A Framework for the Runtime Analysis of Algorithm Configurators

George Thomas Hall

The University of Sheffield
Faculty of Engineering
Department of Computer Science

A thesis submitted in partial fulfilment of the requirements for the degree of
Doctor of Philosophy

September 2021

Declaration

I, George Thomas Hall, confirm that the Thesis is my own work. I am aware of the University's Guidance on the Use of Unfair Means (www.sheffield.ac.uk/ssid/unfair-means). This work has not been previously been presented for an award at this, or any other, university.

George Thomas Hall
September 2021

Abstract

Despite the widespread usage of algorithm configurators to tune algorithmic parameters, there is still little theoretical understanding of their performance. In this thesis, we build a theoretical foundation for the field of algorithm configuration to enable the derivation of specific statements regarding the performance of algorithm configurators. We use the devised framework to prove tight bounds on the time required by specific configurators to identify the optimal parameter values of randomised local search and simple evolutionary algorithms for standard benchmark function classes. Our framework allows us to derive insights regarding the impact of the parameters of algorithm configurators, in particular the cutoff time and performance metric used to compare configurations, as well as to characterise parameter landscapes. In the general case, we present necessary lower bounds and sufficient upper bounds on the cutoff time if the time taken to reach a specific target fitness value is used as the performance metric. For specific simple algorithm configuration scenarios, we show that our general lower bounds are tight and that the same optimal parameter values can be identified using smaller cutoff times if the performance metric is instead taken to be the fitness value obtained within the available time budget, which also reduces the required amount of problem-specific information. Our insights enable the design of mutation operators that are provably asymptotically faster for unimodal and approximately unimodal parameter landscapes and slower by only a logarithmic factor in the worst case. In addition to our contributions to the theory of algorithm configuration, the mathematical techniques derived in this thesis represent a substantial improvement over the state-of-the-art in the field of fixed-budget analysis.

Acknowledgements

Completing a PhD is far from a solitary affair. I would like to take this opportunity to thank the many people whose support has helped me over these past few years.

First, I must thank my supervisor, Pietro Oliveto. Pietro has always gone above and beyond in his efforts to guide me, and in doing so I am sure that he will have turned me into a better researcher. His eye for detail and desire for rigour when I would have settled for less will undoubtedly hugely improve everything that I produce after my PhD. He has always been willing to offer support whenever I need it, and for this I am immensely grateful.

My second supervisor, Dirk Sudholt, has also been incredibly helpful throughout this process. Dirk has been exceedingly generous with both his time and expertise, joining the vast majority of my research meetings and co-authoring all three publications from my PhD. I am sure that the work produced during my PhD is much stronger thanks to Dirk's continued input over the last few years.

I am also indebted to Zemin Ning, with whom I worked several years ago. Working with Zemin was my first serious introduction to the world of research, and had he not shown me the fun to be had and suggested that I pursue an academic career for myself then I probably would not have started down the path that culminated in writing this thesis.

I am very grateful to Pan Peng and Manuel López-Ibáñez for devoting a significant portion of their time to reading my thesis and examining me on it. They have helped to make it stronger thanks to our discussions in my viva. Thanks must also go to John Clark for being my thesis committee chair over the course of my PhD and for asking thought-provoking questions whenever we met, often suggesting perspectives on topics that we had not considered before.

The wider Algorithms Group at Sheffield – Ali, Andrei, Daniel, Dirk, Dogan, Donya, Edgar, Mario, Marios, Pan, Pietro, and Sebastian – fostered a friendly collegiate atmosphere which was a pleasure to be a part of. Extra thanks must be extended to those with whom I attended conferences: thank you for all the happy memories at a number of foreign locales!

I would also like to thank the members of my office at Regent Court for the sociable and enjoyable atmosphere that they created, although I will not name particular people here to avoid missing anyone by accident!

I am grateful to the friends I made along the way for all the happy memories. In particular, I would like to mention the people I joined for lunch and in the pub – Abdullah, Adam, Ali, Alison, Andy, Ben, Dave, Donya, George, Jose, Mario, Mat, Michael, Neil, Owain, Phil, Tom, and Wil. On many days, I only made the commute to Sheffield in order to join you for lunch or in the pub, which hopefully speaks to my enjoyment your company.

I have also spent a significant proportion of this process outside of Sheffield and so my thanks also go to my friends from earlier in life for their support in this pursuit and for providing such nice ways to spend my time when I was not working. Particular thanks must go to Adam, Glen, Josh, and Tom, with whom I spent almost all of my in-person social time during the restriction-heavy final days of my PhD.

Needless to say, I am immensely grateful to my family, both immediate and extended, for their continued support throughout this process and the interest that they have shown in what I have been up to. Naturally, particular thanks must go to my parents, Lisa and Martin, and my sister, Amy, for their unwavering support throughout my life. I hope that you all feel responsible for the parts that you have each played in this endeavour.

Finally, I thank my girlfriend Amelia for her immense support every day of this process. She has been there to hear my problems no matter how many times she has heard them before, tolerated my at times excessive working hours without complaint, and has done a fantastic job of keeping my morale up during some challenging times. Writing at the end of a very strange 18 months, I cannot think of anyone with whom I would rather have been so isolated.

Table of contents

Nomenclature **xv**

I Introduction and Background **1**

1 Introduction **3**

1.1 Overview and Motivation 3

1.2 Structure of this Thesis 12

1.3 Contributions of this Thesis 15

1.4 Underlying Publications 17

2 Algorithm Configuration **19**

2.1 Introduction 19

2.2 Randomised Search Heuristics 19

2.2.1 Types of Parameters 21

2.3 The Algorithm Configuration Problem 22

2.4 Algorithm Configurators 23

2.5 Algorithm Configuration Evaluation 25

2.5.1 Performance Metrics 27

2.5.2 Cutoff Time 29

2.6 Configuration Evaluation Methodologies 31

2.6.1 Static Runs 31

2.6.2 Dynamic Runs 32

2.6.3 Surrogate Models (No Runs) 34

2.7 Configuration Generation Methodologies 35

2.7.1 One-shot methods 35

2.7.2 Black-box methods 35

2.7.3 Experimental design 36

2.7.4 Surrogate model-based techniques 37
2.8 Detailed Algorithm Configurator Descriptions 38
2.8.1 ParamILS 38
2.8.2 Iterated F-Race and *irace* 40
2.8.3 SMAC 41
2.8.4 Structured Procrastination 42
2.8.5 Other Non-Incumbent-Driven Configurators 44
2.9 Theory of Algorithm Configuration 46
2.9.1 Generalisation Analysis 46
2.9.2 Convergence Analysis 49
2.9.3 Worst-Case Performance Analysis 50
2.10 Conclusions 56

3 Mathematical Tools, Configurators, Target Algorithms, Problems Classes 57
3.1 Introduction 57
3.2 Mathematical Tools 58
3.2.1 Runtime Analysis 58
3.2.2 Probability Theory 59
3.2.3 Useful Inequalities 62
3.2.4 Tail Inequalities 63
3.2.5 Random Walks and Gambler's Ruin 65
3.2.6 Drift Analysis 67
3.2.7 Black-Box Complexity 69
3.2.8 Fixed-Budget Analysis 70
3.3 A Simple Algorithm Configurator: ParamRLS 72
3.3.1 The $\pm\{1,\dots,m\}$ Local Search Operator 73
3.3.2 Evaluation of Configurations 75
3.4 Target Algorithms 76
3.4.1 Randomised Local Search 77
3.4.2 The (1+1) EA 78
3.4.3 Scaling and Probabilistic Smoothing (SAPS) 79
3.5 Target Problem Classes 79
3.5.1 RIDGE 80
3.5.2 ONEMAX 82
3.5.3 LEADINGONES 84
3.6 Conclusions 86

II Analysis of Algorithm Configurators 87

4 Fixed-Target Performance Metric Requires Appropriate Cutoff Times 89
4.1 Introduction . . . 89
4.2 A Lower Bound on the Necessary Cutoff Time . . . 90
4.3 An Upper Bound on the Sufficient Cutoff Time . . . 92
4.4 Conclusions . . . 95

5 On the Configuration of the Neighbourhood Size of Randomised Local Search 97
5.1 Introduction . . . 97
5.2 On the Configuration of RLS$_k$ for RIDGE* . . . 98
5.2.1 The Optimisation Time of RLS$_k$ for RIDGE* is Tightly Concentrated 99
5.2.2 Configurators Using the Optimisation-Time Performance Metric Require Quadratic Cutoff Times . . . 101
5.2.3 ParamRLS-F can Identify the Optimal Neighbourhood Size Using Arbitrary Cutoff Times . . . 103
5.3 On the Configuration of RLS$_k$ for ONEMAX* . . . 108
5.3.1 Fixed-Budget Analysis of RLS$_k$ and ONEMAX* . . . 109
5.3.2 Optimisation-Time Requires Superlinear Cutoff Times . . . 122
5.3.3 ParamRLS-F Identifies the Optimal Neighbourhood Size with Arbitrary Cutoff Times . . . 123
5.4 Conclusions . . . 126

6 On the Configuration of the Mutation Rate of a Simple Evolutionary Algorithm 129
6.1 Introduction . . . 129
6.2 On the Configuration of the (1+1)$_\chi$ EA for RIDGE . . . 130
6.2.1 Analysis of the (1+1)$_\chi$ EA for RIDGE . . . 131
6.2.2 Optimisation-Time Requires at Least Quadratic Cutoff Times . . . 135
6.2.3 Linear Cutoff Times Suffice for Best-Fitness . . . 136
6.3 On the Configuration of the (1+1)$_\chi$ EA for LEADINGONES . . . 138
6.3.1 The Parameter Landscape of the (1+1)$_\chi$ EA for LEADINGONES is Unimodal Under Best-Fitness . . . 139
6.3.2 Any Configurator Using the Optimisation-Time Performance Metric is Blind for $\kappa \leq 0.772075n^2$ . . . 152
6.3.3 Best-Fitness Identifies the Optimal Mutation Rate Independent of the Cutoff Time . . . 153
6.4 Conclusions . . . 154

7 Faster Algorithm Configuration by Exploiting Parameter Space Unimodality 155
7.1 Introduction . . . 155
7.2 The Harmonic Mutation Operator . . . 156
7.3 General Lower Bounds for Default Mutation Operators . . . 158
7.4 Performance of the Harmonic Mutation Operator . . . 160
7.5 Experimental Analysis . . . 163
7.5.1 Experimental Setup . . . 164
7.5.2 Results . . . 167
7.6 Conclusions . . . 171

III Conclusions 173

8 Conclusions and Outlook 175
8.1 Summary of Work . . . 175
8.2 Future Work . . . 178

References 181

Appendices 191

List of Algorithms

1 Simulated annealing 20
2 The $(\mu + \lambda)$ EA 20
3 ParamILS 39
4 IterativeFirstImprovement 40
5 irace 40
6 SMAC 41
7 Structured Procrastination 43
8 LEAPSANDBOUNDS 44
9 CAPSANDRUNS 45
10 Structured Procrastination with Confidence 46

11 ParamRLS 74
12 ParamRLS-F 76
13 ParamRLS-T 77
14 RLS$_k$ 78
15 The (1+1)$_\chi$ EA 78

16 ParamHS 157

Nomenclature

Configurator Symbols

κ	The cutoff time.
r	The number of runs per configuration comparison.

Mathematical Objects and Abbreviations

$\mathbb{N}$	The natural numbers: $\{1, 2, \ldots\}$.
$\mathbb{N}_0$	The natural numbers and 0: $\{0\} \cup \mathbb{N}$.
$\mathbb{R}$	The real numbers.
$\mathbb{R}^+$	The positive real numbers.
$\mathbb{R}_0^+$	The non-negative real numbers.
w. o. p.	with overwhelming probability.
u.a.r.	uniformly at random.

Other Abbreviations

LO	LeadingOnes.
x_i	The i-th bit of the bit string x.

Part I

Introduction and Background

Chapter 1

Introduction

1.1 Overview and Motivation

When faced with a new computational problem, an algorithm developer may choose to design an algorithm with good performance guarantees for the specific application. However, this approach requires a good understanding of the problem at hand and the availability of resources (e.g. time and money) required for the design of the problem-specific algorithm.

General-purpose algorithms, such as randomised local search [60], evolutionary algorithms [46], and simulated annealing [77] provide an alternative approach. Whilst they have not been designed with performance guarantees for specific problem classes, they have the advantage that the user need not have an in-depth understanding of the problem for their application. In the case of randomised search heuristics, typically only two decisions must be made: how to represent solutions to the problem and how to measure the quality of solutions (i.e. a *fitness function*).

However, it is well understood that the performance of general-purpose algorithms for specific problems is very sensitive to the settings of their often numerous parameters [65, 80, 110]. According to the no free lunch theorem, optimising these parameters for one problem necessarily leads to degraded performance for others [118]. It is therefore often crucial to choose appropriate parameter values (*parameter configurations*[1]) when applying a general-purpose algorithm to a problem in order to achieve efficient optimisation (i.e. the identification of high quality solutions in a short amount of time).

In the past, it has been common practice to set the values of these parameters manually, either by evaluating a large number of configurations and identifying the one with the best performance, or by using parameter values that have performed well in similar application

[1]Throughout this thesis we often abbreviate this to "configurations".

contexts [45]. However, the former approach is time-consuming and error prone, whilst the latter comes with the risk that the chosen values will not work well in the new setting. Manual approaches may also be biased by the human element involved (reducing reproducibility) and are likely to lead to only a small number of configurations being evaluated (with the evaluations themselves using only a limited set of problem instances) [91].

According to Adenso-Díaz and Laguna, "there is anecdotal evidence that about 10% of the total time dedicated to [the] designing and testing of a new heuristic or metaheuristic is spent on development, and the remaining 90% is consumed by fine-tuning parameters" [1]. Knuth tells us of the difficulty of selecting good parameter values for SAT solvers: "Parameter optimization for general broad-spectrum use is a daunting task, not only because of significant differences between species of SAT instances but also because of the variability due to random choices when solving any specific instances. It's hard to know whether a change of parameter will be beneficial or harmful, especially when running times are so highly erratic" [80].

Despite the importance of choosing good parameter values and the difficulty of doing so, it was only at the end of the last century that the issue began to be seriously addressed by the scientific community [22]. This resulted in methods such as parameter control [75], where parameter values are updated dynamically, and hyper-heuristics [19], where entire algorithms are evolved during the optimisation process.

An increasingly popular method to select good parameter configurations of a *target algorithm* is to automate the aforementioned manual approach, thereby treating the identification of good parameter configurations as an optimisation problem which can be tackled using established techniques. Such automated methods are called *algorithm configurators* (alternatively, *parameter tuners*), and the task that they address is that of *algorithm configuration* (alternatively, *parameter tuning*)[2]. Popular examples of algorithm configurators are ParamILS [62], *irace* [91], and SMAC [64]. Hutter *et al.* list three advantages of automated algorithm configuration over the manual approach [62]:

- It facilitates the development of complex algorithms with many parameters since it eliminates the time-consuming task of setting them wisely.
- It allows algorithms to be used in domains where the user does not have prior knowledge of the impact of each parameter on the performance of the algorithm. Even when the user has some knowledge about the problem, it is still simpler to configure an algorithm automatically rather than manually.

[2]In this thesis, we use the terms "parameter tuning" and "algorithm configuration" interchangeably, and do likewise with "parameter tuner" and "algorithm configurator". Hoos, however, expresses a preference to use "parameter tuning" when there are few parameters and they are mostly real-valued and "algorithm configuration" when there are many parameters and they are not necessarily real-valued [59].

- It allows algorithms to be compared more fairly. If an algorithm outperforms its competitors after all have been configured then it is stronger evidence that it is genuinely better for the problem, rather than simply having better default parameter values.

This approach has now become the standard, and automated algorithm configurators have successfully been used to compare algorithms in numerous settings, for example SAT solving [65].

Despite the widespread usage of algorithm configurators such as *irace* and SMAC in recent years (for example, see [91] for an extensive list of contexts where *irace* has been applied), there is little rigorous understanding of their performance, both in terms of the time that they require to identify good parameter values and the expected quality of the parameter values that they return. In addition, there are no rigorous investigations of how the outcome is affected by the settings of the configurator itself. Thus there is no guidance for practitioners on how to set up the configurator or for how long to run it before it is likely to return satisfactory parameter values. Furthermore, due to the lack of rigorous investigations into the performance of algorithm configurators, it is unclear when the sophisticated features of state-of-the-art configurators are necessary or when simpler (potentially faster) approaches would suffice. Hence practitioners may be using unnecessarily complex tuners at the expense of efficiency.

The only existing runtime analyses for algorithm configurators provide lower bounds on the time required to identify near-optimal configurations in the worst case [78, 79, 115, 116]. However, such analyses do not allow the estimation of the extent to which algorithm configurators are successful in the typical optimisation scenarios encountered in practice, and thus their use to practitioners is limited. Importantly, since such analyses lead to the design of configurators that are tailored to have near-optimal performance in the worst case, they do not contribute to the understanding of the reasons behind the numerous successes of widely-used configurators such as *irace* and SMAC. In this thesis, we build a theoretical foundation to enable the derivation of specific statements regarding the performance of algorithm configurators for specific algorithm configuration scenarios. This framework highlights the reasons for good and bad performance of algorithm configurators by providing rigorous statements about the expected time required by given configurators to identify the optimal parameter values of algorithms for specific problem classes. Such understanding provides insights that can guide the design of algorithm configurators.

Given a problem class and target algorithm, the aim of an automated algorithm configurator is to identify the parameter values that optimise performance of the algorithm for the problem class. Naturally, which parameter values are *optimal* (i.e. optimise the performance of the target algorithm) depends on the measure used to evaluate the performance of the target algorithm. For this reason, it is necessary to use a *performance metric*, the choice of which depends on what is most valuable to the user (for example minimising the time taken to reach a target solution quality, or maximising the solution quality obtained within a time budget). Note that different choices of performance metric may result in different configurations being optimal. A natural performance measure is the time required to identify the global optimum. We refer to such a performance metric as *Optimisation-Time*. Under this metric, the optimal configuration is the set of parameter values that minimises the (usually mean) time required by the target algorithm to identify the global optimum of the problem class instances. However, Optimisation-Time is rarely a relevant performance measure in practical applications, where the expected runtime to reach the global optimum is often prohibitively large. In such cases, more sensible measures of algorithm performance may be the expected time required to reach a solution of a minimal acceptable quality (i.e. an approximate solution) or the solution quality that can be identified within an available time budget. In this thesis, we refer to these performance metrics as *Fixed-Target* and *Best-Fitness*, respectively. We will evaluate the performance of algorithm configurators that use the three most common performance metrics (Optimisation-Time, Fixed-Target, and Best-Fitness) and provide relationships between the settings of the algorithm configurator and the time required to identify the corresponding optimal configuration.

Since it is rarely the case that the performance of each configuration is known ahead of time, it is usually necessary to measure it empirically. This gives rise to two main issues. First, the distribution of the instances of the problem class may be unknown, making it challenging to design a representative training set (i.e. the number of problem instances that may be encountered in practice is prohibitively large). Second, if a solution of minimal quality is sought (i.e. the Fixed-Target performance metric is used), it is unlikely that all configurations will be able to identify a solution of this quality efficiently (i.e. in polynomial time). Thus it is necessary to use a *cutoff time* to control for how long a configuration is run when evaluating its performance for a problem instance. If the time taken to reach a given solution quality is employed as the performance metric, then this prevents slow configurations (i.e. that require at least exponential time to reach the target) from making the entire configuration process require exponential time. On the other hand, when using solution quality as the performance metric (i.e. Best-Fitness), then the cutoff time can simply be set to the time budget for which the user is willing to run their algorithm in practice.

In the algorithm configuration literature, there is no rigorous guidance on how to choose the cutoff time and performance metric. The cutoff time is typically set arbitrarily with little justification of the chosen value. Furthermore, there are no rigorous studies indicating what constitutes a "correct" value for the cutoff time and the price to be paid for choosing an "incorrect" value. The first aim of this thesis is to provide statements regarding how to choose the cutoff time with respect to the performance metric. Whilst for the Best-Fitness performance metric the choice is natural (i.e. the time for which the user will run their tuned algorithm), we provide rigorous necessary lower bounds and sufficient upper bounds on the cutoff times required when Fixed-Target is used in general. In particular, we prove that any algorithm configurator that uses Fixed-Target requires a cutoff time that is at least large enough to allow at least one configuration to reach the target solution quality of at least one training instance with overwhelming probability (i.e. the probability that it fails to do so decreases exponentially in the problem size). If the cutoff time is smaller than this, then the configurator will be *blind* (i.e. it will behave as if all configurations have the same performance and hence return a random configuration). Whilst this result may not appear particularly surprising, it allows us to prove that, if Optimisation-Time is used, then a cutoff time of at least $(n \ln n)/2$ is necessary to configure any unary unbiased algorithm (i.e. an algorithm that perturbs all genes with equal probability) for any optimisation problem with up to $\exp(\sqrt{n}/\log^2 n)$ optima (where n is the problem size), otherwise the tuner will be blind. On the other hand, if the optimal configuration reaches the optimum before every other configuration with overwhelming probability for each instance of the problem class, then a cutoff time large enough to allow this configuration to do so is sufficient for any configurator that samples any given configuration with some positive probability (e.g. ParamILS when configuring an algorithm with only one parameter) to identify this configuration in finite time. We will show that this sufficient condition is tight even for a very simple benchmark problem class called RIDGE, whilst a quadratic cutoff time is necessary for a large range of target solution qualities when using the Fixed-Target performance metric (i.e. a considerably larger cutoff time is required compared to the $(n \ln n)/2$ lower bound that holds in general).

These analyses thus highlight the substantial negative impact of choosing an inappropriately small cutoff time when using the Fixed-Target performance metric and demonstrate the danger of employing this performance metric in the absence of the problem-specific knowledge necessary to set it appropriately. This limits the applicability of Fixed-Target in scenarios where algorithm configurators are commonly applied, where the user may have limited or even no knowledge of the problem at hand ("black-box" optimisation).

We then provide the first indication that choosing the performance metric most similar to the goal of the user may not be optimal, proving that the configuration with the smallest expected optimisation time can also be identified by using solution quality as the performance metric (i.e. Best-Fitness) and breaking ties between configurations by favouring the one that made progress least recently. This disproves an assumption implicit in the algorithm configuration literature that the best choice of performance metric is the one that corresponds to the goal of the user (e.g. if the goal of the user is to minimise the expected time to reach a target solution quality then they should use this quantity as the performance metric).

We then turn our attention towards the analysis of algorithm configurators for specific configuration scenarios, which will allow us to make statements regarding the performance of algorithm configurators for different classes of target algorithm and problems. In particular, assuming that a given configurator *is* able to identify the optimal configuration, we answer the question of how long it is necessary to run it for it to do so. We follow the framework which was successfully used to conduct the initial theoretical analyses of randomised search heuristics for function optimisation [69]. Hence our aim is to identify a minimal algorithm configurator that captures the essential characteristics of the more complex ones that are successfully employed in practice. As was the case with the foundational theoretical analyses of randomised search heuristics, analysing the behaviour of this simple configurator will allow us to build a collection of techniques that can be used to analyse increasingly realistic configurators.

The minimal configurator that we design for our purposes is a simplified version of ParamILS. We call it *ParamRLS* since it searches the space of parameter values using randomised local search, rather than the iterated local search employed by ParamILS. ParamRLS captures the configuration generation / evaluation loop at the heart of any algorithm configurator and thus it is reasonable to assume that positive results for ParamRLS imply that more sophisticated configurators would also succeed. If ParamRLS uses the Fixed-Target performance metric then we call it *ParamRLS-T*, and if it uses the Best-Fitness performance metric and breaks ties in favour of the configuration that made progress least recently then we call it *ParamRLS-F*.

We analyse the performance of ParamRLS on a range of increasingly complex configuration scenarios. Two characteristics of a target algorithm are likely to simplify its configuration (and the analysis thereof). First, the configuration of algorithms with only a single parameter is likely to be simpler to analyse than the configuration of algorithms with multiple parameters as potentially complex inter-parameter interactions will not occur. Second, the configuration of discretely-valued parameters is likely to be simpler to analyse than the configuration of continuously-valued parameters since the difference between the performance of continuous parameters can be made arbitrarily small, whereas this is not usually the case for discrete parameters. Since it satisfies both these criteria, we first analyse the configuration of the neighbourhood size k of randomised local search (RLS$_k$).

In order to simplify our theoretical work further, when selecting problem classes for which to configure a target algorithm it is desirable that the performance of a configuration be the same for each member of the class. This simplifies the analysis as we need not design a representative training set nor analyse the performance of the target algorithm for each member of the problem class. All problem classes considered in this thesis satisfy this criterion.

We first analyse the configuration of RLS$_k$ for the standard RIDGE benchmark problem class. This is a natural first configuration scenario to consider since the amount of progress made by a configuration in each iteration is independent of the current solution quality, and thus the neighbourhood size that maximises solution quality within a given time budget will also do so for any other. In particular, we show that the optimal neighbourhood size under each considered performance metric is $k = 1$. Hence our question is whether ParamRLS can identify the optimal parameter value for RLS$_k$ for this simple problem class, and, if so, what is the expected time required for it to do so. We rigorously prove that if the Best-Fitness performance metric is used then ParamRLS will return $k = 1$ regardless of the cutoff time within an expected quadratic (in the number of parameter values) number of configuration comparisons in the worst case, and within an expected linear number of comparisons for large enough cutoff times. We prove that even the smallest possible cutoff time of 1 is sufficient for ParamRLS-F to return $k = 1$, provided that each configuration is run $n^{3/2}$ times in each evaluation, and that for cutoff times of at least $n^{1+\varepsilon}$, for any constant $\varepsilon > 0$, a single run per evaluation is sufficient to do so (this cutoff time guarantees that, in a comparison between two configurations, the one with k closer to 1 will win with overwhelming probability). Thus ParamRLS-F is not only able to configure RLS$_k$ with respect to the Best-Fitness performance metric for any cutoff time, but is also able to configure it with respect to Optimisation-Time using cutoff times that are up to a quadratic factor smaller than that required by any configurator that actually uses this metric. On the other hand, we also prove that all configurators that use Optimisation-Time are blind for cutoff times of at most $(1 - \varepsilon)n^2$, for any constant $\varepsilon > 0$, which constitutes a lower bound on the necessary cutoff time for any such configurator that is larger than the general lower bound of $(n \ln n)/2$ proved earlier. We prove that for large enough cutoff times ParamILS is able to identify the optimal configuration within a linear number of comparisons after the conclusion of its initialisation procedure.

In practice, however, the configuration that maximises progress towards the optimum may change during execution. We therefore next consider a problem class with the more realistic characteristic that the performance of each neighbourhood size is no longer independent of the current solution quality. For this purpose, we analyse the configuration of RLS$_k$ for the standard ONEMAX problem class, where the fitness of a bit string is the number of bits which match a hidden optimal string. When k is restricted to the set $\{1,2,3,4,5\}$, we prove that ParamRLS-F can identify that $k = 5$ maximises solution quality for small linear cutoff times within 17 configuration comparisons in expectation, whereas for larger linear and all superlinear cutoff times it identifies that $k = 1$ maximises solution quality, again within 17 comparisons in expectation. Thus ParamRLS-F is able to efficiently identify the optimal configuration with respect to the Optimisation-Time performance metric (i.e. $k = 1$) using linear cutoff times whilst, as proven earlier, any Optimisation-Time-based configurator requires a cutoff time of at least $(n \ln n)/2$. Again, we prove that for large enough cutoff times ParamILS is able to identify the optimal configuration within a linear number of comparisons after the conclusion of its initialisation procedure.

