chapter, we consider MAHH$_{\mathrm{OI}}$ with $p = \frac{1}{(1+\varepsilon)n}$. We will show how with this parameter value, as well as hill-climbing efficiently, the hyper-heuristic can escape from difficult local optima effectively.

7.3 When the Hyper-heuristic is Efficient

We first analyse CLIFF$_d$ (with $1 < d < \frac{n}{2}$), a class of benchmark functions with a local optimum that must be escaped from before the global optimum can be found. The CLIFF$_d$ function was originally introduced by Jägersküpper and Storch (2007) as an example where non-elitist algorithms outperform elitist algorithms. We refer to the local optima at $i = n-d$ as the 'cliff', d as the 'length of the cliff' and the two ONEMAX style hillclimbs as the 'first slope' and 'second slope' respectively (See Figure 7.1).

$$\textsc{Cliff}_d(x) := \begin{cases} \textsc{OneMax}(x) & |x|_1 \leq n-d \\ \textsc{OneMax}(x) - d + \frac{1}{2} & \text{otherwise.} \end{cases}$$

To find the global optimum of the CLIFF$_d$ function, it is necessary to escape the local optimum by either dropping down from the cliff and accepting a worse candidate solution and then climbing up the second slope or by making a prohibitive jump to the global optimum on the other side of the cliff (with standard bit mutation, this will require expected exponential time in the length of the cliff and it is not possible at all with local mutations).

We now consider the performance of MAHH$_{\mathrm{OI}}$ on CLIFF$_d$. Clearly, with $p = 1$, MAHH$_{\mathrm{OI}}$ reduces to a random walk across the fitness landscape. Similarly, if $p = 0$, with probability at least $\frac{1}{2}$, the bit-string is initialised with at most $\frac{n}{2}$ 1-bits, and will hillclimb to the top of the cliff. There is no improving step from this position and the global optimum

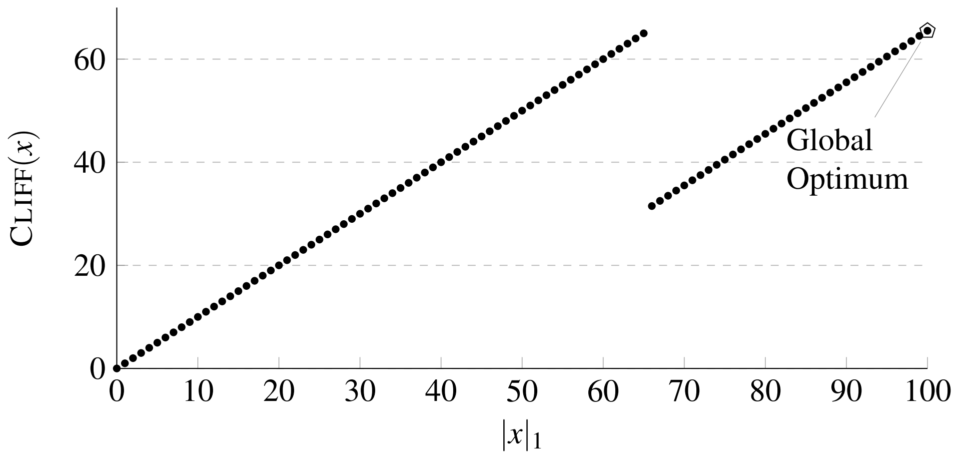

Fig. 7.1 An example of $\text{Cliff}_d(x)$ with $n = 100$ and $d = 35$.

cannot be reached. By the law of total expectation, the expected runtime will be infinite. We will show that using MAHH$_{\text{OI}}$ with $p = \frac{1}{(1+\varepsilon)n}$, which has been shown to hillclimb efficiently (Corollary 7.4), will still allow worsening moves with sufficiently high probability to be able to move down from the cliff and reach the global optimum in expected polynomial time.

Theorem 7.6 bounds the expected runtime of MAHH$_{\text{OI}}$ on Cliff_d from above. We begin, however, by introducing a helper lemma, which was proved by Droste et al. (2001), for trajectory based algorithms which can only change the number of 1-bits in the bit-string by 1. The lemma was subsequently used to analyse the performance of the (1+1) EA for *noisy* OneMax for small noise strength (Droste, 2004). Unlike in noisy optimisation, where the noise represents uncertainty with the respect to the true fitness of solutions, in hyper-heuristics the AM operator is intended to be helpful to the optimisation process by allowing the algorithm to escape from local optima.

Lemma 7.5 (Droste et al., 2001, Lemma 3) *Let* $\text{E}(\text{T}_i^+)$ *be the expected time to reach a state with* $i+1$ *1-bits, given a state with* $i \in \{0,\dots,n-1\}$ *1-bits, and* p_i^+ *and* p_i^- *be the transition probabilities to reach a state with, respectively,* $i+1$ *and* $i-1$ *1-bits (assuming these are the only new states that can be reached). Then:*

$$\text{E}(\text{T}_i^+) = \frac{1}{p_i^+} + \frac{p_i^-}{p_i^+} \cdot \text{E}(\text{T}_{i-1}^+).$$

Within the context of non-elitist local search algorithms such as MAHH$_{\text{OI}}$, the transition probability p_i^+ (p_i^-) refers to the probability of making a local mutation which increases the fitness (respectively decreases) and accepting the new solution.

Theorem 7.6 *The expected runtime of* MAHH$_{\text{OI}}$ *on* CLIFF$_d$*, with* $p = \frac{1}{(1+\varepsilon)n}$ *for any constant* $\varepsilon > 0$*, is* $O\left(n\log n + \frac{n^3}{d^2}\right)$.

Proof: Let i denote the number of 1-bits in the bit-string at any time $t > 0$. We wish to bound the expected runtime from above by separately bounding four 'stages' of the optimisation process, each starting once all earlier stages have ended, and ending once a solution with at least certain number of 1-bits has been constructed for the first time during the optimisation process (i.e., the HH may go backwards afterwards yet will remain in the same stage): the first stage ends when $i \geq n-d$ is reached for the first time, the second when $i \geq n-d+1$, the third when $i \geq n-d+2$, and the fourth when $i = n$, i.e., the optimum has been constructed. We use $\text{T}_1, \dots, \text{T}_4$ to denote the number of iterations the algorithm spends in each of these stages, and note that by definition of these stages, the number of iterations T before the optimum is reached is $\text{T} = \text{T}_1 + \text{T}_2 + \text{T}_3 + \text{T}_4$ and, by the linearity of expectation,

$$\text{E}(\text{T}) = \text{E}(\text{T}_1) + \text{E}(\text{T}_2) + \text{E}(\text{T}_3) + \text{E}(\text{T}_4). \tag{7.3}$$

In the first stage, while $i < n-d$, CLIFF$_d$ resembles the ONEMAX function and we can use the upper bound for the expected runtime of MAHH$_{\text{OI}}$ on ONEMAX with $p = \frac{1}{(1+\varepsilon)n}$ from Theorem 7.3. Hence,

$$E(\text{T}_1) = O(n\log n).$$

The second stage begins with $i \geq n-d$ for the first time and ends once $i \geq n-d+1$ for the first time. When $i = n-d$, there are no improving moves. If the OI operator is selected, there will be no change in the candidate solution. Hence, any move must come from use of the AM operator, which is selected with probability $\frac{1}{(1+\varepsilon)n}$. A mutation step may either increase the number of 1-bits in the bit-string with probability d/n, or decrease the number of 1-bits with probability $\frac{n-d}{n}$. We use Lemma 7.5 to bound $\text{E}(\text{T}_2)$, the expected time to jump down from the cliff,

$$\text{E}(\text{T}_2) = \text{E}(\text{T}^+_{n-d}) = \frac{n^2(1+\varepsilon)}{d} + \frac{n-d}{d} \cdot \text{E}(\text{T}^+_{n-d-1}). \tag{7.4}$$

We now must bound $\text{E}(\text{T}^+_{n-d-1})$. At $i = n-d-1$, the drift is as follows:

$$\Delta(n-d-1) = \frac{d+1}{n} - \frac{1}{(1+\varepsilon)n} \cdot \frac{n-d-1}{n} = \frac{n(d+\varepsilon(d+1))+d+1}{n^2(1+\varepsilon)},$$

since flipping any one of the $(d+1)$ remaining 0-bits increases the fitness by 1 and is accepted by either operator and flipping any one of the $(n-d-1)$ 1-bits decreases the fitness by 1 and is only accepted if the AllMoves operator is chosen, which occurs with probability $p = \frac{1}{(1+\varepsilon)n}$.

Clearly, for all $0 \leq i \leq n-d-1$, the drift is bounded from above by the drift at $i = n-d-1$, while the distance to the required point from $i = n-d-1$ is 1. By the Additive Drift Theorem (Theorem 2.11), we have,

$$\mathrm{E}(\mathrm{T}^+_{n-d-1}) \leq \frac{1}{\Delta(n-d-1)} = \frac{n^2(1+\varepsilon)}{n(d+\varepsilon(d+1))+d+1} = O\left(\frac{n}{d}\right).$$

We can now return to bounding $\mathrm{E}(\mathrm{T}_2)$ in Equation 7.4:

$$\begin{aligned}\mathrm{E}(\mathrm{T}_2) &= \frac{n^2(1+\varepsilon)}{d} + \frac{n-d}{d} \cdot \mathrm{E}(\mathrm{T}^+_{n-d-1}) \\ &= \frac{n^2(1+\varepsilon)}{d} + \frac{n-d}{d} \cdot O\left(\frac{n}{d}\right) = O\left(\frac{n^2}{d}\right). \end{aligned} \tag{7.5}$$

The third stage begins with $i \geq n-d+1$ and ends once $i \geq n-d+2$. When $i = n-d+1$, all moves are improving moves and all moves will be accepted, regardless of the choice of the OI or AM operator. With probability $\frac{d-1}{n}$, the accepted move decreases the number of 1-bits to $i = n-d$ i.e., the hyper-heuristic returns to the local optimum. With probability $\frac{n-d+1}{n}$, the accepted move instead increases the number of 1-bits to $i = n-d+2$. We again use Lemma 7.5 to bound $\mathrm{E}(\mathrm{T}_3)$, the expected time to take one step up the second slope. Noting that $\mathrm{E}(\mathrm{T}^+_{n-d}) = O\left(\frac{n^2}{d}\right)$ by Equation 7.5, we get

$$\begin{aligned}\mathrm{E}(\mathrm{T}_3) = \mathrm{E}(\mathrm{T}^+_{n-d+1}) &= \frac{n}{d-1} + \frac{n-d+1}{d-1} \cdot \mathrm{E}(\mathrm{T}^+_{n-d}) \\ &= \frac{n}{d-1} + \frac{n-d+1}{d-1} \cdot O\left(\frac{n^2}{d}\right) = O\left(\frac{n^3}{d^2}\right).\end{aligned}$$

If $d = 2$, the $\textsc{Cliff}_d$ function will have been optimised at this point. However, for $d \geq 3$, it is necessary to climb further up the second slope. If $d = 3$, we point out that $\mathrm{E}(\mathrm{T}_4) = \mathrm{E}(\mathrm{T}^+_{n-d+2})$, and by applying Lemma 7.5 bound

$$\begin{aligned}\mathrm{E}(\mathrm{T}^+_{n-d+2}) &= \frac{n}{d-2} + \frac{1}{(1+\varepsilon)n} \cdot \frac{n-d+2}{d-2} \cdot \mathrm{E}(\mathrm{T}^+_{n-d+1}) \\ &= \frac{n}{d-2} + \frac{1}{(1+\varepsilon)n} \cdot \frac{n-d+2}{d-2} \cdot O\left(\frac{n^3}{d^2}\right) = \frac{n}{d-2} + O\left(\frac{n^3}{d^3}\right).\end{aligned}$$

For $d > 3$, we know that $i = n-d+2$ and $i = n-d+3$ are points on the second slope and by applying Lemma 7.5 bound

$$\begin{aligned}\mathrm{E}(\mathrm{T}^+_{n-d+3}) &= \frac{n}{d-3} + \frac{1}{(1+\varepsilon)n} \cdot \frac{n-d+3}{d-3} \cdot \mathrm{E}(\mathrm{T}^+_{n-d+2}) \\ &= \frac{n}{d-3} + \frac{1}{(1+\varepsilon)n} \cdot \frac{n-d+3}{d-3} \cdot O\left(\frac{n^3}{d^3}\right) = \frac{n}{d-3} + O\left(\frac{n^3}{d^4}\right).\end{aligned}$$

For $d > 4$ this trend will continue as $\mathrm{MAHH_{OI}}$ progresses closer towards the global optimum; in particular, for $k < d$, we will have

$$\mathrm{E}(\mathrm{T}^+_{n-d+k}) = \frac{n}{d-k} + O\left(\frac{n^3}{d^k}\right).$$

Hence, $\mathrm{E}(\mathrm{T}_4) = \sum_{k=2}^{d-1} \mathrm{E}(\mathrm{T}^+_{n-d+k})$. The terms in the summation will be asymptotically dominated by the $O\left(\frac{n^3}{d^3}\right)$ term in $\mathrm{E}(\mathrm{T}^+_{n-d+2})$ if d is sub-linear, giving $\mathrm{E}(\mathrm{T}_4 \mid d = o(n)) \le d \cdot O\left(\frac{n^3}{d^3}\right) = O\left(\frac{n^3}{d^2}\right)$. However, if d is linear in the problem size, the first terms (i.e., $\frac{n}{d-k}$) will dominate and we will have:

$$\begin{aligned}\mathrm{E}(\mathrm{T}_4 \mid d = \Theta(n)) &= O(1) + \sum_{i=2}^{d-1} \frac{n}{d-i} \le O(1) + \sum_{i=0}^{d-1} \frac{n}{d-i} \\ &= O(1) + n \cdot \sum_{j=1}^{d} \frac{1}{j} = O(n \log n).\end{aligned}$$

Combining the bounds for the sub-linear and linear d, we conclude by the law of total expectation that

$$\mathrm{E}(\mathrm{T}_4) = O\left(n \log n + \frac{n^3}{d^2}\right).$$

We now return to our overall runtime bound from Equation 7.3 and complete the proof:

$$\begin{aligned}\mathrm{E}(\mathrm{T}) &= \mathrm{E}(\mathrm{T}_1) + \mathrm{E}(\mathrm{T}_2) + \mathrm{E}(\mathrm{T}_3) + \mathrm{E}(\mathrm{T}_4) \\ &= O\left(n \log n + \frac{n^2}{d} + \frac{n^3}{d^2} + n \log n + \frac{n^3}{d^2}\right) = O\left(n \log n + \frac{n^3}{d^2}\right).\end{aligned}$$

□

Theorem 7.6 gives an expected runtime bound of $\mathrm{MAHH_{OI}}$ on $\textsc{Cliff}_d$ of $O\left(n \log n + \frac{n^3}{d^2}\right)$. This bound is smallest when d is large, suggesting that the algorithm is fastest when the cliff is hardest for elitist algorithms, i.e., a linear cliff length, $d = \Theta(n)$, gives an expected runtime of $O(n \log n)$. This runtime asymptotically matches the best case expected performance of an artificial immune system, which escapes the local optimum with an ageing operator (i.e., also $O(n \log n)$ (Corus et al., 2017)), which is the best expected runtime known for hard CLIFF$_d$ functions (for problem-independent search heuristics). We suspect MAHH$_{\mathrm{OI}}$ is faster in practice (by considering the leading constants in the runtime), but leave this proof for future work. Mutation based EAs have an expected runtime of $\Theta(n^d)$ for $d \leq \frac{n}{2}$ (Paixão et al., 2017); for $d = \Theta(n)$, this will give exponential runtime in the length of the cliff. Steady-State Genetic Algorithms which use crossover have recently been proven to be faster by at least a linear factor (Dang et al., 2017), but would still require exponential expected runtimes for large cliff lengths.

The worst-case scenario for MAHH$_{\mathrm{OI}}$ is a constant cliff length ($d = \Theta(1)$), giving an expected runtime of $O(n^3)$. This means that the (1+1) EA has an expected runtime lower than the upper bound for MAHH$_{\mathrm{OI}}$ if $d < 3$, but will be slower in expectation for any $3 < d \leq \frac{n}{2}$, with a performance gap that increases exponentially with the length of the cliff. Only an upper bound on the expected runtime of the non-elitist (1,λ) EA is available in the literature; $O(n^{25})$ if λ is not too large or too small (Jägersküpper and Storch, 2007).

We can also compare the performance of MAHH$_{\mathrm{OI}}$ on CLIFF$_d$ with a well-studied non-elitist algorithm. The METROPOLIS algorithm (Algorithm 8) accepts worsening moves with probability $\alpha(n)^{f(y)-f(x)}$, for some $\alpha(n) \geq 1$, and accepts all improvements (Metropolis et al., 1953). Theorem 7.7 shows that MAHH$_{\mathrm{OI}}$ will considerably outperform METROPOLIS on CLIFF$_d$ for all $d > 3$.

Theorem 7.7 *The expected runtime of* METROPOLIS *on* CLIFF$_d$ *is at least*

$$\min\left\{\frac{1}{2}\cdot\frac{n-d+1}{d-1}\cdot\left(\frac{Cn}{\log n}\right)^{d-\frac{3}{2}}, n^{\omega(1)}\right\}$$

for some $C = \Omega(1)$.

Proof: Let i denote the number of 1-bits in the bit-string at any time $t > 0$. The CLIFF$_d$ function has two ONEMAX slopes ($0 \leq i \leq n-d$, $n-d+1 \leq i \leq n-1$). In order to reach the global optimum, it is necessary for METROPOLIS to optimise these two slopes. The runtime for METROPOLIS on ONEMAX is polynomially bounded in the problem size if and only if $\alpha(n) = \Omega\left(\frac{n}{\log n}\right)$ (Jansen and Wegener, 2007, Theorem 7). Hence, $\alpha(n) = \Omega\left(\frac{n}{\log n}\right)$ is necessary in order to optimise the two slopes in polynomial time. However, the jump down to the bottom of the cliff will be difficult for METROPOLIS.

The randomly initialised candidate solution has at most $\frac{n}{2}$ bits with probability at least $\frac{1}{2}$. METROPOLIS accepts the jump down from $i = n-d$ to $i = n-d+1$ with probability at most $\alpha(n)^{(n-d+3/2)-(n-d)} = \alpha(n)^{\frac{3}{2}-d}$. Hence, the expected time for this event to occur is at least $\mathrm{E}(\mathrm{T}^{+}_{n-d}) \geq \alpha(n)^{d-\frac{3}{2}}$. From this state, it is necessary to make at least one improving move to find the global optimum. Given that, at a state with $i = n-d+1$ 1-bits, all moves are accepted, we use Lemma 7.5, with $p^{+}_{n-d+1} = \frac{d-1}{n}$ and $p^{-}_{n-d+1} = \frac{n-d+1}{n}$:

$$\begin{aligned}\mathrm{E}(\mathrm{T}^{+}_{n-d+1}) = \frac{n}{d-1} + \frac{n-d+1}{d-1} \cdot \mathrm{E}(\mathrm{T}^{+}_{n-d}) &\geq \frac{n}{d-1} + \frac{n-d+1}{d-1} \cdot \alpha(n)^{d-\frac{3}{2}} \\ &\geq \left(\frac{n-d+1}{d-1} \cdot \alpha(n)^{d-\frac{3}{2}}\right).\end{aligned}$$

Given that we want to minimise the overall runtime, we have $\alpha(n) = \Omega\left(\frac{n}{\log n}\right)$ such that the hillclimb can be done in polynomial time. However, the jump down the cliff will take in expectation

$$\mathrm{E}(\mathrm{T}^{+}_{n-d+1}) \geq \frac{n-d+1}{d-1} \cdot \left(C \cdot \frac{n}{\log n}\right)^{d-\frac{3}{2}}$$

for some $C = \Omega(1)$.

We note that if $d = \omega(1)$, the term $\mathrm{E}(\mathrm{T}^{+}_{n-d+1})$ is exponential in the problem size, and setting $\alpha = \Omega\left(\frac{n}{\log n}\right)$ might not be optimal. In this case, the lower bound will be some superpolynomial term, $n^{\omega(1)}$, i.e., the time taken for the two hillclimbs.

Overall, in order to optimise the function, the jump down the cliff has to be made (with probability at least $\frac{1}{2}$) and at least one step up the second slope must be made. Hence,

$$\mathrm{E}(\mathrm{T}) \geq \min\left\{\frac{1}{2} \cdot \frac{n-d+1}{d-1} \cdot \left(\frac{Cn}{\log n}\right)^{d-\frac{3}{2}}, n^{\omega(1)}\right\}.$$

$\square$

We have proven that while METROPOLIS cannot optimise hard CLIFF$_d$ variants in expected polynomial time, MAHH$_{\mathrm{OI}}$ is extremely efficient for hard cliffs, and has an expected runtime of $O(n^3)$ in the worst case. In the next section, we introduce the benchmark function class JUMP$_m$ where MAHH$_{\mathrm{OI}}$ is inefficient and compare its expected runtime with that of METROPOLIS again.

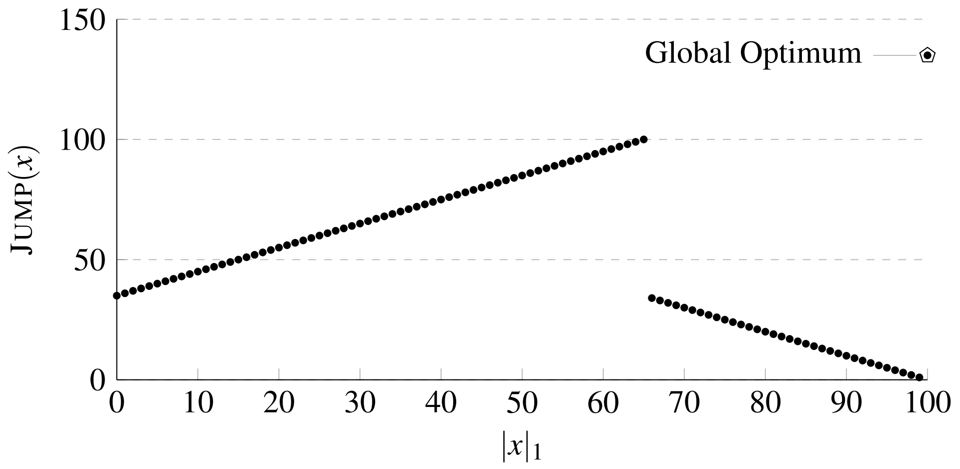

Fig. 7.2 An example of $\text{JUMP}_m(x)$ with $n = 100$ and $m = 35$.