Overall, we have proven that ParamRLS can efficiently identify the optimal neighbourhood sizes of the simple RLS$_k$ algorithm for simple unimodal problem classes. Our analysis reveals that, whilst problem knowledge is required for the proper application of the Fixed-Target performance metric, no advantage is obtained by using this metric, even if seeking a configuration that identifies a fixed solution quality in the smallest amount of time. In particular, in both considered scenarios, ParamRLS-F not only identifies the configuration that achieves the highest fitness within the cutoff time, but also identifies the configuration with the smallest expected optimisation time.

We then consider the more complex algorithm configuration scenario of tuning the mutation rate χ of the well-known (1+1) EA, that in each iteration perturbs each gene of an individual with probability χ/n, where n is the size of the individual.

We once again analyse the configuration of the target algorithm for RIDGE since the performance of each parameter value only changes by a small amount during execution. In this case, $\chi = 1$ is optimal under all three considered performance metrics. We prove that any Optimisation-Time-based configurator is blind for cutoff times up to $(1-\varepsilon)en^2$, for any constant $\varepsilon > 0$. On the other hand, cutoff times that are smaller by a linear factor are sufficient for ParamRLS-F to return $\chi = 1$ within a linear number of configuration comparisons in expectation, again meaning that ParamRLS-F is able to return the configuration with the smallest expected optimisation time using smaller cutoff times than those required by all configurators that explicitly optimise this metric.

We then analyse the configuration of the (1+1) EA for a problem class, LEADINGONES, where the performance of each configuration varies with respect to the current solution quality. Similarly to the neighbourhood size of RLS$_k$ when optimising ONEMAX, the mutation rate that maximises expected progress decreases as the optimum of LEADINGONES is approached. It is known that $\chi \approx 1.59$ results in the smallest expected optimisation time [18] and is thus optimal under the Optimisation-Time performance metric and under Best-Fitness when a large enough cutoff time is used. For smaller cutoff times, we expect larger mutation rates to be optimal with respect to the Best-Fitness metric. We prove that all Optimisation-Time-based configurators are blind for cutoff times of at most $0.772n^2$, whilst ParamRLS-F is able to identify the mutation rate that achieves the highest solution quality within the cutoff time for almost all quadratic cutoff times (and all cutoff times that are at least quadratic), doing so within 61 configuration comparisons in expectation. Furthermore, ParamRLS-F will efficiently return the optimal χ for cutoff times of at least $0.722n^2$ (i.e. $\approx 0.05n^2$ smaller than those required by any Optimisation-Time-based configurator), again doing so within 61 comparisons in expectation.

The analyses of the configuration of RLS_k and the (1+1) EA require us to be able to determine the outcome of comparisons between two configurations given a time budget. However, existing result on the performance of algorithms operating on a given time budget (*fixed-budget analyses* [70]) are either concerned only with the expected solution quality after a given time or provide bounds that are not tight enough for our requirements. In this thesis, we make substantial improvements over the state-of-the-art in fixed-budget analysis, providing bounds on the solution quality after a given time budget that are both tighter and that hold with a higher probability than those found in the literature.

We conclude this thesis by building upon the insights gained from the above analyses to design a mutation operator for faster algorithm configuration. Such work demonstrates how the theoretical understanding of the behaviour and performance of general-purpose algorithms can lead to the design of provably better ones. Given the simplicity of ParamRLS, the only component that may be modified to achieve better performance is the mutation operator. The above analyses enabled us to provide rigorous answers to the question of what structure is present in algorithm configuration problems. They revealed that, for the configuration scenarios considered in this thesis, the *parameter landscape* (the structure over the range of parameter values induced by the configurator) seen by ParamRLS-F is either unimodal or *approximately unimodal* (i.e. rugged with an underlying gradient towards the optimum). We prove that, in such cases, both ParamRLS and ParamILS require a linear number of configuration comparisons in expectation to identify the optimal configuration. Intuitively, this happens because ParamRLS only makes small steps within the parameter space, and is thus slow to follow the gradient, and ParamILS makes steps of random sizes, and thus does not take account of the gradient whatsoever. We modify ParamRLS to use a search operator that favours small step sizes but is also able to make larger ones with a small probability. We prove that, if the parameter landscape is approximately unimodal, then ParamRLS modified to use such an operator identifies an optimal configuration asymptotically faster than both ParamILS and ParamRLS using their default search operators, and is not much slower than them in the worst case (e.g. if the parameter landscape is deceptive with a gradient that leads away from the optimum). We verify empirically that these speed-ups also occur for the configuration of two parameters of a SAT solver.

1.2 Structure of this Thesis

In *Chapter 2*, we introduce the algorithm configuration problem. As an example application, we introduce the field of randomised search heuristics and explain why setting their parameters appropriately is essential, thus motivating the algorithm configuration field. We then formally define the algorithm configuration problem and review the related literature. We discuss different implementations of the two key components of algorithm configurators: methodologies to evaluate the performance of configurations and methodologies to generate new ones. We review historical approaches to automated algorithm configuration before giving an overview of the three most popular configurators in the literature: ParamILS [62], *irace* [91], and SMAC [64]. We conclude the chapter by reviewing the state-of-the-art in the theory of algorithm configuration. We classify the small amount of existing theoretical work into three categories: generalisation analyses, which determine the number of training instances required to ensure that the returned configuration performs well on unseen instances; convergence analyses, which determine whether a configurator will identify an optimal configuration in finite time; and worst-case performance analyses, which derive the time required to guarantee that a near-optimal configuration is returned in the most challenging configuration scenarios.

Chapter 3 provides precise definitions of all the technical ingredients required in the rest of this thesis. We begin with an introduction to the mathematical tools from the literature that we will use in our analyses. We then define the simplified algorithm configurator, ParamRLS, that we analyse. Finally, we define and motivate our choices of target algorithms and problem classes used in this thesis to evaluate the performance of algorithm configurators.

In *Chapter 4*, we present general results related to the use of the Fixed-Target performance metric. We prove a lower bound on the cutoff time that is necessary to prevent any configurator using this performance metric from being blind (i.e. behaving as if each configuration has the same performance and hence returning a random configuration). When configuring a unary unbiased search algorithm for any problem class where each instance has up to $\exp(\sqrt{n}/\log^2 n)$ optima (where n is the problem size), we prove that if the cutoff time is at most $(n \ln n)/2$ then any configurator using the Fixed-Target performance metric will be blind. We then show that setting the cutoff time large enough to allow the optimal configuration to reach the target solution quality for all training instances with overwhelming probability is a sufficient condition for Fixed-Target-based configurators to identify the optimal configuration if the optimal configuration beats all others with overwhelming probability. We emphasise the significance of this result by using it to bound the time required by ParamILS to identify the optimal configuration in such cases. However, we conclude the chapter by showing that similar behaviour can be achieved by using the Best-Fitness performance metric, and thus the above results do not constitute an advantage of Fixed-Target.

In Chapters 5 and 6 we shift our focus to specific algorithm configuration scenarios and analyse the expected time required by the ParamRLS algorithm configurator to identify the optimal parameter values of specific target algorithms for standard benchmark problem classes. We primarily analyse the impact of the choice of the cutoff time κ on the ability of the configurator to optimise the chosen performance metric.

In *Chapter 5*, we consider the configuration of the neighbourhood size k of randomised local search (RLS$_k$) for the standard RIDGE and ONEMAX problem classes. For RIDGE, the neighbourhood size $k = 1$ maximises expected progress regardless of the distance to the optimum, and hence this configuration is optimal under both the Fixed-Target and Best-Fitness performance metrics for any cutoff time. This problem class allows us to evaluate the impact of the cutoff time in scenarios where the optimal parameter value is independent of the chosen metric. Our results show that Best-Fitness allows ParamRLS to identify the optimal parameter value independent of the cutoff time, whereas an appropriate value of the cutoff time is crucial for the configurator to be successful when using the Fixed-Target performance metric.

In particular, we prove that for cutoff times of $\kappa = n^{1+\varepsilon}$, ParamRLS-F identifies that $k = 1$ is optimal, in an expected number of configuration evaluations that is linear in the number of permitted values of k. Remarkably, even if the smallest possible cutoff time of $\kappa = 1$ is used, if configurations are run at least $n^{3/2}$ times in each evaluation then it is still possible to identify that $k = 1$ is optimal. On the other hand, we prove that any configurator that uses Optimisation-Time as the performance metric is blind for all cutoff times $\kappa \leq (1-\varepsilon)n^2$, although ParamRLS-T and ParamILS using Optimisation-Time are both able to identify that $k = 1$ is optimal with cutoff times $\kappa \geq (1+\varepsilon)n^2$. Since ParamRLS-F is able to return $k = 1$ for any cutoff time, it is able to identify the optimal configuration with respect to the Optimisation-Time performance metric for any cutoff time, thus allowing it to perform the same task as Optimisation-Time-based configurators using cutoff times that are smaller by a quadratic factor, resulting in smaller overall configuration times. We then turn our attention to the configuration of RLS$_k$ for ONEMAX with permitted values of $k \in \{1,2,3,4,5\}$. Since, for RLS$_k$ maximising ONEMAX, smaller neighbourhood sizes maximise the expected progress as the distance to the optimum decreases [36], we prove that ParamRLS-F is able to identify the neighbourhood size that achieves the highest expected fitness within the cutoff time for cutoff times satisfying $0.02n \leq \kappa \leq 0.72n$ and for cutoff times $\kappa \geq 0.975n$ it identifies that $k = 1$ achieves the highest expected fitness. The linear cutoff time sufficient for ParamRLS-F to identify that $k = 1$ is optimal under the Optimisation-

Time performance metric is a logarithmic factor smaller than the general lower bound on the cutoff time required by any Optimisation-Time-based configurator, again resulting in smaller overall configuration times.

In *Chapter 6*, we analyse the configuration of the mutation rate χ of the (1+1) EA. This constitutes a more complex configuration scenario than that of tuning RLS$_k$ since the mutation rate of the (1+1) EA is continuous whereas the neighbourhood size of RLS$_k$ is discrete, and thus the difference between the performance of configurations can be arbitrarily small. As in Chapter 5, we first analyse the configuration of the (1+1) EA for RIDGE. Similarly to the optimal neighbourhood size of RLS$_k$ for this problem class, a single parameter value (in this case $\chi = 1$) maximises expected progress regardless of the position in the search space. Hence this configuration is optimal under both the Fixed-Target and Best-Fitness performance metrics. We prove that a cutoff time of $\kappa \geq \varepsilon n$ is sufficient for ParamRLS-F to return $\chi = 1$ within in a linear number of configuration evaluations, in expectation. All configurators using Optimisation-Time, however, are blind if a cutoff time of $\kappa \leq (1-\varepsilon)en^2$ is used, whilst, as shown above, linear cutoff times suffice for ParamRLS-F to identify the optimal configuration under this performance metric. We then consider the configuration of the mutation rate for the LEADINGONES problem class. In this case, we discretised the parameter space to contain the values $\chi \in \{0.1, 0.2, \ldots, 2.9, 3.0\}$. We expect $\chi = 1.6$ to be optimal for large cutoff times since $\chi \approx 1.59\ldots$ yields the smallest optimisation of any mutation rate [18], and for smaller cutoff times we would expect larger χ to be favoured, as with the neighbourhood size of RLS$_k$ for ONEMAX. We prove that this is indeed the case: for practically all quadratic cutoff times, and all cutoff times that are at least quadratic, we show that the optimal configuration under the Best-Fitness performance metric is returned by ParamRLS-F after 61 configuration evaluations, in expectation. Notably, a cutoff time of $\kappa \geq 0.722n^2$ is sufficient for ParamRLS-F to return the optimal configuration under the Optimisation-Time performance metric. We showed that configurators that use the Optimisation-Time performance metric, on the other hand, are blind for cutoff times $\kappa \leq 0.772n^2$ (i.e. $\approx 0.05n^2$ greater than the cutoff time sufficient for ParamRLS-F to configure the (1+1) EA with respect to this performance metric).

In *Chapter 7*, we investigate the performance improvements that can be obtained by modifying ParamRLS and ParamILS to use a local search operator that is designed to have good performance on unimodal parameter landscapes. We first prove that the default operators used in both configurators require at least an expected linear number of configuration evaluations before they identify the optimal configuration. We then show that using a search operator designed to have good performance on unimodal functions [31] allows the configurator to identify the optimal configuration in expected polylogarithmic time if the parameter landscape is approximately unimodal, and in logarithmic time in expectation if it is unimodal, whilst only being a logarithmic factor slower than the standard mutation operators in the worst case. We extend the theoretical results beyond single-parameter configuration scenarios with an empirical analysis, confirming the superiority of the proposed mutation operator in the more complex scenario of configuring two parameters of a SAT solver.

In *Chapter 8*, we summarise the work conducted in this thesis, present our conclusions, and discuss directions of potential future work.

1.3 Contributions of this Thesis

In this thesis, we provide the theoretical foundations for the time complexity analysis of automated algorithm configurators. The main contributions of this thesis are the following:

1. We provide rigorous general upper and lower bounds on the values of the cutoff time required by algorithm configurators using the Fixed-Target performance metric (Chapter 4). In particular:
    - It is necessary that the cutoff time is large enough to allow at least one configuration to reach the target solution quality for at least one training instance, otherwise a random configuration will be returned. A cutoff time large enough to allow the optimal configuration to reach the target solution quality for all training instances with overwhelming probability is sufficient to guarantee convergence to this configuration for algorithm configurators that can sample any configuration with a positive probability.
    - As a corollary, for Optimisation-Time the cutoff time must be at least $(n \ln n)/2$ for the efficient configuration of any unary unbiased algorithm for target problem classes with instance size n with up to $\exp(\sqrt{n}/\log^2 n)$ optima per instance.
2. We show that the given lower bound on the cutoff time when using Optimisation-Time is tight when configuring RLS$_k$ for OneMax, and may be considerably worse even for extremely simple configuration scenarios such as RLS$_k$ for Ridge (Chapters 5 and 6). In particular, we prove that any configurator that uses Optimisation-Time requires cutoff times of at least $(1-\varepsilon)n^2$ and $(n \ln n)/2$ when configuring RLS$_k$ for Ridge and OneMax, respectively, otherwise it is blind. We prove that any such configurator requires cutoff times of at least $(1-\varepsilon)en^2$ and $0.772n^2$ when configuring the (1+1) EA for Ridge and LeadingOnes, respectively, again being blind otherwise.

3. We provide the first rigorous statements regarding the performance of algorithm configurators for specific algorithm configuration scenarios (Chapters 5 and 6). We do so for the configuration of both randomised local search algorithms and evolutionary algorithms using global mutation operators for standard benchmark function classes. In particular, when configuring the neighbourhood size of RLS$_k$ for RIDGE and ONEMAX using Best-Fitness, cutoff times of 1 and $0.02n$ suffice, respectively, to identify the configuration that maximises solution quality within the cutoff time. When configuring the mutation rate of the (1+1) EA for RIDGE and LEADINGONES, εn (for a sufficiently large constant $\varepsilon > 0$) and $0.000001n^2$, respectively, suffice to do so. A quadratic number of configuration comparisons is sufficient for ParamRLS-F to do so in expectation when configuring RLS$_k$ for RIDGE, and in all other cases a linear number of comparisons suffices.

4. We give rigorous proofs of the counter-intuitive insight that the Best-Fitness performance metric allows the configurator to identify the configuration that minimises the optimisation time in considerably smaller expected time than is possible when using Optimisation-Time (Chapters 5 and 6). We show this for various simple configuration scenarios. In particular, we derive ranges of cutoff times that are sufficient for configurators using the Best-Fitness performance metric to identify the configuration with the smallest expected optimisation time. When configuring the neighbourhood size of RLS$_k$ for RIDGE and ONEMAX, cutoff times of 1 and $0.975n$ suffice, respectively. When configuring the mutation rate of the (1+1) EA for RIDGE and LEADINGONES, cutoff times of εn (for a sufficiently large constant $\varepsilon > 0$) and $0.722n^2$ suffice, respectively. These cutoff times are smaller than the expected optimisation time of any configuration (asymptotically smaller in all cases except tuning the (1+1) EA for LEADINGONES) and thus smaller than the cutoff time required by any Optimisation-Time-based tuner, allowing a Best-Fitness-based configurator to identify the configuration with the smallest expected optimisation time configuration faster than any that uses Optimisation-Time. We show that in the worst case ParamRLS-F again requires at most a quadratic (in the number of configurations) number of configuration comparisons to do so when configuring RLS$_k$ for RIDGE, and in all other cases a linear number of comparisons suffices, in expectation.

5. Since our analyses require tight bounds that hold with high probability on the solution quality achieved by given configurations, we considerably improve upon the state-of-the-art in the field of fixed-budget analysis (Chapters 5 and 6). In particular, for RLS$_k$ optimising ONEMAX and the (1+1) EA optimising LEADINGONES, we derive tight

intervals that with overwhelming probability contain the fitness of a configuration after a given time budget, for large ranges of budgets. These bounds are tighter (linear rather than superlinear) and hold with higher probability (overwhelming rather than constant or high) than existing fixed-budget analyses of these scenarios, and the ranges of time budgets for which they hold (almost all linear for ONEMAX and almost all quadratic for LEADINGONES) also exceed those of the state-of-the-art.

6. Our analyses provide rigorous statements regarding the algorithm configuration fitness landscapes for specific configuration scenarios (Chapters 5 and 6). In particular, we prove that the parameter landscape seen by ParamRLS-F is unimodal when configuring RLS$_k$ for RIDGE and when configuring the (1+1) EA for RIDGE and LEADINGONES. When configuring RLS$_k$ for ONEMAX, we prove that the parameter landscape seen by ParamRLS-F is approximately unimodal. Whilst recent experimental works have suggested that the parameter landscapes in the configuration of SAT and TSP solvers are (approximately) unimodal, this is the first time that such claims are rigorously proven for specific scenarios.

7. We exploit our theoretical results by designing a theory-driven mutation operator for algorithm configurators and prove that it considerably outperforms the standard mutation operators for both unimodal and approximately unimodal configuration landscapes, whilst in the worst case it is slower only by at most a logarithmic factor (Chapter 7).

1.4 Underlying Publications

The work presented in this thesis is based on the following publications (where authors are ordered alphabetically). In publications with k authors, the contribution of each author can be roughly quantified as $1/k$.

- Chapter 4 (*Fixed-Target Performance Metric Requires Appropriate Cutoff Times*) and Chapter 5 (*On the Configuration of the Neighbourhood Size of Randomised Local Search*) are based on:

  1. George T. Hall, Pietro S. Oliveto, and Dirk Sudholt (2019). *On the impact of the cutoff time on the performance of algorithm configurators.* In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2019)* (907-915) [51].

2. George T. Hall, Pietro S. Oliveto, and Dirk Sudholt. *On the impact of the cutoff time and the performance metric for algorithm configuration* (journal extension, under review).

- Chapter 6 (*On the Configuration of the Mutation Rate of a Simple Evolutionary Algorithm*) is based on:

3. George T. Hall, Pietro S. Oliveto, and Dirk Sudholt (2020). *Analysis of the performance of algorithm configurators for search heuristics with global mutation operators.* In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2020)* (823–831) [52]. Nominated for the best paper award in the track "General Evolutionary Computation and Hybrids".

- Chapter 7 (*Faster Algorithm Configuration by Exploiting Parameter Space Unimodality*) is based on:

4. George T. Hall, Pietro S. Oliveto, and Dirk Sudholt (2020). *Fast perturbative algorithm configurators.* In *Proceedings of the 16th Parallel Problem Solving from Nature Conference (PPSN XVI)* (19–32) [53].

Chapter 2

Algorithm Configuration

2.1 Introduction

In this chapter, we provide an overview of the field of automatic algorithm configuration and the need for the time complexity analysis of algorithm configurators. We start by introducing the field of general-purpose optimisation and algorithms designed for this task, in particular randomised search heuristics (Section 2.2). This will allow an appreciation of the (often numerous) parameters that must be set for their application and the importance of choosing their values appropriately. We proceed by defining the algorithm configuration problem precisely in Section 2.3, followed by an overview of the field and a classification of parameter tuners in Sections 2.4 to 2.8. Finally, in Section 2.9, we outline existing theoretical analyses of algorithm configuration.

2.2 Randomised Search Heuristics

When faced with a computational problem for which no algorithm exists, a user must decide whether to design an algorithm for the problem or to use a general-purpose one. The first approach requires a good understanding of the problem at hand in order to construct an algorithm that is correct and has good performance guarantees. For complex problems, gathering such knowledge and the algorithm design process itself are often time-consuming tasks (perhaps even prohibitively so). Hence it is often more practical – even necessary – to instead employ a general-purpose algorithm: a method designed to be applicable to a wide variety of problems and that can therefore be readily applied to the task at hand. Examples of such algorithms are local search, randomised local search, and simulated annealing (Algorithm 1), in addition to nature-inspired approaches such as evolutionary algorithms

(Algorithm 2). General-purpose algorithms have been successfully applied in a wide variety of settings, and are able to relieve the user of the burden of designing an algorithm tailored to the specific problem [46]. Furthermore, they do not require that the user have an in-depth understanding of the problem at hand. This characteristic is particularly useful in *black-box optimisation*, where information about the problem (other than a measure of the quality of a solution) is either too complex to be appropriately understood or is unavailable entirely.

Algorithm 1: Simulated annealing for the maximisation of a function f (adapted from [71]). The function $C(t)$ is the cooling schedule, a function that controls the probability of accepting worse solutions at time t.

```
1 $t \leftarrow 1$
2 $x_1 \leftarrow$initial individual
3 while termination criterion not satisfied do
4     $y \leftarrow$ individual generated from $x_t$
5     with probability $\min\left\{1, \exp\left(\frac{f(y)-f(x)}{C(t)}\right)\right\}$ do $x_{t+1} \leftarrow y$
6     otherwise $x_{t+1} \leftarrow x_t$
7     $t \leftarrow t+1$
```

Algorithm 2: $(\mu + \lambda)$ EA for the maximisation of a function f (adapted from [3]).

```
1 Initialise population of $\mu$ individuals u.a.r.; Let multiset $X^0$ be population at time 0.
2 while termination criterion not satisfied do
3     $X' \leftarrow X^{(t)}$
4     for $i = 1, \ldots, \lambda$ do
5         Choose $x \in X'$ using a selection operator
6         Create $x'$ by flipping each bit of $x$ with probability $p$
7         $X' \leftarrow X' \cup \{x'\}$
8     Create multiset $X^{(t+1)}$ by deleting $\lambda$ individuals in $X'$ with lowest $f$-value
9     $t \leftarrow t+1$
```

The algorithms mentioned above (e.g. randomised local search and evolutionary algorithms) are examples of *randomised search heuristics* (RSHs). This class of algorithms makes random choices during the optimisation process in order to explore the space of possible solutions given little knowledge about the problem at hand, usually requiring only a way to represent solutions, a means to generate new solutions, and a method of evaluating solution quality. Given a problem class, however, it is often not obvious which RSH is preferable (i.e. which will have better performance). Furthermore, even once an algorithm has been decided upon, there are still issues that must be resolved before it may be applied.

One of the most important and challenging choices to be made is how to set the values of its often numerous parameters (e.g. the cooling schedule in simulated annealing; the selection pressure, the mutation rate, and the parent and offspring population sizes in the $(\mu+\lambda)$ EA). Often, a randomised search heuristic will only be capable of providing satisfactory solutions to a problem if its parameters are set appropriately.

The general-purpose nature of randomised search heuristics may lead us to hope that it is possible to design an algorithm (and a configuration of its parameters) that performs well for all problems. Unfortunately, however, the no free lunch (NFL) theorem roughly states that when averaged over all optimisation problems, all randomised search heuristics have the same performance [118]. In other words, no RSH outperforms any other over all optimisation problems. Therefore, if Algorithm $\mathcal{A}$ has better performance than Algorithm $\mathcal{B}$ on one set of problems then the NFL theorem implies that $\mathcal{A}$ necessarily has worse performance than Algorithm $\mathcal{B}$ when averaged over all other problems. The NFL theorem demonstrates the need for parameter tuning: if we treat two configurations of an algorithm as two different algorithms entirely then it becomes clear that parameter settings that perform well on one set of problems will be outperformed by other configurations on other problem sets. Since no configuration performs well over all problems, it is advisable to configure an algorithm for the problem class on which we want to apply it. One may object by claiming that only subclasses of problems appear in practice, and that these do not satisfy the conditions of the NFL theorems. Nevertheless, it is widely understood that the success of RSHs depends crucially on the chosen parameter values both from theoretical analyses for specific problems (e.g. [25]) and experience in practical applications [45].

In this thesis, we assume that we have decided which algorithm we want to use, and focus on the task of identifying optimal parameter values for specific problem classes. In recent years, several approaches to algorithm configuration have seen widespread use, most prominently ParamILS, *irace*, and SMAC (see Sections 2.8.1, 2.8.2, and 2.8.3, respectively). However, how well these algorithm configurators actually perform in terms of the quality of the identified parameter values within a given time budget is unclear. In this thesis, we create the theoretical foundations to allow the performance of algorithm configurators to be quantified.

2.2.1 Types of Parameters

There are several types of algorithmic parameter. It is common for a configurators to deal with each type of parameter in a different way. We follow López-Ibáñez *et al.* in defining three types of parameter:

- *Categorical:* There is no order in the values of the parameter (e.g. the selection mechanism used in an evolutionary algorithm).
- *Ordinal:* Similar to categorical parameters, but the values have an implicit order (e.g. a parameter with values $\{$`low, medium, high`$\}$).
- *Numerical:* There is an explicit order in the values of the parameter (e.g. the mutation rate and number of offspring in an evolutionary algorithm).

In addition to these classes, a parameter is called *conditional* if it is only relevant if another parameter takes a certain value. For example, suppose that an evolutionary algorithm has a categorical parameter controlling whether crossover is used. Then any parameter that controls the crossover procedure itself is only relevant if the first parameter is set to `true`. The second parameter is therefore conditional on the value of the first.

2.3 The Algorithm Configuration Problem

In this section, we define the algorithm configuration problem in a formal manner, following the definition given by Stützle and López-Ibáñez [111].

Denote the algorithm to be configured (the *target algorithm*) as $\mathcal{A}$ and the (possibly infinite) set of all permitted configurations of $\mathcal{A}$ as Θ (the *parameter space*). Denote the algorithm $\mathcal{A}$ with its parameters set according to some configuration $\theta \in \Theta$ as $\mathcal{A}(\theta)$. The aim of algorithm configuration is to identify the configuration θ^* that optimises the performance of $\mathcal{A}$ for some problem class Π, consisting of a certain (possibly infinite) number of instances.

Formally, let $\mathcal{I}$ be the distribution of the problem instances[3] over Π, and let $C_{\mathcal{I}}(\mathcal{A}(\theta))$ denote the (performance) cost of the configuration $\mathcal{A}(\theta)$ when run on Π, with respect to the instance distribution $\mathcal{I}$. Then the *algorithm configuration problem* is that of identifying a configuration θ^* such that

$$\theta^* \in \arg\min_{\theta\in\Theta} C_{\mathcal{I}}(\mathcal{A}(\theta)).$$

In practice, two main problems often arise when establishing the cost of a configuration. The first problem is that the distribution $\mathcal{I}$ of the instances over Π is unknown. This issue is generally dealt with by trying to identify a training set of problem instances that is

[3]An instance distribution may be used to capture the likelihood of certain instances arising in practice. Birattari uses the example of an algorithm to be configured for the travelling salesperson problem (TSP) [14]. The problem class Π is an infinite set of instances of the TSP and $\mathcal{I}$ captures the likelihood that each instance is encountered in practice. For example, if the application domain is a delivery company, problem instances including a visit to a block of flats are far more likely to occur in practice than instances including a visit to the middle of the ocean.

representative of the problem class such that the optimal configuration identified is also optimal over all possible problem instances in Π (i.e. it *generalises* well).

It is therefore common to assess the performance of the returned configuration by running it on a distinct *test set* of problem instances. It is crucial that both the training set and the test set are representative of the distribution of problem instances that will be encountered by the target algorithm when it is applied in practice. However, considerable problem knowledge is required to ensure this.

The second problem is that for most problem classes of interest (e.g. NP-Hard problems), it cannot be expected that any configuration θ will allow an algorithm $\mathcal{A}$ to optimise all instances efficiently (i.e. in polynomial time). Hence using the actual optimisation time as a cost measure is often infeasible (unless it is assumed or known that the instances encountered are 'easy'). To overcome this issue, several *performance metrics* have been proposed in the literature (discussed in Section 2.5.1). The choice of the performance metric depends on what is more valuable to the final user of the algorithm (often taken to be optimisation time or solution quality reached within a time budget).

Throughout this thesis, we use the term *parameter landscape*[4] to refer to the fitness landscape induced over the parameter space by a configuration evaluation procedure (i.e. the performance metric and other configurator settings). We say that a parameter landscape is *unimodal* if and only if every non-optimal configuration has a neighbouring configuration (under an appropriate definition of neighbourhood) that is evaluated to have a lower cost than it with overwhelming probability.

2.4 Algorithm Configurators

In this section, we detail the state-of-the-art in algorithm configuration. Virtually all algorithm configurators may be decomposed into two components: a methodology to *generate* new configurations and a methodology to *evaluate* how well a given configuration performs. In a recent review of the field, Huang *et al.* remark "Different generating techniques (also referred to as sampling methods) and evaluation approaches form different tuning algorithms" [61]. Figure 2.1 gives a general framework for algorithm configurators[5]. High-level modularity

[4]Tanabe notes that this terminology is not universal, however "parameter landscape" appears to be the most popular choice in the literature [112].

[5]We diverge from the taxonomy laid out by Huang *et al.* in [61] in that we do not use the distinction of *simple generate evaluate* configurators (where a set of configurations is generated and the best-performing one is identified), *iterative generate-evaluate* configurators (where this process is iterated, i.e. where new configurations are generated based on the best performers in the previous iteration), and *high-level generate-evaluate* configurators (where good configurations are generated by another fast configuration process, and then the best-performing among these is identified). We do not use this taxonomy since all three classes are

can be observed: the two highlighted components (i.e. generation and evaluation) can be altered to create new configurators.

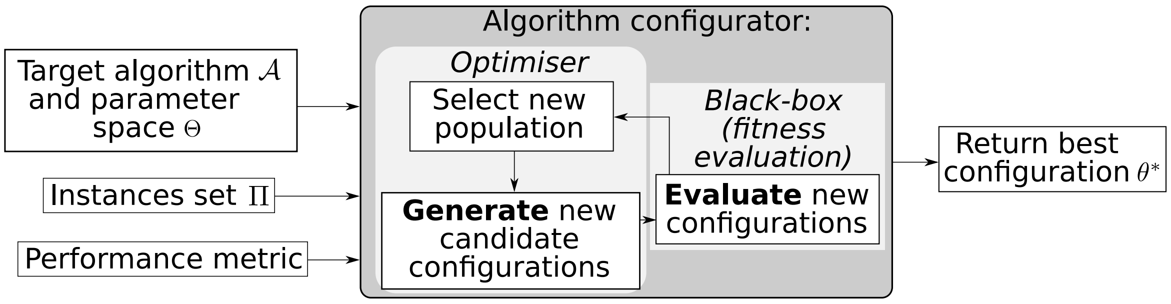

Algorithm configurator:
Optimiser
Target algorithm $\mathcal{A}$ and parameter space Θ
Instances set Π
Performance metric
Select new population
Generate new candidate configurations
Black-box (fitness evaluation)
Evaluate new configurations
Return best configuration θ^*

Fig. 2.1 The general framework of algorithm configurators (based on Figure 1 in [61]). New configurators can be created by modifying the generate and evaluate procedures.

Any general-purpose optimisation algorithm may be used for the configuration generation component of an algorithm configurator, and thus, in principle, there exist at least as many configuration generation techniques as there are optimisation algorithms. Apart from the search space in which they operate (i.e. that of parameter values), what distinguishes algorithm configurators from traditional function optimisers is therefore the configuration evaluation module. Whilst in function optimisation the evaluation of solution quality is often a straightforward (albeit perhaps time-consuming) deterministic task, estimating the performance of an algorithm configuration on a problem class is far from trivial as it is often not possible to run it on all problem instances and its performance on each instance depends on the amount of time for which it is run. Hence, in contrast to traditional black-box optimisation, the configuration evaluation module has to 'learn' how well a candidate configuration performs, a task for which, in principle, any machine learning algorithm may be used. As in traditional machine learning, a training set of instances is provided as input and the aim of the evaluation module is to use the training set to estimate the performance of the candidate configuration in practice (i.e. on unseen instances).

In the following sections, we give a survey of the components that comprise an algorithm configurator. We first give an overview of commonly-used methods for the evaluation of configurations, in Section 2.5, and examine methods used to reduce the number of times that each configuration is run during the tuning process in Section 2.6. We then, in Section 2.7, examine different optimisers that have been used in the literature for the configuration generation procedure. Following this general overview, we then examine in detail the working principles of the three most prominent algorithm configurators: ParamILS [62],

effectively equivalent: high-level generate-evaluate configurators are simple generate-evaluate configurators with a sophisticated initial generation mechanism, and simple generate-evaluate configurators are iterative generate-evaluate configurators that are run only for a single iteration.

irace [91], and SMAC [64] (in Sections 2.8.1, 2.8.2, and 2.8.3, respectively), in addition to a recent family of theory-inspired, configurators designed to have worst-case performance guarantees (Sections 2.8.4 and 2.8.5).

2.5 Algorithm Configuration Evaluation

In practice, one must decide how to determine the cost of a configuration, $C_{\mathcal{I}}(\mathcal{A}(\theta))$. We now give an overview of the decisions that must be made when evaluating a configuration and then examine the ways in these choices may be optimised in practice.

Ideally, as is often the case in randomised search heuristics, the computation of the fitness of a solution (in this case a configuration) will not be noisy (that is, the evaluation of the same configuration will always have the same outcome). However, this is rarely the case in algorithm configuration. In fact, there are two sources of potential stochasticity when evaluating the performance of a configuration. Firstly, its performance may vary greatly when run on different problem instances (this variance will be distributed according to the distribution of problem instances $\mathcal{I}$). Secondly, the runtime of a randomised algorithm (such as a randomised search heuristic) is a random variable with its own inherent variance due to the random choices that are made during its execution.

It is common to take $C_{\mathcal{I}}(\mathcal{A}(\theta))$ to be the cost of running the configuration $\mathcal{A}(\theta)$ on the problem instance π_i, which we denote $C(\mathcal{A}(\theta),\pi_i)$, averaged over problem instances distributed according to $\mathcal{I}$. That is, $C_{\mathcal{I}}(\mathcal{A}(\theta)) = \mathrm{E}_{\pi_i \sim \mathcal{I}}[C(\mathcal{A}(\theta),\pi_i)]$. If configuring a stochastic algorithm, as is often the case in the algorithm configuration literature and is indeed the case throughout this thesis, it is not possible to derive this quantity analytically and instead it must be estimated computationally by running $\mathcal{A}(\theta)$ over multiple problem instances. We call a single computation of $C_{\mathcal{I}}(\mathcal{A}(\theta))$ an *evaluation*.

The different choices that have to be made to perform a computational evaluation of $\mathcal{A}(\theta)$ lead to several algorithm configurator parameters:

- The definition of the training set of problem instances $\Pi' \subseteq \Pi$.
- The number of *runs* r per evaluation (i.e. the number of times an instance is drawn using the distribution $\mathcal{I}$ and the configuration run on it).
- The method used to *aggregate* performance measures over multiple runs of a configuration.
- The *performance metric* (i.e. which quality measure to use to assess the performance of different configurations for an instance π_i, e.g. "the best-found solution within a

time budget” or “the time required to reach a solution of at least a given quality”). It is important to note that different performance metrics may yield different optimal configurations.

- The *cutoff time* κ (i.e. the time for which a configuration is executed in a single run in an evaluation).

Let T be the *runtime of the configurator*, defined as the number of configuration comparisons that it carries out before it is able to return the optimal configuration θ^*. In this thesis, we estimate T for different algorithm configuration scenarios. Since for all considered configuration scenarios it is the case that, with overwhelming probability, the optimal configuration will not lose a comparison once sampled, T is taken to be the number of comparisons before it is sampled for the first time.

If κ and r are static then the *configuration time* is $\mathcal{B} = T \cdot \kappa \cdot r$, and calculating T suffices to fully determine the tuning budget required to identify the optimal configuration. Naturally, many configurators attempt to reduce the amount of time required to learn the relative performance of a set of configurations as quickly as possible, and thus may reduce r and κ (i.e. not use the same values for each evaluation). In this case, $\mathcal{B} \leq T \cdot \kappa \cdot r$. Note that if the performance of each configuration is known (i.e. it is provided by some *oracle*), and thus need not be determined computationally, then $\mathcal{B} = T$.

We now examine how to set each of these parameters. We give the most attention to the performance metric and cutoff time parameters since these play a central role in our analyses in this thesis.

Training set The training set should be chosen such that it is representative of the context in which the algorithm will be applied. That is, performance on the training set should correlate with performance on the instance sets that the target algorithm will encounter in practice. This goal is common in machine learning, and is referred to as *generalisation*. If the instance distribution is known, then generalisation is trivial. Hence optimising the parameters of an algorithm for a given (or chosen) distribution is a much easier task than what happens in more realistic scenarios where the instance distribution also has to be learned.

Number of runs The required number of runs per evaluation r varies with the configuration scenario: it is reasonable to assume that cases where the performance of the target algorithm over the training set has high variance require larger r than cases where this variance is small.

In ParamILS, these runs are distributed uniformly across the training set [62]. For example, in ParamILS, if $r = 100$ and the training set is of size 10, then the configuration will be run exactly 10 times on each training instance. Thus, increasing r leads to a configuration being run on each instance a higher number of times. It is therefore advisable to increase r in proportion to the variance of the performance of configurations on the same training instance.

Aggregation method The aggregation of the results of numerous runs is most commonly implemented as the mean performance over all runs, although other statistical parameters may be used.

2.5.1 Performance Metrics

We now describe the most common performance metrics used in the literature. In optimisation, the user often seeks the best solution identifiable within an available time budget. Indeed, it is common to use a `MAXTIME` parameter as a stopping criterion for an algorithm (where `MAXTIME` indicates the amount of time available for the optimisation). In such cases, it is natural to use the Best-Fitness performance metric.

Best-Fitness: Returns the best solution quality identified within the cutoff time. In the event that two configurations reach the same fitness value within the cutoff time, a natural tie-break is to favour the configuration that reached this fitness first. In this thesis, we assume that this tie-break is always used when Best-Fitness is used. This performance metric avoids the need to specify a target solution quality in advance. The *irace* configurator was originally designed for use with the Best-Fitness performance metric [21]. We say that the configuration with the highest expected solution quality within the cutoff time (and thus sought by Best-Fitness) is *F-optimal*.

However, Best-Fitness may not be the ideal performance metric in other settings. For instance, in some applications solutions of at least a given quality must be sought. In such cases, the Fixed-Target performance metric is more natural.

Fixed-Target: Returns the time taken by a configuration to reach a given solution quality, most commonly the optimum (in which case it is called *Optimisation-Time*). Throughout this thesis, we refer to the time taken by a configuration to reach the target solution quality its "runtime". In order to avoid potentially running a configuration for an infinite amount of time on unsolvable problem instances, it is common to use a cutoff time when evaluating the quality of a configuration under this performance metric. Given a cutoff time κ, the cost of a configuration is the time taken to reach the target solution quality if this time is no greater than κ, and otherwise it is taken to be $\mathcal{P} \cdot \kappa$ for a penalisation constant $\mathcal{P}$. The Fixed-Target (specifically, Optimisation-Time) performance metric is typically used in applications of ParamILS [62] and SMAC [64]. The worst-case-tailored configurator LEAPSANDBOUNDS is also designed for use with the Fixed-Target performance metric [115]. We say that the configuration that has the smallest expected runtime is *T-optimal*, and if the target solution quality is the optimum then we also refer to this configuration as *O-optimal*.

When using the Fixed-Target performance metric, if the runtimes from multiple runs are averaged then the resulting performance measurement is called *penalised average runtime* (PAR) [62]. A penalisation constant of 10 is commonly used in the literature, and in this case the performance measurement is called *PAR10*.

Whilst Best-Fitness and Fixed-Target are the two most commonly used performance metrics in the literature, variants of Fixed-Target have been proposed in attempts to provide a more robust measure of performance.

Penalised quantile runtime (PQR): A recent variant of the Fixed-Target performance metric intended to reduce the effect of anomalous runtimes on the fitness of a configuration (e.g. a configuration A with one very slow run and otherwise fast runs may have a worse mean runtime than another configuration B even if all runs of B are only slightly faster than the slow run of A) [16]. Given $p \in [0,1]$ and m runs of a configuration, PQR returns the penalisation constant $\mathcal{P}$ multiplied by the cutoff time κ if fewer than $\lfloor mp+1 \rfloor$ runs reach the target solution quality within the cutoff time, and otherwise returns the $(100 \cdot p)$-th quantile of its runtimes.

κ-Fixed-Target: This performance metric also reduces the effect of anomalously challenging problem instances by capping the runtime on any problem instance at time κ [78]. Hence the optimal configuration in terms of this performance metric is the configuration with the smallest expected runtime when all runtimes greater than κ are taken to be κ. The optimal configuration under this performance metric is therefore the one with the smallest penalised average runtime with cutoff time κ and a penalisation constant of 1. This metric is subtly different from the conventional Fixed-Target metric: under Fixed-Target, the optimal configuration is the one with the smallest *uncapped* runtime, whereas under κ-Fixed-Target it is the one with the smallest runtime capped at time κ. Fixed-Target is thus equivalent to ∞-Fixed-Target. Structured Procrastination [78] and Structured Procrastination with Confidence [79] are designed for use with the κ-Fixed-Target performance metric.

δ-Percentile-Fixed-Target: The δ-Percentile-Fixed-Target performance metric again mitigates the impact of extremely difficult problem instances by taking the performance of a configuration to be the time that it requires to reach the target solution quality for a $1-\delta$ fraction of problem instances [116]. Fixed-Target is thus equivalent to 0-Percentile-Fixed-Target. This metric differs from penalised quantile runtime since the cost of running a configuration is unbounded: unlike in PQR, there is no cutoff time within which the configuration must optimise an instance, but rather the measurement of the performance of a configuration is as large as is necessary for it to reach the target for the specified

proportion of instances. CapsAndRuns is designed for use with the δ-Percentile-Fixed-Target performance metric [116].

Hypervolume-based performance metrics: López-Ibáñez and Stützle configure algorithms to have good anytime performance (i.e. to optimise their capability to produce good solutions regardless of the amount of time for which they are run). They consider the solution quality obtained by a configuration in conjunction with the time required to obtain it [92]. They optimise for this metric using the *hypervolume* measure common in multi-objective optimisation [120], which, informally, in this context measures the extent to which a configuration is able to identify solutions of high quality in less time than its competitors. This measure converts the multi-objective optimisation problem into a single-objective problem to which established configuration techniques can then be applied.

Multi-objective performance metrics: Bossek *et al.* argue that the Fixed-Target performance metric represents a measure of performance that is a combination of the optimisation time of an algorithm and its probability of success (within the cutoff time). Instead of converting this multi-objective problem into a single-objective one, as done by López-Ibáñez and Stützle with the hypervolume-based metric, they propose minimising these quantities explicitly (removing the potential impact of the choice of penalisation constant) using a multi-objective performance metric [16].

2.5.2 Cutoff Time

The cutoff time is the amount of time for which a configuration is executed in a single run of an evaluation. Algorithm configurators may use either a static cutoff time or a dynamic one that is adapted during the configuration process. The choice of cutoff time and whether it may be set dynamically depends on the choice of performance metric (i.e. Best-Fitness is incompatible with a dynamic cutoff time as it seeks the configuration that achieves the highest solution quality within the cutoff time: a dynamic cutoff time would change the optimal configuration throughout the tuning process). Algorithm configurators for which an upper bound on the maximum cutoff time is fixed throughout the algorithm configuration process have been referred to as *incumbent-driven* [78]. Kleinberg *et al.* define the class of incumbent-driven configurators as follows.

Definition 1 (Incumbent-Driven Configurators [78])**.** A configurator is *incumbent-driven* if the cutoff time of each run of a configuration is static (i.e. set at the beginning of the configuration process) or dynamic and chosen to be at most the runtime of a previously-run configuration.

Configurators with no such pre-determined upper bound have been referred to as *non-incumbent-driven* [78]. Typically, they initially use small cutoff times and increase them if necessary to allow configurations to reach the target solution quality.

Static cutoff times

Without additional problem knowledge, or if the Best-Fitness performance metric is being used, it is reasonable to set the cutoff time as the maximum amount of time for which the user is willing to run the target algorithm in practice (MAXTIME), since different configurations may be the ones that find solutions of better quality for different time budgets.

If the Fixed-Target performance metric (or a variant) is used, and if a bound on the expected time taken by reasonable configurations to reach the target solution quality is known, then the cutoff time may be set to this value. Otherwise, it may be chosen using trial and error.

Adaptive cutoff times

Two opposing techniques have been used in the literature to adapt the cutoff time throughout the configuration process when the Fixed-Target performance metric (or a variant) is used. The first approach is to pre-define a maximum cutoff time for each run and try to use a smaller cutoff time if possible (this method is incumbent-driven). The second methodology, used by non-incumbent-driven configurators, starts with small cutoff times and increases them if a larger cutoff time is required for a configuration to reach the target solution quality.

Decreasing cutoff times (adaptive capping) A technique called *adaptive capping* can reduce the time spent evaluating configurations that are performing poorly if the Fixed-Target performance metric is used [62]. As implemented in ParamILS [62] and SMAC [64], adaptive capping terminates the run of a configuration once it becomes impossible for it to obtain a mean time to reach the target that is smaller than the configuration against which it is being compared (in *trajectory-preserving* adaptive capping); or smaller than the current best-found configuration (in *aggressive* adaptive capping). For example, if, in an evaluation involving 100 runs, the configuration θ_1 takes a total of 100 seconds to reach the target solution quality (and thus has a mean runtime of 1 second), its competitor, θ_2, can be run with a cutoff time of $100+\varepsilon$ seconds on the first problem instance since after that point it becomes impossible for it to obtain a lower mean runtime than θ_1. This upper bound on the cutoff time can be reduced further with each run of the configuration as the lower bound on its runtime increases. Adaptive capping is also implemented in *irace* [91], where the cutoff time is set to the time within which the configuration must optimise the instance in order to improve on the median runtime of the set of best-performing configurations [21].

Increasing cutoff times (non-incumbent-driven configurators) In this approach, small cutoff times are used initially and are only increased if the configuration fails to reach the target solution quality within it [78]. In this way, a configurator is able to use small cutoff times when these are sufficient and large cutoff times when necessary.

Using an incumbent-driven approach presents a problem when used in combination with the Fixed-Target performance metric, since setting the cutoff time too small means that a configurator is unable to differentiate between configurations. By increasing the cutoff time when it proves too small for a configuration to reach the target solution quality, non-incumbent-driven configurators effectively attempt to learn appropriate cutoff times for configuration/instance pairs and avoid having to identify appropriate cutoff times in advance.

2.6 Configuration Evaluation Methodologies

Conducting many observations of the performance of a configuration can be a time-consuming process. In many cases, however, it may be possible to reduce the time spent evaluating a configuration if it is clearly performing worse than the configuration(s) against which it is being compared. We now give an overview of the most common methods to do so. We first define the naïve approach to selecting the number of runs (i.e. where no effort is made to reduce the number of runs even if there is a clear difference between two configuration) and then review the most common means of reducing the number of runs in the evaluation of a configuration.

2.6.1 Static Runs

The simplest method to deal with the stochasticity in the evaluation process is to keep the number of runs in an evaluation *static*. When evaluated, a configuration is run the same number of times regardless of its performance relative to other configurations. Therefore more time than necessary may be spent in an evaluation where the outcome could have been determined earlier. This evaluation technique is employed, for example, by the BasicILS version of ParamILS [62]. Throughout this thesis, the analysed configurators use this simple approach to create a performance baseline for algorithm configuration.

2.6.2 Dynamic Runs

Many configurators reduce the number of runs of poorly-performing configurations. We now review the most prominent approaches.

1. Intensification This method terminates the evaluation of a configuration as soon as it begins to perform worse than the configuration against which it is being compared. When comparing a newly-sampled configuration against one whose performance has already been evaluated, the newly-sampled configuration is evaluated on the same sequence of problem instances as its opponent and after each problem instance the evaluation is terminated if the newly-sampled configuration has a worse aggregated performance. If the end of the sequence of instances is reached (i.e. both configurations have been evaluated on the same sequence of instances) and the new configuration has still not been shown to perform worse then it is concluded that the new configuration is at least as good as its opponent. Intensification is used in the FocusedILS (i.e. default) version of ParamILS [62] and in SMAC [64].

2. Racing (F-Race) As with intensification, racing aims to reduce the time spent evaluating underperforming configurations and thus to spend more time deciding between those with better performance. Racing was introduced by Maron and Moore to compare models in machine learning [95]. It was adapted to the context of algorithm configuration by Birattari *et al.* as F-Race [15] and is employed by the popular *irace* configurator [91]. According to Huang *et al.* "The essential idea of [the] racing method is to evaluate candidate configurations incrementally on a stream of instances. As soon as sufficient (statistical) evidence is gathered against some candidates, these configurations are discarded, and the race continues on [with] the surviving candidates" [61].

In a single race, F-Race compares the performance of each configuration on a sequence of problem instances. That is, for some sequence of instances $\pi_1, \pi_2, \ldots, \pi_m$, F-Race first computes the performance of all configurations in the race on π_1, and then on π_2, and so on. At some point, it begins to eliminate configurations that can be shown to be performing worse than their rivals by an amount that is statistically significant. We illustrate a race between eight configurations ($\theta_1, \ldots, \theta_8$) in Figure 2.2. Each dot represents where a configuration was run on a specific problem instance. All configurations were tested on the first three problem instances, before θ_8 was eliminated. More configurations are eliminated in later rounds. The race is eventually won by θ_1, θ_2, and θ_3 after they have been evaluated on twelve instances.

The fact that F-race evaluates configurations concurrently can yield significant speedups in comparison to intensification, which compares configurations against one that has already been evaluated. For example, when using the Fixed-Target performance metric, intensification

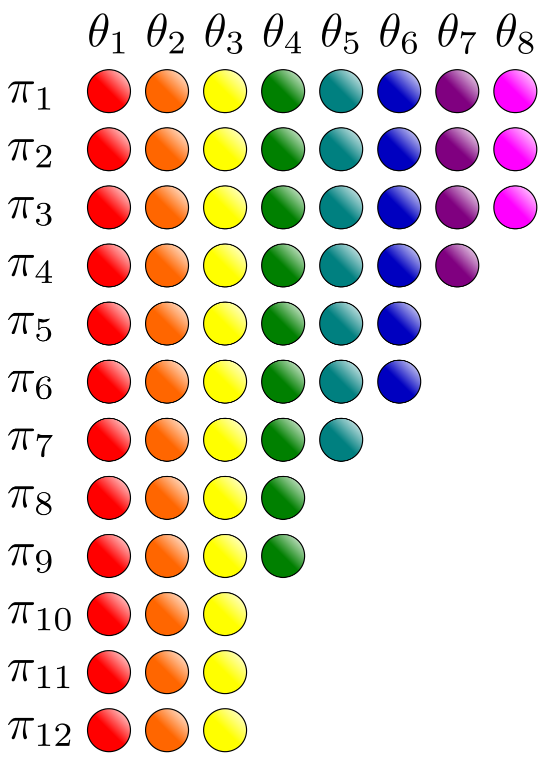


Fig. 2.2 A race between eight configurations.

will be slower than F-race if the configuration that has already been evaluated, θ_1, is much slower than the newly-generated configuration θ_2 against which it is being compared, as much time has already been spent evaluating the performance of the slow θ_1. This problem is avoided in F-race since both θ_1 and θ_2 are evaluated simultaneously, and thus the evaluation can be terminated once it becomes clear that θ_2 outperforms θ_1.

Extending this logic, racing can compare more than two configurations in a more efficient manner than intensification, resulting in considerable time savings if a small number of these configurations clearly outperform the others after being evaluated on only a small number of problem instances. Whilst F-Race will terminate the evaluation once it has identified this behaviour, intensification, on the other hand, is only able to identify the optimal configuration using a series of pairwise comparisons, with the best-performing configuration being used as the opponent in comparisons against the remaining configurations. However, if no configurations with good performance are evaluated until near the end of the evaluation process, then time will be wasted on deciding between configurations with bad performance.

3. Sharpening In this method, all configurations are initially evaluated on the same number of problem instances. However, at some pre-determined time the number of instances on which each configuration is run is doubled. This may happen many times during the configuration process. As a result, better configurations will be evaluated on more instances than worse ones. This method is used by Bartz-Beielstein *et al.* in sequential parameter optimisation (SPO) [10], but only given a name in a survey by Smit and Eiben [109]. Unlike racing and intensification, it has not seen much use in the literature.

4. Hyperband Hyperband is a method for reducing the number of runs when evaluating configurations in hyperparameter optimisation (HPO), a subset of algorithm configuration problems concerned with choosing the "hyperparameters" of a machine learning algorithm that best learn the "parameters" of a dataset[6] [44].

Hyperband iterates a simpler HPO algorithm called SuccessiveHalving [68]. Given a set of configurations, SuccessiveHalving repeatedly eliminates the worst-performing $1-\frac{1}{\eta}$ fraction and then multiplies the number of runs used to evaluate the remaining ones by η (by default, $\eta = 2$, giving SuccessiveHalving its name). However, when using SuccessiveHalving the user must decide the trade-off between the number of hyperparameter configurations to evaluate (i.e. exploration) and the number of runs used to evaluate each one (i.e. exploitation). Hyperband automates this decision by running SuccessiveHalving with different exploration/exploitation trade-offs.

The approach used by Hyperband cannot be applied to algorithm configuration in general as typically target algorithms do not have a training stage in the evaluation process where the number of runs can be reduced (i.e. optimisation algorithms, unlike machine learning algorithms, do not need to be trained in order to be evaluated: see footnote 6) [78].