7.4 When the Hyper-heuristic is Inefficient

In this section we consider the JUMP_m function class ($1 < m < \frac{n}{2}$) as an example of a multimodal function where MAHH_{OI} has a harder time at escaping the local optima. Unlike CLIFF_d, the fitness decreases on the second slope, making it harder to traverse (see Figure 7.2).

$$\text{JUMP}_m(x) := \begin{cases} m + \text{ONEMAX}(x) & |x|_1 \le n - m \text{ or } |x|_1 = n, \\ n - \text{ONEMAX}(x) & \text{otherwise.} \end{cases}$$

In order to optimise the JUMP_m function (Figure 2.6), it is first necessary to reach the local optimum at the top of the slope. Then, either a jump to the global optimum is made, which requires exponential expected time in the length of the jump for unbiased mutation operators, or a jump is made down to the slope of decreasing fitness, which must be traversed before the global optimum is found.

Theorem 7.8 *The runtime of* MAHH_{OI} *on* JUMP_m*, with* $p = \frac{1}{(1+\varepsilon)n}$*, is at least* $\Omega(n \log n + 2^{cm})$*, for some constant* $c > 0$ *and any constant* $\varepsilon > 0$*, with probability at least* $1 - 2^{-\Omega(m)}$.

Proof: Let i represent the number of 1-bits in the bit-string at any time $t > 0$. First, we consider $m = O(\log n)$. The initialised candidate solution has at most $\frac{n}{2}$ bits with probability at least $\frac{1}{2}$. The expected time to optimise the function will be bounded from below by the expected time to reach the local optimum at the top of the slope. If $m = O(\log n)$, the algorithm must at least hillclimb to some point $i = n - k \ln n$, for some constant $k > 0$. The expected time to flip $\frac{n}{2} - k\ln n$ bits correctly is $\Omega(n\log n)$. This can be proved by reusing arguments from the proof of (Droste et al., 2002, Lemma 10).

Similarly, the time to optimise the function will be bounded from below by the time to traverse the slope of decreasing fitness and find the global optimum, that is, to traverse the region where $n-m+1 < i \leq n-1$. Consider the negative drift on the number of 1-bits in the bit-string within this region:

$$\begin{aligned}-\Delta(i) &= \frac{i}{n} - \frac{1}{(1+\varepsilon)n}\cdot\frac{n-i}{n} = \frac{i(n(1+\varepsilon)+1)-n}{n^2(1+\varepsilon)} \geq \frac{\frac{n}{2}(n(1+\varepsilon)+1)-n}{n^2(1+\varepsilon)}\\ &= \frac{n(1+\varepsilon)-1}{2n(1+\varepsilon)} = \frac{1}{2} - \frac{1}{2n(1+\varepsilon)} \geq 0.4.\end{aligned}$$

We also note that the number of 1-bits can only change by 1 in each iteration, so there is no chance of escaping the area of negative drift with large jumps. We can apply the Negative Drift Theorem (Theorem 2.20) for the region from $i = n-m+1$ to $i = n-1$, i.e. of length $m-2$. Hence there exists a constant $c' > 0$ such that, with probability at least $1-2^{-\Omega(m-2)} = 1-2^{-\Omega(m)}$, the global optimum is not found from the area of negative drift within $2^{c'\cdot(m-2)} = 2^{cm}$ steps, where $c = c'\cdot\frac{m-4}{m} > 0$ is a different constant. If $m = \omega(\log n)$, this term will dominate.

By considering both possibilities, we state that, for some constant $c > 0$, the expected time for MAHH$_{\text{OI}}$ to optimise JUMP$_m$ is, with probability at least $1-2^{-\Omega(m)}$, E(T) $= \Omega(n\log n + 2^{cm})$. □

We can also bound the expected time for MAHH$_{\text{OI}}$ to optimise the JUMP$_m$ function from above. The following theorem provides a trivial upper bound on the expected runtime by considering the expected time for the MAHH$_{\text{OI}}$ to accept m consecutive fitness decreasing moves from the local to the global optimum.

Theorem 7.9 *The expected runtime of* MAHH$_{\text{OI}}$ *on* JUMP$_m$*, with* $p = \frac{1}{(1+\varepsilon)n}$*, for any constant* $\varepsilon > 0$*, is*

$$O\left(n\log n + \frac{n^{2m-1}}{m}\right).$$

Proof: Let i denote the number of 1-bits in the bit-string at any time $t > 0$. The expected time to reach a bit-string with $i = n-m$ 1-bits is $O(n\log n)$, by Corollary 7.4.

The probability that the number of 1-bits is increased while $n-m \leq i \leq n-1$ is at least $\frac{1}{n}$, and such a move would be accepted with probability $p = \frac{1}{(1+\varepsilon)n}$. Hence, the probability of accepting a move which increases the number of 1-bits is at least $\frac{1}{n}\cdot\frac{1}{(1+\varepsilon)n} = \frac{1}{(1+\varepsilon)n^2}$. The probability of accepting $m-1$ decreasing moves from a state with $i = n-m$ 1-bits (that is, to find the global optimum) is at least $\left(\frac{1}{(1+\varepsilon)n^2}\right)^{m-1}$.

Let $p^+_{n-m} \geq \left(\frac{1}{(1+\varepsilon)n^2}\right)^{m-1}$ be the probability that the global optimum is reached in $m-1$ consecutive steps from the local optimum at $i = n-m$, and p^-_{n-m} be the probability that a mutation which decreases the fitness to $i = n-m-1$ is accepted. By noting that $\mathrm{E}(\mathrm{T}^+_{n-m-1}) = O\left(\frac{n}{m}\right)$ (by the Additive Drift Theorem (Theorem 2.11)), we can use Lemma 7.5 to bound the expected time to find the global optimum from the local optimum:

$$\begin{aligned}\mathrm{E}(\mathrm{T}^{opt}_{n-m}) &\leq ((1+\varepsilon)n^2)^{m-1} + ((1+\varepsilon)n^2)^{m-1} \cdot O\left(\frac{n}{m}\right)\\ &= O\left(\frac{n^{2m-1}}{m}\right).\end{aligned}$$

Hence, the expected time to reach the local optimum and traverse the slope of decreasing fitness to find the global optimum, is

$$\mathrm{E}(\mathrm{T}) = O\left(n\log n + \frac{n^{2m-1}}{m}\right).$$

□

The expected runtime of the (1+1) EA on JUMP$_m$ is $\Theta(n^m)$ (Droste et al., 2002), and a bound on Steady State (μ+1) Genetic Algorithms of $O(n^{m-1})$ has recently been proved by Dang et al. (2017), both outperforming our upper bound for MAHH$_{\mathrm{OI}}$.

We will now show that METROPOLIS also has poor performance on JUMP$_m$. The large fitness difference between the two points where $|x|_1 = n-m$ and $|x|_1 = n-m+1$ makes it hard for this algorithm to jump down from the local optimum and hence, it will require a long time. Note that the following theorem holds for all values of $m < \frac{n}{2}$ and all functions $\alpha(n)$.

Theorem 7.10 *The runtime of* METROPOLIS *on* JUMP$_m(x)$ *is* $2^{\Omega(n)}$*, with probability at least* $1-2^{-\Omega(n)}$.

Proof: Let i represent the number of 1-bits in the bit-string at any time $t > 0$. The initialised candidate solution has at most $\frac{n}{2}$ 1-bits with probability at least $\frac{1}{2}$. The jump down from the local optimum will be accepted with probability $\alpha(n)^{f(n-m+1)-f(n-m)} = \alpha(n)^{m-1-n}$, and will take expected time at least $\mathrm{E}(\mathrm{T}^+_{n-m}) \geq \alpha(n)^{n+1-m}$.

For $i > n-m$, we can again use Lemma 7.5, with $p_i^+ = \alpha(n)^{-1} \cdot \frac{n-i}{n}$ and $p_i^- = \frac{i}{n}$, to bound the expected time to transition between neighbouring states:

$$\begin{aligned}
\mathrm{E}(\mathrm{T}^+_{n-m+1}) &= \alpha(n) \cdot \frac{n}{m-1} + \alpha(n) \cdot \frac{n-m+1}{m-1} \cdot \mathrm{E}(\mathrm{T}^+_{n-m}) \\
&\geq \alpha(n) \cdot \frac{n}{m-1} + \alpha(n) \cdot \frac{n-m+1}{m-1} \cdot \alpha(n)^{n+1-m} \\
&\geq \alpha(n)^{n+2-m} \\
\mathrm{E}(\mathrm{T}^+_{n-m+2}) &= \alpha(n) \left(\frac{n}{m-2} + \frac{n-m+2}{m-2} \cdot \mathrm{E}(\mathrm{T}^+_{n-m+1}) \right) \\
&\geq \alpha(n) \cdot \frac{n}{m-2} + \alpha(n) \cdot \frac{n-m+2}{m-2} \cdot \alpha(n)^{n+2-m} \\
&\geq \alpha(n)^{n+3-m}.
\end{aligned}$$

We note that, in general, $\mathrm{E}(\mathrm{T}^+_{n-m+k}) \geq \alpha(n)^{n+(k+1)-m}$, and hence

$$\mathrm{E}(\mathrm{T}) \geq \mathrm{E}(\mathrm{T}^+_{n-2}) \geq \alpha(n)^{n+(m-2+1)-m} = \alpha(n)^{n-1}.$$

We now require a lower bound on $\alpha(n)$. If $m > \frac{n}{5}$, consider the negative drift in the number of 1-bits in the region from $i = n-m+1$ to $i = n-1$. Clearly, $\alpha(n) = 1$ is the best choice for this region, yet will still lead to a large area of negative drift:

$$-\Delta(i) = \frac{i}{n} - \frac{n-i}{n} \geq -\Delta\left(\frac{4n}{5}\right) = \frac{3}{5}.$$

Since METROPOLIS is a local search algorithm, this region of negative drift cannot be escaped by a large jump. Hence, by the Negative Drift Theorem (Theorem 2.20), the time to escape this region will be at least $2^{\Omega(n)}$ with overwhelming probability $1 - 2^{-\Omega(n)}$. This will also be a lower bound on the time to optimise the function.

If $m \leq \frac{n}{5}$, with probability at least $\frac{1}{2}$, METROPOLIS must hillclimb from $i = \frac{3n}{4}$ to $i = \frac{4n}{5}$. Consider the drift on the fitness in this region when $\alpha(n) = 2$:

$$-\Delta(i) = \frac{1}{2} \cdot \frac{i}{n} - \frac{n-i}{n} \geq -\Delta\left(\frac{3n}{4}\right) = \frac{1}{8}.$$

Clearly, to avoid a large region of negative drift, $\alpha(n) \geq 2$ is required, giving $\mathrm{E}(\mathrm{T}) \geq 2^{n-1} = 2^{\Omega(n)}$. □

Although JUMP$_m$ is an example of a benchmark function where MAHH$_{\mathrm{OI}}$ does not perform well, with a runtime that is at least exponential in the size of the jump with overwhelming probability, the time for METROPOLIS to optimise the function is exponential in all cases, with overwhelming probability. For small jumps $m = \Theta(1)$, MAHH$_{\text{OI}}$ will by far outperform METROPOLIS, since it will have expected polynomial runtime.

7.5 Chapter Summary and Discussion

This chapter presents the first theoretical analyses of selection-based hyper-heuristic for multimodal optimisation. We have analysed the performance of the Move Acceptance Hyper-heuristic (MAHH$_{\text{OI}}$) which, at each step, applies the ONLYIMPROVING (OI) acceptance operator with probability $1-p$ and the ALLMOVES (AM) acceptance operator with probability p.

We first identified a range of parameter values (i.e., $p < 1/((1+\varepsilon)n)$, for any constant $\varepsilon > 0$) that allow to hillclimb the ONEMAX function in expected $O(n \log n)$ runtime as desired.

Afterwards, we showed characteristics of multimodal optimisation problems where the hyper-heuristic is effective and where it is not. In particular, MAHH$_{\text{OI}}$ performs well on multimodal optimisation landscapes where the candidate solution identifies a gradient of increasing fitness after leaving a local optimum. The CLIFF$_d$ function class was analysed to provide such an example, where MAHH$_{\text{OI}}$ exhibits very good performance, even matching the best runtime known for problem-independent search heuristics, for instances that are prohibitive for elitist Evolutionary Algorithms and METROPOLIS.

On the other hand, for multimodal functions with long slopes of decreasing fitness that have to be traversed, MAHH$_{\text{OI}}$ does not perform as well. In particular, we analysed the JUMP$_m$ function class to provide such an example, proving that MAHH$_{\text{OI}}$ has a runtime that is exponential in the size of the 'jump', with overwhelming probability. However, also the established non-elitist METROPOLIS algorithm was shown to be inefficient on this benchmark function, with an even worse runtime (i.e., at least exponential in the problem size, with overwhelming probability). This is due to the large difference in fitness between two points that the METROPOLIS algorithm must traverse.

All the statements given in the chapter for MAHH$_{\text{OI}}$ also hold for MAHH$_{\text{IE}}$, a variant of MAHH$_{\text{OI}}$ which chooses the IMPROVINGANDEQUAL acceptance operator with probability $1-p$ and the ALLMOVES acceptance operator with probability p.

Chapter 7 is based on the following publication:

1. Lissovoi, A., Oliveto, P. S., and Warwicker, J. A. (2018). On the Time Complexity of Algorithm Selection Hyper-heuristics for Multimodal Optimisation. AAAI ‘19 (to appear).

Chapter 8

Hyper-heuristics with Larger Choices of Low-level Heuristics can be Faster

In this chapter, we consider the previously analysed generalised mutation-based hyper-heuristics introduced in Chapter 4 (i.e., Generalised Greedy, Generalised Random Gradient and Generalised Greedy Gradient) using a more realistic set of low-level heuristics. Rather than selecting from two different heuristics, we increase the size of the set of low-level heuristics H such that it may contain an arbitrary number of heuristics to more accurately reflect the use of hyper-heuristics in the literature. This is the first time hyper-heuristics with $|H| > 2$ have been theoretically analysed. All the theoretical results in this chapter will hold for hyper-heuristics allowed to choose between $k = \Theta(1)$ (i.e., any constant number of heuristics) operators.

Our main result is to show that increasing the number of low-level heuristics may lead to improved performance. In particular, we prove that the Generalised Random Gradient hyper-heuristic using $k = \Theta(1)$ mutation operators as low-level heuristics outperforms, in expectation, any algorithm using $m < k$ mutation operators on the LEADINGONES benchmark function. Thus, an example is provided where the inclusion of more heuristics is provably beneficial. Furthermore, we show that with appropriate lengths of the learning period, the hyper-heuristic runs in the best possible expected time achievable with the k low-level heuristics. On the other hand, if no learning period is used, then increasing the number of low-level heuristics is detrimental. We conclude the chapter with an experimental analysis of the performance of Generalised Random Gradient, Generalised Greedy and Generalised Greedy Gradient when using larger sets of low-level heuristics, for different problem sizes (up to $n = 10^8$). These experiments confirm that for Generalised Random Gradient, including more operators is beneficial. The same result holds for Generalised Greedy Gradient. However, for

Generalised Greedy, the inclusion of more operators is detrimental to its performance. Yet, it is still able to outperform RLS.

8.1 The Best-Possible Runtime Achievable on LEADINGONES

Rigorous runtime analyses of hyper-heuristics are easier if the algorithms can only choose between two operators. However, in realistic contexts, hyper-heuristics have to choose between many more operators. For example, Cowling et al. (2001) used hyper-heuristics selecting from a set of ten low-level heuristics and Cowling et al. (2002b) presented a hyper-heuristic selecting from a set of eight heuristics.

In this section, we derive the best-possible expected runtime achievable by any hyper-heuristic using $k = \Theta(1)$ stochastic mutation operators on LEADINGONES. Before we prove this result we introduce the following two helpful lemmata.

Lemma 8.1 *The probability of improvement* mBITFLIP *(for some constant* $m > 0$*) on* LEADINGONES*, at the state* $\mathrm{LO}(x) = i$*, is*

$$\Pr(\mathrm{IMP}_m) = m \cdot \frac{1}{n} \cdot \left(\frac{n-i-1}{n}\right)^{m-1} + O(n^{-2}).$$

Proof: In the context of the LEADINGONES function, the way for the mBITFLIP operator to make an improvement is to flip the first 0-bit after the leading 1-bits, whilst keeping the leading 1-bits in the prefix unchanged. This occurs with probability $m \cdot \frac{1}{n} \cdot \left(\frac{n-i-1}{n}\right)^{m-1}$.

It is also possible to make fitness improvements while one or more bits of the prefix (i.e., within the first $i+1$ bits) are flipped, each an even number of times. This can occur by flipping and reflipping between 1 and $\lfloor \frac{m-1}{2} \rfloor$ bits. Firstly, the first 0-bit must be flipped into a 1-bit, which occurs with probability at most $\frac{m}{n}$. Then, an even number of the remaining bitflips can be used to flip bits in the prefix, which need to be reflipped to maintain the prefix. The remaining flips will occur within the suffix $n-i-1$ bits and these events can occur in any order. Hence, the probability of producing an improvement while flipping and reflipping an even number of bits (> 0) in the prefix is at most:

$$\frac{m}{n} \cdot \sum_{j=1}^{\lfloor \frac{m-1}{2} \rfloor} \left[\binom{m-1}{2j} \cdot \frac{1}{i+1} \cdot \left(\frac{i+1}{n-1}\right)^{2j} \cdot \left(\frac{n-i-1}{n-1}\right)^{m-2j-1} \right]$$
$$= \frac{m}{n} \cdot \binom{m-1}{2} \cdot \frac{1}{i+1} \cdot \left(\frac{i+1}{n-1}\right)^{2} \cdot \left(\frac{n-i-1}{n-1}\right)^{m-3} + O\left(\frac{1}{n^2}\right) = O\left(\frac{1}{n^2}\right),$$

given $m = \Theta(1)$. Hence, the probability of improvement of the mBITFLIP operator for LEADINGONES is

$$\Pr(\text{IMP}_m) = m \cdot \frac{1}{n} \cdot \left(\frac{n-i-1}{n}\right)^{m-1} + O\left(\frac{1}{n^2}\right).$$

□

We note that through a simple calculation, the 1BITFLIP operator is always the best choice in the second half of the search space ($\frac{n-1}{2} \le i < n$). We can thus leave the formula in this state as the $\frac{1}{n}$ success probability of the 1BITFLIP operator will always dominate any $O(n^{-2})$ terms, and the $\Pr(\text{IMP}_m)$ terms are all $O\left(\frac{1}{n}\right)$ in the first half of the search space.

Lemma 8.2 *Given a bit-string x with $\text{LO}(x) = i$, consider two mutation operators flipping $a = \Theta(1)$ and $b = \Theta(1)$ bits with replacement respectively (a*BITFLIP *and b*BITFLIP*), with $b > a$. Then (excluding $i = n-1$) b*BITFLIP *has a higher probability of success than a*BITFLIP *(i.e., $\Pr(\text{IMP}_b) > \Pr(\text{IMP}_a)$) when*

$$i < n\left(1 - \left(\frac{b}{a}\right)^{\frac{1}{a-b}}\right) - 1.$$

This follows from Lemma 8.1. In particular, an operator flipping m bits outperforms in expectation an operator flipping $m-1$ bits when $i < \frac{n}{m} - 1$. It is worth noting that only an operator which flips an even number of bits can make progress when $\text{LO}(x) = n-1$, and the 1BITFLIP operator has the best success probability at this point ($\frac{1}{n}$). We note that for the LEADINGONES benchmark function, maximising the probability of success implies minimising the expected waiting times for an improvement since each successful move increases the fitness in expectation by the same amount.

We now use Lemmata 8.1 and 8.2 to present the following statement on the best-possible expected runtime achievable for any algorithm using k stochastic mutation operators. Similar results have been presented by Doerr and Wagner (2018) and Doerr (2018a) for mutation operators flipping bits without replacement (note that the results are equivalent up to lower order terms).

Theorem 8.3 *The best-possible expected runtime on* LEADINGONES *for any mechanism using $(1,\ldots,k)$*BITFLIP*, where $k = \Theta(1)$, is*

$$\text{E}(\text{T}_{k,\text{opt}}) = \frac{1}{2}\left(\sum_{i=0}^{n/k-1} \frac{1}{k \cdot \frac{1}{n} \cdot \left(\frac{n-i-1}{n}\right)^{k-1}} + \sum_{m=1}^{k-1} \sum_{i=n/(m+1)}^{n/m-1} \frac{1}{m \cdot \frac{1}{n} \cdot \left(\frac{n-i-1}{n}\right)^{m-1}}\right) \pm o(n^2).$$

Proof: The optimal algorithm uses the mutation operator with the highest probability of success based on the current LO value at each $\mathrm{LO}(x) = i$. We use Theorem 2.13 to give the theorem statement, where the A_i values are the expected time for an improvement from the state $\mathrm{LO}(x) = i$ using the operator with the highest success probability.

From Lemma 8.2, we can say the mBITFLIP operator is optimal during the time when $\frac{n}{m+1} \leq i \leq \frac{n}{m} - 1$ (unless $m = k$, in which case it is optimal for $0 \leq i \leq \frac{n}{k} - 1$). Since the 1BITFLIP operator has success probability $O(n^{-1})$, and is chosen with the same probability as all other operators, the leading term in any success probability is $O((nk)^{-1}) = \omega(n^{-2})$ and any $O(n^{-2})$ terms are insignificant. Thus, they can be grouped into a lower order $o(n^2)$ term.

Hence, by standard waiting time arguments, coupled with Theorem 2.13, we have that the optimal expected runtime when using k operators is

$$\mathrm{E}(\mathrm{T}_{k,\mathrm{opt}}) = \frac{1}{2}\left(\sum_{i=0}^{n/k-1} \frac{1}{k \cdot \frac{1}{n} \cdot \left(\frac{n-i-1}{n}\right)^{k-1}} + \sum_{m=1}^{k-1} \sum_{i=n/(m+1)}^{n/m-1} \frac{1}{m \cdot \frac{1}{n} \cdot \left(\frac{n-i-1}{n}\right)^{m-1}}\right) \pm o(n^2).$$

□

In particular, taking limits as $n \to \infty$, we have $\mathrm{E}(\mathrm{T}_{1,\mathrm{opt}}) = \frac{1}{2}n^2$, $\mathrm{E}(\mathrm{T}_{2,\mathrm{opt}}) = \frac{1+\ln(2)}{4}n^2 \approx 0.423n^2$, $\mathrm{E}(\mathrm{T}_{3,\mathrm{opt}}) = \left(\frac{1}{3} + \frac{\ln(2)}{2} - \frac{\ln(3)}{4}\right)n^2 \approx 0.405n^2$, $\mathrm{E}(\mathrm{T}_{5,\mathrm{opt}}) = \left(\frac{3721}{11520} + \frac{\ln(2)}{2} - \frac{\ln(3)}{4}\right)n^2 \approx 0.395n^2$. A closed form result for $E(T_{k,\mathrm{opt}})$ is difficult to find as is the limit for the best-possible expected runtime as $k \to \infty$. A numerical analysis by Doerr and Wagner (2018) suggests an expected runtime of $E(T_{\infty,\mathrm{opt}}) \approx 0.388n^2 \pm o(n^2)$.