2.6.3 Surrogate Models (No Runs)

The evaluation of the performance of a configuration is usually the most time-consuming component of algorithm configuration. Some configurators therefore replace the majority of evaluations on the actual training set with the evaluation of a less costly 'surrogate' function that predicts the performance of the configuration. This method therefore eliminates runs in the majority of configuration evaluations since in most cases configurations are evaluated using a surrogate model. Model-based configuration evaluations are used in the configuration generation procedures of SPOT and SMAC. We cover these procedures and the model-based framework in Section 2.7.4.

[6]In general, the configuration evaluation process in hyperparameter optimisation has two stages. First, the target algorithm with its hyperparameters set according to the configuration is trained on a training dataset, and second, its performance is assessed by running the trained algorithm on the test dataset. Hyperband adapts the size of the training set, aiming to avoid wasting time training poorly-performing configurations on large datasets.

2.7 Configuration Generation Methodologies

In this section, we give an overview of the optimisers used to select and generate new configurations. As this can be done using standard black-box optimisation algorithms, there are in principle at least as many choices for methods for configuration generation as there are optimisation algorithms.

2.7.1 One-shot methods

The most simple algorithm configurators only involve a single round of configuration generation. In some cases, such as F-Race [15] and Structured Procrastination [78], the best member of a given set of candidate configurations is determined, an approach called *one-shot optimisation* [29]. If all possible configurations in this set are generated and evaluated, then the configurator is an example of *full factorial design* [48]. However, since generating and evaluating all possible configurations will be time-consuming for large parameter spaces, random sampling may be used instead, with samples being drawn either from the input set of candidate configurations or generated given constraints on the parameter space. If information about the parameter space is already known, then this can be used to guide the sampling procedure, otherwise configurations may be sampled uniformly at random. F-Race has been modified to sample configurations uniformly at random given bounds on the range of each parameter, and the configurator using this technique was shown to outperform its full factorial variant [7].

2.7.2 Black-box methods

Due to the black-box optimisation nature of algorithm configuration, many tuners use heuristic methods to generate new configurations. One of the earliest efforts to address the algorithm configuration problem was a heuristic approach (specifically, a genetic algorithm). Grefenstette proposed a *meta-GA* to optimise the parameter values of another genetic algorithm [49]. In the meta-GA, a population of 50 configurations was run on a set of problem instances and then the best-performing configurations were used as the parents for the parameters of the next generation of GAs. A static number of runs were used to evaluate configurations. The tuned algorithm exhibited a 3% performance improvement over a GA using parameter values recommended by de Jong in an earlier work [30].

A more sophisticated approach also using a genetic algorithm for generating new configurations was proposed by Ansótegui *et al.* with the introduction of the *gender-based genetic algorithm (GGA)* [2]. GGA assigns each offspring configuration one of two genders: either competitive or non-competitive. The configurations in the top ten percent of the competitive population (with respect to fitness) are then recombined with the same number of configurations from the non-competitive population (chosen uniformly at random). An ageing mechanism is used to encourage diversity of configurations by removing from the population any configuration that is older than a pre-determined limit. Configurations are evaluated using racing, as this allows for the parallel evaluation of many configurations. Ansótegui *et al.* show that GGA considerably outperforms a standard (i.e. non-gender-based) genetic algorithm. When both tuners are run for the same number of iterations, GGA terminates twenty times earlier and returns configurations that are of a substantially higher quality than those returned by the standard GA. They also show that it outperforms the well-known ParamILS configurator (see Section 2.8.1) when configuring the SAT solvers SAPS, SPEAR, and SAT4J. Both configurators were run for the same time budget and the configurations generated by GGA had better performance than those yielded by ParamILS [2]. However, despite these promising initial results, GGA has failed to gain widespread use, and configurators such as *irace* and SMAC have been shown to outperform it considerably [64, 105].

Other types of evolutionary algorithms have also been applied to the problem of configuration generation. For example, Yuan *et al.* used the state-of-the-art evolution strategy (a family of evolutionary algorithm for continuous search spaces) CMA-ES [54] to generate configurations of swarm intelligence algorithms, evaluating configurations using either a static number of runs or racing [119]. Yuan *et al.* compared the configurations generated by CMA-ES against those generated by two black-box approaches common in numerical optimisation, BOBYQA [102] and mesh adaptive direct search (MADS) [4]. BOBYQA is shown to generate configurations that outperform those generated by CMA-ES and MADS, and those generated by CMA-ES outperform those generated by MADS.

The popular configurators ParamILS and *irace* both use heuristic methods to generate new configurations. ParamILS employs iterated local search (ILS) [93] and *irace* generates new configurations using an estimation of distribution algorithm (EDA) [83]. These approaches are discussed in more detail in Sections 2.8.1 and 2.8.2, respectively.

2.7.3 Experimental design

The field of *experimental design* is concerned with how to best investigate the "response" of a system (in this case, the target algorithm) to different "factors" (in this case, parameter values) [48]. The CALIBRA configurator employs a method from experimental design, called "Taguchi fractional factorial experimental designs" [107], to follow ever-improving paths of configurations. Taguchi fractional factorial experimental designs are a systematic method to vary the factors of a system such that a small number of experiments are needed to identify the response of the system to each factor. In each iteration, CALIBRA creates nine new configurations using the Taguchi method and identifies the best-performing of these. It then iterates this process, initially generating nine new configurations based on this configuration. It restarts the search process once it reaches a local optimum. However, the Taguchi framework is only applicable to scenarios with five factors, and thus CALIBRA is limited to configuring only five parameters. Therefore, whilst it has been shown to improve the performance of a number of algorithms, for example simulated annealing and tabu search, for combinatorial optimisation problems such as job scheduling [1], its restriction to the configuration of five parameters appears to have limited its widespread adoption.

2.7.4 Surrogate model-based techniques

This family of methods uses a model of the parameter landscape to sample new configurations. These are then evaluated on the training set, and this information is used to update the model and the process is iterated, with the new model being used to generate new configurations. A general framework for this approach is given in Figure 2.3. Recall that surrogate models are not only a means of generating new configurations but also a means of evaluating them (by predicting their performance), as discussed in Section 2.6.3.

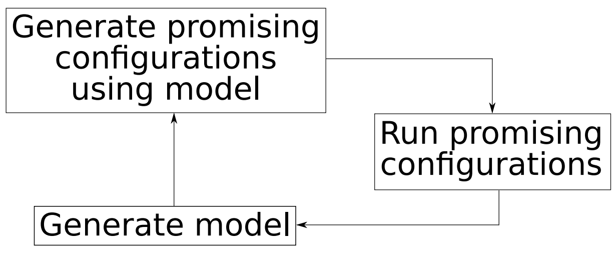

Generate promising configurations using model
Run promising configurations
Generate model

Fig. 2.3 The surrogate-model-based configuration generation framework.

Hutter *et al.* call a configurator that generates new configurations in this manner a *sequential model-based optimiser* (SMBO) [64]. In this approach, the configurator generates an initial set of configurations and evaluates them on the training set. The tuner then fits a model to the performance data obtained from these evaluations. Using this model, the tuner generates a set of new configurations in areas of the model with low confidence or which appear to be close to the optimum, called the “areas of greatest expected potential gain”. The configurator then evaluates this new set of configurations on the training set, and generates a new model to reflect the new information which has been learned. The process is then repeated, beginning by generating a new set of parameters based on this new model.

Bartz-Beielstein *et al.* were the first to apply this methodology to algorithm configuration with a technique called *sequential parameter optimisation (SPO)* [10], implemented in the

SPO Toolbox (SPOT) [11]. Evaluation of configurations is carried out using sharpening. However, SPOT can only optimise parameters for single problem instances: it is unable to tune an algorithm for classes of problems, as is often desired in real-world applications. In addition, it could initially only handle numerical parameters (although this shortcoming has subsequently been addressed by extending it to be able to handle numerical ones [12]).

To address these shortcomings, Hutter *et al.* introduced SMAC [64] (discussed in detail in Section 2.8.3). The models used by SMAC allow it to optimise categorical parameters in addition to numerical ones. Furthermore, SMAC is able to optimise parameter values over multiple problem instances.

2.8 Detailed Algorithm Configurator Descriptions

In this section, we first present ParamILS in detail, which until recently was the state-of-the-art, and for which we derive performance guarantees in this thesis. We then examine the two most widely-used configurators in the literature, *irace* and SMAC, giving an in-depth look at the state-of-the-art. The final two sections give an overview of a recently-introduced class of configurators that, uniquely for the algorithm configuration literature prior to this thesis, has performance guarantees.

2.8.1 ParamILS

ParamILS generates new configurations using an iterated local search [62]. In contrast to many other tuners, it maintains a population of only a single configuration.

We give the pseudocode for ParamILS in Algorithm 3. Essentially, after having selected the best initial configuration out of $R+1$ randomly-generated ones, ParamILS repeats the following loop:

1. Apply an iterated local search procedure, called *IterativeFirstImprovement*, to the current solution, thus locating a local optimum;

2. Perturb the identified local optimum by performing a random walk of length s through the parameter space (that is, perform s random moves where in each move it selects a configuration that differs from the current one in a single parameter value);

3. Re-initialise the search procedure with probability p_{restart}.

The *IterativeFirstImprovement* procedure is described in Algorithm 4 and defines the neighbourhood of a configuration as all configurations that differ from it in a single parameter value. Given an input configuration θ, the procedure visits its undiscovered[7] neighbours (all the neighbours that have not been evaluated in the current call of *IterativeFirstImprovement*) $\mathit{UndiscNbh}(\theta)$ in a randomised order and then accepts the first one it finds that is at least as good as it. It then performs the same procedure on this newly discovered configuration. This process is repeated until it reaches a configuration with no undiscovered neighbours that are at least as good as it.

One of two different approaches may be applied for configuration evaluation. A simple approach, called "BasicILS", uses a static number of runs, whereas "FocusedILS", uses intensification. Both methods optionally use adaptive capping. The configuration evaluation procedure is called $\mathit{better}(\theta_1, \theta_2)$ in Algorithms 3 and 4 and returns `true` if and only if configuration θ_1 is evaluated to be better than configuration θ_2.

In this thesis, we will use ParamILS as the basis for the design of a bare-bones algorithm configurator, called ParamRLS (Section 3.3), which we use to set up the mathematical basis for the time complexity analysis of algorithm configurators.

Algorithm 3: ParamILS pseudocode, adapted from [62].

```
Input: Initial configuration $\theta_0 \in \Theta$, parameters $R, p_{\text{restart}}$, and $s$.
Output: Best parameter configuration $\theta$ found.
1  for $i = 1, \ldots, R$ do
2      $\theta \leftarrow$ random $\theta \in \Theta$
3      if $\mathit{better}(\theta, \theta_0)$ then $\theta_0 \leftarrow \theta$
4  $\theta_{\text{inc}} \leftarrow \theta_{\text{ils}} \leftarrow \mathit{IterativeFirstImprovement}(\theta_0)$
5  while not TerminationCriterion do
6      $\theta \leftarrow \theta_{\text{ils}}$
7      for $i = 1, \ldots, s$ do
           // Random perturbation step of size $s$
           // Nbh contains all neighbours of a configuration
8          $\theta \leftarrow$ random $\theta' \in \mathit{Nbh}(\theta)$
9      $\theta \leftarrow \mathit{IterativeFirstImprovement}(\theta)$
10     if $\mathit{better}(\theta, \theta_{\text{ils}})$ then $\theta_{\text{ils}} \leftarrow \theta$
11     if $\mathit{better}(\theta_{\text{ils}}, \theta_{\text{inc}})$ then $\theta_{\text{inc}} \leftarrow \theta_{\text{ils}}$
12     with probability $p_{\text{restart}}$ do $\theta_{\text{ils}} \leftarrow$ random $\theta \in \Theta$
13 return $\theta_{\text{inc}}$
```

[7]The consideration of *undiscovered* neighbours fixes a typo in the pseudocode given in [62] which could lead to an infinite loop if equally good configurations belong to the same neighbourhood. The pseudocode in Algorithm 4 follows the implementation of ParamILS available at `http://www.cs.ubc.ca/labs/beta/Projects/ParamILS`.

Algorithm 4: IterativeFirstImprovement(θ) procedure, adapted from [62].

```
1 repeat
2     $\theta' \leftarrow \theta$
3     forall $\theta'' \in UndiscNbh(\theta')$ in randomised order do
          // UndiscNbh contains all undiscovered neighbours of a configuration
4         if better($\theta'', \theta'$) then $\theta \leftarrow \theta''$; break
5 until $\theta' == \theta$
6 return $\theta$
```

2.8.2 Iterated F-Race and *irace*

The Iterated F-Race (I/F-Race) configurator maintains a population of configurations and repeatedly generates new ones from probability distributions biased towards the best performers, where sets of configurations are evaluated using F-Race (outlined in Section 2.6.2) [7]. This approach makes it an *estimation of distribution* algorithm (EDA), a class of evolutionary algorithms [83]. The I/F-Race framework is implemented, generalised, and improved upon in the *irace* configurator [91]. One improvement of *irace* over I/F-Race is to use an *elitist* iterated racing procedure, which ensures that the best-performing configurations have been evaluated on the highest number of problem instances (similarly to intensification – Section 2.6.2). We give the pseudocode of *irace* in Algorithm 5.

Algorithm 5: *irace* pseudocode, adapted from [91]. Given parameter space Θ, total allowed tuning budget B. B_i is the budget allocated to race i, B^{used} stores current budget used.

```
1 $\Theta_1 \leftarrow \text{SampleUniform}(\Theta)$ // Initialise u.a.r.
2 $\Theta^{\text{elite}} \leftarrow \text{Race}(\Theta_1, B_1)$ // $\Theta^{\text{elite}}$ now contains winners of first race.
3 $j \leftarrow 1$
4 while $B^{used} \leq B$ do
5     $j \leftarrow j + 1$
6     $\Theta^{\text{new}} \leftarrow \text{Sample}(\Theta, \Theta^{\text{elite}})$ // Sample new configurations from distributions biased
        towards $\Theta^{\text{elite}}$.
7     $\Theta_j \leftarrow \Theta^{\text{new}} \cup \Theta^{\text{elite}}$
8     $\Theta^{\text{elite}} \leftarrow \text{Race}(\Theta_j, B_j)$ // $\Theta^{\text{elite}}$ now contains winners of $j$-th race.
9 return $\Theta^{\text{elite}}$
```

Given a parent configuration, *irace* samples new values for numerical parameters from a truncated normal distribution[8]. The mean of this distribution is taken to be the value of the

[8]A variant of the normal distribution for which the probability density function is 0 outside a given range of permitted values.

parameter in the parent configuration and its standard deviation decreases with the length of time that the tuner has been running. Categorical parameters are sampled from distributions that are biased towards the value of that parameter in the parent configuration. In this case, the distribution is updated each time a new configuration is sampled and is inherited by the child configuration.

Historically, *irace* has focused on the maximisation of solution quality (i.e. the Best-Fitness performance metric) rather than the minimisation of runtime. However, the addition of adaptive capping to *irace* has enabled it to achieve competitive performance also when run using the Fixed-Target performance metric [21].

2.8.3 SMAC

The sequential model-based optimiser (see Section 2.7) SMAC ('sequential model-based algorithm configuration') uses a model of the parameter landscape in order to generate new configurations that are expected to have good performance [64]. It evaluates these newly-generated configurations on the training set and generates a new model to reflect the information that has been learned. SMAC uses a variant of intensification to evaluate the performance of configurations. A configuration is initially run on only one instance, then on two, then four, and so on. When the length of the sequence list is doubled, if the new configuration has a higher cost than the current best-found, then it is eliminated and the evaluation is terminated. We give the pseudocode for SMAC in Algorithm 6.

Algorithm 6: Pseudocode for SMAC, adapted from [64]. Given parameter space Θ, the array D stores information about performance of configurations from previous evaluations.

```
1 $(D, \Theta_{\text{inc}}) \leftarrow \text{Initialise}(\Theta)$
2 repeat
3     $\mathcal{M} \leftarrow \text{FitModel}(D)$
4     $\Theta_{\text{new}} \leftarrow \text{SelectConfigurations}(\mathcal{M}, \Theta_{\text{inc}})$ // Select configurations with highest expected
        positive improvement based on model $\mathcal{M}$.
5     $(D, \Theta_{\text{inc}}) \leftarrow \text{Intensify}(\Theta_{\text{new}}, \Theta_{\text{inc}}, \mathcal{M}, D)$ // Identify best configuration in $\Theta_{\text{new}}$.
6 until termination condition is satisfied
7 return $\Theta_{\text{inc}}$
```

SMAC generates new configurations by identifying those with high *expected positive improvement* (EPI) over the best configuration seen so far. Intuitively, EPI is maximised for configurations where the model of the parameter space has high uncertainty and low predicted cost. This allows SMAC to balance exploration and exploitation. SMAC uses a multi-start local search from the ten previously-run configurations with the highest EPI in order to find ten new configurations with locally optimal EPI. In addition, it samples 20,010 configurations uniformly at random. It compares the newly-sampled configurations against the current best-found (using evaluations on the training set rather than predictions using the model) and replaces it if a better-performing configuration has been discovered.

Experimental evidence suggests that SMAC yields better algorithm configurations than ParamILS and GGA for a range of tuning problems [64]. According to the recent review by Huang *et al.*, "SMAC is the currently most powerful automatic parameter tuning method" [61].

2.8.4 Structured Procrastination

In this section, we provide an overview of Structured Procrastination (SP), the first non-incumbent-driven configurator. In Section 2.8.5, we outline variants of this configurator that have subsequently been developed. We detail this class of configurators because it has worst-case performance guarantees that we will review when describing the state-of-the-art in the theory of algorithm configuration in Section 2.9.3.

By default, all configurators in this class are examples of one-shot optimisation (defined in Section 2.7.1): given a set of permitted configurations, they attempt to identify the member of this set with the best performance. They are all designed for use with variants of the Fixed-Target performance metric. Structured Procrastination seeks the optimal configuration according to the κ-Fixed-Target performance metric.

Structured Procrastination maintains a list of configurations and, associated with each of them, a queue of problem instances and cutoff times. In each queue, the instance with the lowest associated cutoff time is always at the head. SP also estimates the mean runtime of each configuration based on its execution thus far. In each iteration, it runs the configuration with the smallest estimated mean runtime on the problem instance at the head of its corresponding queue. If the run is successful (i.e. the configuration reaches the optimum of the problem instance within the associated cutoff time) then this problem instance is removed from the queue. Otherwise, the cutoff time associated with the instance is doubled and the instance is sent to the back of the queue. Upon termination, SP returns the configuration with the smallest mean runtime. The user must first specify the amount by which the returned configuration can be worse than the optimal one, ε, and also the probability ζ with which this guarantee can fail to hold. When the algorithm is terminated, it returns the value of δ for which its chosen configuration is (ε, δ)-optimal[9] with probability at least $1 - \zeta$. We give the pseudocode for SP in Algorithm 7.

Algorithm 7: Structured Procrastination, adapted from [78].

```
Input: Parameter space $\Theta$, precision parameter $\varepsilon \in (0, \frac{1}{3})$, failure probability
       $\zeta \in (0,1)$, lower and upper runtime bounds $\kappa_0$ and $\bar{\kappa}$, sequence $j_1, j_2, \ldots$ of
       (instance, seed) pairs.
1  $\beta \leftarrow \log_2(\bar{\kappa}/\kappa_0)$
2  for $\theta \in \Theta$ do
3      $k_\theta \leftarrow 0$
4      $\ell_\theta \leftarrow \lceil 12\varepsilon^{-2} \ln(2\beta|\Theta|/\zeta) \rceil$
5      $Q_\theta \leftarrow$ empty double-ended queue
6      for $\ell = 1, \ldots, \ell_\theta$ do
7          $R_{\theta\ell} \leftarrow 0$
8          Insert $(\ell, \kappa_0)$ at tail of $Q_\theta$
9  repeat
10     $i \leftarrow \arg\min_{\theta' \in \Theta} \left\{ \frac{1}{k_{\theta'}} \sum_{\ell=1}^{k_1} R_{\theta'\ell} \right\}$
11     Remove $(\ell, \kappa)$ from head of $Q_i$ // i.e. $j_\ell$ is instance at head of $Q_i$
12     if $R_{i\ell} == 0$ then
13         $k_i \leftarrow k_i + 1$
14         $q_i \leftarrow \lceil 12\varepsilon^{-2} \ln(3\beta|\Theta|(k_i)^2/\zeta) \rceil$
15     if RUN$(i, j_\ell, \kappa)$ terminates in time $t \leq \kappa$ // RUN$(i, j_\ell, \kappa)$: run config. $i$ on $j_\ell$ with cutoff $\kappa$
16      then
17         $R_{i\ell} \leftarrow t$
18     else
19         $R_{i\ell} \leftarrow \kappa$
20         Insert $(\ell, 2\kappa)$ at tail of $Q_i$
21     while $|Q_i| < q_i$ do
22         $\ell_i \leftarrow \ell_i + 1$
23         $R_{i,\ell_i} \leftarrow 0$ Insert $(\ell_i, \kappa)$ at head of $Q_i$
24 until termination criterion satisfied
25 return $\theta^* \leftarrow \arg\max_{\theta \in \Theta} \left\{ \sum_{\ell=1}^{k_\theta} R_{\theta\ell} \right\}$, $\delta = \frac{\sqrt{1+\varepsilon} q_{\theta^*}}{k_{\theta^*}}$
```

[9]We define the notion of (ε, δ)-optimality precisely in Section 2.9.3. Briefly, a configuration is (ε, δ)-optimal if with probability at least $1 - \delta$ its average runtime is larger than that of the optimal configuration by a factor of at most $1 + \varepsilon$.

2.8.5 Other Non-Incumbent-Driven Configurators

Inspired by SP, Weisz *et al.* created the non-incumbent-driven configurator LEAPSANDBOUNDS (LB) [115], and then subsequently built upon it with CAPSANDRUNS (CR) [116]. LB and CR are designed for use with the Fixed-Target and the δ-Percentile-Fixed-Target performance metrics, respectively.

LB estimates the runtime of the optimal configuration (initially as $16\kappa_0/7$ for a given lower runtime bound κ_0) and then attempts to find a configuration with a smaller mean runtime than this by estimating the mean runtime of all configurations using runs with this cutoff time. If it finds any such configurations then it returns the one with the smallest mean runtime. If it cannot find such a configuration then it doubles its estimate of the optimal runtime (i.e. doubles the cutoff time) and repeats the search. LB allows the runtime estimation routine for each configuration to be terminated early if the variance of its runtimes across problem instances is low. We give the pseudocode for LB in Algorithm 8.

Algorithm 8: LEAPSANDBOUNDS, adapted from [115].

```
Input : Parameter space $\Theta$, precision parameter $\varepsilon \in (0, \frac{1}{3})$, percentile parameter
        $\delta \in (0,1)$, failure probability $\zeta \in (0,1)$, lower runtime bound $\kappa_0$, instance
        distribution $\mathcal{I}$.
1  $\kappa \leftarrow \frac{16}{7}\kappa_0, k \leftarrow 0, \mathcal{J} \leftarrow$ empty list
2  while True do
3      $k \leftarrow k+1$
4      $b \leftarrow \lceil 44 \log_2 \left( \frac{6|\Theta|k(k+1)}{\zeta} \frac{1}{\delta\varepsilon^2} \right) \rceil$
5      Add $b - |\mathcal{J}|$ new instances sampled from $\mathcal{I}$ to $\mathcal{J}$
6      for $\theta \in \Theta$ do
7          $\bar{Q}_\theta \leftarrow$ RUNTIMEEST$(\theta, \mathcal{J}, \delta, \kappa)$ // Calcuate mean runtime of config. $\theta$ using cutoff $\kappa$
8      if $\min_\theta \bar{Q}_\theta < \kappa$ then
9          return $\arg\min_\theta \bar{Q}_\theta$
10     $\kappa \leftarrow 2\kappa$
```

Like LB, CR operates in two phases. In the first phase, it estimates a cutoff time for each configuration within which it will solve at least a $1-\delta$ fraction of instances. In the second phase, it uses these cutoff times to compute the mean runtime of each configuration. It does so using a modified Bernstein race[10] [96] in which if it becomes clear that the configuration cannot outperform the best-found configuration then the evaluation is stopped. CR computes the average runtime for each configuration and returns the configuration with the smallest

[10]Bernstein races are modified version of F-Races that additionally allow the evaluation process to be terminated early if there is low variance in the runtime of configurations.

corresponding value. Like LB, CR terminates the evaluation of the runtime of a configuration if these times exhibit low variance. Additionally, CR stops evaluating configurations if they can be shown to perform significantly worse than another configuration. We give the pseudocode for CR in Algorithm 9.

Algorithm 9: CAPSANDRUNS, adapted from [116].

```
Input: Parameter space $\Theta$, precision parameter $\varepsilon \in (0, \frac{1}{3})$, percentile parameter
       $\delta \in (0,1)$, failure probability $\zeta \in (0, \frac{1}{6})$, instance distribution $\mathcal{I}$.
1  $\Theta' \leftarrow \Theta$
2  for configuration $\theta \in \Theta$ in parallel do
3      $t_\theta \leftarrow$ QUANTILEEST$(\theta, 1-\delta, \zeta)$ // Estimate $1-\delta$-th percentile of runtimes of $\theta$
4      if QUANTILEEST$(\theta, 1-\delta, \zeta)$ aborted // i.e. $\theta$ will not be faster than best-found
5       then
6          Remove $\theta$ from $\Theta'$
7      else
8          Run RUNTIMEEST$(\theta, t_\theta, \varepsilon, \zeta)$, abort if $|\Theta'| = 1$ // Est. runtime of $\theta$ with cutoff $t_\theta$
9          if RUNTIMEEST$(\theta, t_\theta, \varepsilon, \zeta)$ rejected $\theta$ // i.e. $\theta$ will not be faster than best-found
10          then
11             Remove $\theta$ from $\Theta'$
12         else
13             $\bar{Y}(\theta) \leftarrow$ return value of RUNTIMEEST$(\theta, t_\theta)$ // Upper bound on runtime of $\theta$
14 return $\theta^* = \arg\min_{\theta \in \Theta'} \bar{Y}(\theta)$ and $t_{\theta^*}$
```

LB and CR differ from SP since the latter always assumes the worst-case scenario and therefore runs each configuration long enough to be able to distinguish between it and its opponents in the worst case. LB and CR make no such assumption, as can be seen in their ability to terminate evaluations early if it is already clear that the configuration is not optimal. Thus LB and CR are likely to perform better than SP in non-worst-case scenarios. Unlike SP, however, LB and CR are not *anytime*: the user must specify δ at the beginning of the process and then wait until the method terminates, whereas SP successively tightens δ throughout the run, and hence the best-found configuration can be returned at any time.

Recently, Kleinberg *et al.* created a modification of SP, called *Structured Procrastination with Confidence* (SPC) that "maintains the anytime property [of SP] while aiming to observe only as many [runs] as necessary to separate the runtime of each configuration from that of the best alternative" [79]. They show that SPC exhibits near-optimal worst-case running time (as with SP) but, unlike SP, it also has near-optimal runtime in non-worst-case scenarios. They prove that SPC is faster than SP in configuration scenarios where many configurations have poor performance. They conduct experiments comparing SPC against SP and LB for the configuration of the SAT solver *minisat* and the results suggest that it is able to identify good configurations (in the top 1%) in a fraction of the time required by LB. Unfortunately, there is no comparison against CR since they were developed concurrently.