This result gives a theoretical lower bound on the expected runtime of any algorithm using the k operators on LEADINGONES. We see that including further operators can improve the expected runtime if they are used appropriately. In the following section, we will analyse the performance of the previously considered 'simple' learning mechanisms.

8.2 Simple Learning Mechanisms are Slower

In this section, we will show that incorporating more operators is detrimental to the performance of the 'simple' mechanisms on LEADINGONES. Like in Chapter 5, we start by stating the expected runtime of the Simple Random mechanism and use this as a basis to prove poor performance also for the other three mechanisms. Recall that the standard Simple Random mechanism chooses each operator uniformly at random in each iteration (i.e., with probability $\frac{1}{k}$ when using k operators).

Theorem 8.4 *The expected runtime of the Simple Random mechanism on* LEADINGONES *using* $(1,\dots,k)$BITFLIP, *with* $k = \Theta(1)$, *is*

$$\mathrm{E}(\mathrm{T}_{SR_k}) = \frac{k}{2} \cdot \sum_{i=0}^{n-1} \frac{1}{\sum_{m=1}^{k} m \cdot \frac{1}{n} \cdot \left(\frac{n-i-1}{n}\right)^{m-1}} - o(n^2).$$

Note that as a result of Theorem 8.4, the expected runtime of the Simple Random mechanism increases as k increases, implying that incorporating more operators is detrimental to the performance of the 'simple' mechanisms. In particular, the expected runtimes when using 1, 2 and 3 operators respectively are $\frac{1}{2}n^2 = 0.5n^2$, $\frac{\ln(3)}{2}n^2 \approx 0.549n^2$ and $\left(\frac{3\sqrt{2}}{4}\arctan\left(\frac{\sqrt{2}}{2}\right)\right)n^2 \approx 0.653n^2$.

Proof: [Of Theorem 8.4]

We have seen previously in Lemma 8.1 that the probability of an improvement when flipping m indistinct bits is

$$\Pr(\textsc{imp}_m) = m \cdot \frac{1}{n} \cdot \left(\frac{n-i-1}{n}\right)^{m-1} + O(n^{-2}).$$

Since we have k operators, the probability of an improvement in a single iteration is

$$\begin{aligned}
&\frac{1}{k} \cdot \sum_{m=1}^{k} \left(m \cdot \frac{1}{n} \cdot \left(\frac{n-i-1}{n}\right)^{m-1} + O(n^{-2}) \right) \\
&= \frac{1}{kn} + \frac{1}{k} \cdot \sum_{m=2}^{k} \left(m \cdot \frac{1}{n} \cdot \left(\frac{n-i-1}{n}\right)^{m-1} + O(n^{-2}) \right) = \Omega\left(\frac{1}{n}\right)
\end{aligned}$$

and hence, using the same nomenclature as in Theorem 2.13 (which we can use since the mutation operators fit in to the unbiased (1+1) black box algorithm setting), the expected time for an improvement given a distance of $n-i$ to the global optimum, is

$$A_{n-i} = \frac{1}{\frac{1}{k} \cdot \sum_{m=1}^{k} m \cdot \frac{1}{n} \cdot \left(\frac{n-i-1}{n}\right)^{m-1}} - o(n).$$

We can use Theorem 2.13 to give the result:

$$\mathrm{E}(\mathrm{T}_{\mathrm{SR}_k}) = \frac{1}{2}\sum_{i=0}^{n-1} A_{n-i} = \frac{1}{2} \cdot \sum_{i=0}^{n-1} \frac{1}{\frac{1}{k} \cdot \sum_{m=1}^{k} m \cdot \frac{1}{n} \cdot \left(\frac{n-i-1}{n}\right)^{m-1}} - o(n^2).$$

$$= \frac{k}{2} \cdot \sum_{i=0}^{n-1} \frac{1}{\sum_{m=1}^{k} m \cdot \frac{1}{n} \cdot \left(\frac{n-i-1}{n}\right)^{m-1}} - o(n^2).$$

□

We now prove the same deteriorating performance for the other 'simple' hyper-heuristics.

Corollary 8.5 *The expected runtime of the Permutation, Greedy and Random Gradient mechanisms on* LEADINGONES *with* $(1,\ldots,k)$BITFLIP*,* $k = \Theta(1)$*, is*

$$\mathrm{E}(\mathrm{T}_k) = \frac{k}{2} \cdot \sum_{i=0}^{n-1} \frac{1}{\sum_{m=1}^{k} m \cdot \frac{1}{n} \cdot \left(\frac{n-i-1}{n}\right)^{m-1}} \pm o(n^2).$$

Proof: Let p_i be the probability that at least one fitness improvement is constructed within k fitness function evaluations. For the Greedy and Permutation mechanisms, we have

$$p_i = \Pr(\text{IMP}_1) + (1 - \Pr(\text{IMP}_1)) \cdot \Pr(\text{IMP}_2) + \cdots + \left(\prod_{j=1}^{k-1} (1 - \Pr(\text{IMP}_j))\right) \cdot p_k \tag{8.1}$$

$$= \sum_{m=1}^{k} \left(\left(\prod_{j=1}^{m-1} \left(1 - j \cdot \frac{1}{n} \cdot \left(\frac{n-i-1}{n}\right)^{j-1} \right) \right) \cdot m \cdot \frac{1}{n} \cdot \left(\frac{n-i-1}{n}\right)^{m-1} \right) \tag{8.2}$$

$$= \sum_{m=1}^{k} \left((1 - o(1)) \cdot m \cdot \frac{1}{n} \cdot \left(\frac{n-i-1}{n}\right)^{m-1} \right). \tag{8.3}$$

To move from the Equation 8.2 to Equation 8.3, we note that $\Pr_{\text{IMP}_j} = O(\frac{1}{n})$ and $\prod_{j=1}^{m-1}(1 - \Pr_{\text{IMP}_j}) = \left(1 - O\left(\frac{1}{n}\right)\right)^m = 1 - o(1)$. To upper bound the expected optimisation time, we note that the difference between $\frac{k}{p_i}$ (i.e., the expected waiting time for an improvement to be constructed in terms of fitness evaluations) and the A_{n-i} expected waiting times used to prove Theorem 8.4 ($A_{n-i} = \frac{1}{\frac{1}{k} \cdot \sum_{m=1}^{k} m \cdot \frac{1}{n} \cdot \left(\frac{n-i-1}{n}\right)^{m-1}} - o(n)$) is at most constant, and thus the difference between the expected runtimes of these mechanisms and Simple Random is limited to lower-order terms (through the application of Theorem 2.13). We note that with probability at most $\frac{k(k-1)}{n^2}$, at least two mutations (of the k) considered are improvements (two parallel successes of the Greedy mechanism, or two mutations performed sequentially by the Permuation mechanism). As this occurs at most a sublinear number of times in expectation, and the maximum expected waiting time for any improving step is $O(n)$, the lower bound differs from the upper bound by at most an $o(n^2)$ term. Thus, the expected runtime of these mechanisms is also $\frac{k}{2} \cdot \sum_{i=0}^{n-1} \frac{1}{\sum_{m=1}^{k} m \cdot \frac{1}{n} \cdot \left(\frac{n-i-1}{n}\right)^{m-1}} \pm o(n^2)$.

For the Random Gradient mechanism, we note that the probability that an operator, when repeated following a success, is successful again is at most $\frac{k}{n}$ (an upper bound on the success probability of the kBITFLIP operator by Lemma 8.1). Since there are at most n improvements to be made throughout the search space, the expected number of repeats which produce an improvement is at most $k = \Theta(1)$. If the chosen operator is not successful, the Random Gradient mechanism behaves identically to the Simple Random mechanism. Its expected runtime is therefore at least the expected runtime of the Simple Random mechanism less an $o(n^2)$ term, and at most the expected runtime of the Simple Random mechanism plus n (as there are at most n iterations where the mechanisms differ in operator selection). Thus, its expected runtime is also $\frac{k}{2} \cdot \sum_{i=0}^{n-1} \frac{1}{\sum_{m=1}^{k} m \cdot \frac{1}{n} \cdot \left(\frac{n-i-1}{n}\right)^{m-1}} \pm o(n^2)$. □

8.3 Generalised Random Gradient is Faster

In this subsection, we present a rigorous theoretical analysis of the expected runtime of the Generalised Random Gradient hyper-heuristic (introduced in Chapter 4) on LEADINGONES using $k = \Theta(1)$ operators. We present an experimental analysis of the Generalised Greedy and Generalised Greedy Gradient hyper-heuristics in Section 8.4.

The following theorem bounds the expected runtime of the Generalised Random Gradient hyper-heuristic for any value of τ smaller than $\frac{1}{k} n \ln n$ for any values of $k = \Theta(1)$. The theorem allows us to identify values of τ for which the expected runtime of the hyper-heuristic for the function is the optimal expected runtime that may be achieved by using k operators; this result will be highlighted in Corollary 8.7 for values of $\tau = \omega(n)$. The main result of this section will be presented in Theorem 8.8 which rigorously proves that increasing the number of operators leads to improved expected runtimes.

Theorem 8.6 *The expected runtime of the Generalised Random Gradient hyper-heuristic on* LEADINGONES *for* $\tau \leq \left(\frac{1}{k} - \varepsilon\right) n \ln(n)$*, for some constant* $\varepsilon > 0$ *with* $(1,\ldots,k)$BITFLIP*,* $k = \Theta(1)$*, is at most:*

$$\frac{n^2}{2} \cdot \sum_{j=1}^{w} \left(\frac{\left(k \cdot \frac{\tau}{n} + \left[\sum_{m=1}^{k} e^{m\frac{\tau}{n}(1-(j-1)/w)^{m-1}} \cdot M_m(j,w)\right]\right)}{w \cdot \left(\left[\sum_{m=1}^{k} e^{m\frac{\tau}{n}(1-j/w)^{m-1}}\right] - k\right)} \right) \pm o(n^2).$$

k	$\tau = 5n$	$\tau = 50n$	$\tau = 100n$	$\tau = \frac{1}{10}n\ln(n)$
2	0.46493	0.42363	0.42329	0.42329
3	0.46802	0.40579	0.40525	0.40525
4	0.48102	0.39897	0.39830	0.39830
5	0.49630	0.39568	0.39492	0.39492
11	3.090×10^{23}	8785.8	0.38987	0.38987
18	1.886×10^{44}	5.363×10^{24}	1034.8	0.38899

Table 8.1 The upper bounds in the leading constants found from various parameter combinations of the number of operators k and the learning period τ in Theorem 8.6 with $w = 100{,}000$. The values reported in the fourth column are the best possible for each k. As the number of operators increases, larger values of τ are required for good performance ($\tau = \omega(n)$ is necessary to achieve optimal performance). Note that while Theorem 8.6 allows us to show optimal leading constants for $\tau = \omega(n)$, the upper bounds it provides for large values of k and small values of τ are far from tight due to the generality of the theorem.

for any integer $w = \Omega(1) \cap o\left(\frac{n}{\exp(k\tau/n)}\right)$*, where* $M_1(j,w) = \min\left\{\frac{\tau}{n}, 1\right\}$ *and for* $m \geq 2$*,*

$$M_m(j,w) := \begin{cases} \frac{\tau}{n} & \text{if } j = w \text{ or } \frac{1}{m(1-j/w)^{m-1}} > \frac{\tau}{n} \\ \frac{1}{m(1-j/w)^{m-1}} & \text{otherwise.} \end{cases}$$

Theorem 8.6 is very general as it provides bounds on the expected runtime of Generalised Random Gradient for any parameter values $\tau \leq \left(\frac{1}{k} - \varepsilon\right) n\ln(n)$ for some constant $\varepsilon > 0$ and $k = \Theta(1)$. In Table 8.1 we present some of the most interesting parameter combinations of k and τ that arise from this result. When $\tau = 1$, the Generalised Random Gradient hyper-heuristic acts the same as the Simple Random mechanism. When $\tau = 0.1n\ln(n)$ GRG is able to find the best possible runtime achievable with the low-level heuristics, up to lower order terms, as will be shown in Theorem 8.7.

Doerr and Wagner (2018) calculated that the best expected runtime for any unbiased (1+1) black box algorithm using such mutation operators for LEADINGONES is (up to lower order terms) $\approx 0.388n^2$. Corollary 8.7 states that GRG can match this theoretical performance limit up to one decimal place with 4 low-level heuristics, up to two decimal places with 11 low-level heuristics and up to three decimal places with 18 low-level heuristics. In Table 8.1 we present some of the most interesting parameter combinations of k and τ.

Proof: [Of Theorem 8.6]

We follow the proof idea for the Generalised Random Gradient hyper-heuristic for two operators (Theorem 5.11 in Chapter 5). That is, we partition the optimisation process into w chunks based on the value of the LEADINGONES fitness function: during chunk j, the LO value of the current solution is at least $\frac{(j-1)n}{w}$ and less than $\frac{j \cdot n}{w}$. After all w chunks have been completed, the global optimum, with a LO value of n, will have been found.

We upper bound the expectation of the runtime T of the Generalised Random Gradient hyper-heuristic on LO by the sum of the expected values of T_j, the runtimes on each chunk of the optimisation process, plus the expected times taken to begin a chunk. As our analysis of $\mathrm{E}(\mathrm{T}_j)$ requires the chunk to start with a random choice of mutation operator, we bound $\mathrm{E}(\mathrm{T}) \leq \sum_{j=1}^{w} \left(\mathrm{E}(\mathrm{T}_j) + \mathrm{E}(S_{j+1})\right)$ where S_{j+1} is a random variable denoting the expected number of iterations between the first solution in chunk $j+1$ being constructed and the following random operator choice. We will later show that with proper parameter choices, the contribution of S_{j+1} terms can be bounded by $o(n^2)$, and therefore does not affect the leading constant in the overall bound.

Let us now consider T_j, the number of iterations the hyper-heuristic spends in chunk j. Recall that a mutation operator is selected uniformly at random, and is allowed to run until it fails to produce an improvement for τ sequential iterations. Let N_j be a random variable denoting the number of random operator choices the hyper-heuristic performs during chunk j, and $X_{j,1}, \ldots, X_{j,N_j}$ be the number of iterations the hyper-heuristic runs each chosen operator for, in which case $\mathrm{T}_j = \sum_{\kappa=1}^{N_j} X_{j,\kappa}$, and by applying Wald's equation (Theorem 5.4 in Chapter 5), using $\mathrm{E}(X_j)$ to denote an upper bound on all $\mathrm{E}(X_{j,\kappa})$ in chunk j:

$$\mathrm{E}(\mathrm{T}_j) = \textstyle\sum_{\kappa=1}^{N_j} \mathrm{E}(X_{j,\kappa}) \leq \mathrm{E}(N_j)\,\mathrm{E}(X_j). \tag{8.4}$$

To bound $\mathrm{E}(N_j)$, the expected number of times the random operator selection is performed during chunk j, we lower bound the expected number of improvements found following operator selection, and apply the Additive Drift Theorem (Theorem 2.11) to find the expected number of random operator selections occurring before a sufficient number of improvements have been found to enter the next chunk.

Recall from Lemma 8.1 that the improvement probability for a mBITFLIP operator is

$$\Pr(\text{IMP}_m) = m \cdot \frac{1}{n} \cdot \left(\frac{n-i-1}{n}\right)^{m-1} + O(n^{-2}).$$

Since either the $m \cdot \frac{1}{n} \cdot \left(\frac{n-i-1}{n}\right)^{m-1}$ terms will asymptotically dominate, or the $\Pr(\text{IMP}_1) = \frac{1}{n}$ term will asymptotically dominate, the lower order terms will be insignificant. In particular, we can combine the impact of the $O(n^{-2})$ terms into the $o(n^2)$ in the final bound on the expected runtime and omit them from the $\Pr(\text{IMP}_m)$ terms in the calculations.

Let F_m denote the event that the mBITFLIP operator fails to find an improvement during τ iterations. This will occur with probability $\Pr(F_m) = (1-\Pr(\text{IMP}_m))^{\tau}$. Within chunk j, the ancestor individual has at most $jn/w-1$, and at least $(j-1)n/w$, leading 1-bits. Hence:

$$\Pr(F_m) \le \left(1 - m\cdot\frac{1}{n}\left(\frac{n-(jn/w-1)-1}{n}\right)^{m-1}\right)^{\tau} \le \left(1-m\cdot\frac{1}{n}\left(1-\frac{j}{w}\right)^{m-1}\right)^{\tau}$$
$$\le e^{-\frac{m\tau}{n}(1-\frac{j}{w})^{m-1}};$$

$$\Pr(F_m) \ge \left(1 - m\cdot\frac{1}{n}\left(\frac{n-((j-1)n/w)-1}{n}\right)^{m-1}\right)^{\tau} \ge \left(1-m\cdot\frac{1}{n}\left(1-\frac{j-1}{w}\right)^{m-1}\right)^{\tau}$$
$$\ge e^{-\frac{m\tau}{n}(1-\frac{j-1}{w})^{m-1}} - \frac{1}{n}.$$

We note that a geometric distribution with parameter $p = \Pr(F_m)$ can be used to model the number of improvements that the mBITFLIP operator finds prior to failing to find an improvement for τ iterations; the expectation of this distribution is $(1-p)/p = 1/p-1$. Combined over all k operators, the expected number of improvements D_j produced following a single random operator selection during chunk j, $\mathrm{E}(D_j)$, is greater than:

$$\mathrm{E}(D_j) \ge \sum_{m=1}^{k}\frac{1}{k}\left(\frac{1}{\Pr(F_m)}-1\right) \ge \frac{1}{k}\left(\sum_{m=1}^{k} e^{\frac{m\tau}{n}(1-\frac{j}{w})^{m-1}}\right) - 1$$

by inserting the upper bounds for $\Pr(F_m)$. We use this expectation as the drift in the Additive Drift Theorem to upper bound $\mathrm{E}(N_j)$. Recall that each chunk consists of advancing through at most n/w fitness values; as bits beyond the leading ones prefix and the first zero bit remain uniformly distributed, in expectation $n/(2w)$ improvements by mutation are required. If each step of a random process in expectation contributes $\mathrm{E}(D_j)$ improvements by mutation, the expected number of steps required to complete chunk j is at most:

$$\mathrm{E}(N_j) \le \frac{n/(2w)}{\mathrm{E}(D_j)} \le \frac{k}{2}\cdot\frac{n}{w}\cdot\frac{1}{\left(\sum_{m=1}^{k} e^{\frac{m\tau}{n}(1-\frac{j}{w})^{m-1}}\right)-k} \tag{8.5}$$

by the Additive Drift Theorem.

To bound $\mathrm{E}(X_j)$, the expected number of iterations before a selected mutation operator fails to produce an improvement for τ iterations, we apply Wald's equation: let S be the number of improvements found by the operator before it fails, and $W_1, \dots, W_S$ be the number of iterations it took to find each of those improvements. Then, once selected, the mBITFLIP operator runs for:

$$\mathrm{E}(X_j|m) = \tau + \sum_{s=1}^{S} \mathrm{E}(W_s) = \mathrm{E}(S)\,\mathrm{E}(W_m \mid S \geq 1)$$

where τ accounts for the iterations immediately before failure, and the sum for the iterations preceding each constructed improvement. Recall that $\mathrm{E}(S) = 1/\Pr(F_m) - 1$ by the properties of the geometric distribution, and observe that $\mathrm{E}(W_m \mid S \geq 1) = \mathrm{E}(W_m \mid W_m \leq \tau) \leq \min\{\tau, \mathrm{E}(W_m)\}$. Using a waiting time argument for $\mathrm{E}(W_m)$ being equal to the reciprocal of the improvement probability for the mBITFLIP operator, we get:

$$\begin{aligned}
\mathrm{E}(X_j \mid m) &\leq \tau + \left(\frac{1}{\Pr(F_m)} - 1\right) \cdot \min\left\{\tau, \frac{1}{m \cdot \frac{1}{n} \cdot \left(\frac{n-(jn/w-1)-1}{n}\right)^{m-1}}\right\} \\
&= \tau + \left(\frac{1}{e^{-\frac{m\tau}{n}(1-\frac{j-1}{w})^{m-1}} - \frac{1}{n}} - 1\right) \cdot \min\left\{\tau, \frac{n}{m(1-j/w)^{m-1}}\right\} \\
&\leq \tau + e^{\frac{m\tau}{n}(1-\frac{j-1}{w})^{m-1}} \cdot \min\left\{\tau, \frac{n}{m(1-\frac{j}{w})^{m-1}}\right\} + O(1).
\end{aligned}$$

Combining these conditional expectations yields

$$\begin{aligned}
\mathrm{E}(X_j) \leq \sum_{m=1}^{k} \frac{1}{k} \cdot \mathrm{E}(X_j \mid m) &\leq \tau + \sum_{m=1}^{k} \left[\frac{1}{k} \cdot e^{\frac{m\tau}{n}(1-\frac{j-1}{w})^{m-1}} \cdot \min\left\{\tau, \frac{n}{m(1-j/w)^{m-1}}\right\}\right] + O(k) \\
&\leq n\left(\frac{\tau}{n} + \sum_{m=1}^{k}\left[\frac{1}{k} \cdot e^{\frac{m\tau}{n}(1-\frac{j-1}{w})^{m-1}} \cdot \min\left\{\frac{\tau}{n}, \frac{1}{m(1-j/w)^{m-1}}\right\}\right]\right) + O(k) \\
&\leq \frac{n}{k}\left(\frac{k\tau}{n} + \sum_{m=1}^{k}\left[e^{\frac{m\tau}{n}(1-\frac{j-1}{w})^{m-1}} \cdot \min\left\{\frac{\tau}{n}, \frac{1}{m(1-j/w)^{m-1}}\right\}\right]\right) + O(k).
\end{aligned} \tag{8.6}$$

At $j = w$, we have that $\min\left\{\frac{\tau}{n}, \frac{1}{m(1-j/w)^{m-1}}\right\}$ is undefined. However, we know that the chosen operator would only be applied for a maximum of τ steps. Hence, we define

$M_1(j,w) = \min\{\frac{\tau}{n}, 1\}$ and for $m \geq 2$,

$$M_m(j,w) := \begin{cases} \frac{\tau}{n} & \text{if } j = w \text{ or } \frac{1}{m(1-j/w)^{m-1}} > \frac{\tau}{n} \\ \frac{1}{m(1-j/w)^{m-1}} & \text{otherwise.} \end{cases}$$

Finally, we return to bounding the overall expected runtime of the hyper-heuristic. Recall that S_j denoted the number of iterations the algorithm spends in chunk j while using the random operator chosen during chunk $j-1$, and as $\mathrm{E}(S_j) \leq \max\left(\mathrm{E}(X_j \mid m)\right)_{m=1,\dots,k} < k \cdot \mathrm{E}(X_j) = O\left(n\exp\left(\frac{k\tau}{n}\right)\right)$ and since $w = o\left(\frac{n}{\exp(k\tau/n)}\right)$, $\sum_{j=1}^{w} \mathrm{E}(S_{j+1}) = o(n^2)$. Substituting the bounds (8.5) and (8.6) into (8.4) yields the theorem statement:

$$\begin{aligned}\mathrm{E(T)} &\leq \sum_{j=1}^{w}\left(\mathrm{E}(\mathrm{T}_j) + \mathrm{E}(S_{j+1})\right) \leq \left(\sum_{j=1}^{w} \mathrm{E}(N_j)\mathrm{E}(X_j)\right) + o(n^2) \\ &\leq o(n^2) + \frac{n^2}{2} \times \sum_{j=1}^{w} \frac{1}{w} \cdot \frac{\frac{k\tau}{n} + \sum_{m=1}^{k}\left[e^{\frac{m\tau}{n}(1-\frac{j-1}{w})^{m-1}} \cdot M_m(j,w)\right]}{\left(\sum_{m=1}^{k} e^{\frac{m\tau}{n}(1-\frac{j}{w})^{m-1}}\right) - k}.\end{aligned}$$

□

We now show that for appropriate values of the parameter τ, the k-operator Generalised Random Gradient hyper-heuristic can achieve the best-possible expected runtime available for a mechanism using k-operators, up to lower order terms, from Theorem 8.3. The following Corollary provides an upper bound on the expected runtime for sufficiently large learning periods (i.e., $\tau = \omega(n)$).

Corollary 8.7 *The expected runtime of the Generalised Random Gradient hyper-heuristic on* LEADINGONES *using* $(1,\dots,k)$BITFLIP, *where* $k = \Theta(1)$, *with* τ *that satisfies both* $\tau = \omega(n)$ *and* $\tau \leq \left(\frac{1}{k} - \varepsilon\right) n\ln(n)$, *for some constant* $\varepsilon > 0$, *is at most*

$$\frac{1}{2}\left(\sum_{i=0}^{n/k-1} \frac{1}{k \cdot \frac{1}{n} \cdot \left(\frac{n-i-1}{n}\right)^{k-1}} + \sum_{m=1}^{k-1} \sum_{i=n/(m+1)}^{n/m-1} \frac{1}{m \cdot \frac{1}{n} \cdot \left(\frac{n-i-1}{n}\right)^{m-1}}\right) \pm o(n^2).$$

Proof: This proof follows a similar structure to that of Corollary 5.10 in Chapter 5.

We set $w = \log^2 n$. We want to simplify the sum from the statement of Theorem 8.6, which states that

$$\mathrm{E(T)} \leq \frac{n^2}{2} \cdot \sum_{j=1}^{w} \left(\frac{\left(k \cdot \frac{\tau}{n} + \left[\sum_{m=1}^{k} e^{m\frac{\tau}{n}(1-(j-1)/w)^{m-1}} \cdot M_m(j,w)\right]\right)}{w \cdot \left(\left[\sum_{m=1}^{k} e^{m\frac{\tau}{n}(1-j/w)^{m-1}}\right] - k\right)} \right) \pm o(n^2).$$

Since $\tau = \omega(n)$, we have that $\tau/n = \omega(1)$. Hence, we can simplify the M terms. Firstly, we have that $M_1(j,w) = \min\{\frac{\tau}{n}, 1\} = 1$. Recall that, for $m \geq 2$,

$$M_m(j,w) := \begin{cases} \frac{\tau}{n} & \text{if } j = w \text{ or } \frac{1}{m(1-j/w)^{m-1}} > \frac{\tau}{n} \\ \frac{1}{m(1-j/w)^{m-1}} & \text{otherwise.} \end{cases}$$

Since $\frac{\tau}{n} = \omega(1)$, we have that $\frac{\tau}{n} > \frac{1}{m(1-j/w)^{m-1}}$. Hence, we can simplify:

$$M_m(j,w) := \begin{cases} \frac{\tau}{n} & \text{if } j = w \\ \frac{1}{m(1-j/w)^{m-1}} & \text{otherwise.} \end{cases}$$

In particular, for $j \neq w$, we have that $M_1(j,w) = \Theta(1)$ and, for $m \geq 2$, $M_m(j,w) = \Theta(1)$.

We can further simplify the sum from Theorem 8.6 by noting that, since $\frac{\tau}{n} = \omega(1)$, the exponential term in each summand will be asymptotically dominant. In particular, the $\frac{k\tau}{n}$ term in the numerator, and the $-k$ term in the denominator, will be asymptotically dominated. Hence, we can relegate these terms into the lower order $o(n^2)$ term.

Furthermore, the difference between the exponential terms in the numerator and denominator in the sum from Theorem 8.6 differ only by a small multiplicative term, $1 + o(1)$. Hence, we can consider both as $e^{m\frac{\tau}{n}(1-\frac{j}{w})^{m-1}}$, with the difference again being relegated into the $o(n^2)$ term.

With these modifications, we can simplify the sum:

$$\begin{aligned} \mathrm{E(T)} &\leq \frac{n^2}{2} \cdot \sum_{j=1}^{w} \left(\frac{\left(k \cdot \frac{\tau}{n} + \left[\sum_{m=1}^{k} e^{m\frac{\tau}{n}(1-\frac{j-1}{w})^{m-1}} \cdot M_m(j,w)\right]\right)}{w \cdot \left(\left[\sum_{m=1}^{k} e^{m\frac{\tau}{n}(1-\frac{j}{w})^{m-1}}\right] - k\right)} \right) \pm o(n^2) \\ &\leq \frac{n^2}{2} \cdot \sum_{j=1}^{w} \left(\frac{\left(\sum_{m=1}^{k} e^{m\frac{\tau}{n}(1-\frac{j}{w})^{m-1}} \cdot M_m(j,w)\right)}{w \cdot \left(\left[\sum_{m=1}^{k} e^{m\frac{\tau}{n}(1-\frac{j}{w})^{m-1}}\right]\right)} \right) \pm o(n^2). \end{aligned}$$

To further simplify the sum, we consider when each of the k exponential summand terms (i.e., the $\exp(m\tau/n(1-(j-1)/w)^{m-1})$ terms) will be asymptotically dominant in the numerator. Since $\frac{\tau}{n} = \omega(1)$, whichever exponential term has the largest exponent will asymptotically dominate the other terms. In particular, the m^{th} term will dominate when $\frac{m\tau}{n}(1-\frac{j}{w})^{m-1}$ is the largest amongst $1 \leq m \leq k$. Comparing the terms at m and $m+1$, we see that the m^{th} term dominates the $(m+1)^{th}$ term when $j \geq \frac{w}{m+1}$. Continuing these calculations, we have that the m^{th} term will dominate all others (i.e., have the largest exponent) when $\frac{w}{m+1} \leq j \leq \frac{w}{m} - 1$ for $m < k$, and the k^{th} term will dominate when $j \leq \frac{w}{k} - 1$. If a term is asymptotically dominated, we can relegate it into the $o(n^2)$ lower order term. Hence, the only terms remaining in the numerator in the period when the m^{th} operator dominates will be the m^{th} exponential term multiplied by $M_m(j,w)$; the denominator will similarly only contain the m^{th} exponential term, multiplied by w. The sum will hence simplify to a sum of $M_m(j/w)$ terms:

$$\begin{aligned}
\mathrm{E(T)} &\leq \frac{n^2}{2} \cdot \left(\sum_{j=1}^{w/k} M_k(j,w) + \sum_{m=1}^{k-1} \sum_{j=w/(m+1)+1}^{w/m} M_m(j,w) \right) \pm o(n^2). \\
&= \frac{n^2}{2} \cdot \left(\sum_{j=1}^{w/k} \frac{1}{k(1-j/w)^{k-1}} + \sum_{m=1}^{k-1} \sum_{j=w/(m+1)+1}^{w/m} \frac{1}{m(1-j/w)^{m-1}} \right) \pm o(n^2). \\
&= \frac{n}{2} \cdot \left(\sum_{j=1}^{w/k} \frac{1}{k \cdot \frac{1}{n} \cdot (1-j/w)^{k-1}} + \sum_{m=1}^{k-1} \sum_{j=w/(m+1)+1}^{w/m} \frac{1}{m \cdot \frac{1}{n} \cdot (1-j/w)^{m-1}} \right) \pm o(n^2).
\end{aligned}$$

Recall that, for $i = \mathrm{LO}(x)$, the chunk j refers to when $\frac{(j-1)n}{w} \leq i \leq \frac{jn}{w} - 1$. In particular, if we substitute $i \geq \frac{(j-1)n}{w}$, (or $j \leq \frac{iw}{n} + 1$) into each summand, we will find

$$\frac{1}{m \cdot \frac{1}{n} \cdot (1-\frac{j}{w})^{m-1}} \leq \frac{1}{m \cdot \frac{1}{n} \cdot \left(\frac{n-i-1}{n}\right)^{m-1}}.$$

Furthermore, using the upper and lower bounds on i to adjust the bounds on the sums (which will cancel out the multiplicative n term) gives the result:

$$\begin{aligned}
\mathrm{E(T)} &\leq \frac{n}{2} \cdot \left(\sum_{j=1}^{w/k} \frac{1}{k \cdot \frac{1}{n} \cdot (1-\frac{j}{w})^{k-1}} + \sum_{m=1}^{k-1} \sum_{j=w/(m+1)+1}^{w/m} \frac{1}{m \cdot \frac{1}{n} \cdot (1-\frac{j}{w})^{m-1}} \right) \pm o(n^2) \\
&\leq \frac{1}{2} \left(\sum_{i=0}^{n/k-1} \frac{1}{k \cdot \frac{1}{n} \cdot \left(\frac{n-i-1}{n}\right)^{k-1}} + \sum_{m=1}^{k-1} \sum_{i=n/(m+1)}^{n/m-1} \frac{1}{m \cdot \frac{1}{n} \cdot \left(\frac{n-i-1}{n}\right)^{m-1}} \right) \pm o(n^2).
\end{aligned}$$

□

Thus, we have shown that the k-operator Generalised Random Gradient hyper-heuristic can match, up to lower order terms, the best-possible performance of any algorithm using the same k stochastic mutation operators.

We now present the main result of this chapter, which states that the best case performance of the k-operator variant of the Generalised Random Gradient hyper-heuristic beats the best-possible expected runtime for an algorithm using any $m < k$ bitflip operators.

Theorem 8.8 *Let τ satisfy both $\tau = \omega(n)$ and $\tau \leq \left(\frac{1}{k} - \varepsilon\right) n \ln(n)$, for some constant $\varepsilon > 0$. Then the expected runtime for the Generalised Random Gradient hyper-heuristic on* LEADINGONES *using* $(1,\ldots,k)$BITFLIP*, where $k = \Theta(1)$, is less than the best-possible expected runtime on* LEADINGONES *for any unary unbiased algorithm using any strict subset of* $(1,\ldots,k)$BITFLIP.

Proof: Corollary 8.7 shows that the Generalised Random Gradient hyper-heuristic with k mutation operators can match the best-possible peformance for an algorithm with k mutation operators from Theorem 8.3, up to lower order terms. The results from Corollary 8.7 and Theorem 8.3 imply that the only difference in the best case asymptotic expected runtime for the Generalised Random Gradient hyper-heuristic with k operators ($(1,\ldots,k)$BITFLIP), and the best-possible expected runtime for an algorithm with $k-1$ operators (i.e., using (1,...,K-1)BITFLIP, occurs when $0 \leq i \leq \frac{n}{k} - 1$. In this region, the best case expected performance of the k-operator variant matches the expected performance of only using the kBITFLIP operator in this area, up to lower order terms, while the expected performance of the best-possible $k-1$-operator algorithm matches the expected performance of using the (K-1)BITFLIP operator in this area.

We know from Lemma 8.2 that the expected runtime to optimise this area when applying the kBITFLIP operator will be less than the expected runtime when applying the (K-1)BITFLIP operator, and thus using the kBITFLIP operator will be faster. Since in the rest of the search space the best case k-operator Generalised Random Gradient hyper-heuristic and the best-possible $k-1$-operator algorithm will have the same expected performance (up to lower order terms), the best case expected runtime of the Generalised Random Gradient hyper-heuristic with k operators will be faster than the best-possible expected runtime of an algorithm $k-1$-operators.

Furthermore, continuing the argument for the best-possible performance of an algorithm with (1,...,M)BITFLIP operators ($m < k-1$) implies that the best case expected runtime of the k-operator Generalised Random Gradient hyper-heuristic is faster than the best-possible performance of any algorithm with $m < k$ operators. In particular, Generalised Random Gradient with k operators ($(1,\ldots,k)$BITFLIP) is faster than the best-possible algorithm using

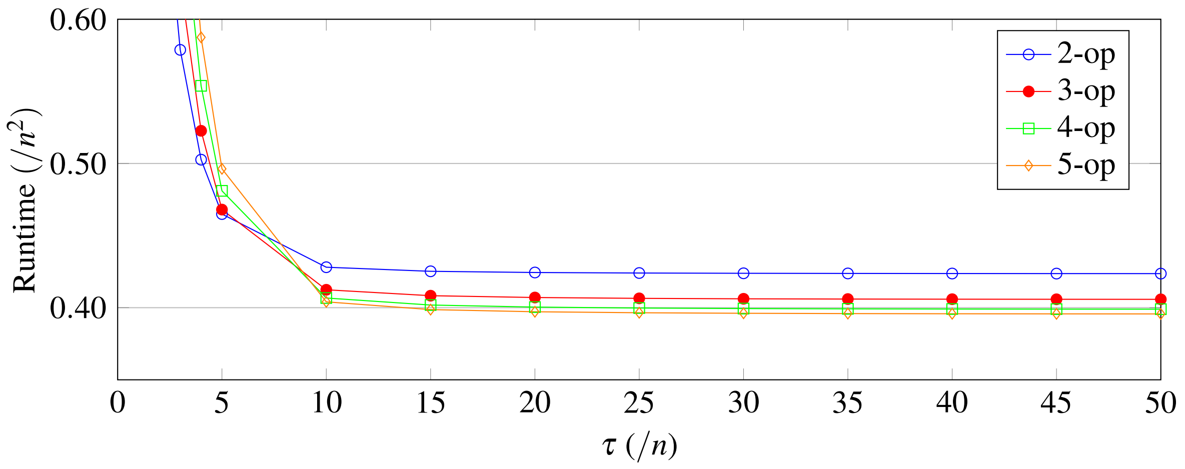

Fig. 8.1 The leading constants in the theoretical upper bounds on the average number of fitness function evaluations required by the k-operator Generalised Random Gradient hyper-heuristic to find the LEADINGONES optimum (we have used a value of $w = 100{,}000$ in the result of Theorem 8.6).

any strict subset of $(1,\ldots,k)$BITFLIP. The result from Lemma 8.2 (and a similar argument as used above with $k-1$ operators) implies that Generalised Random Gradient will outperform the best-possible algorithm in the area of the search space corresponding to where the best-possible algorithm is missing any of the $(1,\ldots,k)$BITFLIP operators (i.e., if the best-possible algorithm is missing the mBITFLIP operator, Generalised Random Gradient will outperform it in the region where the LO fitness value satisfies $\frac{n}{m+1} \leq \text{LO}(x) \leq \frac{n}{m} - 1)$, leading to a faster expected runtime. $\square$

Figure 8.1 shows the relevant theoretical upper bounds for the k-operator variant of the Generalised Random Gradient hyper-heuristic from Theorem 8.6 for linear values of the learning period τ. The upper bounds found by the hyper-heuristics with more operators are better than the ones with less operators, as given by Theorem 8.6. In particular, the upper bounds for any hyper-heuristic with k-operators outperforms the best-possible expected runtime for any hyper-heuristic with less than k operators, as implied by Corollary 8.7 and Theorem 8.8. We depict this result explicitly for $k = 3$ and $k = 5$ in Figure 8.2, as a comparison with the results from Theorem 8.3.

8.4 Experimental Supplements

In the previous section we proved that the Generalised Random Gradient hyper-heuristics performs efficiently with extended sets of low-level heuristics for the LEADINGONES bench-

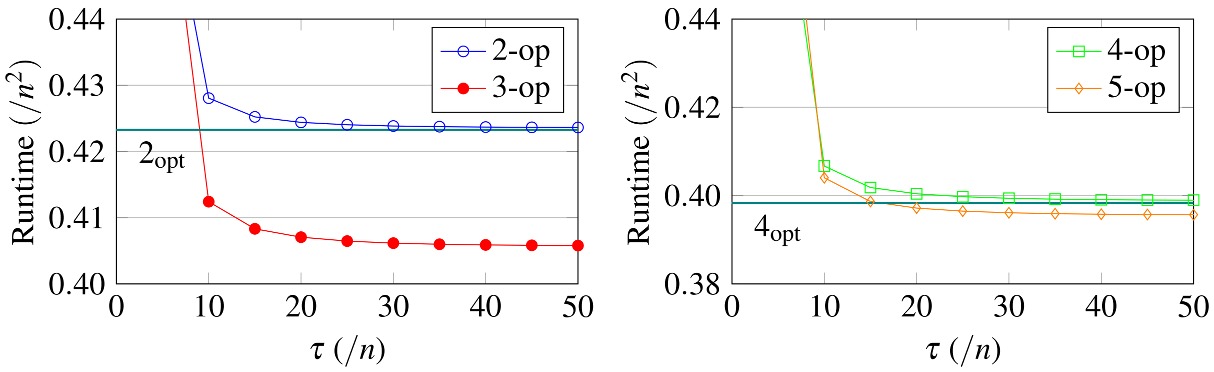

Fig. 8.2 A comparison of the optimal expected runtimes of the Generalised Random Gradient hyper-heuristic with k-operators against the leading constant in the theoretical upper bound of the expected runtime of the Generalised Random Gradient hyper-heuristic with $k+1$-operators; for $k = 2$ and $k = 4$. 2_{opt} ($\approx 0.42329n^2$) and 4_{opt} ($\approx 0.39830n^2$) are the best possible runtimes achievable by the Generalised Random Gradient hyper-heuristic with access to 2 and 4 low-level heuristics respectively, from Corollary 8.7.

mark function, for large enough problem sizes n. In this section we present an experimental study of the three generalised hyper-heuristics (Generalised Random Gradient, Generalised Greedy and Generalised Greedy Gradient), introduced in Chapter 4 and analysed choosing from two heuristics in Chapter 5, in order to shed light on their performance for realistic problem sizes up to $n = 10^8$. All parameter combinations have been simulated 10,000 times. We consider the generalised hyper-heuristics using k operators, $(1,\ldots,k)$BITFLIP, and look at the impact of the parameters τ, k and n.

Figure 8.3 shows the runtimes for the Generalised Greedy hyper-heuristic when choosing from 2 to 5 separate mutation operators on LEADINGONES, for a problem size of $n =$ 100,000. The Generalised Greedy hyper-heuristic, unlike the Generalised Random Gradient hyper-heuristic and the 'simple' mechanisms (see Theorem 8.4, Corollary 8.5), displays a detrimental performance when more operators are included. Unlike the Generalised Random Gradient hyper-heuristic, which exploits a high-performing operator while running for a shorter time with low-performing operators, the Generalised Greedy hyper-heuristic runs each chosen operator for a fixed period of time. Involving more operators means a lower chance of choosing the best operator in the Decision Stage.

Figure 8.4 shows the same results for a much larger range of τ values. We can see that the performance of the 2-operator hyper-heuristic remains the best performing hyper-heuristic. Although the performance of the 5-operator hyper-heuristic is the worst, it is worth noting than even this algorithm outperforms the single operator algorithm RLS (i.e., $0.5n^2$). For this hyper-heuristic, the detriment in performance between $\tau = 50n$ and $\tau = 750n$ is

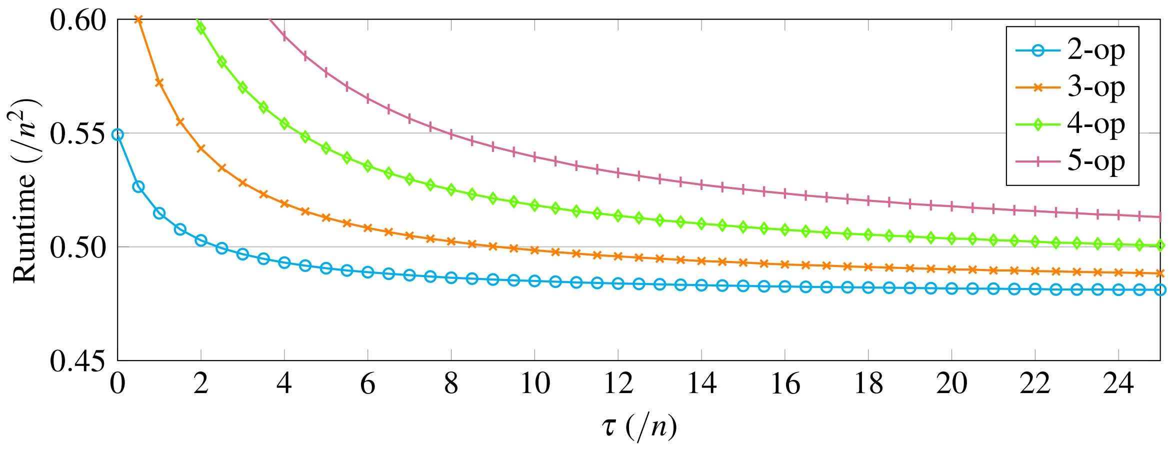

Fig. 8.3 Average number of fitness function evaluations required by the Generalised Greedy hyper-heuristic to find the LEADINGONES optimum, $n = 100{,}000$.