Similarly to SP, SPC maintains a queue of problem instances and cutoff times for each configuration. However, unlike the mean runtime that is tracked by SP, SPC instead maintains a lower confidence bound on the runtime of each configuration which becomes tighter the more times the configuration is run. In each iteration of the main loop, the configuration with the smallest lower bound is selected and run on the (problem instance, random seed) pair at the head of its associated queue. If the optimum is reached within this cutoff time then the lower bound is updated, otherwise the cutoff time is doubled and the instance sent to the back of the queue. At the end of the tuning process, the configuration that has been run on the most problem instances is returned. We give the pseudocode for SPC in Algorithm 10.

Algorithm 10: Structured Procrastination with Confidence, adapted from [79].

```
  Input : Parameter space $\Theta$.
1 for $\theta \in \Theta$ do
2     $C_\theta \leftarrow$ new configuration tester for $\theta$ // Class for testing configuration $\theta$.
3     $C_\theta$.initialise() // Create double-ended instance queue
4 while termination criterion not satisfied do
5     $\theta \leftarrow \arg\min_{\theta' \in \Theta}\{C_{\theta'}.\texttt{GetLCB()}\}$ // identify config. with smallest lower confidence bound
6     $C_\theta$.ExecuteStep() // Execute a run of configuration $\theta$
7 return configuration with most instances either completed or in queue
```

2.9 Theory of Algorithm Configuration

In this section, we provide an overview of the literature regarding the theory of algorithm configuration. We classify the literature into three broad themes: analyses of the necessary size of the training set and distribution of runs across it to ensure good generalisation of the performance of the identified configuration (Section 2.9.1); proofs of convergence for SMAC and the FocusedILS version of ParamILS (Section 2.9.2); worst-case runtime analyses (Section 2.9.3).

2.9.1 Generalisation Analysis

An important choice when running an algorithm configurator is how to distribute the configuration budget across the training set (i.e. how many training instances to use and how many times to run the target algorithm on each instance). Birattari provides a simple answer: for a configuration scenario with an infinite training set and where we are willing to run a configuration N times in each evaluation, a configuration should be run once on N distinct problem instances [13]. This minimises the variance of the estimate of the performance of the configuration.

Recently, Liu *et al.* built on this result by considering the more realistic scenario where the training set is finite [90]. They assume that the training set is of size K and that we are again willing to run a configuration N times in each evaluation. Liu *et al.* prove that to minimise the variance of the obtained performance estimate a configuration should be run n_i times on instance i, where $n_i \in \{\lfloor \frac{N}{K} \rfloor, \lceil \frac{N}{K} \rceil\}$ and $\sum_{i=1}^{K} n_i = N$. This implies that runs should be distributed between problem instances as evenly as possible. These works provide theoretical validation for the approaches used in ParamILS, SMAC, and *irace*.

Related to the optimal number of times to run a configuration on each training instance is the question of how many training instances should be used to increase the probability that configurations with good performance across unseen problem instances are identified. Called the *sample complexity*, this quantity is commonly studied in the analysis of machine learning algorithms (*learning theory*), where the impact of the size of the training set is of interest. In the remainder of this section, we survey results that derive the sample complexity of configuration evaluation.

Balcan *et al.* analysed the number of training instances required for the performance of an algorithm during the configuration process to be close to its expected performance across all problem instances (i.e. including unseen instances) [8]. They derive sample complexity results for cases where the class of functions expressing the performance of the target algorithm with respect to its parameters is piecewise-structured. This condition holds if the parameter space can be split (using a set of *boundary functions*) into sections within which the performance-expressing function is well-behaved (i.e. belongs to a function class of low complexity, such as constant functions or linear functions). This structure has been shown to arise in multiple real-world algorithm configuration scenarios [8].

The size of the training set required for good generalisation depends on the complexity of both the class of boundary functions and the class that quantifies the performance of the target algorithm within each section. The following theorem shows that the number of training instances required to sample a configuration with performance that generalises well grows with the complexity of these two function classes. The complexity of these classes is defined in terms of *pseudo-dimension* and *VC-dimension*, measures which we define in Appendix A (page 193).

Theorem 1 (Adapted from the text of Section 1.1 in [8])**.** Let $\mathcal{C}(\theta,\pi_i)$ denote the performance of configuration θ on problem instance π_i. Let $\mathcal{F} = \{\mathcal{C}(\theta,\pi_i) \mid \pi_i \in \Pi\}$ and let $\mathcal{G}$ be a class of functions that partitions the parameter space into sections within which the performance of the target algorithm is described by a function in $\mathcal{F}$. If there are at most k boundary functions, and if the performance of an algorithm is in $[0,H]$ then, with probability $1-\delta$, the difference between the average performance of the algorithm on the training set of size N and its expected performance is

$$\tilde{O}\left(H\sqrt{\frac{1}{N}\left(\mathrm{Pdim}(\mathcal{F}^*) + \mathrm{VCdim}(\mathcal{G}^*)\ln k + \ln\frac{1}{\delta}\right)}\right),$$

where $\mathcal{F}^*$ and $\mathcal{G}^*$ are the duals of $\mathcal{F}$ and $\mathcal{G}$, respectively, and Pdim and VCdim are pseudo-dimension and VC-dimension, respectively, and where $\tilde{O}$ ignores logarithmic factors.

Balcan *et al.* generalise this result, showing that good generalisation guarantees can also be obtained in cases where the class of performance-quantifying functions (i.e. $\mathcal{F}$ in Theorem 1) is not piecewise-structured [9]. Letting $\mathcal{C}(\theta,\pi_i)$ denote the performance of configuration θ on problem instance π_i, they show that if the class $\mathcal{F} = \{\mathcal{C}(\theta,\pi_i) \mid \pi_i \in \Pi\}$ can be approximated by a function class $\mathcal{H}$ under the L^∞-norm (i.e. the absolute difference between $\mathcal{F}$ and $\mathcal{H}$ is small at all points in the parameter space), and if $\mathcal{H}$ is of low complexity, then the number of training instances required for good generalisation is small. However, if this approximation only holds under the L^p-norm, for some $p < \infty$ (i.e. the difference between the two function classes is small under the L^p-norm but *not* under the L^∞-norm), then they show that it is not possible to obtain such guarantees.

Theorem 2 formalises the first of these claims, showing that the upper bound on the absolute difference between the performance of a configuration during the tuning process and its performance across the complete set of problem instances decreases as the complexity of the dual of the approximating function class $\mathcal{H}$ decreases. The complexity of a function class is this time quantified in terms of *empirical Rademacher complexity*, which we define in Appendix A (page 193).

Theorem 2 (Combination of Theorems 4.2 and 4.3 in [9])**.** Given a training set $\mathcal{S}$ of size N, let $\mathcal{C}(\theta,\pi_i)$ denote the performance of configuration θ on training instance π_i and $\mathcal{C}(\theta)$ denote the performance of θ over all problem instances Π, distributed according to $\mathcal{I}$. Then, with high probability over the draw of the training set $\mathcal{S}$, for any configuration θ,

$$\left|\frac{1}{N}\sum_{\pi_i\in\mathcal{S}}\mathcal{C}(\theta,\pi_i) - \mathcal{C}(\theta)\right| = \tilde{O}\left(\frac{1}{N}\sum_{\pi_i\in\mathcal{S}}\sup_{\pi_i}|\mathcal{C}(\theta,\pi_i) - h_{\pi_i}(\theta)| + \mathfrak{R}_{\mathcal{S}}(\mathcal{H}^*) + \sqrt{\frac{1}{N}}\right),$$

where $h_\theta(\pi_i)$ is an approximation of $\mathcal{C}(\theta, \pi_i)$ that holds under the L^∞-norm, $\mathcal{H} = \{h_\theta(\pi_i) \mid \pi_i \in \Pi\}$, $\mathcal{H}^*$ is the class of dual functions of $\mathcal{H}$, and $\mathfrak{R}_\mathcal{S}(\mathcal{H}^*)$ is its empirical Rademacher complexity given the training set $\mathcal{S}$. $\tilde{O}$ suppresses logarithmic factors.

We do not reproduce the negative result of Balcan *et al.* that demonstrates that similar claims of good generalisation do not hold if the approximation of the dual function class holds only under the L^p-norm, for some $p < \infty$, since the theorem statement would introduce much unnecessary notation. However, we will briefly give an intuition of the result. The positive claim (Theorem 2) holds because if $\mathcal{F}$ can be approximated by $\mathcal{H}$ under the L^∞-norm and the dual function class of $\mathcal{H}$, $\mathcal{H}^*$, has low complexity, then $\mathcal{F}$ also has low complexity, which implies the claim. However, if the approximation only holds under the L^p-norm, for some $p < \infty$, then this is not the case: it is possible for $\mathcal{H}^*$ to have low complexity and $\mathcal{F}$ to have high complexity.

2.9.2 Convergence Analysis

Hutter *et al.* prove that both SMAC and the FocusedILS version of ParamILS converge to an optimal configuration when operating in a finite parameter space [62, 63]. These results require that the cost of a configuration is measured using a consistent estimator, i.e. that grows closer to the true value of the cost as the number of runs of the target algorithm increases:

Definition 2. An estimator of the cost of a configuration θ after running it N times, $\hat{c}_N(\theta)$, is a *consistent estimator* of the cost of the configuration, $c(\theta)$, if and only if, $\forall \varepsilon > 0$,

$$\lim_{N \to \infty} \Pr(|\hat{c}_N(\theta) - c(\theta)| < \varepsilon) = 1.$$

This definition allows us to now state the convergence results of Hutter *et al.*:

Theorem 3 (Combination of [62, Theorem 9] and [63, Theorem 4])**.** Consider FocusedILS or SMAC configuring an algorithm using a consistent estimator of the cost of a configuration. If the configuration space is finite, then the probability that the configurator finds the optimal configuration approaches one as the time spent running the configurator approaches infinity.

For both configurators, the proof of this theorem uses the following reasoning. First, each iteration of the configurator takes finite time and thus the number of times that each configuration has been evaluated approaches infinity as the number of iterations of the configurator grows. Second, after many comparisons between two configurations, the configuration with the lower cost will be evaluated to be better than the worse configuration tends toward 1 (since the cost estimator is consistent). Third, since the configuration space is finite, all pairwise comparisons of configurations will have been carried out as the number of iterations of the configurator tends to infinity. Combining these observations, we conclude that the probability that the optimal configuration has been found approaches 1 as the total number of iterations of the configurator tends to infinity.

2.9.3 Worst-Case Performance Analysis

In the following, we call a pair consisting of a problem instance and a random seed simply an "instance", thus making the runtime of a randomised algorithm on an instance deterministic. Let the runtime of the configuration θ on the instance π_i be denoted by $R(\theta, \pi_i)$[11] and denote the τ-capped runtime of θ on π_i as $R(\theta, \pi_i, \tau) = \min\{R(\theta, \pi_i), \tau\}$. Thus the τ-capped runtime of a configuration on an instance is the time required by the configuration to reach the target solution quality for the instance if this time is at most τ, and is otherwise it is equal to τ. We define $R_\tau(\theta) = \mathrm{E}_{\pi_i \sim \mathcal{I}}[R(\theta, \pi_i, \tau)]$ as the expected τ-capped runtime taken over the distribution of instances $\mathcal{I}$. Given some runtime $\bar{\kappa}$ that we are never willing to exceed, the optimal configuration with respect to the κ-Fixed-Target performance metric (and thus the configuration that Structured Procrastination seeks) is defined as $\theta^*_{\mathrm{SP}} = \arg\min_\theta\{R_{\bar{\kappa}}(\theta)\}$. Hence, θ^*_{SP} is the configuration with the smallest expected $\bar{\kappa}$-capped runtime, i.e. the configuration with smallest expected runtime when runtimes greater than $\bar{\kappa}$ are taken to be $\bar{\kappa}$.

Kleinberg *et al.* [78, 79] provide the following definition of an near-optimal solution to the algorithm configuration problem:

Definition 3 ((ε, δ)-SP-optimality [78, 79])**.** A configuration θ is (ε, δ)-SP-optimal if there exists some threshold τ such that $\Pr_{\pi_i \sim \mathcal{I}}(R(\theta, \pi_i) > \tau) \leq \delta$ and $R_\tau(\theta) \leq (1+\varepsilon) R_{\bar{\kappa}}(\theta^*_{\mathrm{SP}})$. Otherwise, we say θ is (ε, δ)-SP-suboptimal.

By Definition 3, a configuration is (ε, δ)-SP-optimal if there exists a cutoff time τ within which it reaches the target solution quality for all but a δ fraction of problem instances in a $\bar{\kappa}$-capped time that is no more than a factor of $(1+\varepsilon)$ larger than the $\bar{\kappa}$-capped runtime of the optimal configuration θ^*_{SP}.

The following theorem provides a lower bound on the expected time required by any algorithm configurator to identify an (ε, δ)-SP-optimal configuration.

[11] We do not denote the runtime of a configuration as $T(\theta, \pi_i)$, as would more closely follow the literature related to the theory of evolutionary computation, since: (1) we use T throughout this thesis to denote the time required by a configurator to identify an optimal configuration and want to avoid confusion here; (2) we want to be consistent with the literature regarding worst-case analysis of configurators, which employs the notation used in this section.

Theorem 4 (Worst-case configuration time under the $\bar{\kappa}$-Fixed-Target performance metric, adapted from Theorem 6.1 in [78]). Suppose an algorithm configuration procedure is guaranteed to select an (ε, δ)-SP-optimal configuration with probability at least $1/2$. In the setting with a finite number $|\Theta|$ of configurations, the worst-case configuration time of any configurator is at least $\Omega\left(\frac{|\Theta|}{\delta\varepsilon^2} R_{\bar{\kappa}}(\theta^*_{\mathrm{SP}})\right)$, in expectation.

This lower bound can be understood as capturing the need to sample an (ε, δ)-SP-optimal configuration (accounting for the $|\Theta|$ term), and then needing to run this configuration enough times to sample a run whose runtime exceeds $\bar{\kappa}$ (accounting for the $1/\delta$ term) and needing to observe enough slow runs to be able to estimate their probability of occurring within a factor of $1-2\varepsilon$ (accounting for the $1/\varepsilon^2$ term) [78]. The factor of $R_{\bar{\kappa}}(\theta^*_{\mathrm{SP}})$ is because all configurations must be run long enough to allow at least one (the optimum, θ^*_{SP}) to reach the target solution quality.

Kleinberg *et al.* show that, in the worst case, all incumbent-driven algorithm configurators (defined formally in Section 2.5.2) using the Fixed-Target, $\bar{\kappa}$-Fixed-Target, or δ-Percentile-Fixed-Target performance metric have arbitrarily bad performance with high probability [78]. This includes all configurators where the cutoff time is static and those that attempt to reduce it (e.g. with adaptive capping).

Theorem 5 (Worst-case incumbent-driven configuration time, adapted from Theorem 3.2 in [78]). For any incumbent-driven configurator, any $\gamma > 0$, let R_1 and R_2 be the runtimes of two configurations. Then for $R_1 < R_2$, there is an algorithm configuration problem in which a γ fraction of all possible configurations have an average runtime no greater than R_1, but, with probability at least $1-\gamma$, the configuration procedure will require time at least R_2 to identify a configuration with runtime R_1.

Note that a cutoff time of at least R_2 must be used in order to distinguish between non-optimal configurations. Consider a configuration scenario where, when an instance is of size n, an $1/n$ fraction of configurations have runtime $R_1 = n$ and all others have runtime $R_2 = n^n$ (for simplicity, assume that the target algorithm is deterministic). Then, in terms of Theorem 5, $\gamma = 1/n$ and thus, with probability at least $1-1/n$, the configurator will require time at least n^n to return an optimal configuration with runtime n.

Kleinberg *et al.* show that, on the other hand, in the worst case the expected configuration time of the non-incumbent-driven Structured Procrastination (SP) is only larger than the lower bound given in Theorem 4 by a logarithmic factor [78]. The version of the bound that we state below is a simplification due to Weisz *et al.* [115].

Theorem 6 (Runtime of Structured Procrastination [115]). For any $\delta > 0$ and cutoff time $\bar{\kappa}$ that we are not willing to exceed, Structured Procrastination identifies an (ε, δ)-SP-optimal configuration in time

$$O\left(R_{\bar{\kappa}}(\theta^*_{\text{SP}})\frac{|\Theta|}{\varepsilon^2\delta}\log\left(\frac{|\Theta|\log\bar{\kappa}}{\zeta\varepsilon^2\delta}\right)\right),$$

with probability at least $1-\zeta$.

The logarithmic factor by which this upper bound is larger than the worst-case lower bound given in Theorem 4 is due to the number of problem instances that SP must run a configuration on before it can identify it as (ε,δ)-SP-optimal with the required probability of $1-\zeta$.

Weisz *et al.* prove a similar upper bound on the runtime of LEAPSANDBOUNDS (LB) [115]. However, their definition of optimality (and near-optimality) is subtly different from that used by Kleinberg *et al.* in their analysis of SP. Instead of defining the optimal configuration in terms of $\bar{\kappa}$-capped runtimes, Weisz *et al.* do so in terms of *uncapped* runtimes (this is equivalent to simply using the Fixed-Target performance metric). That is, Weisz *et al.* define the optimal configuration $\theta^*_{\text{LB}} = \arg\min_\theta\{R_\infty(\theta)\}$. Under this definition, the optimal configuration θ^*_{LB} has the smallest expected runtime over the distribution of instances without any runs being capped. This has the disadvantage that even the optimal configuration θ^*_{LB} could have infinite runtime, but if this is not the case then this definition arguably provides a more meaningful indicator of which configuration is optimal since its uncapped nature means that the optimal configuration will also be optimal for larger cutoff times (unlike with the $\bar{\kappa}$-Fixed-Target metric, where larger values of $\bar{\kappa}$ may yield different optimal configurations).

Definition 4 ((ε,δ)-LB-optimality [115])**.** A configuration θ is (ε,δ)-LB-optimal if there exists some threshold τ such that $\Pr_{i\sim\mathcal{I}}(R(\theta,i)>\tau)\leq\delta$ and $R_\tau(\theta)\leq(1+\varepsilon)R_\infty(\theta^*_{\text{LB}})$. Otherwise, we say θ is (ε,δ)-LB-suboptimal.

The following upper bound on the configuration time of LB replaces the $\bar{\kappa}$ term (the largest cutoff time that we are willing to use) in the upper bound for SP (Theorem 6) with the term $R_\infty(\theta^*_{\text{LB}})$. $R_\infty(\theta^*_{\text{LB}})$ will be be equal to $\bar{\kappa}$ if $\bar{\kappa}$ is large enough to allow all configurations to reach the target solution quality. Theorem 7 also removes the logarithmic dependency on $\varepsilon^{-2}\delta^{-1}$ present in Theorem 6.

Theorem 7 (Theorem 13 in [115])**.** LEAPSANDBOUNDS identifies an (ε,δ)-LB-optimal configuration in time

$$O\left(R_\infty(\theta^*_{\text{LB}})\frac{|\Theta|}{\varepsilon^2\delta}\log\left(\frac{|\Theta|\log R_\infty(\theta^*_{\text{LB}})}{\zeta}\right)\right),$$

with probability at least $1-\zeta$.

Kleinberg *et al.* derive the following upper bound on the runtime of Structured Procrastination with Confidence (SPC).

Theorem 8 (Theorem 3.4 in [79]). Fix ε and δ and let S be the set of (ε, δ)-SP-optimal configurations. For each $\pi_i \notin S$ suppose that π_i is $(\varepsilon_{\pi_i}, \delta_{\pi_i})$-SP-suboptimal, with $\varepsilon_{\pi_i} > \varepsilon$ and $\delta_{\pi_i} > \delta$. Then if the total configuration time is

$$\Omega\left(R_{\bar{\kappa}}(\theta^*_{\mathrm{SP}})\left(|S| \cdot \frac{\log(\zeta \log(1/\delta))}{\varepsilon^2 \delta} + \sum_{\pi_i \notin S} \frac{\log(\zeta \log(1/\delta_{\pi_i}))}{\varepsilon^2_{\pi_i} \delta_{\pi_i}}\right)\right),$$

then SPC will return an (ε, δ)-SP-optimal configuration when it is terminated, with high probability in ζ.

This upper bound on the configuration time of SPC is smaller than those for SP and LB (Theorems 6 and 7) since, for every configuration, the runtimes of SP and LB include a term of $\varepsilon^{-2}\delta^{-1}$ whereas, in the runtime of SPC, this term is replaced with the term $\varepsilon^{-2}_{\pi_i}\delta^{-1}_{\pi_i}$ (smaller than $\varepsilon^{-2}\delta^{-1}$ since $\varepsilon_{\pi_i} > \varepsilon$ and $\delta_{\pi_i} > \delta$) for each configuration i that is not (ε, δ)-SP-optimal.

When analysing CAPSANDRUNS, Weisz *et al.* again redefine the runtime of a configuration (and hence the notion of near-optimality) as they note that the $\bar{\kappa}$-capped runtime of a configuration, as used in the analysis of SP and SPC, can be significantly larger than the time required by the configuration to reach the target solution quality for a $1-\delta$ fraction of problem instances. Therefore upper bounds on the configuration time that involve $\bar{\kappa}$-capped runtimes (e.g. Theorems 6 and 8) are larger than those that could be derived by replacing the dependency on $\bar{\kappa}$-capped runtimes with a dependency on the time required to reach the target solution quality for a $1-\delta$ fraction of instances. Weisz *et al.* thus define the runtime of a configuration as the time that it requires to reach the target solution quality for a $1-\delta$ fraction of instances [116]. This time is never larger than the definition of runtime used in LB (i.e. $R_\infty(\theta)$) and can only be larger than the definition of runtime used in SP and SPC (i.e. $R_{\bar{\kappa}}(\theta)$) if $\bar{\kappa}$ is not large enough to allow the configuration to reach the target solution quality for a $1-\delta$ fraction of problem instances.

We now formally define the notion of runtime used in the analysis of CR. Let us denote δ-quantile of the runtimes of configuration θ as $t_\delta(\theta) = \inf_{t \in \mathbb{R}}\{t \mid \Pr_{\pi_i \sim \mathcal{I}}(R(\theta, \pi_i) > t) \leq \delta\}$. That is, $t_\delta(\theta)$ is the smallest time at which θ has failed to reach the target solution quality for at most a δ fraction of instances. Define the runtime of configuration θ below the δ-quantile as $R^\delta(\theta) = R_{t_\delta(\theta)}(\theta)$ and define the configuration with the smallest expected runtime over a $1-\delta$ fraction of instances as $\theta^*_{CR}(\delta) = \arg\min_\theta R^\delta(\theta)$. That is, $\theta^*_{CR}(\delta)$ is the configuration that reaches the target solution quality for a $1-\delta$ fraction of instances in the smallest expected time. Thus $\theta^*_{CR}(\delta)$ is the optimal configuration with respect to the δ-Percentile-Fixed-Target performance metric. Weisz *et al.* use the following definition of (ε,δ)-optimality:

Definition 5 ((ε,δ)-CR-optimality as in [116])**.** A configuration θ is (ε,δ)-CR-optimal if $R^\delta(\theta) \leq (1+\varepsilon)R(\theta^*_{CR}(\delta/2))$. Otherwise, we say θ is (ε,δ)-CR-suboptimal.

The presence of $\theta^*_{CR}(\delta/2)$ instead of $\theta^*_{CR}(\delta)$ in the above definition is necessary to allow the required property to be verified with high probability [116].

The upper bound on the runtime of CAPSANDRUNS is considerably more complex than the other runtime bounds seen thus far, but as a result it expresses the impact of the complexity of the configuration scenario on the runtime of the configurator. In the following, let $\hat{\theta}$ denote the configuration with the smallest runtime out of all configurations not eliminated in the first phase of CAPSANDRUNS. For any configuration θ, denote the amount by which it is suboptimal by $\Delta_\theta = 1 - \frac{R^{\delta/2}(\theta^*_{CR}(\delta/2))}{R^\delta(\theta)}$ and let σ^2_τ denote the variance of its runtimes using cutoff time τ, its maximum relative variance as $\hat{\sigma}^2(\theta) = \sup_{\tau\in[t_\delta(\theta),t_{\delta/2}(\theta)]} \frac{\sigma^2_\tau(\theta)}{R^2_\tau(\theta)}$, and maximum relative range as $\rho(\theta) = \sup_{\tau\in[t_\delta(\theta),t_{\delta/2}(\theta)]} \frac{\tau}{R_\tau(\theta)}$.

Theorem 9 (Theorem 1 in [116])**.** For a failure parameter $\zeta \in (0,1/6)$, with probability at least $1-6\zeta$, CAPSANDRUNS finds an (ε,δ)-CR-optimal configuration in total configuration time

$$O\left(R^{\frac{\delta}{2}}\left(\theta^*_{CR}\left(\frac{\delta}{2}\right)\right)\left[\frac{|\Theta|}{\delta}\log\frac{|\Theta|}{\zeta} + \sum_{\theta\in\Theta}\left(\max\left\{\frac{\max\{\hat{\sigma}^2(\theta),\hat{\sigma}^2(\hat{\theta})\}}{\max\{\varepsilon^2,\Delta^2_\theta\}}, \frac{\max\{\rho(\theta),\rho(\hat{\theta})\}}{\max\{\varepsilon,\Delta_\theta\}}\right\}\cdot\right.\right.$$
$$\left.\left.\left(\log^2\frac{|\Theta|}{\zeta} + \log\frac{|\Theta|}{\zeta}\log\frac{\max\{\rho(\theta),\rho(\hat{\theta})\}}{\max\{\varepsilon,\Delta^2_\theta\}}\right)\right)\right]\right).$$

This bound shows that it is possible for CAPSANDRUNS to identify a near-optimal configuration in less time than the worst-case bound stated in Theorem 4 if the configuration problem has favourable characteristics (i.e. it is easy to differentiate the performance of different configurations) [116]. This does not contradict the lower bound claim of Theorem 4 since that theorem seeks a configuration that is (ε,δ)-SP-optimal whereas Theorem 9 seeks a configuration that is (ε,δ)-CR-optimal.

We now make explicit the features of the upper bound in Theorem 9 that make it possible to fall below the lower bound given in Theorem 4. Firstly, the configuration time scales with the reciprocal of the sub-optimality gaps Δ_θ instead of with $1/\varepsilon^2$, and thus the runtime can be reduced if many configurations are significantly suboptimal. Secondly, it removes the large $(\delta\varepsilon^2)^{-1}$ term and replaces it with separate terms for ε and δ, thus reducing the configuration time further if one of these terms is chosen to be small despite the other being large. Finally, the term $1/\varepsilon$ is multiplied by the range $\rho(\theta)$, whereas $1/\varepsilon^2$ is only multiplied by the much smaller relative variance $\hat{\sigma}^2(\theta)$.