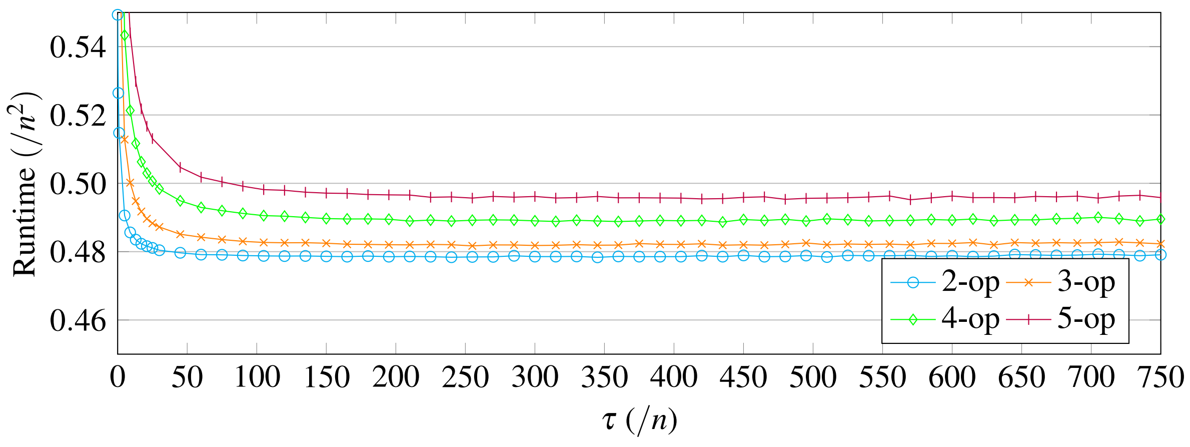

Fig. 8.4 Average number of fitness function evaluations required by the Generalised Greedy hyper-heuristic to find the LEADINGONES optimum, $n = 100{,}000$.

approximately $0.01n^2$, confirming that the Generalised Greedy hyper-heuristic is robust to the choice of the learning period τ.

Figure 8.5 shows the runtimes for the Generalised Random Gradient hyper-heuristic when choosing from 2 to 5 separate mutation operators on LEADINGONES, for a problem size of $n = 100{,}000$. We see that incorporating more operators can be beneficial to the performance of the Generalised Random Gradient hyper-heuristic. Whilst the 2-operator hyper-heuristic achieves a best performance of approximately $0.44n^2$, the 5-operator hyper-heuristic achieves a best performance of around $0.425n^2$. It is, however, important to set τ optimally to achieve the best performance. For $0 \leq 0.35n\ln(n) \leq \tau$ and $\tau \geq 1.5n\ln(n)$, the 2-operator hyper-heuristic outperforms the others. The results imply that the more operators

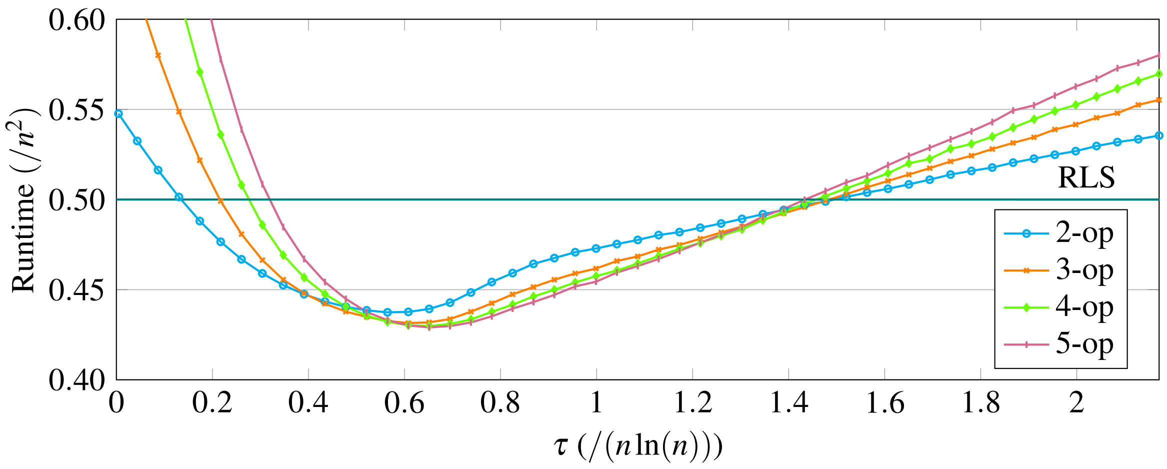

Fig. 8.5 Average number of fitness function evaluations required by the Generalised Random Gradient hyper-heuristic to find the LEADINGONES optimum, $n = 100{,}000$.

that are incorporated in the hyper-heuristic, the shorter the range of τ values for which it performs the best, in comparison with the hyper-heuristics with fewer operators.

Figure 8.6 shows the average runtime for the Generalised Greedy Gradient hyper-heuristic on LEADINGONES for $n = 100{,}000$. While the hyper-heuristic displays similar performance to the Generalised Random Gradient hyper-heuristic, we note that the best performance of each k outperforms the respective selection for the Generalised Random Gradient hyper-heuristic with the same k. This implies that, while the Decision Stage can be detrimental in the case of the Generalised Greedy hyper-heuristic (see Figure 8.3), the Exploitation Stage idea can correct for poor operator choices or expensive Decision Stages, leading to better performance. We can see, for example, that the best performance of the 5-operator variant of the Generalised Greedy Gradient hyper-heuristic, for $\tau \approx 0.62n\ln(n)$, runs on average in approximately $0.418n^2$, outperforming the best-possible 2-operator expected runtime, shown in Corollary 6.3, of $\approx 0.423n^2$.

8.5 Chapter Summary and Discussion

In Chapters 4 and 5 we saw that the Generalised Random Gradient hyper-heuristic is able to learn to prefer certain heuristics in different areas of the search space, and exploit this knowledge to achieve optimal performance for LEADINGONES.

In this chapter, we rigorously analysed the performance of the Generalised Random Gradient hyper-heuristic with more realistic sets of low-level heuristics of arbitrary size greater than 2. We have shown that, while the performance of four 'simple' hyper-heuristic

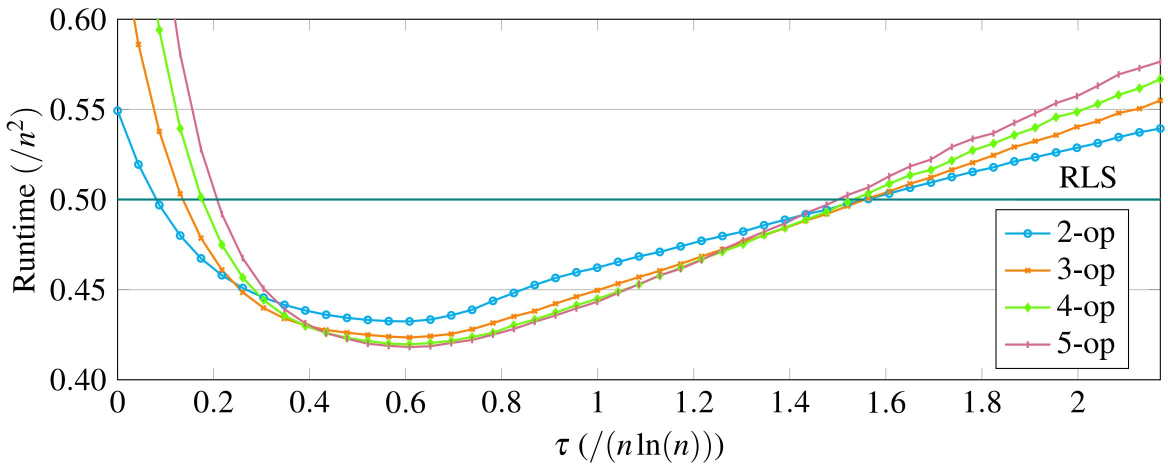

Fig. 8.6 Average number of fitness function evaluations required by the Generalised Greedy Gradient hyper-heuristic to find the LEADINGONES optimum, $n = 100{,}000$.

mechanisms (Simple Random, Permutation, Random Gradient, Greedy; introduced in Section 3.1) deteriorates with the inclusion of more operators, the Generalised Random Gradient hyper-heuristic shows improved performance when able to choose between more operators. The hyper-heuristic can achieve the best-possible performance for an algorithm able to choose between the same k operators (up to lower order terms), and thus is able to beat the performance of any algorithms using $m < k$ operators.

We have complemented the theory with experiments for realistic problem sizes. The experiments show that the Generalised Greedy Gradient hyper-heuristic also has improved performance with the increase of the number of operators. On the other hand, the performance of Generalised Greedy deteriorates with the increase of operators. Yet, this detriment is slow such that up to $k = 5$ it still outperforms the single operator algorithm Randomised Local Search on LEADINGONES.

Chapter 8 is based on the following publications:

1. Lissovoi, A., Oliveto, P. S., and Warwicker, J. A. (2017). On the Runtime Analysis of Selection Hyper-heuristics for Pseudo-Boolean Optimisation. *Proceedings of the Annual Conference on Genetic and Evolutionary Computation (GECCO '17)*, pages 849-856. ACM.

2. Lissovoi, A., Oliveto, P. S., and Warwicker, J. A. (2019). Simple Hyper-heuristics Control the Neighbourhood Size of Randomised Local Search Optimally for LeadingOnes. *Evolutionary Computation* (to appear).

Chapter 9

Hyper-heuristics can Automatically Adapt their Learning Periods to Optimality

In this chapter, we extend the power of the generalised hyper-heuristics by allowing the learning period τ to automatically adapt its length throughout the run. We introduce a simple adaptive update rule for τ, inspired by the $1/5^{th}$ rule from continuous optimisation. The goal is that the learning period should adapt such that the currently optimal heuristic succeeds to find σ improvements during the learning period, while the suboptimal heuristics fail to find the requisite successes, where σ is a (superconstant) parameter describing the speed of the adaptation. Such performance would give a $1 - o(1)$ ratio of successes to failures, i.e., we would see successes with high probability.

Our main result is that the presented Adaptive Random Gradient hyper-heuristic with self adjusting learning period adapts its learning period to optimality. This is shown by proving that Adaptive Random Gradient has optimal performance on the LEADINGONES benchmark function. In contrast to the parameter τ in the previous work, the hyper-parameter σ here is not very critical – all we require for our proof is that $\sigma = \Omega(\log^4 n) \cap o(\sqrt{n/\log n})$, which is a wide range of possible values. We complement our theoretical results with some experiments which show that Adaptive Random Gradient converges to optimality faster than Generalised Random Gradient with fixed values of τ i.e., it is faster for realistic problem sizes.

9.1 A Hyper-heuristic with Adaptive Learning Periods

We have seen that the Generalised Random Gradient hyper-heuristic is able to learn to prefer certain low-level heuristics throughout the optimisation process (Chapter 4), is able to achieve optimal performance on LEADINGONES (Chapter 5) and that the performance improves the larger the set of low-level heuristics on LEADINGONES (Chapter 8).

These previous results were achieved with a smart, static, a priori choice of the learning period τ. In this section, we introduce an updated version of the Generalised Random Gradient hyper-heuristic where we aim for it to automatically adjust the learning period τ throughout the run. Apart from removing the burden of having to identify an appropriate value for the learning period, the best parameter value may change at different stages of the optimisation process, hence no fixed value of τ may be optimal.

In particular, we modify the Generalised Random Gradient hyper-heuristic by introducing a simple self-adjusting mechanism inspired by the $1/5^{th}$ rule from continuous optimisation (Beyer and Schwefel, 2002). The main idea is that the learning period τ should be large enough for the current best low-level heuristic to succeed, but also small enough such that sub-optimal heuristics fail. We define a heuristic to be *successful* if it achieves at least σ fitness improvements during the learning period τ. If less than σ improvements occur (i.e., the heuristic fails), then τ is increased by a multiplicative factor $F^{1/\sigma}$ and a new operator is chosen at random. Otherwise, once the heuristic is successful, τ it is decreased by a smaller factor F^{1/σ^2} and a new learning period is started with the successful heuristic. This self-adjusting rule strives to adapt the learning period such that a failure occurs after approximately every σ successes, and to maintain a success probability of $1-1/\sigma$, i.e., a $1-o(1)$ rule. In contrast to the one success after several failures of traditional 1/5 rule algorithms (Doerr and Doerr, 2015; Kern et al., 2004), the innovation behind the $1-o(1)$ rule is that it aims to achieve many successes before a failure to maintain a high success probability. The parameter F should be a constant greater than 1 (previously 1.5 has been used in discrete search spaces (Doerr et al., 2015)). The Adaptive Random Gradient hyper-heuristic is formally described in Algorithm 15.

9.2 Proof of Optimality on LEADINGONES

In this section, we will prove that the Adaptive Random Gradient hyper-heuristic has optimal runtime for the LEADINGONES (LO) benchmark function, up to lower order terms. Recall:

$$\textsc{LeadingOnes}(x) := \sum_{i=1}^{n}\prod_{j=1}^{i} x_j.$$

Algorithm 15 Adaptive Random Gradient Hyper-heuristic

```
1: $\tau \leftarrow \tau_0$
2: Choose $x \in S$ uniformly at random
3: while optimum not found do
4:     Choose $h \in H$ uniformly at random
5:     $c_t \leftarrow 0$; $c_s \leftarrow 0$
6:     while $c_t < \tau$ do
7:         $c_t \leftarrow c_t + 1$; $x' \leftarrow h(x)$
8:         if $f(x') > f(x)$ then
9:             $c_s \leftarrow c_s + 1$; $x \leftarrow x'$
10:        end if
11:        if $c_s \geq \sigma$ then
12:            $c_s \leftarrow 0$; $c_t \leftarrow 0$
13:            $\tau \leftarrow \tau \cdot F^{-1/\sigma^2}$
14:        end if
15:    end while
16:    $\tau \leftarrow \tau \cdot F^{1/\sigma}$
17: end while
```

We consider the expected runtime of the Adaptive Random Gradient hyper-heuristic, with access to the same 1-bit and 2-bit mutation operators, 1BITFLIP and 2BITFLIP, which respectively flip one and two bits of the bit-string chosen uniformly at random with replacement, as used by Alanazi and Lehre (2014) and Chapters 4, 5, 6 and 8.

Adaptive Random Gradient picks an initial mutation operator uniformly at random and applies it for a number of iterations (the learning period τ). If the mutation operator produces at least σ improvements within these τ iterations, the learning period τ is decreased to $\tau \cdot F^{-1/\sigma^2}$, and the same operator is applied again. If fewer than σ improvements are produced in τ iterations, τ is increased to $\tau \cdot F^{1/\sigma}$, and a mutation operator is once again chosen uniformly at random.

Recall from Chapter 5 that the improvement probabilities (i.e., the probability that a mutation produces an individual with a higher fitness value than its ancestor) of the two operators on an individual with a LEADINGONES value of i are: $p_1(i) = \frac{1}{n}$ and $p_2(i) = \frac{2(n-i-1)}{n^2}$. We refer to the operator with the greatest probability of producing an improvement at the current search point (i.e., 2BITFLIP for $i < \frac{n-1}{2}$, and 1BITFLIP for $i \geq \frac{n-1}{2}$) as the *optimal* operator, and use $p_{\text{opt}}(i)$ to denote its improvement probability. Conversely, we call the operator with the smaller improvement probability *non-optimal*, and use $p_{\neg\text{opt}}(i)$ to denote its probability of producing an improvement in one iteration.

Recall from Theorem 5.1 in Chapter 5 that the best-possible expected runtime on LEADINGONES for an algorithm using only the 1BITFLIP and 2BITFLIP operators is

$\mathrm{E}(\mathrm{T_{OPT}}) = \frac{1+\ln(2)}{4}n^2 + O(n) \approx 0.42329n^2$. Theorem 9.1 shows that the Adaptive Random Gradient hyper-heuristic matches this best-possible performance (i.e., it is optimal for LEADINGONES). The proof requires Lemmata 9.2, 9.3, 9.4, 9.5 and 9.6, which are stated and proved after the theorem.

Theorem 9.1 *The expected runtime of the Adaptive Random Gradient hyper-heuristic on* LEADINGONES *for* $\tau_0 = 1$, $\sigma = \Omega(\log^4 n) \cap o\left(\sqrt{n/\log n}\right)$ *and* $F > 1$ *a constant, is*

$$\mathrm{E}(\mathrm{T_{ARG}}) \leq \left(\frac{1+\ln(2)}{4}\right) n^2 + o(n^2).$$

Proof: We call a period of at most τ iterations in which the mutation operator produces σ LEADINGONES improvements by mutation a *successful phase*, and a period of τ iterations in which the mutation operator produces less than σ LEADINGONES improvements a *failed phase*. We will bound $\mathrm{T_{ARG}} = \mathrm{T_{mid}} + \mathrm{T_S} + \mathrm{T_{NS}} + \mathrm{T_F}$ by bounding each of the four contributing components:

- $\mathrm{T_{mid}}$, the number of iterations spent in the 'middle region', where $|\mathrm{LO}(x) - (\frac{n-1}{2})| < \beta n$, and $\mathrm{LO}(x)$ is the LEADINGONES value of the current solution. By Lemma 9.4, $\mathrm{E}(\mathrm{T_{mid}}) = o(n^2)$ for any $\beta = o(1)$.

- $\mathrm{T_S}$, the number of iterations spent in successful phases applying the optimal operator outside the middle region,

- $\mathrm{T_{NS}}$, the number of iterations spent in successful phases applying the non-optimal operator outside the middle region,

- $\mathrm{T_F}$, the number of iterations spent in failed phases outside the middle region.

To prove the theorem we will bound $\mathrm{E}(\mathrm{T_S}) \leq \mathrm{E}(\mathrm{T_{OPT}})$ and show that the expected values of the other contributing terms are at most $o(n^2)$.

We define $\tau_{\max}(i) := \left(1 + \frac{4}{\ln n}\right) \cdot \frac{\sigma}{p_{\mathrm{opt}}(i)}$, where $p_{\mathrm{opt}}(i)$ is the improvement probability of the optimal operator, i is the LEADINGONES value of the current solution and we will make use of the following:

1. With high probability, τ remains below $\tau_{\max}(i)$ throughout the optimisation process per Lemma 9.5.

2. While $|\mathrm{LO}(x) - \left(\frac{n}{2} - 1\right)| > \beta n$ and $\tau < \tau_{\max}(i)$, the non-optimal operator fails a phase with probability $1 - e^{-\Omega(\beta\sigma)}$ per Lemma 9.6.

For T_S, we note that $E(T_S) \leq E(T_{OPT})$, as the optimum would be found in expectation after $E(T_{OPT})$ iterations of applying the optimal mutation operator, while Adaptive Random Gradient can additionally make progress toward the optimum applying the non-optimal operator, as well as in periods which fail to produce σ improvements.

For T_{NS} and T_F, bounds on the number of successful phases with the non-optimal operator as well as the total number of failed phases are needed. Let L_5 denote the event that τ remains below $\tau_{\max}(i)$ throughout the optimisation process, N_S and N_F denote the number of successful and failed phases (respectively) that occur before the global optimum is constructed, and τ_{end} be the value of τ when the global optimum is constructed. N_F can be bounded by observing that, regardless of the order of the phases, the following balance relation is valid: $\tau_{\text{end}} = \tau_0 F^{N_F/\sigma - N_S/\sigma^2}$.

Conditional on L_5, we have $\tau_{\text{end}} < \tau_{\max}(n)$, and hence given that $\tau_0 = 1$,

$$\log_F \tau_{\max}(n) \geq \frac{N_F}{\sigma} - \frac{N_S}{\sigma^2}.$$

As each successful phase provides at least σ LEADINGONES improvements, we can bound $N_S \leq \frac{n}{\sigma}$, and then bound N_F given that $\tau_{\max}(n) = O(\sigma n)$:

$$N_F \leq \frac{N_S}{\sigma} + \sigma \log_F(\tau_{\max}(n)) = O\left(\frac{n}{\sigma^2} + \sigma \log n\right).$$

This bounds the number of iterations spent in failed phases:

$$E(T_F \mid L_5) \leq \tau_{\max}(n) \cdot N_F \leq (1 + o(1))\sigma n \cdot O\left(\frac{n}{\sigma^2} + \sigma \log n\right) = o(n^2).$$

Except during the iterations spent in the middle region of the search space (which are counted by T_{mid}), the probability that a non-optimal operator produces σ improvements within a single phase is at most $o(1)$ by Lemma 9.6. Conditional on L_5, there are at most $N_S + N_F = O(n/\sigma)$ phases in total, and hence at most $o(1) \cdot O(n/\sigma) = o(n/\sigma)$ successful phases with the non-optimal operator, which, combined with $\tau < \tau_{\max}(i) \leq \tau_{\max}(n)$ yields a bound on the number of iterations spent in successful phases using the non-optimal operator:

$$E(T_{NS} \mid L_5) \leq \tau_{\max}(n) \cdot o\left(\frac{n}{\sigma}\right) = o(n^2).$$

Combining the four contributing factors, by the linearity of expectation, the expected runtime of Adaptive Random Gradient is:

$$E(T_{ARG} \mid L_5) \leq E(T_S \mid L_5) + E(T_F \mid L_5) + E(T_{NS} \mid L_5) + E(T_{\text{mid}} \mid L_5)$$

$$\leq \mathrm{E}(\mathrm{T}_{\mathrm{OPT}}) + o(n^2),$$

noting that the bounds on the expected values of T_{S} and $\mathrm{T}_{\mathrm{mid}}$ derived previously hold also when conditioned on L_5.

For an unconditional expectation, we use Lemma 9.3 to bound the expected runtime of Adaptive Random Gradient when L_5 fails to hold as $O(\sigma n^3)$. By Lemma 9.5, the probability of τ exceeding $\tau_{\max}(i)$ at any point before the global optimum is found can be made $n^{-c'}$ small for any constant $c' > 0$, and hence, using the law of total expectation for, e.g., $c' = 2$:

$$\begin{aligned}\mathrm{E}(\mathrm{T}_{\mathrm{ARG}}) &= \Pr(L_5)E(\mathrm{T}_{\mathrm{ARG}} \mid L_5) + \Pr(\overline{L_5})E(\mathrm{T}_{\mathrm{ARG}} \mid \overline{L_5}) \\ &= \mathrm{E}(\mathrm{T}_{\mathrm{OPT}}) + o(n^2) + O(n^{-2}) \cdot O(\sigma n^3) \\ &= \mathrm{E}(\mathrm{T}_{\mathrm{OPT}}) + o(n^2),\end{aligned}$$

as $\sigma = o(\sqrt{n/\log n})$. □

Note that the proof above not only shows that Adaptive Random Gradient has (apart from lower-order terms) the optimal runtime achievable with the given set of operators, but also that, with probability $1 - o(1)$, only a fraction ($o(1)$) of the iterations use the non-optimal operator. Hence our self-adjusting mechanism in an extremely good manner manages to select the most suitable mutation operator.

In the rest of this section we will provide the lemmata required to complete the proof of Theorem 9.1. Before this, we present the following technical lemma, which allows us to bound the probability that a process on the non-negative integers which in expectation decreases in value remains above its initial value after a certain amount of time. A similar result has been proven by Kötzing et al. (2015). As witnessed by the $\ln\left(\frac{1}{(a+1)p}\right)$ term, our bound is stronger when the expected movement away from zero is significantly smaller than the expected movement towards zero (note that state zero is not necessarily reflecting). We need this stronger result in Lemma 9.3.