Approximation worst-case analysis has also been provided for the Hyperband configurator (outlined in Section 2.6.2) that optimises the hyperparameters of machine learning algorithms. Li *et al.* present bounds on the difference between the optimal hyperparameter configuration and the one returned by Hyperband [88] after using a total budget of τ (in terms of the resource being allocated dynamically, most commonly the number of runs used to evaluate a configuration). Adopting the bandit-theory-related language used in [88], we identify configuration θ with 'arm' θ, and assume that if we 'pull' the θ-th arm k times then we observe a loss $\ell_{\theta,k}$. Thinking of the loss of an arm as the error of the model learned by the target algorithm with the corresponding hyperparameter configuration, we can see that choosing the optimal hyperparameter configuration is equivalent to identifying the arm with the smallest loss. Assume that:

- For each $\theta \in \mathbb{N}$, the limit $\lim_{k\to\infty} \ell_{\theta,k}$ exists and is equal to ν_θ;
- Each ν_θ is a bounded i.i.d. random variable with cumulative distribution F.

Let ν_* be the arm with the smallest loss (i.e. the optimal hyperparameter configuration) and let $\gamma(j)$ be the pointwise smallest monotonically decreasing function that bounds from above the deviation of the loss of an arm from its limit value as the number of pulls j increases. Also assume that there exists some constant $\alpha > 0$ such that $\gamma(j) \simeq$ [12] $(1/j)^{1/\alpha}$ and that there exists a constant $\beta > 0$ such that $F(x) \simeq (x - \nu_*)^\beta$ if $x \geq \nu_*$ and 0 otherwise. Then the sub-optimality of the configuration returned by Hyperband after a total budget of τ (i.e. after using τ of the resources that are being distributed, e.g. number of instances used to train the model) can bounded from above as follows:

Theorem 10 (Adapted from Theorem 5 in [88])**.** For any $\tau \in \mathbb{N}$, let $\hat{\theta}_\tau$ be the empirically best-performing arm output in the final round of Hyperband after exhausting a total budget of τ. Fix $\delta \in (0,1)$. Then the sub-optimality of configuration $\hat{\theta}_\tau$ is at most

$$\nu_{\hat{\theta}_\tau} - \nu_* \leq c \cdot \left(\frac{(\overline{\log}\ \tau)^3 \overline{\log}((\log\ \tau)/\delta)}{\tau} \right)^{1/\max\{\alpha,\beta\}}$$

for some constant $c = \exp(O(\max\{\alpha,\beta\}))$, and where $\overline{\log}(x) := \log(x)\log\log(x)$.

The bound on sub-optimality given in Theorem 10 is

[12] $f \simeq g$ if and only if there exists constants c, c' such that $cg(x) \leq f(x) \leq c'g(x)$.

$$O\left(\left(\frac{(\log(\tau))^6}{\tau}\right)^{1/\max\{\alpha,\beta\}}\right) = o\left(\tau^{-1/2\max\{\alpha,\beta\}}\right),$$

implying that the gap between the optimal hyperparameter configuration and the one returned by Hyperband decreases polynomially in relation to the budget τ.

The performance analyses outlined in this section provide upper bounds on the performance of configurators in the worst case. They therefore do not give insights into the performance of algorithm configurators in realistic scenarios, and hence cannot be used to explain the success of configurators in practice. In this thesis, we develop the theoretical foundations of algorithm configuration to enable performance statements for algorithm configurators in specific scenarios.

2.10 Conclusions

In this chapter, we have introduced the algorithm configuration problem and explained its importance. We have shown that automated methods designed to identify good algorithm configurations are composed of two components, a method to generate new configurations and a method for their evaluation. We gave an overview of the way in which these components are designed in practice, including methods used to dynamically update their settings with the aim of reducing the time required to identify good configurations. We then gave an overview of the most popular algorithm configurators in the literature. Finally, we detailed the state-of-the-art of the theory of algorithm configuration. In the worst case, using small cutoff times initially and then increasing them when necessary provides better approximations of optimal configurations in a shorter time compared to using pre-defined upper bounds on the cutoff time. However, this technique relies on the availability of appropriately large configuration budgets which may not be assumed to be always available in real-world configuration scenarios. In particular, no results are available regarding the approximation quality that any algorithm configurator can guarantee efficiently (i.e. in a time that is a polynomial function of the size of the configuration search space). In this thesis, we will set up the theoretical foundations for such analyses for the cases where a minimal solution quality is sought (i.e. the Fixed-Target performance metric) and where a fixed configuration budget is available (i.e. the Best-Fitness performance metric).

Chapter 3

Mathematical Tools, Configurators, Target Algorithms, Problems Classes

3.1 Introduction

In this chapter, we provide an overview of the 'ingredients' that we will use to build a theoretical foundation to assess the performance of algorithm configurators. We begin with a summary in Section 3.2 of the mathematical techniques from the literature that we employ to construct this theoretical foundation. In Section 3.3, we present a bare-bones stochastic local search algorithm configurator, which we call ParamRLS to indicate that it is a simplified version of the popular ParamILS configurator. Its simplicity allows us to identify algorithm configuration problem characteristics where minimal algorithmic sophistication suffices and others where more complex configurators are necessary. By building upon the mathematical tools developed for the analysis of ParamRLS, we will extend our analysis to the more sophisticated ParamILS algorithm configurator. Throughout this thesis, the performance of the algorithm configurators will be assessed by estimating the expected time they require to identify optimal parameter values for some target algorithms for benchmark problem classes from the literature designed to have characteristics commonly encountered in practice. In Section 3.4, we present the two target algorithms – RLS$_k$ and the (1+1) EA– whilst in Section 3.5 we describe the three standard benchmark problem classes – RIDGE, ONEMAX, and LEADINGONES – for which their parameters are tuned.

3.2 Mathematical Tools

Over the last two decades, the framework has been laid for the rigorous theoretical analysis of randomised search heuristics [69]. In this section, we review the tools and techniques developed in this field that we use in this thesis. We begin by defining the terminology that is common in the field and used throughout our work. We then review useful results from probability theory. We finally give an overview of the techniques that are used in the analysis of randomised search heuristics.

3.2.1 Runtime Analysis

The performance of an algorithm is often measured in terms of its *runtime*. In the context of optimisation algorithms, this may be called the *optimisation time*. Given an algorithm $\mathcal{A}$, an instance i of an optimisation problem, and a set of optimal search points X^*, let the best-found search point after t iterations be denoted as X_t. Then the optimisation time of this algorithm on this problem instance is defined as $T = \min\{t \in \mathbb{N} \mid X_t \in X^*\}$.

The runtime of a randomised algorithm is a random variable. In runtime analysis, the quantity of interest is often the expected value (defined in Section 3.2.2) of its runtime, called its *expected runtime*. It is often also useful to analyse its runtime with a specified failure probability (i.e. probability of not reaching the optimum within that time).

It is common to express the order of growth of the runtime as a function of the problem size using *asymptotic notation* (also called Landau notation) [24]. This framework originated in mathematics where it is used to classify the growth of a function. It can thus be used to analyse the relative growth of functions that represent the runtimes of different algorithms in relation to the problem size. If, as the problem size increases, the runtime of an algorithm can be shown to grow asymptotically faster than the runtime of another, then we can conclude that it is faster for all sufficiently large inputs.

Definition 6 (Asymptotic notation)**.** Given two functions $f, g : \mathbb{N}_0 \to \mathbb{R}$ we use the following terms to express their relative growth.

- We say that f grows at most as fast as g if and only if there exist constants $n_0 \in \mathbb{N}_0$ and $c \in \mathbb{R}^+$ such that, for all $n \geq n_0$, it holds that $f(n) \leq c \cdot g(n)$. We write $f(n) = O(g(n))$.
- We say that f grows at least as fast as g if and only if $g(n) = O(f(n))$. We write $f(n) = \Omega(g(n))$.
- We say that f and g grow at the same rate if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$. We write $f(n) = \Theta(g(n))$.

- We say that f grows slower than g if and only if $\lim_{n\to\infty} f(n)/g(n) = 0$. We write $f(n) = o(g(n))$.
- We say that f grows faster than g if and only if $g(n) = o(f(n))$. We write $f(n) = \omega(g(n))$.

In the theory of computation, an algorithm is typically considered *efficient* if its runtime grows as a polynomial function of the size of its input and *inefficient* if this relationship is instead superpolynomial.

Definition 7 (Classes of growth)**.** A function $f : \mathbb{N}_0 \to \mathbb{R}$ is:

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

In this thesis, we also make frequent use of these classes of growth when calculating the probabilities of events that are very likely (or very unlikely) to happen. In the strongest case, we say that an event occurs ‘with overwhelming probability’, to indicate that the probability that this event does not occur is exponentially decreasing. We now use the above terminology to define this concept, along with its weaker counterpart ‘with high probability’.

Definition 8 (High/Overwhelming probability)**.** We say that an event A occurs:

- *with high probability* (w.h.p.) if and only if $1 - \Pr(A)$ is polynomially small (i.e. $\Pr(A) = 1 - O(n^{-k})$ for some constant $k \in \mathbb{R}^+$).
- *with overwhelming probability* (w.o.p.) if and only if $1 - \Pr(A)$ is exponentially small (i.e. $\Pr(A) = 1 - 2^{-\Omega(n^k)}$ for some constant $k \in \mathbb{R}^+$).

3.2.2 Probability Theory

We now give an overview of the elements of probability theory that we use throughout this thesis. Unless otherwise stated, all definitions and results in this section are given in the review by Doerr [32].

Basic Definitions

A *random variable* is a function that captures the range of possible outcomes of some random process. The probability with which it yields each outcome is defined by a *probability distribution*. We call each value of X an *outcome*, and define an *event* to be a set containing some number of outcomes. We denote the probability that an event A occurs by $\Pr(A)$ and denote the probability that the random variable X takes the value x by $\Pr(X = x)$. Given a set of events $B = \{b_1, \ldots, b_m\}$, we denote the probability that at least one event in B occurs by $\Pr(b_1 \cup \ldots \cup b_m)$. In this thesis, we assume that we deal with *discrete random variables* (i.e. random variables that only yield a countable number of values). Some theorems discussed later in this chapter require that a random process has been "adapted to a filtration $\mathcal{F}_t$". We do not define this notion formally here since doing so would introduce several otherwise unnecessary definitions. For our purposes, it suffices to interpret $\mathcal{F}_t$ as the history of the process up to time t.

The *expected value* of a random variable X (interchangeably referred to as its *expectation* in this thesis) is defined as $\mathrm{E}[X] = \sum_{i \in Y} i \cdot \Pr(X = i)$, where Y is the set of all outcomes of X.

Bernoulli Random Variables A Bernoulli random variable only takes the values 0 and 1. The expected value of a Bernoulli random variable is hence $\mathrm{E}[X] = \Pr(X = 1)$. Bernoulli random variables are commonly used as indicator variables, where a variable takes the value 1 if an event of interest occurs and 0 otherwise.

Binomial Random Variables A binomial random variable counts the number of successes in a given number of trials, for example the number of heads when a coin is flipped n times. Let us define X as the random variable counting the number of successes in n trials where in every trial a success occurs with probability p. We write $X \sim \mathrm{Bin}(n, p)$ and have $\mathrm{E}[X] = np$.

Geometric Random Variables A geometric random variable counts the number of trials before a success is observed for the first time[13]. Given an infinite sequence of Bernoulli random variables $X_1, X_2, \ldots$ with common success probability p, the corresponding geometric random variable $Y \sim \mathrm{Geom}(p)$ is defined as $Y = \min\{k \in \mathbb{N} \mid X_k = 1\}$. Then $\Pr(Y = k) = (1-p)^{k-1} \cdot p$ and $\mathrm{E}[Y] = 1/p$. This gives rise to the *waiting time argument*: in expectation we wait time $1/p$ for an event with success probability p to occur. For example, in expectation a fair die must be rolled six times before a one is rolled.

[13]Geometric random variables are sometimes alternatively defined as the number of failures that occur before the first success, thus corresponding to the random variable $Y - 1$ in our definition.

Union Bound

It is common to bound the probability that any event in a set occurs. Since it lacks any assumptions, the *union bound* provides a simple and readily applicable means of doing so:

Theorem 11 (Union bound)**.** Given a set of events $\{A_1, \ldots, A_m\}$ in some probability space,

$$\Pr(A_1 \cup \ldots \cup A_m) \leq \sum_{i=1}^{m} \Pr(A_i).$$

Linearity of Expectation

Given a number of random variables, the *linearity of expectation* allows us to derive the expected value of a linear combination expected values. Like the union bound, it makes no assumptions about the random variables at hand.

Theorem 12 (Linearity of expectation)**.** Given a set of random variables $\{X_1, \ldots, X_m\}$ and a set of real numbers $\{a_1, \ldots, a_m\}$, let $Y = \sum_{i=1}^{m} a_i X_i$. Then $\mathrm{E}[Y] = \mathrm{E}[\sum_{i=1}^{m} a_i X_i] = \sum_{i=1}^{m} a_i \mathrm{E}[X_i]$.

Stochastic Domination

The concept of *stochastic domination* gives a rigorous definition to scenarios where we intuitively feel that the value of one random variable is ‘smaller’ than another. For example, consider a scenario where a player must choose which game to play given three options (this example is adapted from [32]). In game A, they win one coin with probability 1/2 and three coins also with probability 1/2. In game B, they win one coin with probability 1/4, two coins with probability 1/4, and three coins with probability 1/2. In game C, they win 100 coins with probability 1/20 and otherwise win nothing. Game B is clearly preferable to game A, since the player is more likely to win more coins by playing game B rather than game A. However, the question of whether to choose to play game B or game C is not so easy to answer. Whilst the expected number of coins won in game C is greater than the expected number won in B, the player may wish to play a game that guarantees a return of *some* coins (game B) rather than a game where with probability 0.95 they win nothing (game C).

Whilst the latter choice is a matter of preference, the intuition that game B is preferable to game A can be made formal using the notion of stochastic domination:

Definition 9 (Stochastic domination (adapted from Definition 1.8.1 in [32]))**.** Let X and Y be two random variables. We say that Y *stochastically dominates* X if for all $\alpha \in \mathbb{R}$ we have $\Pr(X \leq \alpha) \geq \Pr(Y \leq \alpha)$.

The number of coins won in game B thus stochastically dominates the number won in game A, explaining why we intuitively feel that one game is preferable to the other.

3.2.3 Useful Inequalities

In the analysis of randomised algorithms, it can be cumbersome to manipulate exact expressions of probabilities. In this section, we review inequalities that can be used to bound these probabilities in order to simplify the analysis. Unless otherwise stated, the statements of these theorems are adapted from [32].

It is useful to be able to switch between polynomial terms and exponential terms. The following inequalities, whilst simple, are often sufficient for our purposes.

Theorem 13. (i) For all $x \in \mathbb{R}$, $1+x \leq e^x$.

(ii) For all $x < 1$, $e^x \leq 1 + x/(1-x)$.

The next two inequalities provide bounds on quantities that frequently arise in the analysis of randomised algorithms, most commonly when dealing with the probability that an event does not occur in a given number of trials:

Theorem 14. For all $r \geq 1$ and $0 \leq s \leq r$,

$$\left(1 - \frac{s}{r}\right)^r \leq e^{-s} \leq \left(1 - \frac{s}{r}\right)^{r-s}$$

Theorem 15. For all $x > -1$,

$$\exp\left(\frac{x}{1+x}\right) \leq 1 + x.$$

For all $x, y > 0$,

$$\exp\left(\frac{xy}{x+y}\right) \leq \left(1 + \frac{x}{y}\right)^y \leq \exp(x).$$

We also make use of the following bound on the probability that an event occurs at least once in a given number of trials.

Theorem 16 (Adapted from Lemma 10 in [6]). Let p_α be the probability of having at least one success in α trials of a Bernoulli random variable with success probability p. Then

$$p_\alpha \geq \frac{\alpha p}{1 + \alpha p}.$$

The binomial coefficients that appear regularly in our analyses are difficult to handle directly due to the presence of factorials in their definitions. Bounds on their values are therefore greatly useful to us. We make use of the following standard result:

Theorem 17. For all $n \in \mathbb{N}$ and $k \in \{1, \ldots, n\}$, $\left(\frac{n}{k}\right)^k \leq \binom{n}{k} \leq \left(\frac{ne}{k}\right)^k$.

3.2.4 Tail Inequalities

Tail inequalities allow us to bound the probability that a random variable deviates from its expected value by a given amount. This allows us to "turn expected run times into upper bounds which hold with overwhelming probability" [114]. Furthermore, they are often used when an algorithm is unlikely to take its worst-case runtime, i.e. an algorithm may have exponential runtime in the worst case but may also identify the optimum in polynomial time with overwhelming probability. Hence, if it is run many times it is very likely to be efficient in the vast majority of cases.

Markov's Inequality

This is the simplest tail bound commonly encountered in the literature. Its strength is that it imposes only a single constraint on the random variable to which it is applied, only requiring that it be non-negative.

Theorem 18 (Markov's inequality)**.** For a non-negative random variable X with $\mathrm{E}[X] > 0$ and for all $\alpha > 0$,

$$\Pr(X \geq \alpha \,\mathrm{E}[X]) \leq \frac{1}{\alpha}.$$

Whilst Markov's inequality is useful as it is applicable in many scenarios, this comes at a cost: the bound that it yields may not be as strong as desired. We now outline a family of tail bounds that yield significantly stronger guarantees than Markov's inequality, but in turn impose further conditions that that limit the scenarios in which they may be applied.

Chernoff Bounds

Chernoff bounds are a family of powerful tail inequalities that provide much stronger probability bounds on the value that a random variable may assume than those obtained using Markov's inequality. In the case of Theorems 19 and 20, the stronger bounds are achieved by using the information that the random variable X is a sum of independent random variables each of which only takes values in $\{0,1\}$.

We first give the bounds on the probability that a random variable deviates from its expectation by a multiplicative factor.

Theorem 19 (Theorems 1.10.1 and 1.10.5 in [32])**.** Let $X_1,\ldots,X_m$ be independent random variables taking values in $\{0,1\}$. Let $X = \sum_{i=1}^m X_i$. Then, for $0 \leq \delta \leq 1$,

$$\Pr(X \leq (1-\delta)\,\mathrm{E}[X]) \leq \exp\left(-\frac{\delta^2\,\mathrm{E}[X]}{2}\right)$$

and, for $\delta \geq 0$

$$\Pr(X \geq (1+\delta)\,\mathrm{E}[X]) \leq \exp\left(-\frac{\min\{\delta, \delta^2\}\,\mathrm{E}[X]}{3}\right).$$

We occasionally use the following additive analogue of these bounds.

Theorem 20 (Theorem 1.10.7 in [32])**.** Let $X_1, \ldots, X_m$ be independent random variables taking values in $\{0,1\}$. Let $X = \sum_{i=1}^{m} X_i$. Then, for all $\delta \geq 0$,

$$\Pr(X \geq \mathrm{E}[X] + \delta) \leq \exp\left(-\frac{2\delta^2}{m}\right)$$

$$\Pr(X \leq \mathrm{E}[X] - \delta) \leq \exp\left(-\frac{2\delta^2}{m}\right).$$

We also make use of Chernoff bounds for sums of geometric random variables. As with the earlier bounds, it is required that each random variable in the sum be independent.

Theorem 21 (Theorem 1.10.32 in [32])**.** Let $X_1, \ldots, X_m$ be independent geometric random variables with a common success probability $p > 0$. Let $X := \sum_{i=1}^{m} X_i$ and $\mu := \mathrm{E}[X] = m/p$. Then:

(i) For all $\delta \geq 0$,

$$\Pr(X \geq (1+\delta)\mu) \leq \exp\left(-\frac{\delta^2(m-1)}{2(1+\delta)}\right).$$

(ii) For all $0 \leq \delta \leq 1$,

$$\Pr(X \leq (1-\delta)\mu) \leq \exp\left(-\frac{\delta^2 m}{2 - \frac{4}{3}\delta}\right).$$

Method of Bounded Martingale Differences

In some cases, it may be necessary to derive tail bounds for random variables that are more complex functions of other random variables (i.e. not simply a sum). In these cases, Chernoff bounds are not applicable. However, if each variable that contributes to the function has only a limited influence on its value then it is still possible to bound the extent to which this function deviates from its expectation. One means of dealing with such cases that requires only that these random variables be bounded is the method of bounded martingale differences.

Theorem 22 (Method of bounded martingale differences (Theorem 3.67 in [108])). Let $X_1, \ldots, X_m$ be an arbitrary set of random variables and let f be a function satisfying the property that for each $i \in \{1, \ldots, m\}$ there is a $c_i \geq 0$ such that

$$|\mathrm{E}[f \mid X_1, \ldots, X_i] - E[f \mid X_1, \ldots, X_{i-1}]| \leq c_i.$$

Then

$$\Pr(f \geq \mathrm{E}[f] + t) \leq \exp\left(-\frac{t^2}{2\sum_{i=1}^{m} c_i^2}\right)$$

and

$$\Pr(f \leq \mathrm{E}[f] - t) \leq \exp\left(-\frac{t^2}{2\sum_{i=1}^{m} c_i^2}\right).$$

Note that the X_i terms are *not* required to be independent.

3.2.5 Random Walks and Gambler's Ruin

We now review two related topics that are useful when analysing the performance of algorithms for functions with plateaus of fitness values (i.e. regions of the search space in which all points have the same fitness). We first define key terms related to random walks, and we then show how the theory related to the 'gambler's ruin problem' can be used to analyse such scenarios.

Random Walks

Given a set of states and a definition of a neighbourhood, a *random walk* is a process that at each step moves from the current state to a neighbouring state chosen according some probability distribution. A random walk that moves to any neighbouring state with the same probability is called a *symmetric random walk*. A *lazy random walk* is a random walk that additionally may remain in the same state with some positive probability. An *absorbing state* is a state that, once reached, cannot be left (i.e. the probability of leaving it is 0).

Gambler's Ruin

Consider a player and an adversary playing a game where in each round they both bet a coin and then one of them wins both coins with some probability. The probability with which either player wins a round does not change during the game. Determining the probability that the player eventually loses all their coins to the adversary along with the expected time for either participant to lose all their coins is called the *gambler's ruin* problem [47]. The gambler's ruin scenario is therefore a random walk across the state space of the wealth of the player where the neighbours of each non-absorbing state correspond to increasing or decreasing the wealth of the player by a single coin.

Assume that the player and the adversary start with z and $a - z$ coins, respectively. In each round, the player wins both coins with probability p, with $0 < p \leq 1$, and the adversary with probability $q = 1 - p$. Then the probability q_z of the player's ruin (that is, the probability that the player eventually loses all their coins to the adversary) is

$$q_z = \begin{cases} \frac{(q/p)^a - (q/p)^z}{(q/p)^a - 1}, & \text{if } p \neq q \\ 1 - \frac{z}{a}, & \text{if } p = q = 1/2 \end{cases}$$

and the expected duration of the game D_z (i.e. the expected number of rounds before either of the participants has lost all their coins to the other) is

$$D_z = \begin{cases} \frac{z}{q-p} - \frac{a}{q-p} \frac{1-(q/p)^z}{1-(q/p)^a}, & \text{if } p \neq q \\ z(a-z), & \text{if } p = q = 1/2 \end{cases}$$

In this thesis, we analyse the expected hitting time of a random walk with a *reflecting barrier* at one end of the search space (i.e. where there is only a single absorbing state). In the context of gambler's ruin, only the player is able to win the game: if the adversary is about to win then they give the player a coin and continue playing. Figure 3.1 shows an example of such a scenario.

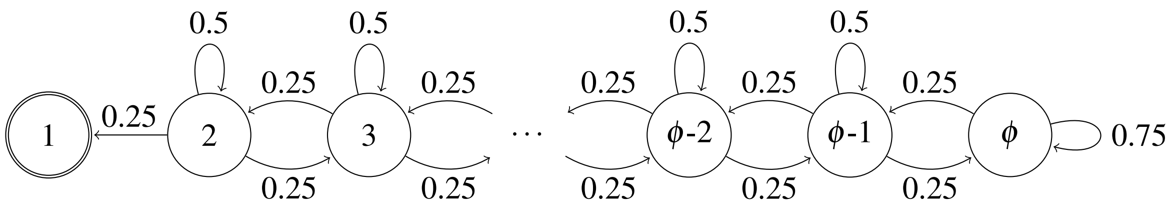


Fig. 3.1 A lazy random walk. State 1 is the only absorbing state.

We analyse it using a method similar to that used by Bossek and Sudholt [17, Lemma 27]. Note that there are two differences with the scenario shown in Figure 3.1 and conventional gambler's ruin. Firstly, with each step there is a probability of 0.5 of remaining in a state. This is not a large problem for the analysis: it will simply double the expected time to reach either end of the state space. The more significant difference between the two scenarios is that there is only one absorbing state (state 1), as opposed to the standard gambler's ruin scenario where there are two (corresponding to either the player or the adversary winning the game).

To deal with this, we instead consider the lazy random walk shown in Figure 3.2. Notice that the new model has two absorbing states: 1 and 2ϕ. Note also that the time required to reach the absorbing state in the first scenario is the same as the time required to reach either absorbing state in the second, as the probability of remaining in the states ϕ and $\phi+1$ in Figure 3.2 is the same as remaining in state ϕ in Figure 3.1 and both lazy random walks with one end at these states are identical to that presented in Figure 3.1. We can thus analyse the expected time to reach state 1 in the first scenario by analysing the expected duration of the game in the second scenario. Using these arguments, the theory of gambler's ruin yields that, in the worst case (i.e. when initialised at state ϕ), the expected hitting time of state 1 in the first scenario is at most $2\phi(\phi-1)$.

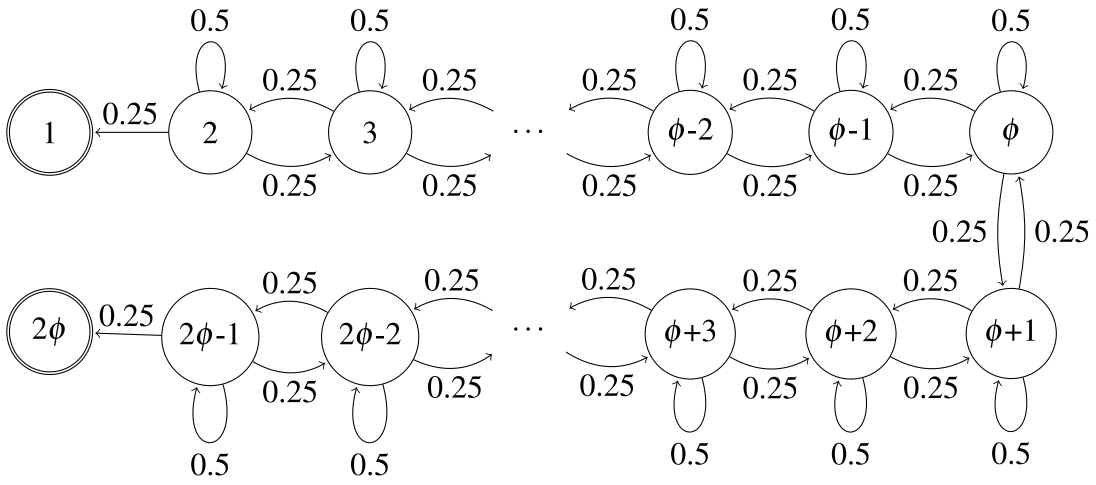


Fig. 3.2 The modified lazy random walk from Figure 3.1. States $1,\dots,\phi$ have been duplicated as states $\phi+1,\dots,2\phi$. States 1 and 2ϕ are both absorbing states.