Lemma 9.2 *Let $X_0, X_1, \ldots$ be a random process on the non-negative integers. Assume that there are $a \in \mathbb{N}_{\geq 1}$ and $p \in (0, \frac{1}{e(a+1)})$ such that for all t and all $k \geq 1$, we have $\Pr(X_{t+1} = X_t + a \mid X_t = k) \leq p$ and $\Pr(X_{t+1} = X_t - 1 \mid X_t = k) = 1 - \Pr(X_{t+1} = X_t + a \mid X_t = k)$. Assume further that $X_0 = 0$. Then for all $k \in \mathbb{N}$ and all $t \in \mathbb{N}$, we have $\Pr(X_t \geq k) \leq \exp\Bigg(-\frac{(k-1)(1-(a+1)p)}{a(a+1)}\left(\ln\left(\frac{1}{(a+1)p}\right) - 1\right)\Bigg)K(a+1)$, where K is an absolute constant.*

Proof: Let $k,t \in \mathbb{N}_{\geq 1}$. To have $X_t \geq k$, there must be a $t_0 \in [0..t]$ such that $X_{t_0} = 1$ and $X_{t'} \geq 1$ for all $t' \in [t_0,t]$. Let $Y_t, t \in [t_0..t]$ be a random process with

- $Y_{t_0} = 1$,
- $\Pr(Y_{t+1} = Y_t + a \mid Y_t = k) = \Pr(X_{t+1} = X_t + a \mid X_t = k) \leq p$ and $\Pr(Y_{t+1} = Y_t - 1 \mid Y_t = k) = \Pr(X_{t+1} = X_t - 1 \mid X_t = k)$ for all $k \in \mathbb{N}_{\geq 1}$, and
- $\Pr(Y_{t+1} = Y_t + a \mid Y_t = k) = p$ and $\Pr(Y_{t+1} = Y_t - 1 \mid Y_t = k) = 1 - p$ for all $k \leq 0$.

Note that this process is obtained from the process $(X_t \mid X_{t_0} = 1)$ by modifying it only on the non-positive integers. Consequently, $\Pr(X_t \geq k \wedge \forall t' \in [t_0,t] : X_{t'} \geq 1 \mid X_{t_0} = 1) = \Pr(Y_t \geq k \wedge \forall t' \in [t_0,t] : Y_{t'} \geq 1) \leq \Pr(Y_t \geq k)$. Note that Y_t is stochastically dominated by a random variable $\tilde{Y}_t = 1 + \sum_{t'=t_0}^{t-1} Z_{t'}$ with independent $Z_{t'}$ such that $\Pr(Z_{t'} = a) = p$ and $\Pr(Z_{t'} = -1) = 1 - p$. We have $\mathrm{E}(\tilde{Y}_t) = 1 + (t - t_0)(ap - (1-p))$, $\mathrm{Var}[\tilde{Y}_t] = \sum_{t'=t_0}^{t-1} \mathrm{Var}[Z_{t'}] \leq (t - t_0)(a^2 p - ((ap - (1-p))^2 - (1-p))) \leq (t - t_0)a^2 p$, and trivially $Z_{t'} \leq \mathrm{E}(Z_{t'}) + (a+1)$ for all t'. Note that for $t_0 > t - (k-1)/a$, we trivially have $\Pr(X_t \geq k \mid X_{t_0} = 1) = 0$. Hence let $(t - t_0) \geq (k-1)/a$. By the above, we have $\Pr(X_t \geq k \wedge \forall t' \in [t_0,t] : X_{t'} \geq 1 \mid X_{t_0} = 1) \leq \Pr(\tilde{Y}_t \geq k) \leq \Pr(\tilde{Y}_t \geq 1) = \Pr(\tilde{Y}_t \leq \mathrm{E}(\tilde{Y}_t) - \lambda)$ for $\lambda = -(t - t_0)(ap - (1-p))$. Putting $b := a + 1$, we use the variance-based Chernoff bound (see e.g., Hoeffding (1963) or Theorem 1.12 and the subsequent text in Doerr (2011)) and compute, writing $p = \frac{c}{a+1}$,

$$\begin{aligned}
&\Pr(\tilde{Y}_t \leq \mathrm{E}(\tilde{Y}_t) - \lambda) \\
&\leq \exp\left(-\frac{\lambda}{b}\left(\left(1 + \frac{\mathrm{Var}[\tilde{Y}_t]}{b\lambda}\right)\ln\left(1 + \frac{b\lambda}{\mathrm{Var}[\tilde{Y}_t]}\right) - 1\right)\right) \\
&\leq \exp\left(-(t - t_0)\frac{1 - p(a+1)}{a+1}\left(\ln\left(1 + \frac{(a+1)(1 - p(a+1))}{a^2 p}\right) - 1\right)\right) \\
&\leq \exp\left(-(t - t_0)\frac{1-c}{a+1}\left(\ln\left(\tfrac{1}{c}\right) - 1\right)\right).
\end{aligned}$$

Hence,

$$\begin{aligned}
\Pr(X_t \geq k) &\leq \sum_{t_0=0}^{t-(k-1)/a} \Pr(X_t \geq k \wedge \forall t' \in [t_0,t] : X_{t'} \geq 1 \mid X_{t_0} = 1) \\
&\leq \sum_{t_0=0}^{t-(k-1)/a} \Pr(\tilde{Y}_t \leq \mathrm{E}(\tilde{Y}_t) - \lambda) \\
&\leq \sum_{\delta=0}^{\infty} \exp\left(-\left(\delta + \frac{k-1}{a}\right)\frac{1-c}{a+1}\left(\ln\left(\frac{1}{c}\right) - 1\right)\right)
\end{aligned}$$

$$\begin{aligned}&\leq \exp\left(-\frac{(k-1)(1-c)}{a(a+1)}\left(\ln\left(\frac{1}{c}\right)-1\right)\right)\cdot\sum_{\delta=0}^{\infty}\exp\left(-\frac{1-c}{a+1}\left(\ln\left(\frac{1}{c}\right)-1\right)\right)^{\delta}\\&=\exp\left(-\frac{(k-1)(1-c)}{a(a+1)}\left(\ln\left(\frac{1}{c}\right)-1\right)\right)\cdot K(a+1),\end{aligned}$$

where K can be chosen as an absolute constant (independent from a and c, provided that $c<1/e$). □

Lemma 9.3 *Consider a run of the Adaptive Random Gradient, started with an arbitrary initial search point x and an arbitrary initial period length $\tau_0 \leq n^3$, on the* LEADINGONES *problem. Let* T *be the runtime, that is, the number of fitness evaluations performed up to the point when for the first time the optimal solution is evaluated. Then* $E(T) = O(\sigma n^3)$.

Proof: While the fitness is less than $n-1$, each of the two operators has a probability of at least $1/n^2$ of finding an improvement. Hence by a simple fitness level argument, the time T_0 to reach a fitness of at least $n-1$ satisfies $E(T_0) \leq n^3$.

Throughout this first part of the optimisation process, we have that a period starting with a τ-value of n^3 or more is successful with probability $1-\exp(-\Theta(n/\sigma)) =: 1-p$. This is because the expected number of improvements is at least τ/n^2, whereas for a success we need only σ improvements. Hence, a Chernoff bound for geometrically distributed random variables (Theorem 2.10) shows this claim.

Let i be minimal such that $\tau' := \tau_0 F^{i/\sigma^2} \geq n^3$. In other words, τ' is the smallest τ value not smaller than n^3 which we could encounter in this run of the algorithm. For any time t, let $X_t = \max\{0, \log_{F^{1/\sigma^2}}(\tau_t/\tau')\}$. In other words, if $\tau_t \geq \tau'$, then X_t is such that $\tau_t = \tau'(F^{1/\sigma^2})^{X_t}$; otherwise $X_t = 0$. By definition, we have $X_0 = 0$. Also, for all $t \geq 0$ and all $k \geq 1$, we have $\Pr(X_{t+1} = X_t + \sigma \mid X_t = k) \leq p$ and $\Pr(X_{t+1} = X_t - 1 \mid X_t = k) = 1 - \Pr(X_{t+1} = X_t + \sigma \mid X_t = k)$. By Lemma 9.2, we have

$$\begin{aligned}\Pr(X_t \geq k) &\leq \exp\left(-\frac{(k-1)(1-(\sigma+1)p)}{\sigma(\sigma+1)}\cdot\left(\ln\left(\frac{1}{(\sigma+1)p}\right)-1\right)\right)\Theta(\sigma+1)\\&=\exp\left(-\Theta\left(\frac{nk}{\sigma^3}\right)\right)\end{aligned}$$

for all $t,k \geq 1$, where the implicit constants can be chosen independently of n,k,σ provided that n is sufficiently large. We use this to compute $E(\tau_{T_0}) \leq \sum_{k=0}^{\infty}\Pr(X_{T_0} = k)\tau' F^{k/\sigma^2} \leq \tau' + \sum_{k=1}^{\infty}\tau' F^{k/\sigma^2}\exp(-\Theta(nk/\sigma^3)) = O(\tau') = O(n^3)$.

We shall use this estimate of τ_{T_0}, the τ-value at time T_0, to obtain an estimate for the remaining runtime T_1 starting from time T_0, that is, when the fitness reached or exceeded

$n-1$. If at time T_0 we already have a fitness of n, then $T_1 = 0$ and there is nothing to show. So let us assume that the fitness at time T_0 is $n-1$ and estimate the time it takes to find the last missing bit. Observe that in this situation, 2BITFLIP has no chance to find the missing bit, whereas 1BITFLIP has a probability of $\frac{1}{n}$. If we find the missing bit in the current period (that is, in the period in which we reached a fitness of $n-1$), then we have $T_1 \leq \tau_{T_0}$. Otherwise, for each of the following periods P_i, $i = 1, 2, \ldots$ up to the point when the optimum is found, the following holds. (i) The τ-value $\tau^{(i)}$ in period P_i is exactly $\tau^{(i)} = \tau^{(1)} F^{(i-1)/\sigma}$ for $i > 1$ and satisfies $\tau^{(1)} \leq \tau_{T_0} F^{1/\sigma}$. (ii) If the optimum has not been found earlier, then with probability $q_i := \frac{1}{2}\left(1 - \left(1 - \frac{1}{n}\right)^{\tau^{(i)}}\right)$, the optimum is found in P_i. Note, trivially, that if the optimum is found in P_i, then $T_1 \leq \tau_{T_0} + \sum_{j=1}^{i} \tau^{(j)}$.

Let r_i be the probability that the optimum is found in P_i and let $R_i := \sum_{j=i}^{\infty} r_i$ be the probability that it is found in period P_i or later. Let $i_0 \geq 1$ be minimal such that $\tau^{(i_0)} \geq n$. Note that for each $i \geq i_0$, we have $q_i \geq \frac{1}{2}(1 - \frac{1}{e})$. Consequently, $R_i \leq \min\{1, (1 - \frac{1}{2}(1 - \frac{1}{e}))^{i - i_0}\}$. This gives $E(T_1 \mid \tau_{T_0}) \leq \sum_{i=1}^{\infty} r_i (\tau_{T_0} + \sum_{j=1}^{i} \tau^{(j)}) = \tau_{T_0} + \sum_{j=1}^{\infty} \tau^{(j)} \sum_{i=j}^{\infty} r_j = \tau_{T_0} + \sum_{j=1}^{\infty} R_j \tau^{(j)}$. We estimate $\sum_{j=1}^{i_0} R_j \tau^{(j)} \leq \sum_{j=1}^{i_0} \tau^{(j)} = \tau^{(1)} \sum_{j=0}^{i_0 - 1} F^{j/\sigma} = \tau^{(1)} \frac{F^{i_0/\sigma} - 1}{F^{1/\sigma} - 1} \leq \tau^{(1)} F^{(i_0)/\sigma} \sigma / \ln(F) = \frac{\sigma F}{\ln(F)} \tau^{(i_0)}$, where we used the estimate $F^{1/\sigma} - 1 \geq \ln(F)/\sigma$. We estimate $\sum_{j=i_0+1}^{\infty} R_j \tau^{(j)} \leq \sum_{j=1}^{\infty} (1 - \frac{1}{2}(1 - \frac{1}{e}))^j F^{j/\sigma} \tau^{(i_0)} \leq C \tau^{(i_0)}$ for some constant C (assuming that n is sufficiently large). Note that $\tau^{(i_0)} \leq \max\{n F^{1/\sigma}, \tau^{(1)}\} \leq \max\{n, \tau_{T_0}\} F^{1/\sigma}$. Consequently, $E(T_1 \mid \tau_{T_0}) = O(\sigma(n + \tau_{T_0}))$. By the law of total expectation, we have $E(T_1) = O(\sigma(n + E(\tau_{T_0}))) = O(\sigma n^3)$.

□

Lemma 9.4 *The expected runtime of any combination of* 1BITFLIP *and* 2BITFLIP *for the* $|LO(x) - (\frac{n-1}{2})| < \beta n$ *region of the search space, where* $\beta = 1/\sqrt[4]{\ln(n)} = o(1)$, *is at most* $o(n^2)$.

Proof: This region contains $2\beta n = o(n)$ fitness values. The expected waiting time for an improvement within this region is smaller than $1/p_{\neg\text{opt}}(3n/4 - 1) = 4n$. This is not within this region and will provide an upper bound on the expected wait time, as the expected waiting time for each fitness improvement increases as $\text{LO}(x)$ increases. Hence, a simple artificial fitness level argument (see Section 2.5) gives that the region is optimised after $2\beta n \cdot 4n = o(n^2)$ iterations in expectation. □

Lemma 9.5 *With high probability (at least $1-n^{-c'}$ for any $c'>0$), Adaptive Random Gradient finds the global optimum before $\tau \geq \tau_{\max}(i)$ occurs, where i is the* LEADINGONES *value of the ancestor individual in any given iteration.*

Proof: We prove the claim by showing that at any time of the algorithm ($\mathrm{LO}(x)=i$) where we have $\tau \leq \tau_1(i) = (1+1/\ln(n))\cdot\sigma/p_{\mathrm{opt}}(i)$, and then failures increase τ over $\tau_1(i)$, the probability that the algorithm increases τ to over $\tau_2(i)=\tau_1(i)\cdot F^{(\log_2(n))^2/\sigma} < \tau_{\max}(i)$ or more without first reaching or going below $\tau_1(i)$ is at most n^{-c} for any $c>0$. We then prove that, with probability at least $1-n^{-c'}$, this does not happen at any point throughout the optimisation process.

We first show that $\tau_2(i) < \tau_{\max}(i)$. Note that, by a Maclaurin series argument (see e.g., Mitzenmacher and Upfal 2005), $F^{(\log_2(n))^2/\sigma} = 1+\frac{\ln(F)(\log_2(n))^2}{\sigma}+o\left(\frac{\ln(F)(\log_2(n))^2}{\sigma}\right) = 1+O((\log(n))^{-2})$, and hence:

$$\begin{aligned}\tau_2(i) &= \tau_1(i)\cdot F^{(\log_2(n))^2/\sigma}\\ &= (1+O((\log(n))^{-2}))\cdot(1+1/\ln(n))\cdot\sigma/p_{\mathrm{opt}}(i)\\ &\leq (1+1/\ln(n)+o(1/\ln(n)))\cdot\sigma/p_{\mathrm{opt}}(i)\\ &\leq (1+4/\ln(n))\cdot\sigma/p_{\mathrm{opt}}(i) = \tau_{\max}(i).\end{aligned}$$

To increase τ to $\tau > \tau_2(i)$ from $\tau \leq \tau_1(i)$, at least $(\log_2(n))^2$ failures are necessary; hence, at least $(\log_2(n))^2$ random operator choices are necessary. With probability at least $1-n^{-\log_2(n)}$, at least one of these choices is with the optimal operator.

With overwhelming probability, the LEADINGONES value of the current solution is increased by no more than 3σ in a successful phase. Let Y be a binomially-distributed random variable with parameters $\tau_1(i)$ and $p_{\mathrm{opt}}(i+3\sigma)$; the number of improvements produced by the optimal operator in a period of $\tau \geq \tau_1(i)$ iterations stochastically dominates Y. Applying a classic multiplicative Chernoff bound (see, e.g., Theorem 2.10 in Chapter 2) with $\mathrm{E}(Y) \geq \mu := \left(1+\frac{1}{\ln(n)}\right)\cdot\left(\frac{n-6\sigma}{n}\right)\cdot\sigma = \Theta(\sigma)$ and $\delta := \frac{n-6\sigma\ln(n)-6\sigma}{(1+\ln(n))(n-6\sigma)} = \Theta(1/\log(n))$ (such that $(1-\delta)\mu=\sigma$) yields

$$\begin{aligned}&\Pr(Y<\sigma) \leq \Pr(Y\leq\sigma)\\ &\leq \exp\left(-\left(\frac{n-6\sigma\ln(n)-6\sigma}{(1+\ln(n))(n-6\sigma)}\right)^2\cdot\left(1+\frac{1}{\ln(n)}\right)\cdot\frac{n-6\sigma}{n}\cdot\frac{\sigma}{2}\right)\\ &= \exp\left(-\Theta\left(\frac{\sigma}{\log^2(n)}\right)\right) = n^{-\Omega(\log(n))},\end{aligned}$$

since $\sigma = \Omega(\log^4(n))$.

Adaptive Random Gradient requires $\sigma \cdot (\log_2(n))^2$ consecutive successes by the optimal operator to cancel out the increase in τ by $(\log_2(n))^2$ failures from the non-optimal operator. Given a success probability of at least $1 - n^{-\Omega(\log(n))}$, the probability of $\sigma \cdot (\log_2(n))^2$ consecutive successes given that each success contributes a fitness increase of at most 3σ (denoted as the event Y') is, by a union bound,

$$\begin{aligned}\Pr(Y') &\geq \left(1 - (1 - n^{-\Omega(\log n)}) \cdot n^{-\Omega(\log(n))} - n^{-\Omega(\log n)}\right)^{\sigma(\log_2(n))^2} \\ &\geq 1 - 2 \cdot n^{-\Omega(\log(n))} \cdot \sigma(\log_2(n))^2 = 1 - n^{-\Omega(\log(n))}.\end{aligned}$$

Hence, the probability of τ exceeding $\tau_2(i) < \tau_{\max}(i)$ before returning below $\tau_1(i)$ (denoted as the event Z) is at most

$$\Pr(Z) \geq \left(1 - n^{-\Omega(\log(n))}\right) \cdot \left(1 - n^{-\log_2(n)}\right) \geq 1 - 2n^{-\Omega(\log(n))}.$$

There will be at most n/σ successful phases throughout the optimisation process, and hence at most n/σ times $\tau < \tau_1(i)$ occurs (which invokes the previous argument). Hence, the probability of τ not exceeding $\tau_{\max}(i)$ throughout the optimisation process (denoted as the event L_5) is at most

$$\begin{aligned}\Pr(L_5) &\geq \left(1 - 2n^{-\Omega(\log(n))}\right)^{n/\sigma} \geq 1 - 2n^{-\Omega(\log(n))} \cdot n/\sigma \\ &= 1 - n^{-\Omega(\log(n))},\end{aligned}$$

which is greater than $1 - n^{-c'}$ for any constant $c' > 0$ when n is sufficiently large. □

Lemma 9.6 *While $\tau < \tau_{\max}(i)$ and $|\mathrm{LO}(x) - (n/2-1)| > \beta n$, $\beta = 1/\sqrt[4]{\ln(n)} = o(1)$, the non-optimal mutation operator produces at least σ improvements within τ iterations with probability at most $o(1)$.*

Proof: We prove the claim by showing that at $i = n/2 - 1 \pm \beta n$, the non-optimal mutation operator produces σ improvements within τ iterations with probability at most $o(1)$. Clearly, for any point for which $i < n/2 - 1 - \beta n$ and $i > n/2 - 1 + \beta n$ will have the same, or lower success probability, since $p_{\neg\mathrm{opt}}(i) = 1/n$ for $i < n/2$, and is a decreasing function for $i \geq n/2 - 1$.

Let Y be a binomially-distributed random variable with parameters $\tau_{\max}(n/2 - 1 - \beta n) = (1 + 4/\ln(n)) \cdot \frac{\sigma n}{2\beta+1}$ and $p_{\neg\mathrm{opt}}(n/2 - 1 - \beta n) = 1/n$. Since $p_{\neg\mathrm{opt}}(i)$ is a decreasing function, the number of improvements produced by the non-optimal operator in a period of $\tau < \tau_{\max}(n/2-1-\beta n)$ iterations is stochastically dominated by Y. Applying the classic multiplicative Chernoff bound with $\mu := \mathrm{E}(Y) = (1+4/\ln(n)) \cdot \frac{\sigma}{2\beta+1}$, and $\delta := \frac{2\ln(n)\beta-4}{\ln(n)+4}$ (such that $(1+\delta)\mu = \sigma$) yields

$$\begin{aligned}\Pr(Y \geq \sigma) &\leq \exp\left(-\left(\frac{2\ln(n)\beta-4}{\ln(n)+4}\right)^2 \cdot \left(1+\frac{4}{\ln(n)}\right) \cdot \frac{\sigma}{2\beta+1} \cdot \frac{1}{3}\right) \\ &= \exp(-\Theta(\beta\sigma)) = o(1).\end{aligned}$$

A similar argument holds at $i = n/2-1+\beta n$ where a binomially-distributed random variable Y' with parameters $\tau_{\max}(n/2-1+\beta n) = (1+4/\ln n) \cdot \sigma n$ and $p_{\neg\mathrm{opt}}(n/2-1+\beta n) = (1-2\beta)/n$ stochastically dominates the number of improvements produces by the non-optimal operator in $\tau < \tau_{\max}(n/2-1+\beta n)$ iterations. A Chernoff bound with $\mu := \mathrm{E}(Y) = (1+4/\ln(n)) \cdot (1-2\beta) \cdot \sigma$ and $\delta := \frac{2(\beta\ln(n)+4\beta-2)}{(1-2\beta)\ln(n)-(8\beta+4)}$ (such that $(1+\delta)\mu = \sigma$) clearly yields the same result: $\Pr(Y' \geq \sigma) = o(1)$. $\square$

In the next section we will provide an experimental analysis of the Adaptive Random Gradient hyper-heuristic for realistic problem sizes and we will analyse how the value of τ adapts as well as the importance of the parameter σ.

9.3 Experimental Supplements

In the previous section, we proved that for large enough problem sizes n, the Adaptive Random Gradient hyper-heuristic has the best performance achievable with the available low-level heuristics for LEADINGONES, up to lower order terms. In this section we perform a set of experiments to assess the performance of Adaptive Random Gradient for realistic problem sizes. Our theoretical results rely on the adaptation parameter σ to grow slowly with the problem size, i.e., asymptotically between $\Omega(\log^4 n)$ and $o(\sqrt{n/\log n})$. We experiment with various super-constant values within and outside the theoretical bounds. We introduce multiplicative constants c^* such that σ is sufficiently small compared to n, for small problem sizes. We decide to arbitrarily set c^* such that for our smallest problem size ($n = 10^3$), $\sigma = 4$. We set the initial learning period trivially to $\tau_0 = 1$. We then set $F = 1.5$ as suggested by Doerr and Doerr (2015) and run all algorithms 100 times.