3.2.6 Drift Analysis

Introduced in a general form by Hajek [50], drift analysis was first applied to the analysis of evolutionary algorithms by He and Yao [56]. Since its introduction, many variants have been created to enable the analysis of evolutionary algorithms with a range of behaviours. It has become one of the most popular and powerful techniques in the field [86].

All members of this family of techniques bound the runtime of an algorithm by bounding the expected amount by which in each iteration the algorithm decreases (or increases) its distance from some target (e.g. the optimum), called the *drift*.

There are many variants of this approach, for example those that allow bounds on the optimisation time when the drift is additive [56], multiplicative [39], and variable [74].

However, in this thesis these latter, more sophisticated methods are unnecessary, since we are generally uninterested in calculating optimisation times, being concerned instead with bounding the fitness of the individual in an algorithm at any point in time (as discussed in Section 3.2.8).

We first formally state the approach of drift analysis and then present the theorems used in this thesis to bound the runtime of an algorithm using its drift.

Formal Outline of Drift Analysis

The outline given here follows that of Lengler [86]. Let $(X_t)_{t\geq 0}$ be a sequence of non-negative random variables with finite state space $\mathcal{S} \subseteq \mathbb{R}_0^+$ where each X_t describes the distance to the optimum at time t. Denote the minimum positive state as $\mathrm{s_{min}} = \min\{\mathcal{S} \setminus 0\}$. The drift of the process is defined as $\mathrm{E}[X_t - X_{t+1} \mid X_t = s]$. The drift of the process at time t given $X_t = s$ is usually denoted as $\Delta_t(s)$.

We are typically interested in calculating the first point in time where $X_t = 0$. That is, we seek the time value $\tau = \min\{t \mid X_t = 0\}$. This value of t is commonly called the *hitting time* of the process.

Additive Drift Analysis

Additive drift analysis was the form in which drift analysis was first applied to the analysis of evolutionary algorithms [57]. It requires a bound on the drift that does not change over the run. The following statement is based on the formulation by Lengler [86].

Theorem 23 (Additive Drift Theorem, Theorem 1 in [86])**.** Let X_t and τ be defined as in Section 3.2.6. If there exists $\delta > 0$ such that, for all states $s \in \{\mathcal{S} \setminus 0\}$ and for all $t \geq 0$, $\mathrm{E}[X_t - X_{t+1} \mid X_t] \geq \delta$, then $\mathrm{E}[\tau] \leq \mathrm{E}[X_0]/\delta$. Similarly, if there exists $\delta > 0$ such that, for all states $s \in \{\mathcal{S} \setminus 0\}$ and for all $t \geq 0$ $\mathrm{E}[X_t - X_{t+1} \mid X_t] \leq \delta$, then $\mathrm{E}[\tau] \geq \mathrm{E}[X_0]/\delta$.

Negative Drift / Simplified Drift Theorem

In addition to deriving runtimes of randomised algorithms, drift analysis can also be used to prove that an algorithm is likely to be inefficient for a problem. The standard negative drift theorem was introduced by Oliveto and Witt and provides exponential lower bounds on the runtime of an algorithm with drift that points away from the optimum [99, 100].

In this thesis, we use a variant of this technique, introduced by Rowe and Sudholt, that allows for time steps where the value of the random process frequently does not change by removing the condition on a continual drift away from the optimum that is present in the original formulation of this result [106].

Theorem 24 (Negative Drift with Self-loops (Theorem 4 in [106])). Suppose that for all $t \geq 0$ the following two conditions hold:

1. There exist integers a, b with $0 < a < b \leq m$ and $\varepsilon > 0$ such that $\mathrm{E}[X_{t+1} - X_t] < -\varepsilon(1 - \Pr(X_{t+1} = k \mid X_t = k))$ for all $a \leq k \leq b$.

2. There exist constants $r, \delta > 0$ (i.e. independent of m) such that for all $k \geq 1$ and all $d \geq 1$ both $\Pr(X_{t+1} = k - d \mid X_t = k)$ and $\Pr(X_{t+1} = k + d \mid X_t = k)$ are at most

$$\frac{r \cdot (1 - \Pr(X_{t+1} = k \mid X_t = k))}{(1+\delta)^d}.$$

Let τ be the hitting time when starting from $X_0 \geq b$. Let $m = b - a$. Then there is a constant $c > 0$ such that

$$\Pr(\tau \leq 2^{cm/r}) = 2^{-\Omega(m/r)}.$$

Multiplicative Drift Tail Bounds

If the drift of a process is a multiplicative factor of its current value, then it can be analysed using multiplicative drift analysis. In this thesis, we use the version with tail bounds derived by Doerr and Goldberg [37]. The formulation we give here is due to Lengler [86].

Theorem 25 (Adapted from Theorem 18 in [86]). Let X_t, $s_{\min}$, and τ be defined as in Section 3.2.6. If there exists $\delta > 0$ such that, for all states $s \in \{\mathcal{S} \setminus 0\}$ and for all $t \geq 0$,

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

then for all $r \geq 0$,

$$\Pr\left(\tau > \left\lceil \frac{r + \ln(s_0/s_{\min})}{\delta} \right\rceil\right) \leq \exp(-r).$$

3.2.7 Black-Box Complexity

Black-box complexity analyses describe the minimum number of fitness function evaluations required by the best black-box search algorithm in a given class to optimise the hardest instance of a given class of functions. They thus capture the intrinsic difficulty of optimising the function class at hand.

Lehre and Witt performed such an analysis for the class of *unary unbiased* algorithms. This class of algorithms generates new search points using a unary operator (i.e. an operator that takes one search point as input and produce another search point as output) that does not focus on any particular region of the search space (e.g. for solutions represented as bit strings, all bit values and all bit positions are treated in the same way) [85]. Lehre and Sudholt extended this model to include algorithms that generate search points in parallel, such as the $(\mu+\lambda)$ EA (Algorithm 2) [84]. They proved a lower bound for any member of this class of algorithms to optimise any function with up to an exponential number of optima.

Theorem 26 (Adapted from Theorem 21 in [84] with $\delta := 1/2$)**.** Every unary unbiased black-box algorithm needs more than $(n\ln n)/2$ fitness evaluations, with probability $1-\exp(-\Omega(\sqrt{n}/\log(n)))$, to find a global optimum for all functions with $\exp(o(\sqrt{n}/\log n))$ optima.

We use this result in Section 4.2 to prove a lower bound on the required cutoff time for configurators using the Optimisation-Time performance metric.

3.2.8 Fixed-Budget Analysis

The majority of the existing theoretical analyses of randomised search heuristics are concerned with estimating the time required to reach the optimum (or an approximation thereof). However, this differs from how these algorithms are often used in practice. In particular, they are *anytime* algorithms, i.e. they can be halted at any time and return a solution. Jansen and Zarges argue that it would be of more interest to practitioners to calculate the quality of a solution found when the algorithm is run for a specific time *budget* [73]. A similar line of analysis is to calculate the improvement in the solution quality obtained by, for example, doubling its original time budget. Works addressing these questions are called *fixed-budget analyses*.

Due to its relatively recent introduction and high complexity, there are few techniques for carrying out fixed-budget analysis. Many of the works in this area favour direct analyses of specific problems rather than using more elegant, general frameworks. Since estimating the solution quality achieved by different parameter settings within a given time budget is crucial for the analysis of algorithm configurators, in this section we give an overview of the few existing techniques for conducting fixed-budget analysis. We provide problem-specific fixed-budget results when defining the target problem classes in Section 3.5. The fixed-budget results available in the literature are insufficient for our purposes for some combination of the following reasons:

- They are only concerned with bounds on the expected fitness value or bounds that hold with non-overwhelming probability, whereas we require fitness bounds that hold with overwhelming probability.

- The bounds on the fitness are not precise enough.
- They do not deal with generalised parameter values, instead holding only for a single configuration.

The first generalised framework for carrying out fixed-budget analyses was developed by Doerr *et al.* It allows results from runtime analysis to be turned into fixed-budget results [38]. This method is based on the intuition that if there exists a function that yields the expected time for an algorithm to reach some fitness value then the inverse of this function corresponds to the expected fitness value at each time step. However, this intuition is only true for deterministic algorithms: for stochastic algorithms it is easy to find counterexamples. The following method uses this insight but is applicable to both stochastic and deterministic algorithms.

First, expected optimisation times are transformed into expected times required by the algorithm to reach an arbitrary fitness value. Second, deviation bounds (i.e. bounds on the probability of this time deviating from its expectation by a given amount) are derived for these times. Given a function that returns the time required to reach a given fitness value that holds with some probability, the inverse of this function can then used to compute the fitness value at a given time which holds with the same probability.

We now formalise this method following [38]. Let us denote by $T(a)$ the time for the algorithm to reach the fitness a. Let the probability p_u and the function $t_u(a)$ be such that $P(T(a) \geq t_u(a)) \leq p_u$. Then, denoting the inverse of $t_u(a)$ as $t_u^{-1}(a)$, we have a lower bound $t_u^{-1}(a)$ on the fitness at time t that holds with probability at least $1 - p_u$: $P(f(x_t) \leq t_u^{-1}(a)) \leq p_u$. We can similarly derive corresponding upper bounds on the fitness at an arbitrary time t by deriving p_l and $t_l(a)$.

Lengler and Spooner also created a method to allow results from runtime analysis to be used in a fixed-budget context if the algorithm exhibits multiplicative drift [87].

Theorem 27. (Theorem 1 in [87]) Let X_t be defined as in Section 3.2.6. If, for $0 < \delta < 1$ and $t \in \mathbb{N}$, we can bound the expected progress of X_t as $\mathrm{E}[X_t - X_{t+1} \mid X_t = x] \geq \delta x$ then, given X_0 (the initial value of X_t), $\mathrm{E}[X_t \mid X_0] \leq X_0 e^{-\delta t}$. Similarly, if $\mathrm{E}[X_t - X_{t+1} \mid X_t = x] \leq \delta x$, then $\mathrm{E}[X_t \mid X_0] \geq X_0 e^{-2\delta t}$.

Kötzing and Witt recently presented a method to calculate an upper bound on the value of a stochastic process at a given point in time if it exhibits variable drift [82].

Theorem 28 (Theorem 5 in [82])**.** Let X_t and τ be defined as in Section 3.2.6, and let X_t be adapted to a filtration $\mathcal{F}_t$. Let $h : S \to \mathbb{R}^{\geq 0}$ be a differentiable and convex function such that

$\tilde{h}$ is monotone non-decreasing and $\tilde{h}'(0) \in (0,1]$, where $\tilde{h} = x - h(x)$. If $\mathrm{E}[X_t - X_{t+1} \mid \mathcal{F}_t, t < \tau] \geq h(X_t)$, then

$$\mathrm{E}[X_t \mid \mathcal{F}_0] \leq \tilde{h}^t(X_0) + \frac{\tilde{h}(0)}{\tilde{h}'(0)},$$

and, in particular,

$$\mathrm{E}[X_t] \leq \tilde{h}^t(\mathrm{E}[X_0]) - \frac{\tilde{h}(0)}{\tilde{h}'(0)} \cdot \Pr(t \geq \tau \mid \mathcal{F}_0),$$

where $\tilde{h}^t$ is the t-times composition of $\tilde{h}$.

Theorem 28 can be generalised to deal with the case where the drift is not monotone.

Theorem 29 (Fixed-budget variable drift (Theorem 7 in [82])). Let X_t and τ be defined as in Section 3.2.6, and let X_t be adapted to a filtration $\mathcal{F}_t$. Let h be a non-decreasing function such that

$$\mathrm{E}[X_t - X_{t+1} \mid \mathcal{F}_t; t < \tau] \geq h(X_t).$$

Define $g : S \to \mathbb{R}$ by

$$g(x) = \begin{cases} \frac{x_{\min}}{h(x_{\min})} + \int_{x_{\min}}^{x} \frac{1}{h(z)} dz, & \text{if } x \geq x_{\min} \\ 0, & \text{otherwise} \end{cases} .$$

Then it holds that

$$\mathrm{E}[g(X_t \mid \mathcal{F}_0)] \leq g(X_0) - \sum_{s=0}^{t-1} \Pr(s < T).$$

Pérez Heredia derives fixed-budget versions of additive and multiplicative drift theorems using stochastic differential equations (SDEs) [58]. Applying these theorems to randomised algorithms requires the assumption, called the *diffusion approximation*, that the behaviour of the algorithm can be approximated by a family of well-studied stochastic processes. There is evidence, both experimental and from comparing the fixed-budget results obtained under this assumption with other results in the literature, that the diffusion approximation is approximately satisfied for evolutionary algorithms [58]. Using this approximation, Pérez Heredia derives a number of fixed-budget results on the behaviour of RSHs on standard test problems.

3.3 A Simple Algorithm Configurator: ParamRLS

The first step towards creating a theoretical foundation for algorithm configuration is the definition of a simple algorithm configurator. Similarly to the well-studied (1+1) EA in evolutionary computation, this basic parameter tuner should allow the performance analysis to be simple whilst at the same time possessing the right modularity to enable the addition of more sophistication, eventually allowing for the analysis of realistic algorithm configurators as used in practice.

In particular, this basic configurator should have characteristics that allow the development of general mathematical techniques created for its analysis that can be extended without too much effort for the analysis of more sophisticated and realistic configurators. To this end, we design a bare-bones version of the popular ParamILS configurator (Section 2.8.1) obtained by stripping it down to the essentials and removing all additional features.

This configurator, which we call ParamRLS, allows us to assess a baseline performance for algorithm configurators which can be reasonably considered to be a lower bound on the expected performance of more sophisticated configurators.

ParamRLS captures the iterative evolutionary loop at the heart of all state-of-the-art algorithm configurators. Specifically, it uses the following basic framework laid out for ParamILS [62]:

1. Initialise with some configuration θ;
2. Mutate θ by modifying a single parameter to create θ';
3. Accept the new configuration θ' if it performs at least as well as θ in a *comparison*;
4. Go to Step 2 until the termination criterion is satisfied.

We call the configuration θ that is mutated (and possibly replaced) in each iteration the *active parameter*. ParamILS and ParamRLS both follow this simple framework. The difference is that the former mutates configurations using an iterated local search mechanism whereas the latter replaces this with a simplified stochastic local search step (i.e. where it moves to a single neighbouring configuration in each iteration). The resulting configurator is described in Algorithm 11.

3.3.1 The $\pm\{1,\dots,m\}$ Local Search Operator

ParamRLS is designed to optimise numerical parameters. For some maximum step size m, we call the local search operator "$\pm\{1,\dots,m\}$" (for instance, if the maximum step size is 2 then we call the operator $\pm\{1,2\}$). This operator mutates a configuration by first selecting a parameter uniformly at random and then selecting a new value for this parameter from all values at distance at most m. For example, consider using the $\pm\{1,2\}$ operator to mutate a configuration of a single-parameter algorithm, where the parameter only takes values

Algorithm 11: ParamRLS $(\mathcal{A},\Theta,\Pi',\kappa,r)$

```
Input : target algorithm ($\mathcal{A}$), parameter space ($\Theta$), training instances ($\Pi'$), cutoff time
        ($\kappa$), number of runs per evaluation ($r$).
1 $\theta \leftarrow$initial configuration chosen uniformly at random // Step 1; initialise active parameter
2 while termination condition not satisfied do
3     $\theta' \leftarrow$ mutate($\theta$) // Step 2: choose $\theta'$ u.a.r. with replacement from neighbourhood of $\theta$
4     $\theta \leftarrow$ better($\mathcal{A},\theta,\theta',\Pi',\kappa,r$) // Step 3: compare $\theta$ and $\theta'$ w.r.t. performance metric
5 return $\theta$
```

in $\{1,2,\ldots,10\}$. If mutating the configuration where the parameter has value 5, then the mutation operator selects a new parameter value uniformly at random from the set $\{3,4,6,7\}$.

To enable the configuration of continuously-valued parameters, we follow ParamILS and discretise the parameter space. We do so using a *discretisation factor d*, defining the parameter space as the set

$$\left\{\frac{1}{d},\frac{2}{d},\frac{3}{d},\ldots,\phi\cdot\left(1-\frac{2}{d}\right),\phi\cdot\left(1-\frac{1}{d}\right),\phi\right\},$$

where ϕ is the largest permitted value for the parameter (a discretisation factor of $d=1$ thus corresponds to discrete parameters). For example, if $d=3$ and $\phi=3$ then the possible values of the parameter are

$$\left\{\frac{1}{3},\frac{2}{3},1,\frac{4}{3},\frac{5}{3},2,\frac{7}{3},\frac{8}{3},3\right\}.$$

If this parameter is selected for mutation, then, supposing that it currently has the value α, the $\pm\{1,\ldots,m\}$ operator chooses a new value uniformly at random from the set

$$\left\{\alpha-\frac{m}{d},\alpha-\frac{m-1}{d},\ldots,\alpha-\frac{1}{d},\alpha+\frac{1}{d},\ldots,\alpha+\frac{m-1}{d},\alpha+\frac{m}{d}\right\}.$$

We allow infeasible parameter values (i.e. values outside the parameter space) to be generated, and assume that any perturbation that oversteps a boundary is considered infeasible by ParamRLS and thus automatically loses any comparison against a feasible configuration. This constitutes a pessimistic assumption in terms of the time taken to identify an optimal configuration since it would be more efficient simply not to generate infeasible configurations, but it has minimal impact on runtime and simplifies the analysis.

The operator $\pm\{1\}$ has previously been analysed for the optimisation of functions defined over search spaces with larger alphabets than those that can be represented using bit strings [33].

The $\pm\{1,\ldots,m\}$ search operator differs from that used in ParamILS in four ways. First, configurations are generated with replacement, whereas in the iterative first improvement procedure of ParamILS (Section 2.8.1, Algorithm 4) configurations are generated without replacement. Second, the neighbourhood size of ParamILS is considerably larger than that of ParamRLS (i.e. in ParamILS, $m = \phi$). Thirdly, the $\pm\{1,\ldots,m\}$ operator is able to generate infeasible configurations whereas ParamILS is not. Finally, multiple parameters are dealt with in a subtly different manner. In ParamRLS, a parameter is selected u.a.r. and then perturbed, whereas in ParamILS, all configurations that differ in exactly one parameter value may be selected with the same probability (i.e. there is a larger chance of perturbing a parameter with relatively few values when using the search operator in ParamRLS rather than the one in ParamILS).

3.3.2 Evaluation of Configurations

We call ParamRLS using the Best-Fitness performance metric *ParamRLS-F* whilst if it uses the Optimisation-Time performance metric then we refer to it as *ParamRLS-T*. ParamRLS evaluates configurations using a static number of runs, r, as in the BasicILS version of ParamILS. Naturally, more sophisticated methodologies to reduce the number of runs per evaluation (e.g. as in FocusedILS) may also be applied in ParamRLS, but we do not do so in this thesis.

In practice, it may be desirable to cache the performances of configurations. However, we assume throughout this thesis that no caching takes place. Apart from simplifying the analysis, re-evaluating the performance of configurations may also be useful since it allows the configurator to overcome local optima introduced by the noisy evaluation of configurations.

In each run, ParamRLS-F executes θ and θ' on the same r problem instances and compares the fitness value achieved within the cutoff time κ. A configuration is considered to "win" a run if it achieves a higher fitness value than its opponent within the cutoff time. If both configurations achieve the same fitness value, then the first configuration to do so is considered the winner. The configuration that "wins" the most runs is the winner of the overall comparison. If both configurations have won the same number of runs, then the winner of the comparison is chosen uniformly at random. The pseudocode for ParamRLS-F is given in Algorithm 12.

In ParamRLS-T, we denote the time taken by $\mathcal{A}(\theta)$ to reach the target fitness value $\mathcal{T}$ for instance π using cutoff time κ and penalty constant $\mathcal{P}$ by $\texttt{CapTargetTime}(\mathcal{A}(\theta),\kappa,\pi,\mathcal{P},\mathcal{T})$. In each run, `CapTargetTime` is computed for both configurations in the comparison. The sum of these times is compared and the configuration with the lowest total runtime is

Algorithm 12: ParamRLS-F $(\mathcal{A}, \Theta, \Pi', \kappa, r)$

```
Input: target algorithm ($\mathcal{A}$), parameter space ($\Theta$), training instances ($\Pi'$), cutoff time
       ($\kappa$), number of runs per evaluation ($r$).
1  $\theta \leftarrow$ initial configuration chosen uniformly at random
2  while termination condition not satisfied do
3      $\theta' \leftarrow$ mutate($\theta$)
4      repeat $r$ times
5          $\pi \leftarrow$ problem instance drawn from $\Pi'$ according to $\mathcal{I}$
6          Fit $\leftarrow \mathcal{A}(\theta)$ fitness after $\kappa$ iterations on $\pi$
7          Fit$' \leftarrow \mathcal{A}(\theta')$ fitness after $\kappa$ iterations on $\pi$
8          ImprTime $\leftarrow$ time of last improvement of $\mathcal{A}(\theta)$ on $\pi$
9          ImprTime$' \leftarrow$ time of last improvement of $\mathcal{A}(\theta')$ on $\pi$
10         if Fit $>$ Fit$'$ then W $\leftarrow$ W $+ 1$
11         else if Fit$' >$ Fit then W$' \leftarrow$ W$' + 1$
12         else
13             if ImprTime $<$ ImprTime$'$ then W $\leftarrow$ W $+ 1$
14             else if ImprTime$' <$ ImprTime then W$' \leftarrow$ W$' + 1$
15     if W$' >$ W then $\theta \leftarrow \theta'$
16     else if W $==$ W$'$ then with probability 0.5 do $\theta \leftarrow \theta'$
17 return $\theta$
```

considered the winner of the comparison. In the case of a tie, the winner is chosen uniformly at random. The pseudocode for ParamRLS-T is given in Algorithm 13.

As stated in Section 2.5, we define the runtime of an algorithm configurator as the number of configuration comparisons that it requires before it is able to return the optimal configuration for the considered performance metric. When using the Best-Fitness performance metric, the optimal configuration is that which achieves the highest expected solution quality within the cutoff time. When using the Fixed-Target performance metric, the optimal configuration is that which in expectation takes the least time to reach the target solution quality (the global optimum, in the case of the Optimisation-Time performance metric).

3.4 Target Algorithms

In this section, we introduce the simple target algorithms that will be used throughout this thesis to evaluate the expected time required by algorithm configurators to identify their optimal parameter values. For the theoretical analyses in this thesis, we select the two most simple randomised search heuristics in the literature: randomised local search (RLS) and the (1+1) evolutionary algorithm (the (1+1) EA). These simple algorithms both have

Algorithm 13: ParamRLS-T $(\mathcal{A}, \Theta, \Pi', \kappa, r)$

```
Input : target algorithm ($\mathcal{A}$), parameter space ($\Theta$), training instances ($\Pi'$), cutoff time
        ($\kappa$), number of runs per evaluation ($r$).
1  $\theta \leftarrow$initial configuration chosen uniformly at random
2  while termination condition not satisfied do
3      $\theta' \leftarrow \texttt{mutate}(\theta)$
4      TargetTime $\leftarrow 0$ // runtime counter for $\mathcal{A}(\theta)$
5      TargetTime$'$ $\leftarrow 0$ // runtime counter for $\mathcal{A}(\theta')$
6      repeat $r$ times
7          $\pi \leftarrow$ problem instance drawn from $\Pi'$ according to $\mathcal{I}$
8          TargetTime $\leftarrow$ TargetTime $+$ $\texttt{CapTargetTime}(\mathcal{A}(\theta), \kappa, \pi, \mathcal{P}, \mathcal{T})$
9          TargetTime$'$ $\leftarrow$ TargetTime$'$ $+$ $\texttt{CapTargetTime}(\mathcal{A}(\theta'), \kappa, \pi, \mathcal{P}, \mathcal{T})$
10     if TargetTime$'$ $<$ TargetTime then $\theta \leftarrow \theta'$
11     else with probability 0.5 do $\theta \leftarrow \theta'$
12 return $\theta$
```

only a single parameter to be configured (i.e. the neighbourhood size and mutation rate, respectively), and thus are ideal for conducting initial analyses to establish a theoretical foundation of algorithm configuration. RLS$_k$ and the (1+1) EA both belong to the class of unary unbiased black-box algorithms (Section 3.2.7). For the experimental work conducted in Chapter 7, we additionally configure two parameters of the SAPS algorithm for the MAX-SAT optimisation problem. This simple randomised local search has been used as a target algorithm in several algorithm configuration papers [62, 64].

3.4.1 Randomised Local Search

Given a notion of distance between any two points in the search space, randomised local search with neighbourhood size k (RLS$_k$) evaluates a search point chosen uniformly at random from the neighbourhood of size k around the current best-found solution and replaces the best-found with it if it is no worse. That is, it follows a path of non-worsening neighbours (possibly evaluating solutions more than once). Since it can only move between neighbouring search points, randomised local search is liable to get stuck in local optima.

When a bit string representation is used, as is the case throughout this thesis, the neighbourhood of size k of a bit string x is the set of all bit strings that are exactly a Hamming distance of k from x (i.e. that differ from x in exactly k bits). We give the pseudocode for RLS$_k$ operating on bit strings in Algorithm 14.

The neighbourhood of RLS$_k$ is most commonly set to 1 (i.e. RLS$_1$) and this default version of the algorithm is usually simply referred to as 'RLS'. However, for different

Algorithm 14: RLS$_k$ for the maximisation of a function f [60].

```
1 initialise $x$ // according to initialisation scheme
2 while termination criterion not met do
3     $x' \leftarrow x$ with exactly $k$ distinct bits flipped, chosen uniformly at random
4     if $f(x') \geq f(x)$ then $x \leftarrow x'$
```

problems and time budgets, different neighbourhood sizes are preferable. The aim of the algorithm configurator is to identify the optimal neighbourhood size k for a given problem class. In our analyses, we will estimate the expected time required by different algorithm configurators to identify the optimal value for k according to the considered performance metric.

3.4.2 The (1+1) EA

Evolutionary algorithms as used in practice are often complex and therefore challenging to study theoretically. The (1+1) evolutionary algorithm ((1+1) EA) is a simplified evolutionary algorithm designed to ease initial analyses that can subsequently be built upon to assess the performance of more sophisticated variants. When using a bit string representation of length n, in each iteration the (1+1) EA creates a new solution by flipping each bit in the best-found solution independently with probability χ/n, for some *mutation rate* χ. As in RLS$_k$, this offspring replaces its parent if its fitness is no worse than it. We call the (1+1) EA with mutation rate χ the "(1+1)$_\chi$ EA". We give the pseudocode for the (1+1)$_\chi$ EA operating on bit strings of length n in Algorithm 15.

Algorithm 15: The (1+1)$_\chi$ EA for the maximisation of a function f [43].

```
1 initialise $x$ // according to initialisation scheme
2 while termination criterion not met do
3     $x' \leftarrow x$ with each bit flipped with probability $\chi/n$
4     if $f(x') \geq f(x)$ then $x \leftarrow x'$
```

Even though it is unrealistic as it does not use a population of candidate solutions, the (1+1) EA was defined to enable initial rigorous performance analyses of EAs [43], with similar intentions to ours in the creation of ParamRLS for the initial analysis of algorithm configurators. Indeed, the (1+1) EA has allowed the development of mathematical techniques that nowadays enable the analysis of evolutionary algorithms used in practice [5, 41, 69, 97].