In Figure 9.1, we plot the average runtimes of the Adaptive Random Gradient hyper-heuristic as the problem size increases and compare their performance against the Generalised Random Gradient hyper-heuristic (introduced in Chapter 4 and analysed on LEADINGONES

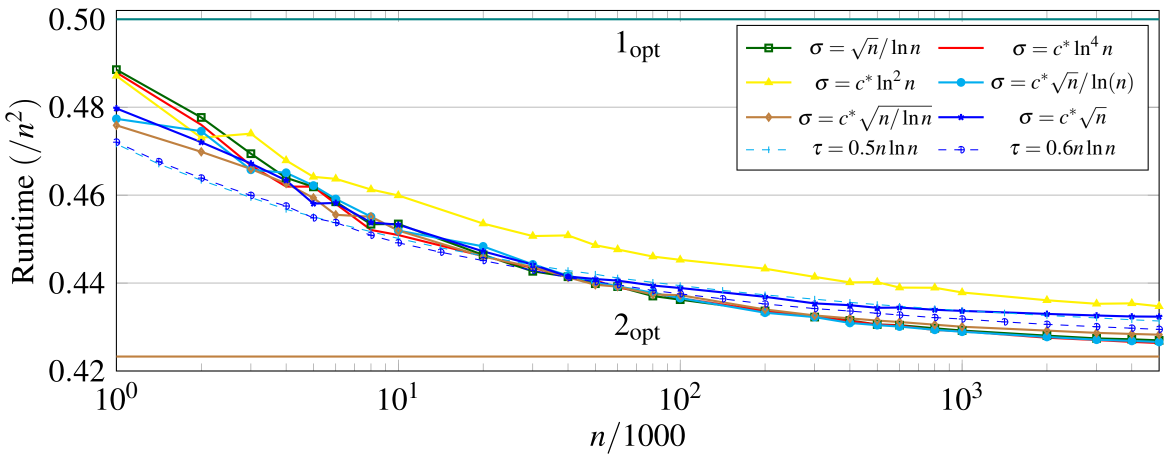

Fig. 9.1 Average number of fitness function evaluations required by the Adaptive Random Gradient and Generalised Random Gradient hyper-heuristics (with parameters σ and τ respectively) to find the LEADINGONES optimum as the problem size n increases. 1_{opt} and 2_{opt} show the best runtime achievable by using respectively one low-level heuristic or both.

in Chapter 5) with fixed τ values. We plot the best performing versions of Generalised Random Gradient identified in Chapter 5 (i.e., $\tau = 0.5 n \ln n$ and the best, $0.6 n \ln n$, for $n = 100{,}000$). While the runtimes of all the Adaptive Random Gradient hyper-heuristics are comparable with the three Generalised Random Gradient variants, some outperform them. These include all the ones with σ values within the theoretical bounds. Hence, for realistic problem sizes, better runtimes can be achieved by adapting τ during the run.

Figure 9.2 shows the adaptation of τ for five individual runs for $n = 10^7$ of the Adaptive Random Gradient hyper-heuristic with $\sigma = \sqrt{n}/\ln n$, as well as an average over 100 runs. As expected, the learning period τ is quickly increased into the range where the optimal operator produces more than σ improvements per τ iterations in expectation. The adapted learning period tracks the increasing waiting times for an improvement by the optimal operator in the first half the search space, while remaining stable in the second half where the waiting time does not change. τ_{max} is included to show how the parameter τ adapts as suggested within the proof of Theorem 9.1. The average runtime of Adaptive Random Gradient over 100 runs in this setting was $0.42634n^2$.

Figure 9.3 shows the percentage of iterations where the optimal operator is used by Adaptive Random Gradient with $\sigma = \sqrt{n}/\ln n$ for problem sizes $n = 10^5$ and $n = 10^7$, averaged over 100 independent runs. We divide the $n+1$ fitness values into 100 ranges and plot the percentage of iterations where the optimal operator is employed. We see that Adaptive Random Gradient exhibits the behaviour predicted by Theorem 9.1, already for these problem sizes. Naturally, as n increases, the curves are smoother and the optimal

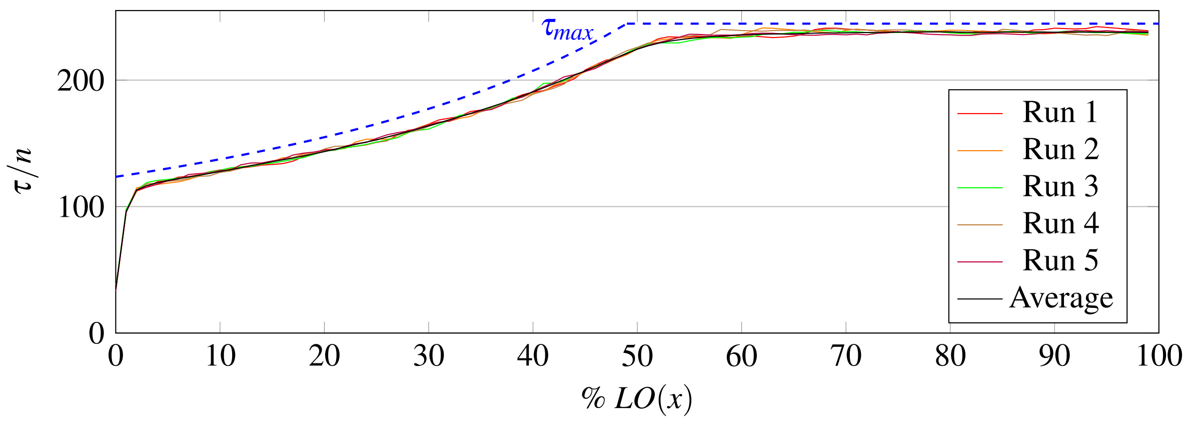

Fig. 9.2 Adapted value of τ over time in five typical runs for $\sigma = \sqrt{n}/\ln n$, $n = 10^7$, and an average over 100 runs.

operator is used more often. As expected, both operators are used in the middle section where they have similar improvement probabilities. Outside this area, the optimal operator is used most of the time as desired.

9.4 Chapter Summary and Discussion

In this chapter we have presented an Adaptive Random Gradient hyper-heuristic that automatically adjusts the learning period throughout the run. Adaptive Random Gradient uses an innovative $1-o(1)$ self-adjusting rule that strives to adapt the learning period such that a failure occurs after approximately every $\omega(1)$ successes. The novelty consists of seeking many successes before a failure in contrast to a success after many failures of traditional adaptive algorithms (Doerr and Doerr, 2015; Kern et al., 2004).

We have rigorously proved that Adaptive Random Gradient optimises LEADINGONES in the best, up to lower order terms, runtime achievable using the 1BITFLIP and 2BITFLIP low level heuristics. Our proof also shows that with probability $1-o(1)$, only a fraction ($o(1)$) of the iterations use the non-optimal operator. Hence, the optimal operator is used most of the time as desired.

We believe, without proof, that analogous results would also hold for larger sets of low-level heuristics, e.g., for all KBITFLIP heuristics for k from some constant-size set of alternatives (similar to the results presented in Chapter 8). The reason is that again, after excluding a constant number of lower-order ranges of the objective space where two operators have similar improvement probabilities (as the middle range was for 1BITFLIP and 2BITFLIP), the best operator is sufficiently better than the others and the adjusting mechanism is sufficiently sensitive so that the τ-value stays in a range in which essentially

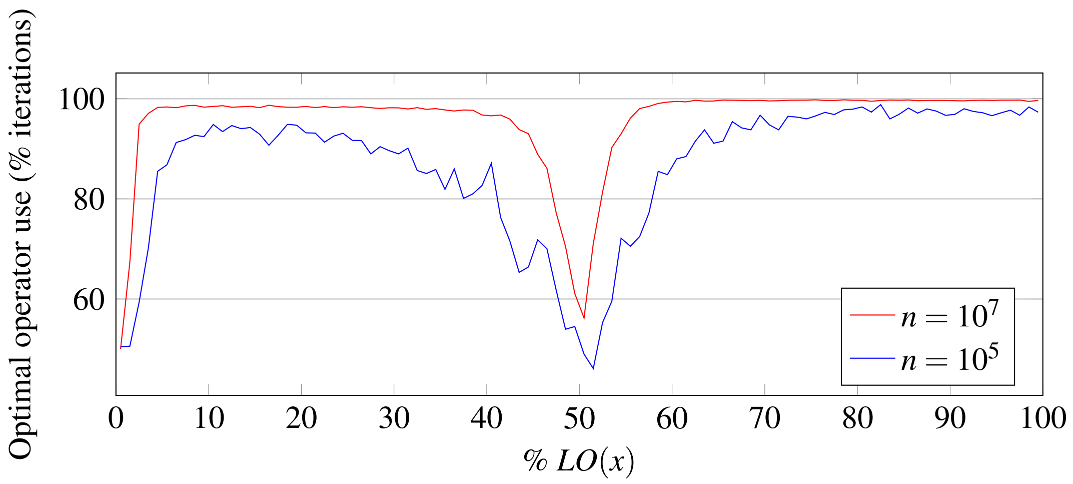

Fig. 9.3 Percentage of the iterations Adaptive Random Gradient applies the optimal mutation operator for $\sigma = \sqrt{n}/\ln n$, average over 100 runs. The dip around the middle of the graph corresponds to the middle region of the optimisation process, where both operators perform similarly well.

only the optimal operator can lead to successful phases. We believe that also non-constant numbers of operators could be handled by our algorithm, again giving optimal performance apart from lower order terms.

We have complemented the theory with experiments for realistic problem sizes. The results show that the parameter σ, indicating the ratio of successes to failures that maintains a stable τ value (i.e., σ successes will cancel out a single failure), is fairly robust. If it is set within the range of values predicted by our theoretical analysis, then Adaptive Random Gradient outperforms the best hyper-heuristics with fixed τ reported in the literature.

Chapter 9 is based on the following publication:

1. Doerr, B., Lissovoi, A., Oliveto, P. S., Warwicker, J. A. (2018). On the Runtime Analysis of Selection Hyper-heuristics with Adaptive Learning Periods. *Proceedings of the Annual Conference on Genetic and Evolutionary Computation (GECCO '18)*, pages 1015-1022. ACM.

Part III

Conclusions

Chapter 10

Conclusions and Outlook

This thesis has provided the theoretical underpinnings to the understanding of selection hyper-heuristics. Our work has significantly expanded on previous theoretical results (Alanazi and Lehre, 2014, 2016; Lehre and Özcan, 2013; Qian et al., 2016) and brought an understanding of the performance of hyper-heuristics on a range of problem classes and their application as general-purpose problem solvers.

10.1 Summary of the Results

In Chapter 2, we introduced the broad field of Evolutionary Computation and the importance of general-purpose problem solvers. If an optimisation problem is not well understood, it is necessary to apply a general-purpose algorithm which requires no problem-specific knowledge. Evolutionary Algorithms (EAs) are a class of general-purpose algorithms which take inspiration from biological mechanisms. The theoretical analysis of stochastic search algorithms is an important tool to understand the performance of EAs and allows us to answer questions regarding the expected runtime of EAs, on which classes of problems EAs are efficient or inefficient and how the setting of the inherent parameters (including population size, mutation rate and selection pressure) affects their performance. We discussed the techniques available for the rigorous runtime analysis of EAs in Section 2.5. The field of hyper-heuristics was introduced in Chapter 3, as well a discussion on parameter setting techniques and the importance of choosing the parameters of an algorithm smartly. Regarding hyper-heuristics, we discussed their classification, the methodologies that have been used in the literature, examples of successful applications and existing theoretical results. It is well understood that algorithms may be effective on one class of problems, and ineffective on others, and thus it is necessary to understand when and why a given algorithm is efficient (or inefficient) and the role the parameters play. By learning to select and generate high-performing heuristics, the aim of hyper-heuristics is to automate the algorithm and parameter selection process. We have derived a plethora of rigorous results regarding hyper-heuristics in this thesis.

While many of the commonly used hyper-heuristics in their current format essentially choose low-level heuristics at random, a clever choice of low-level heuristics and smart methodologies to exploit these choices can lead to improved performance on a range of unimodal and multimodal problem classes where single-operator mechanisms are already efficient for the problem at hand. This suggests that our work will bring a deeper understanding of how to apply hyper-heuristics and their importance as general-purpose problem solvers.

In Chapter 4 we introduced a generalised framework for two well-established hyper-heurstics, namely Generalised Greedy and Generalised Random Gradient. As part of this framework, both generalised hyper-heuristics kept their original heuristic decision process and exploited the chosen heuristic over a 'learning period' of length τ, rather than in a single iteration. The goal of the generalised hyper-heuristics is to exploit good heuristic choices. We motivated the use of hyper-heuristics with extended learning periods (choosing between two mutation operators) by presenting an example function called GENERALISEDGAPPATH-WITHTRAPS where it is necessary to learn to prefer one operator over another. We saw in Chapter 5 that the hyper-heuristics without a learning period essentially choose low-level heuristics at random in each iteration and perform worse in expectation than Randomised Local Search, one of the inherent low-level heuristics. However, the Generalised Greedy hyper-heuristic equipped with a smart choice of learning period (choosing between two mutation operators) is able to outperform those without a learning period and the effective single operator algorithm Randomised Local Search on the benchmark function LEADINGONES. On the same function, the Generalised Random Gradient hyper-heuristic (choosing between two mutation operators) is able to achieve the best-possible performance achievable with the available low-level heuristics, up to lower order terms.

Based on these theoretical insights, the Generalised Greedy Gradient hyper-heuristic, which combines the learning mechanisms of the Greedy and Random Gradient hyper-heuristics, was introduced in Chapter 6. By making a smart heuristic choice and exploiting the heuristic so long as it is successful, the goal is to see improved performance upon the other generalised hyper-heuristics. We proved that Generalised Greedy Gradient (choosing between two mutation operators) is able to achieve the same best-possible performance, up to lower order terms, on LEADINGONES as Generalised Random Gradient. Furthermore, experiments for different problem sizes show Generalised Greedy Gradient is able to outperform Generalised Random Gradient for a wide range of values of the learning period.

We also presented the first theoretical analyses of selection-based hyper-heuristics for multimodal optimisation in Chapter 7, considering a move-acceptance hyper-heuristic (MAHH_{OI}) choosing between elitist and non-elitist selection operators applied to two classes of multimodal functions. We saw that MAHH_{OI} is able to find the best-possible asymptotic expected performance for (unary and unbiased) problem-independent search heuristics on the benchmark function class CLIFF, for instances that elitist algorithms such as the (1+1) EA find difficult. MAHH_{OI} also outperforms in expectation the well-studied non-elitist METROPOLIS algorithm on Cliff and also on the benchmark function class JUMP, despite MAHH_{OI} also having inefficient performance.

Moreover, we have also analysed hyper-heuristics with more realistic sets of low-level heuristics, compared to previous theoretical analyses which have considered hyper-heuristics selecting between just two low-level heuristics. In Chapter 8 we considered the commonly used 'simple' hyper-heuristic mechanisms and the Generalised Random Gradient hyper-heuristic, when given access to a set of $k \geq 2$ low-level heuristics. Whereas the performance of the simple mechanisms (those without a learning period) deteriorates with more operators, the Generalised Random Gradient hyper-heuristic is able to match the leading constant in the best possible expected runtime on LEADINGONES given the operators available. In particular, Generalised Random Gradient with access to k low-level heuristics is able to outperform, in expectation, any algorithm, including the best-possible one, using $m < k$ of the low-level heuristics as operators.

We have also followed a recent trend in the parameter control field in Chapter 9 where we incorporated, within the hyper-heuristic framework, an update rule to automatically adapt the value of the learning period of the Random Gradient hyper-heuristic as it progresses throughout the search space. The new hyper-heuristic, Adaptive Random Gradient, was designed such that it should use the optimal operator with high probability in each iteration. We rigorously proved that this is the case on LEADINGONES when choosing from two stochastic mutation operators of different neighbourhood size. Furthermore, the value of the learning period was adapted successfully and Adaptive Random Gradient was able to achieve the best-possible expected performance achievable with the heuristics available, up to lower order terms. Experimentally, Adaptive Random Gradient was shown to outperform Generalised Random Gradient, suggesting that adapting the learning period is preferable, as the best choice may not be static throughout the optimisation process.

As a general overview, this thesis has yielded many contributions towards the theoretical analysis of stochastic search algorithms. The main contribution of this thesis was the plethora of results regarding hyper-heuristics. We have shown hyper-heuristics are able to achieve fast performance on a wide range of unimodal and multimodal fitness landscapes, and they are able to automatically adapt to changing fitness landscapes, learning which operators are preferable.

Summarising, the major contributions of this thesis are:

- We rigorously proved that four simple hyper-heuristic mechanisms (namely Simple Random, Permutation, Greedy and Random Gradient) do not learn as intended by an analysis on the function GENERALISEDGAPPATHWITHTRAPS. Furthermore, we proved that the four mechanisms essentially choose low-level heuristics at random. This was shown with an analysis on the LEADINGONES benchmark function where all four hyper-heuristics exhibited the same expected runtime, up to lower order terms.
- We showed that introducing a learning period, such that the performance of the heuristics can be evaluated over a longer period of time, can lead to improved performance on a range of pseudo-Boolean function classes including GENERALISEDGAPPATHWITHTRAPS. Particularly, on LEADINGONES, the newly introduced Generalised Random Gradient hyper-heuristic, equipped with a sufficiently large learning period, is able to achieve the best-possible performance with the (up to a constant number of) heuristics available, up to lower order terms. This performance is an improvement upon many commonly used algorithms, including Randomised Local Search and the (1+1) Evolutionary Algorithm, which have been shown to exhibit efficient performance.
- Based on theoretical insights, we introduced a new hyper-heuristic which combines the learning mechanisms from Greedy and Random Gradient. The new hyper-heuristic, Generalised Greedy Gradient, aims to make a smart heuristic choice and exploit the chosen heuristic so long as it is successful. This hyper-heuristic is able to match the performance of the Generalised Random Gradient hyper-heuristic theoretically and outperforms it experimentally on LEADINGONES.
- We presented the Adaptive Random Gradient hyper-heuristic, which automatically adapts the value of the learning period of the Random Gradient hyper-heuristic throughout the optimisation process. A rigorous analysis on LEADINGONES shows that Adaptive Random Gradient also matches the performance of the Generalised Random Gradient hyper-heuristic theoretically and outperforms it experimentally for any static value of the learning period. Hence, this suggests that setting a static value of the learning period of the hyper-heuristics is sub-optimal.
- We presented one of the first theoretical analyses of selection hyper-heuristics selecting between different move acceptance operators. We proved that such a hyper-heuristic, while still being able to hillclimb efficiently, solves two benchmark multimodal function

classes faster than well-studied elitist and non-elitist algorithms, including Randomised Local Search, Evolutionary Algorithms and Simulated Annealing.

10.2 Future Work

There are still many open questions regarding the understanding of hyper-heuristics. Further work may follow several research directions.

Firstly, applying the hyper-heuristics presented in this thesis to different problems such as ONEMAX would give more insights as to their performance. Additionally, an analysis of hyper-heuristics for more realistic problems from the literature, such as benchmark problems from combinatorial optimisation where hyper-heuristics have shown successful performance (e.g., Scheduling (Cowling et al., 2001) or Timetabling (Burke et al., 2003)), would be of great interest. A series of benchmark functions and the necessary software to apply hyper-heuristic methodologies in a Java framework is available from the HyFlex library (Ochoa et al., 2012a). Combinatorial benchmark function classes available through the HyFlex framework include satisfiability problems, bin packing and personnel scheduling. Applying Generalised Random Gradient (from Chapter 4), Move Acceptance Hyper-heuristic (from Chapter 7) and Adaptive Random Gradient (from Chapter 9) within this framework would produce results that are easily comparable with a wide range of results regarding the performance of sophisticated hyper-heuristics from the literature.

Moreover, a natural next step to further the understanding of hyper-heuristics is to consider a greater variety of operators as low-level heuristics. In this thesis we have studied hyper-heuristics in the framework of trajectory-based algorithms. By introducing populations, crossover of individuals is used to create offspring that share characteristics with their parents. Further parameters can be updated by hyper-heuristic mechanisms, including population size and crossover rate.

Furthermore, while the hyper-heuristic mechanisms we have studied have elicited good performance in the literature, more sophisticated mechanisms, such as the Choice Function approach of Cowling et al. (2001) (where the performance of low-level heuristics is tracked through a number of metrics and are selected based on their 'scores' in these metrics), exist. It would be interesting to understand from a theoretical perspective why these approaches have provided good results. Given the analysis of simpler hyper-heuristic approaches presented in this thesis, a foundational theoretical underpinning of analysis techniques can be further built up, with the goal of analysing state-of-the-art generation and selection hyper-heuristics (see e.g., work by Sabar et al. (2015) or Li and Kendall (2017) for recent successful applications of sophisticated hyper-heuristics).

All of these open questions and future research directions imply that the theoretical analysis of hyper-heuristics remains an interesting, challenging and rewarding research area.

Bibliography

Afshani, P., Agrawal, M., Doerr, B., Doerr, C., Larsen, K. G., and Mehlhorn, K. (2013). The query complexity of finding a hidden permutation. In Brodnik, A., López-Ortiz, A., Raman, V., and Viola, A., editors, *Space-Efficient Data Structures, Streams, and Algorithms: Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, pages 1–11. Springer.

Ahmadi, S., Barone, R., Cheng, P., Cowling, P., and McCollum, B. (2003). Perturbation based variable neighbourhood search in heuristic space for examination timetabling problem. In *Proceedings of the Multidisciplinary International Conference on Scheduling: Theory and Applications*, MISTA '03, pages 155–173.

Alanazi, F. and Lehre, P. K. (2014). Runtime analysis of selection hyper-heuristics with classical learning mechanisms. In *Proceedings of the IEEE Congress on Evolutionary Computation*, CEC '14, pages 2515–2523. IEEE.

Alanazi, F. and Lehre, P. K. (2016). Limits to learning in reinforcement learning hyper-heuristics. In *Proceedings of the European Conference on Evolutionary Computation in Combinatorial Optimization*, EvoCOP '16, pages 170–185. Springer.

Anil, G. and Wiegand, R. P. (2009). Black-box search by elimination of fitness functions. In *Proceedings of the Workshop on Foundations of Genetic Algorithms*, FOGA '09, pages 67–78. ACM.

Ayob, M. and Kendall, G. (2003). A monte carlo hyper-heuristic to optimise component placement sequencing for multi head placement machine. In *Proceedings of the International Conference on Intelligent Technologies*, InTech '03, pages 132–141. Springer.

Bader-El-Den, M. and Poli, R. (2008). Generating SAT local-search heuristics using a GP hyper-heuristic framework. In *Proceedings of the International Conference on Artificial Evolution (Evolution Artificielle)*, EA '08, pages 37–49. Springer.

Bader-El-Den, M., Poli, R., and Fatima, S. (2009). Evolving timetabling heuristics using a grammar-based genetic programming hyper-heuristic framework. *Memetic Computing*, 1(3):205.

Bai, R., Blazewicz, J., Burke, E. K., Kendall, G., and McCollum, B. (2012). A simulated annealing hyper-heuristic methodology for flexible decision support. *4OR: A Quarterly Journal of Operations Research*, 10:43–66.

Bai, R., Burke, E. K., Gendreau, M., Kendall, G., and McCollum, B. (2007). Memory length in hyper-heuristics: An empirical study. In *Symposium on Computational Intelligence in Scheduling*, CISched '07, pages 173–178. IEEE.

Bai, R., Burke, E. K., and Kendall, G. (2008). Heuristic, meta-heuristic and hyper-heuristic approaches for fresh produce inventory control and shelf space allocation. *Journal of the Operational Research Society*, 59(10):1387–1397.

Bai, R. and Kendall, G. (2005). *An Investigation of Automated Planograms Using a Simulated Annealing Based Hyper-Heuristic*, pages 87–108. Springer.

Balaprakash, P., Birattari, M., and Stützle, T. (2007). Improvement strategies for the F-race algorithm: Sampling design and iterative refinement. In *Proceedings of the International Workshop on Hybrid Metaheuristics*, HM '07, pages 108–122. Springer.

Beyer, H.-G. and Schwefel, H.-P. (2002). Evolution strategies - a comprehensive introduction. *Natural Computing*, 1(1):3–52.

Bilgin, B., Özcan, E., and Korkmaz, E. E. (2007). An experimental study on hyper-heuristics and exam timetabling. In *Proceedings of Practice and Theory of Automated Timetabling*, PATAT '07, pages 394–412. Springer.

Birattari, M., Stützle, T., Paquete, L., and Varrentrapp, K. (2002). A racing algorithm for configuring metaheuristics. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '02, pages 11–18. ACM.

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

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 *Congress on Evolutionary Computation*, CEC '10, pages 1–8. IEEE.

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 *Proceedings of the Congress on Evolutionary Computation*, volume 3 of *CEC '05*, pages 2263–2270 Vol. 3. IEEE.

Burke, E., Kendall, G., and Soubeiga, E. (2003). A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics*, 9(6):451–470.

Burke, E. and Soubeiga, E. (2003). Scheduling nurses using a tabu-search hyperheuristic. In *Proceedings of the Multidisciplinary International Conference on Scheduling: Theory and Applications*, MISTA '03, pages 197–218.

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*, pages 1695–1724.

Burke, E. K., Gendreau, M., Ochoa, G., and Walker, J. D. (2011). Adaptive iterated local search for cross-domain optimisation. In *Proceedings of Conference on Genetic and Evolutionary Computation*, GECCO '11, 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 Gendreau, M. and Potvin, J.-Y., editors, *Handbook of Metaheuristics*, pages 449–468. Springer.

Burke, E. K., Silva, J. D. L., and Soubeiga, E. (2005b). *Multi-Objective Hyper-Heuristic Approaches for Space Allocation and Timetabling*, pages 129–158. Springer.

Buzdalov, M. and Buzdalova, A. (2015). Can onemax help optimizing leadingones using the EA+RL method? In *Proceedings of the IEEE Congress on Evolutionary Computation*, CEC '15, pages 1762–1768. IEEE.

Castro, L., de Castro, L., and Timmis, J. (2002). *Artificial Immune Systems: A New Computational Intelligence Approach*. Springer.

Chakhlevitch, K. and Cowling, P. (2008). Hyperheuristics: Recent developments. In Cotta, C., Sevaux, M., and Sörensen, K., editors, *Adaptive and Multilevel Metaheuristics*, pages 3–29. Springer.

Chan, C. Y., Xue, F., Ip, W. H., and Cheung, C. F. (2012). A hyper-heuristic inspired by pearl hunting. In Hamadi, Y. and Schoenauer, M., editors, *Learning and Intelligent Optimization*, LION '12, pages 349–353. Springer.

Chen, P., Kendall, G., and Berghe, G. V. (2007). An ant based hyper-heuristic for the travelling tournament problem. In *Symposium on Computational Intelligence in Scheduling*, CISched '07, pages 19–26. IEEE.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms*. The MIT Press, 3rd edition.

Corus, D., Dang, D., Eremeev, A. V., and Lehre, P. K. (2018). Level-based analysis of genetic algorithms and other search processes. *IEEE Transactions on Evolutionary Computation*, 22(5):707–719.

Corus, D., Oliveto, P. S., and Yazdani, D. (2017). On the runtime analysis of the opt-IA artificial immune system. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, pages 83–90. ACM.

Cowling, P. and Chakhlevitch, K. (2003). Hyperheuristics for managing a large collection of low level heuristics to schedule personnel. In *Proceedings of the Congress on Evolutionary Computation*, CEC '03, pages 1214–1221. IEEE.

Cowling, P., Kendall, G., and Han, L. (2002a). An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem. In *Proceedings of the Congress on Evolutionary Computation*, CEC '02, pages 1185–1190. IEEE.

Cowling, P., Kendall, G., and Soubeiga, E. (2001). A hyperheuristic approach to scheduling a sales summit. In *Proceedings of the Conference on Practice and Theory of Automated Timetabling*, PATAT '01, pages 176–190. Springer.

Cowling, P., Kendall, G., and Soubeiga, E. (2002b). Hyperheuristics: A tool for rapid prototyping in scheduling and optimisation. In *Proceedings of the Workshop on Applications of Evolutionary Computing*, EvoWorkshops ‘02, pages 1–10. Springer.

Dang, D. C., Friedrich, T., Kötzing, T., Krejca, M. S., Lehre, P. K., Oliveto, P. S., Sudholt, D., and Sutton, A. M. (2017). Escaping local optima using crossover with emergent diversity. *IEEE Transactions on Evolutionary Computation*, 22(3):484–497.

Darwin, C. (1859). *On the Origin of Species by Means of Natural Selection*. Murray.

Demeester, P., Bilgin, B., De Causmaecker, P., and Vanden Berghe, G. (2012). A hyperheuristic approach to examination timetabling problems: benchmarks and a new problem from practice. *Journal of Scheduling*, 15(1):83–103.

Di Gaspero, L. and Urli, T. (2012). Evaluation of a family of reinforcement learning cross-domain optimization heuristics. In Hamadi, Y. and Schoenauer, M., editors, *Learning and Intelligent Optimization*, LION ‘12, pages 384–389. Springer.

Doerr, B. (2011). Analyzing randomized search heuristics: Tools from probability theory. In Auger, A. and Doerr, B., editors, *Theory of Randomized Search Heuristics*, pages 1–20. World Scientific.

Doerr, B. (2018a). Better runtime guarantees via stochastic domination. In *Proceedings of the European Conference on Evolutionary Computation in Combinatorial Optimization*, EvoCOP ‘18, pages 1–17. Springer.

Doerr, B. and Doerr, C. (2015). Optimal parameter choices through self-adjustment: Applying the 1/5-th rule in discrete settings. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO ’15, pages 1335–1342. ACM.

Doerr, B. and Doerr, C. (2016). The impact of random initialization on the runtime of randomized search heuristics. *Algorithmica*, 75(3):529–553.

Doerr, B. and Doerr, C. (2018). Theory of parameter control for discrete black-box optimization: Provable performance gains through dynamic parameter choices. *CoRR*, abs/1804.05650.

Doerr, B., Doerr, C., and Ebel, F. (2015). From black-box complexity to designing new genetic algorithms. *Theoretical Computer Science*, 567:87–104.

Doerr, B., Doerr, C., and Yang, J. (2016a). k-bit mutation with self-adjusting k outperforms standard bit mutation. In *Proceedings of the International Conference on Parallel Problem Solving from Nature*, PPSN ’16, pages 824–834. Springer.

Doerr, B., Doerr, C., and Yang, J. (2016b). Optimal parameter choices via precise black-box analysis. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO ’16, pages 1123–1130. ACM.

Doerr, B. and Goldberg, L. A. (2010). Drift analysis with tail bounds. In *Proceedings of the International Conference on Parallel Problem Solving from Nature*, PPSN ’10, pages 174–183. Springer.

Doerr, B., Johannsen, D., and Winzen, C. (2012). Multiplicative drift analysis. *Algorithmica*, 64(4):673–697.

Doerr, B., Lissovoi, A., Oliveto, P. S., and Warwicker, J. A. (2018). On the runtime analysis of selection hyper-heuristics with adaptive learning periods. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '18, pages 1015–1022. ACM.

Doerr, B., Sudholt, D., and Witt, C. (2013). When do evolutionary algorithms optimize separable functions in parallel? In *Proceedings of the International Workshop on Foundations of Genetic Algorithms*, FOGA '13, pages 51–64. ACM.

Doerr, C. (2018b). Dynamic parameter choices in evolutionary computation. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '18, pages 800–830. ACM.

Doerr, C. and Wagner, M. (2018). Simple on-the-fly parameter selection mechanisms for two classical discrete black-box optimization benchmark problems. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '18, pages 943–950. ACM.

Dorigo, M. and Stützle, T. (2004). *Ant Colony Optimization*. Bradford Company.

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 *Proceedings of the International Conference on Parallel Problem Solving from Nature*, PPSN '12, pages 307–316. Springer.

Droste, S. (2004). Analysis of the (1+1) EA for a noisy onemax. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '04, pages 1088–1099. Springer.

Droste, S., Jansen, T., Tinnefeld, K., and Wegener, I. (2003). A new framework for the valuation of algorithms for black-box optimization. In *Proceedings of the International Workshop on Foundations of Genetic Algorithms*, FOGA '03, pages 253–270. ACM.

Droste, S., Jansen, T., and Wegener, I. (2001). Dynamic parameter control in simple evolutionary algorithms. In *Proceedings of the International Workshop on Foundations of Genetic Algorithms*, FOGA '01, pages 275 – 294. ACM.

Droste, S., Jansen, T., and Wegener, I. (2002). On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, pages 51–81.

Eiben, A. E., Hinterding, R., and Michalewicz, Z. (1999). Parameter control in evolutionary algorithms. *IEEE Transations on Evolutionary Computation*, 3(2):124–141.

Eiben, A. E. and Smith, J. E. (2015). *Introduction to Evolutionary Computing*. Springer, 2nd edition.

Epitropakis, M. G. and Burke, E. K. (2018). *Hyper-heuristics*, pages 1–57. Springer.

Feller, W. (1968). *An Introduction to Probability Theory and Its Applications*. Wiley.

Fogel, D. B., Hays, T. J., Hahn, S. L., and Quon, J. (2004). A self-learning evolutionary chess program. *Proceedings of the IEEE*, 92(12):1947–1954.

Fogel, L., Owens, A., and Walsh, M. (1966). *Artificial intelligence through simulated evolution*. Wiley.

Gibbs, J., Kendall, G., and Özcan, E. (2010). Scheduling english football fixtures over the holiday period using hyper-heuristics. In *Proceedings of the International Conference on Parallel Problem Solving from Nature*, PPSN ‘10, pages 496–505. Springer.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533 – 549.

Hajek, B. (1982). Hitting-time and occupation-time bounds implied by drift analysis with applications. *Advances in Applied Probability*, 14(3):502–525.

Hall, G. T., Oliveto, P. S., and Sudholt, D. (2019). On the impact of the cutoff time on the performance of algorithm configurators. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO ’19, pages 907–915.

Han, L. and Kendall, G. (2003). Guided operators for a hyper-heuristic genetic algorithm. In *Proceedings of the Australian Conference on Artificial Intelligence*, AI ‘03, pages 807–820.

He, J., He, F., and Dong, H. (2012). Pure strategy or mixed strategy? In *Proceedings of the European Conference on Evolutionary Computation in Combinatorial Optimization*, EvoCOP ’12, pages 218–229. Springer.

He, J., Hou, W., Dong, H., and He, F. (2013). Mixed strategy may outperform pure strategy: An initial study. In *Proceedings of the IEEE Congress on Evolutionary Computation*, CEC ‘14, pages 562–569. IEEE.

He, J. and Yao, X. (2001). Drift analysis and average time complexity of evolutionary algorithms. *Artificial Intelligence*, 127(1):57–85.

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 2nd edition.

Hoos, H. and Stützle, T. (2004). *Stochastic Local Search: Foundations & Applications*. Morgan Kaufmann.

Hsiao, P.-C., Chiang, T.-C., and Fu, L.-C. (2012). A VNS-based hyper-heuristic with adaptive computational budget of local search. In *Congress on Evolutionary Computation*, CEC ‘12, pages 1–8.

Hutter, F., Hoos, H. H., Leyton-Brown, K., and Stützle, T. (2009). ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36(1):267–306.

Immerman, N. (1999). *Descriptive Complexity*. Springer.

Jägersküpper, J. and Storch, T. (2007). When the plus strategy outperforms the comma strategy and when not. In *Proceedings of IEEE Symposium on Foundations of Computational Intelligence*, FOCI '07, pages 25–32.

Jansen and Wegener (2002). The analysis of evolutionary algorithms—a proof that crossover really can help. *Algorithmica*, 34(1):47–66.

Jansen, T. (2011). Simulated annealing. In Auger, A. and Doerr, B., editors, *Analyzing Randomized Search Heuristics: Tools from Probability Theory*, pages 171–196. World Scientific.

Jansen, T. (2013). *Analyzing Evolutionary Algorithms: The Computer Science Perspective*. Springer.

Jansen, T., De Jong, K. A., and Wegener, I. (2005). On the choice of the offspring population size in evolutionary algorithms. *Evolutionary Computation*, 13(4):413–440.

Jansen, T., Oliveto, P. S., and Zarges, C. (2011). On the analysis of the immune-inspired b-cell algorithm for the vertex cover problem. In *Proceedings of the International Conference on Artificial Immune Systems*, ICARIS '11, pages 117–131. Springer.

Jansen, T. and Wegener, I. (2007). A comparison of simulated annealing with a simple evolutionary algorithm on pseudo-boolean functions of unitation. *Theoretical Computer Science*, pages 73 – 93.

Jansen, T. and Zarges, C. (2012). Fixed budget computations: A different perspective on run time analysis. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '12, pages 1325–1332. ACM.

Johannsen, D. (2010). *Random combinatorial structures and randomized search heuristics*. PhD thesis, Universität des Saarlandes, Saarbrücken.

Kaelbling, L. P., Littman, M. L., and Moore, A. P. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285.

Kendall, G. and Mohamad, M. (2004). Channel assignment in cellular communication using a great deluge hyper-heuristic. In *Proceedings of the IEEE International Conference on Networks*, ICON '04, pages 769–773.

Kern, S., Müller, S. D., Hansen, N., Büche, D., Ocenasek, J., and Koumoutsakos, P. (2004). Learning probability distributions in continuous evolutionary algorithms - a comparative review. *Natural Computing*, 3:77–112.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

Kötzing, T., Lissovoi, A., and Witt, C. (2015). (1+1) EA on generalized dynamic onemax. In *Proceedings of the International Workshop on Foundations of Genetic Algorithms*, FOGA '15, pages 40–51. ACM.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press.

Lehre, P. K. and Oliveto, P. S. (2018). *Theoretical Analysis of Stochastic Search Algorithms*, pages 849–884. Springer.

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 International Workshop on Foundations of Genetic Algorithms*, FOGA '13, pages 97–104. ACM.

Lehre, P. K. and Witt, C. (2012). Black-box search by unbiased variation. *Algorithmica*, pages 623–642.

Li, J. and Kendall, G. (2017). A hyperheuristic methodology to generate adaptive strategies for games. *IEEE Transactions on Computational Intelligence and AI in Games*, 9(1):1–10.

Lissovoi, A., Oliveto, P. S., and Warwicker, J. A. (2017). On the runtime analysis of generalised selection hyper-heuristics for pseudo-boolean optimisation. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, pages 849–856. ACM.

Lissovoi, A., Oliveto, P. S., and Warwicker, J. A. (2018). On the performance of move acceptance hyper-heuristics for multimodal optimisation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, AAAI '19. To appear.

Lissovoi, A., Oliveto, P. S., and Warwicker, J. A. (2019). Simple hyper-heuristics control the neighbourhood size of randomised local search optimally for leadingones. *Evolutionary Computation Journal*. To appear.

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., and Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43 – 58.

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

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, pages 1087–1092.

Michiels, W., Aarts, E., and Korst, J. (2007). *Theoretical Aspects of Local Search*. Springer.

Mitzenmacher, M. and Upfal, E. (2005). *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press.

Motwani, R. and Raghavan, P. (1995). *Randomized Algorithms*. Cambridge University Press.

Mısır, M. (2019). Hyper-heuristic bibliography. https://mustafamisir.github.io/hh.html. Accessed: 2018-12-10.

Mısır, M., Vancroonenburg, W., Verbeeck, K., and Vanden Berghe, G. (2011). A selection hyper-heuristic for scheduling deliveries of ready-mixed concrete. In *Proceedings of Metaheuristics International Conference*, MIC '11.

Mısır, M., Verbeeck, K., Causmaecker, P., and Berghe, G. (2012). An intelligent hyper-heuristic framework for CHeSC 2011. In *International Conference on Learning and Intelligent Optimization*, LION '06, pages 461–466. Springer.

Mısır, M., Verbeeck, K., Causmaecker, P. D., and Berghe, G. V. (2010). Hyper-heuristics with a dynamic heuristic set for the home care scheduling problem. In *Proceedings of the Congress on Evolutionary Computation*, CEC '10, pages 1–8. IEEE.

Mısır, M., Wauters, T., Verbeeck, K., and Berghe, G. V. (2009). A new learning hyper-heuristic for the traveling tournament problem. In *Proceedings of the Metaheuristics International Conference*, MIC '09.

Mühlenbein, H. (1992). How genetic algorithms really work: Mutation and hillclimbing. In *Proceedings of the International Conference on Parallel Problem Solving from Nature*, PPSN '92, pages 15–26.

Nallaperuma, S., Oliveto, P. S., Pérez Heredia, J., and Sudholt, D. (2019). On the analysis of trajectory-based search algorithms: When is it beneficial to reject improvements? *Algorithmica*, 81(2):858–885.

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. (2010). *Bioinspired Computation in Combinatorial Optimization: Algorithms and Their Computational Complexity*. Springer, 1st edition.

Ochoa, G., Hyde, M., Curtois, T., Vazquez-Rodriguez, J. A., Walker, J., Gendreau, M., Kendall, G., McCollum, B., Parkes, A. J., Petrovic, S., and Burke, E. K. (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 *Proceedings of the International Conference on Parallel Problem Solving from Nature*, PPSN '12, 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.

Oliveto, P. S. and Witt, C. (2011). Simplified drift analysis for proving lower bounds in evolutionary computation. *Algorithmica*, pages 369–386.

Oliveto, P. S. and Witt, C. (2012). Erratum: Simplified drift analysis for proving lower bounds in evolutionary computation. *CoRR*, abs/1211.7184.

Oliveto, P. S. and Witt, C. (2015). Improved time complexity analysis of the simple genetic algorithm. *Theoretical Computer Science*, 605:21–41.

Oliveto, P. S. and Yao, X. (2011). *Runtime Analysis of Evolutionary Algorithms for Discrete Optimization*, pages 21–52. World Scientific.

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 *Congress on Evolutionary Computation*, CEC '09, pages 997–1004. IEEE.

Özcan, E. and Kheiri, A. (2012). A hyper-heuristic based on random gradient, greedy and dominance. In Gelenbe, E., Lent, R., and Sakellari, G., editors, *Computer and Information Sciences II*, pages 557–563. Springer.

Özcan, E., Mısır, M., Ochoa, G., and Burke, E. K. (2010). A reinforcement learning-great-deluge hyper-heuristic for examination timetabling. *International Journal of Applied Metaheuristic Computing*, 1(1):39–59.

Paixão, T., Pérez Heredia, J., Sudholt, D., and Trubenová, B. (2017). Towards a runtime comparison of natural and artificial evolution. *Algorithmica*, pages 681–713.

Pisinger, D. and Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers and Operations Research*, 34(8):2403–2435.

Qian, C., Tang, K., and Zhou, Z.-H. (2016). Selection hyper-heuristics can provably be helpful in evolutionary multi-objective optimization. In *Proceedings of the International Conference on Parallel Problem Solving from Nature*, PPSN '16, pages 835–846. Springer.

Rizzo, M. (2007). *Statistical Computing with R.* Chapman & Hall/CRC The R Series. Taylor & Francis.

Ross, P. (2005). Hyper-heuristics. In Burke, E. K. and Kendall, G., editors, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, pages 529–556. Springer.

Rowe, J. E. and Sudholt, D. (2014). The choice of the offspring population size in the (1,λ) evolutionary algorithm. *Theoretical Computer Science*, 545:20 – 38.

Rudolph, G. (1997). *Convergence Properties of Evolutionary Algorithms.* Kovač.

Rudolph, G. (1998). Finite markov chain results in evolutionary computation: A tour d'horizon. *Fundamenta Informaticae*, 35(1-4):67–89.

Sabar, N. R., Ayob, M., Kendall, G., and Qu, R. (2015). Automatic design of a hyper-heuristic framework with gene expression programming for combinatorial optimization problems. *IEEE Transactions on Evolutionary Computation*, 19(3):309–325.

Schwefel, H. P. (1975). *Evolutionsstrategie und numerische optimierung*. PhD thesis, Technical University of Berlin, Berlin.

Turing, A. M. (1948). Intelligent machinery, a heretical theory. *Philosophia Mathematica*, 4(3):256–260.

Wald, A. (1944). On cumulative sums of random variables. *The Annals of Mathematical Statistics*, 15(3):283–296.

Walker, J. D., Ochoa, G., Gendreau, M., and Burke, E. K. (2012). Vehicle routing and adaptive iterated local search within the hyflex hyper-heuristic framework. In *Learning and Intelligent Optimization*, LION '12, pages 265–276. Springer.

Witt, C. (2006). Runtime analysis of the $(\mu + 1)$ EA on simple pseudo-boolean functions. *Evolutionary Computation*, 14(1):65–86.

Witt, C. (2013). Tight bounds on the optimization time of a randomized search heuristic on linear functions. *Combinatorics, Probability and Computing*, 22(2):294–318.

Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82.