Like RLS$_k$, the (1+1)$_\chi$ EA is ideal for our purposes as it has only one parameter (the mutation rate χ). However, configuring the (1+1)$_\chi$ EA constitutes a more challenging task than configuring RLS$_k$ as the mutation rate is continuous, unlike the discrete neighbourhood of RLS$_k$. Thus ideally a configurator should be able to distinguish between arbitrarily small differences in parameter values. Traditionally, a mutation probability of $1/n$ (i.e. $\chi = 1$) is recommended [117]. This has been proven to be optimal for all linear functions (i.e. where the fitness of a bit string is a linear combination of the value of each bit) [117]. However, for other problem classes, or even for identifying approximate solutions for linear functions, other mutation rates are preferable. We will analyse the time required by ParamRLS to identify the optimal mutation rate for the (1+1)$_\chi$ EA optimising the benchmark problem classes under different performance metrics.

3.4.3 Scaling and Probabilistic Smoothing (SAPS)

For solving the MAX-SAT problem, randomised local search algorithms typically use the number of unsatisfied clauses as the fitness of a solution (i.e. an assignment of truth values to the variables). To avoid getting stuck in local optima, the Scaling and Probabilistic Smoothing (*SAPS*) algorithm uses an alternative measure of fitness by assigning weights to each clause and defining the fitness of a solution as the sum of the weights of unsatisfied clauses [67]. These weights are updated during the optimisation process, and thus can transform the fitness function, allowing local optima to be escaped.

In each iteration, SAPS first performs a search step, in which it selects the solution with highest fitness from all those where a single variable appearing in an unsatisfied clause of the parent solution has been flipped. It then, with probability wp, flips a variable selected uniformly at random. Finally, it updates the weights of unsatisfied clauses. SAPS updates weights in two stages, first scaling the weights of unsatisfied clauses by multiplying them by the value α, and then, with probability P_{smooth}, moves each weight w closer to the mean of all weights $\bar{w}$ by applying the transformation $w = (\rho \cdot w) + (1 - \rho)\bar{w}$ (thus the parameter ρ controls the extent to which weights are drawn towards the mean: $\rho = 1$ results in all weights being set to $\bar{w}$, whereas $\rho = 0$ results in no smoothing).

In the experimental section of Chapter 7, we configure α and ρ and leave P_{smooth} and wp as their default values of 0.05 and 0.01, respectively.

3.5 Target Problem Classes

We now introduce the standard benchmark problem classes from evolutionary computation for which we will assess the performance of the configurators when tuning the parameters of RLS$_k$ and the (1+1)$_\chi$ EA. An important reason for the selection of these particular problem classes is that, for either target algorithm, a configuration will have the same performance for any instance in the class. Thus performance may be assessed without worrying about generalisation to unseen instances as it is not influenced by the chosen training set. For each problem class, we therefore define the training set as consisting of only the canonical problem instance (i.e. the problem instance commonly analysed in the evolutionary computation literature), which we specify for each class.

3.5.1 RIDGE

The RIDGE problem is that of maximising the length of the prefix that matches a secret bit string when the remainder of the bit string is the complement of the remainder of the secret bit string. The member of the RIDGE benchmark problem class corresponding to the secret bit string a consists of a ridge of bit strings with a prefix that matches a and where all bits not in this matching prefix differ from a [43][14]. On this ridge, fitness increases with the length of the matching prefix. All points not on the ridge have a lower fitness than all points on it, and thus once RLS$_k$ or the (1+1)$_\chi$ EA is on the ridge it will never accept a newly-generated search point that is not on it.

The function RIDGE(x) is defined as

$$\text{RIDGE}(x) = \begin{cases} n + \sum_{i=1}^{n} x_i, & \text{if } x \text{ is in the form } 1^i 0^{n-i} \\ n - \sum_{i=1}^{n} x_i, & \text{otherwise} \end{cases} ,$$

and the fitness of a bit string given the hidden bit string a is

$$\text{RIDGE}_a(x) := \text{RIDGE}(x_1 \leftrightarrow a_1 \ldots x_n \leftrightarrow a_n)$$

where "$\leftrightarrow$" is the "biconditional" operator (true if and only if both bits have the same value) [42]. The RIDGE problem class thus consists of 2^n functions, each corresponding to a distinct secret bit string $a \in \{0,1\}^n$.

The canonical instance of this problem class corresponds to $a = 1^n$ (i.e. the ridge contains bit strings of the form $1^i 0^{n-i}$). Note that $\text{RIDGE}_{1^n}(x) = \text{RIDGE}(x)$. Without loss of generality,

[14]Droste *et al.* define the fitness of a bit string using the "exclusive or" operator [42], whereas in this section we use the biconditional operator. The two operators are complementary: the biconditional is true if and only if two variables have the same value, whilst the exclusive or is true if and only if two variable differ. This modification yields the more natural situation where the secret bit string corresponds to the optimum and thus the goal is to maximise the quality of the bit string with respect to the secret bit string (this quality is specific to each problem class). These two definitions yield the same problem classes: the instance corresponding to a secret bit string in the definition of Droste *et al.* can be obtained using our terminology using the complement of the secret bit string.

we will use this instance for the analysis in this thesis. The performance will be the same on any other instance of this class.

To ease the analysis, we assume throughout this thesis that the algorithm is initialised at the beginning of the ridge (i.e. at the bit string $\bar{a}$, the complement of a). This assumption has already been made in the literature in other settings, including for fixed-budget analyses [72] and the analysis of hyper-heuristics [89].

Since RLS$_k$ always flips exactly k bits, it can only makes steps of size k towards the optimum, as flipping bits other than the initial k that disagree with the secret bit string a will generate a search point off the ridge. Therefore it is not possible to reach the optimum if n is not a multiple of k. The optimal value of RIDGE which RLS$_k$ can reach is in fact $\lfloor \frac{n}{k} \rfloor k$. In order to avoid an infinite expected optimisation time, in this thesis we will consider reaching a fitness of at least $2n - \sqrt{n} + 1$ as having optimised the function (i.e. the final $\sqrt{n}$ points on the ridge are all considered optimal), as this allows all configurations with $1 \leq k \leq \sqrt{n}$ to reach the optimum. We do not consider neighbourhood sizes of $k > \sqrt{n}$ since these values lead to an almost random search. We ensure that all optima have the same fitness by defining $\text{RIDGE*}_a(x) := \min\{\text{RIDGE}_a(x), 2n - \sqrt{n} + 1\}$ and instead configure RLS$_k$ for this class (again tuning for the canonical instance where $a = 1^n$).

Once on the ridge, RLS$_k$ makes a step of exactly k towards the optimum with probability $1/\binom{n}{k}$ (observe that this probability is independent of the current fitness). For $1 \leq k \leq \sqrt{n}$, this probability is maximised for $k = 1$. By the waiting time argument, RLS$_1$ therefore has an expected optimisation time of $(2n - \sqrt{n} + 1) \cdot n = \Theta(n^2)$ for RIDGE*. The differences in the improvement probabilities between different configurations are highly pronounced, with RLS$_m$ requiring a linear amount of time more, in expectation, to make progress than RLS$_{m+1}$. This implies that it should be easy for an algorithm configurator to determine which configuration in a comparison is better.

Similar behaviour can be observed for the (1+1)$_\chi$ EA. Once on the ridge, the drift of the individual is roughly $(\chi/n)(1 - (\chi/n))^{n-1}$, which is maximised for $\chi = 1$. Note that, like with RLS$_k$, the expected progress of the algorithm does not depend on the current fitness. The expected optimisation time when initialised at $\bar{a}$ can be shown to be en^2 using additive drift analysis.

The fact that, for both target algorithms, the expected progress is independent of the position in the search space and, for RLS$_k$, the gap between the performance of different configurations is highly pronounced, makes RIDGE (resp. RIDGE*) ideal for laying the foundations to be built upon for the analysis of the more complex problem classes considered later. In particular, these characteristics imply that the optimal parameter values for RLS$_k$

and the (1+1)$_\chi$ EA are $k = 1$ and $\chi = 1$, respectively, independent of the chosen performance metric.

Jansen and Zarges give bounds on the expected fitness of RLS$_1$ when run for t steps [73]:

Theorem 30 (Theorem 7 in [73])**.** Consider RLS$_1$ initialised at 0^n optimising $\textsc{Ridge}_{1^n}$. Then after t iterations, with $t \leq (1-\varepsilon)n^2$ for some constant $0 < \varepsilon < 1$,

$$t\left(1-\frac{1}{n}\right) - e^{-\Omega(n)} \leq \mathrm{E}[\textsc{Ridge}_{1^n}(x_t)] \leq t\left(1-\frac{1}{n}\right).$$

For our analysis, we will need to extend this result to hold for any parameter value k and derive bounds on the identified solution quality that hold with overwhelming probability rather than in expectation. We must also derive similar results for the (1+1)$_\chi$ EA.

3.5.2 OneMax

The OneMax problem is that of minimising the Hamming distance to a secret bit string. The class therefore consists of 2^n functions, each corresponding to a different secret bit string a [42]. For each instance, the fitness of an individual increases with the Hamming distance to a. The function OneMax(x) is defined as

$$\textsc{OneMax}(x) = \sum_{i=1}^{n} x_i,$$

which counts the number of ones in a bit string. The fitness of an individual for a problem instance corresponding to the secret bit string a is given by

$$\textsc{OneMax}_a(x) = \textsc{OneMax}(x_1 \leftrightarrow a_1 \ldots x_n \leftrightarrow a_n).$$

Thus $\textsc{OneMax}_a(x)$ counts the number of bits in which x matches a. The canonical member of this problem class corresponds to $a = 1^n$, and therefore the fitness of a bit string is the number of one bits:

$$\textsc{OneMax}_{0^n}(x) = \textsc{OneMax}(x).$$

As with Ridge, it is not always possible for RLS$_k$ to reach the optimum of this function. This occurs if the distance to the optimum is at most $\lfloor k/2 \rfloor$. To avoid infinite expected optimisation times, we treat any search point with a distance to the optimum of at most $\lfloor \phi/2 \rfloor$ as an optimum, where ϕ is the largest permitted value of k in the configuration scenario. It is relevant for later analyses to note that this implies that the number of optima is therefore, using $\binom{n}{m} \le (en/m)^m$,

$$\sum_{i=0}^{\lfloor \phi/2 \rfloor} \binom{n}{i} \le \left(\left\lfloor \frac{\phi}{2} \right\rfloor + 1 \right) \binom{n}{\lfloor \phi/2 \rfloor} \le \left(\left\lfloor \frac{\phi}{2} \right\rfloor + 1 \right) \cdot \left(\frac{e}{\lfloor \phi/2 \rfloor} \right)^{\lfloor \phi/2 \rfloor} \cdot n^{\lfloor \phi/2 \rfloor},$$

which is polynomial for $\phi = O(1)$. As with the definition of RIDGE*, we ensure that all optima have the same fitness by defining ONEMAX*$(x) := \min\{$ONEMAX$(x), n - \lfloor \phi/2 \rfloor\}$.

We analyse the configuration of RLS$_k$ for ONEMAX* and assume that the algorithm is initialised uniformly at random. This analysis is more complex than analysing the configuration of RLS$_k$ for RIDGE* since, for ONEMAX*, the optimal neighbourhood size depends on the fitness of the individual. To maximise expected progress, it is necessary to use larger neighbourhood sizes when far from the optimum and to decrease them when closer. This is similar to real-world scenarios, where often it is preferable to increase the size of the perturbation in correlation with the distance to the optimum. This behaviour implies that large values of k will be optimal when the cutoff time is small (under the Best-Fitness performance metric) or when the target solution quality is far from the optimum (under the Fixed-Target performance metric), whereas small k will be preferable when using larger cutoff times or when the target solution quality is close to the optimum. The challenge of detecting these different optimal values of k makes it more challenging to configure for ONEMAX* than for RIDGE*.

To simplify the analysis, when configuring RLS$_k$ for ONEMAX* we only allow $k \in \{1,2,3,4,5\}$. When at a distance of s from the optimum, RLS$_1$ has drift of s/n, RLS$_k$ with $k \in \{2,3\}$ makes expected progress of approximately $k(s/n)^2$ and RLS$_k$ with $k \in \{4,5\}$ makes expected progress in the range $[k(s/n)^3, 2k(s/n)^3]$. The neighbourhood size $k = 1$ has an expected optimisation time of $n \log n + O(n)$, which follows from simple coupon collector's arguments[15][98]. This is the smallest expected optimisation time among all configurations of RLS$_k$, and is in fact optimal within the class of unary unbiased black-box algorithms [35, 36] up to lower order terms of $\pm O(n)$. RLS$_k$ with $k \in \{2,3,4,5\}$ can be seen to have an expected optimisation time of $\Omega(n^2)$ by considering the expected time required to make the final improvement for each configuration. Regardless of the fitness of the individual, flipping $2c$ bits never yields higher drift than flipping $2c+1$ bits (for any positive integer c)[16] [35, 36]

[15]Given a set of n "coupons" and the ability to buy a single coupon chosen u.a.r., the coupon collector's problem is that of calculating the expected number of purchases necessary before all n different coupons have been collected. This corresponds to RLS$_1$ optimising ONEMAX when initialised at 0^n and thus provides an upper bound on the expected optimisation time of RLS$_1$ for the problem considered here.

[16]Although it is not necessarily optimal to maximise the drift in order to minimise the expected optimisation time for ONEMAX [20].

In terms of fixed-budget analyses, Jansen and Zarges provide a bound on the expected fitness of RLS$_1$ for OneMax after t iterations [73].

Theorem 31 (Theorem 5 in [73])**.** Consider RLS$_1$ initialised uniformly at random optimising OneMax. Then after t iterations

$$\mathrm{E}[\textsc{OneMax}(x_t)] = \frac{n}{2} + \frac{n}{2} \cdot \left(1 - \left(1 - \frac{1}{n}\right)^t\right).$$

3.5.3 LeadingOnes

The LeadingOnes problem (occasionally abbreviated to ‘LO’ in this thesis) is that of maximising the length of the prefix that matches the prefix of a secret bit string. We analyse the configuration of the $(1+1)_\chi$ EA for this problem class and assume that the algorithm is initialised uniformly at random.

The LeadingOnes function is defined as

$$\textsc{LeadingOnes}(x) = \sum_{i=1}^{n} \prod_{j=1}^{i} x_j,$$

which counts the number of consecutive one bits at the start of a bit string. The fitness of an individual for the LeadingOnes instance with secret bit string a is given by

$$\textsc{LeadingOnes}_a(x) = \textsc{LeadingOnes}(x_1 \leftrightarrow a_1 \dots x_n \leftrightarrow a_n),$$

which returns the length of the longest prefix of the bit string that matches the prefix of a [42].

Therefore, the LeadingOnes problem class again consists of 2^n functions, each corresponding to a different secret bit string. The canonical instance considered in this thesis corresponds to the secret bit string $a = 1^n$, where the fitness of an individual is the length of the prefix of consecutive one bits. Note that $\textsc{LeadingOnes}_{1^n}(x) = \textsc{LeadingOnes}(x)$.

It has been shown that a mutation rate of $\chi \approx 1.59$ yields the smallest expected optimisation time (approximately $0.77n^2$) for the $(1+1)_\chi$ EA and LeadingOnes [18]. As with OneMax, in order to maximise expected progress it is necessary to use larger perturbations as the fitness distance to the optimum increases, and hence runs with shorter cutoff times (under the Best-Fitness performance metric) or with the target solution quality far from the optimum (under the Fixed-Target performance metric) will favour larger values of χ, whereas runs with larger cutoff times or where the target solution quality is close to the optimum will favour smaller χ.

In terms of fixed-budget analysis, Jansen and Zarges derive bounds that hold with probability $1 - o(1)$ on the fitness of the $(1+1)_1$ EA when optimising LeadingOnes [73]:

Theorem 32 (Theorem 14 in [73])**.** Consider the (1+1)$_1$ EA, initialised uniformly at random, optimising LEADINGONES. Then, for a number of iterations $t = cn^2$, for any constant c with $0 < c < 1/2$,

$$\Pr\left(ce^{-c}(1+\exp(\exp(-c\exp(-c)))) \le \frac{\mathrm{LO}(x_t)}{n} \le c(1+\exp(-c\exp(-c)))\right) = 1 - o(1).$$

Doerr *et al.* used their general framework for fixed-budget analysis (outlined in Section 3.2.8) to derive bounds on the fixed-budget fitness of the (1+1) EA [1] for LEADINGONES [38]. The bound with the tightest concentration is the following.

Theorem 33 (Consequence of Theorem 9 in [38])**.** If the following bounds do not exceed $n - \log n$ then, with probability $1 - 2^{-\Omega(n^{\varepsilon})}$, after t iterations optimising LEADINGONES the fitness of the (1+1)$_1$ EA can be bounded as

$$\frac{\ln\left(\frac{n^2-n}{n^2-n+2b-2d-o(d)}\right)}{\ln(1-1/n)} \le \mathrm{LO}(x_t) \le \frac{\ln\left(\frac{n^2-n}{n^2-n+2b+2d+o(d)}\right)}{\ln(1-1/n)},$$

for any $d = \Omega(n^{3/2+\varepsilon})$, where ε is an arbitrarily small constant.

Theorem 28 (page 71) can be applied to bound the fixed-budget fitness of the (1+1) EA [1] when optimising LEADINGONES.

Theorem 34 (Theorem 8 in [82])**.** Consider the (1+1)$_1$ EA for LEADINGONES. After t iterations, we have

$$\mathrm{E}[\mathrm{LO}(x_t)] \ge \begin{cases} \frac{2t}{n} - O(1), \text{ if } t = O(n^{3/2}) \\ \frac{2t}{n} - (1-o(1)), \text{ if } t = o(n^2) \\ n\ln(1+\frac{2t}{n^2}) - O(1), \text{ if } t \le \frac{e-1}{2}n^2 - n^{3/2} \end{cases}.$$

Applying Theorem 29 (page 72) in this context is shown to be simpler than using Theorem 28, but, due to its weaker conditions on the drift (i.e. that the drift need not be monotonically non-decreasing), the fitness bounds yielded are weaker.

Kötzing and Witt also derive tail bounds on the expected fitness of the (1+1)$_1$ EA when optimising LEADINGONES. We do not reproduce these bounds here as doing so would introduce several extra definitions, but we do reproduce the result derived using these bounds: an interval that contains the fitness at time t with probability at least $1 - \frac{1}{n}$, derived using the technique described by Doerr *et al.* in [38].

Theorem 35 (Theorem 10 in [82])**.** Consider the (1+1)$_1$ EA for LEADINGONES. Then, for $t = \omega(n \log n)$ and $t \le (e-1)n^2/2 - cn^{3/2}\sqrt{\log n}$, where c is a sufficiently large constant,

$$-n\ln\left(1-\frac{2t}{n^2}+O\left(\frac{\sqrt{t}\log n}{n^{3/2}}\right)\right) \le \mathrm{LO}(x_t) \le -n\ln\left(1-\frac{2t}{n^2}-O\left(\frac{\sqrt{t}\log n}{n^{3/2}}\right)\right),$$

with probability at least $1 - 1/n^3$.

For small t, these bounds are tighter than those given in Theorem 33. However, as t increases they soon become less precise.

3.6 Conclusions

In this chapter, we introduced the prerequisites required for our analysis of algorithm configurators. We first provided an overview of the mathematical techniques from the literature that we use in this thesis. We then defined the configurator ParamRLS and the target algorithms and problem classes that will be used in the analysis, as well as providing the motivations behind their selection.

Part II

Analysis of Algorithm Configurators

Chapter 4

Fixed-Target Performance Metric Requires Appropriate Cutoff Times

4.1 Introduction

The Fixed-Target performance metric (Section 2.5.1) is arguably the most commonly used in the algorithm configuration literature. This metric is designed to be applied when a minimal solution quality is sought as it returns the time required by a candidate configuration to reach the target solution quality. It requires the use of a cutoff time to terminate the evaluation of poor parameter configurations that require a prohibitively long time to reach the desired solution quality. A common approach for dealing with such configurations (used, for example, in ParamILS) is the penalised average runtime methodology (PAR, see Section 2.5.1). For instances where the target solution quality is not reached within the cutoff time, PAR returns the cutoff time multiplied by a penalisation constant. If the cutoff time is so small that a large proportion of the configurations do not reach the target solution quality, then the algorithm configurator will struggle to identify a gradient towards high-quality configurations. Naturally, the choice of an appropriate cutoff time requires problem-specific knowledge, both in terms of the hardness of the problem instances that are expected to be encountered by the target algorithm and in terms of the sensitivity of the performance of the target algorithm to different parameter settings.

In this chapter, we investigate the impact of the cutoff time on the performance of algorithm configurators that employ the Fixed-Target performance metric. We first provide a formal proof that if the cutoff time is too small for any configuration to reach the target solution quality, with overwhelming probability, then any configurator using the Fixed-Target performance metric will behave as if all configurations have the same performance. This result allows us to derive a general lower bound on the cutoff time required by any algorithm configurator to be efficient if the global optimum is chosen as the target solution quality (i.e. the Optimisation-Time performance metric). This bound is of particular interest since this is the most commonly selected target in practical examples of algorithm configuration [16].

We then turn to positive results by proving an upper bound on the cutoff time that is sufficient for configurators that use global mutation operators to guarantee convergence to the optimal configuration. We prove that if the cutoff time is large enough for the T-optimal configuration to reach the target solution quality of every problem instance with overwhelming probability, and if it does so for each instance in less time than all other configurations, also with overwhelming probability, then the expected number of configuration samples required by the configurator to return this configuration is linear in the size of the parameter space. This bound is independent of whether any other configuration can reach the optimum within the cutoff time. As a corollary, we show that ParamILS can identify the T-optimal configuration of a single-parameter target algorithm using the identified bounds on the cutoff time. We conclude the chapter by showing that, even if the O-optimal configuration is sought by the user, the same performance can be achieved by using the Best-Fitness performance metric. Thus the derived positive result does not highlight an advantage of using the Fixed-Target performance metric over others.

4.2 A Lower Bound on the Necessary Cutoff Time

If the cutoff time is small enough such that with overwhelming probability no configuration of the target algorithm reaches the sought solution quality for any member of the training set, then any configurator using the Fixed-Target performance metric will clearly not be able to distinguish between the performance of the different configurations. We call a configurator that behaves in this way "blind". We start by defining this notion formally and then provide a simple proof of this fact.

Definition 10. We call a configurator *blind* if there is an event A that occurs with overwhelming probability and, conditional on A, the configurator returns a configuration chosen according to the probability distribution that would be generated if all configurations had the same performance.

The event A characterises a 'typical' run of a tuner (i.e. a run with no occurrences of extremely unlikely events); it is necessary since in an "atypical" run (i.e. where at least one extremely unlikely event occurs), we may not know the output of the tuner. Our notion of blindness implies that if ParamRLS (or ParamILS) is blind then their output is virtually indistinguishable from a uniform random distribution of parameter values after any polynomial number of comparisons.

Theorem 36. Consider a configurator that uses the Fixed-Target performance metric tuning a target algorithm $\mathcal{A}$ for a problem class where the largest instance has size n. Assume that with overwhelming probability no configuration of the target algorithm $\mathcal{A}$ reaches the target solution quality for any member of the training set within the cutoff time. Then, after a number of comparisons that is polynomial in n, where each evaluation consists of at most $r \in \text{poly}(n)$ runs, the configurator is blind.

Proof. Since there are polynomially many comparisons, each of which consists of polynomially many runs, by the union bound no configuration will reach the target solution quality for any member of the training set within the cutoff time, with overwhelming probability. Therefore every configuration will have the same fitness. □

A corollary of Theorem 36 allows us to provide a lower bound on the cutoff time required by any algorithm configuration that uses the global optimum as the target solution quality for the Fixed-Target performance metric. In particular, we now prove that any configurator that uses the Optimisation-Time performance metric with a cutoff time of at most $\kappa \leq (n \ln n)/2$, where n is the size of the largest instance in the training set, is unable to tune any unary unbiased search heuristic when the training set consists of problem instances each with up to $\exp(\sqrt{n}/\log^2 n)$ optima. To prove the result, we apply the black-box complexity result of Lehre and Sudholt (given in Section 3.2.7) that states that no unary unbiased search heuristic optimises any such function within $(n \ln n)/2$ iterations, with overwhelming probability.

Corollary 37. Consider any configurator that uses the Optimisation-Time performance metric tuning any unary unbiased black-box target algorithm using a training set where each instance has maximum problem size n and has up to $\exp(\sqrt{n}/\log^2 n)$ optima. If the cutoff time is $\kappa \leq (n \ln n)/2$ and the number of runs per evaluation $r \in \text{poly}(n)$, then the configurator is blind.

Proof. From Theorem 26 (Section 3.2.7) we have that all unary unbiased black-box algorithms require at least $(n \ln n)/2$ iterations to reach the optimum of any function with up to $\exp(\sqrt{n}/\log^2 n)$ optima, with probability $1 - \exp(-\Omega(\sqrt{n}/\log n))$. By the union bound, the probability that none of the polynomially many runs of the algorithm reach the optimum within $(n \ln n)/2$ iterations is still overwhelming. Then Theorem 36 implies that the tuner is blind. □

4.3 An Upper Bound on the Sufficient Cutoff Time

In this section, we derive upper bounds on the cutoff time that are sufficient for configurators to return the T-optimal configuration.

We first prove that, if the cutoff time is set large enough such that the T-optimal configuration reaches the target solution quality of each training instance with overwhelming probability then any configurator that uses the Fixed-Target performance metric can identify the optimal configuration.

We point out that the condition that the T-optimal configuration reaches the target for all instances is required. For instance, the weaker condition that at least one configuration reaches the target for all instances would not be strong enough since it is possible for a non-T-optimal configuration to reach the target solution quality for all instances and for the T-optimal configuration not to do so, e.g. if it performs badly on a small number of instances.

Theorem 38. Consider a configurator that uses the Optimisation-Time performance metric tuning an algorithm $\mathcal{A}$ using a training set Π' with instance sizes n. Assume that in each iteration the configurator samples a new configuration and compares it against the best-found so far. Assume also that there are $|\Theta|$ configurations of $\mathcal{A}$, that the configurator uses at most r runs per configuration evaluation, that the cutoff time is large enough such that the T-optimal configuration, θ^*, reaches the optimum of every instance $\pi \in \Pi'$, with overwhelming probability, and that θ^* reaches the optimum of every instance $\pi \in \Pi'$ in less time than all other configurations, with overwhelming probability.

- If the configurator uses a mutation operator that samples configurations uniformly at random, then in expectation the number of comparisons sufficient to return θ^* is $|\Theta|$. After running for t comparisons, the probability that the best-found configuration is θ^* is at least
$$(1-\exp(-\Omega(n^{\varepsilon})))^{rt} \cdot \left(1-\left(1-\frac{1}{|\Theta|}\right)^t\right),$$
for some constant $\varepsilon > 0$.
- Assume that the configurator uses a global mutation operator: at every step it samples every configuration with probability at least $p_{\min} > 0$. Then, for every constant $\varepsilon > 0$, after $t = \Omega(n^{\varepsilon}/p_{\min})$ comparisons, assuming $rt \in \text{poly}(n)$, the best-found configuration is θ^*.

Proof. By the waiting time argument, the expected number of configuration samples required before sampling θ^* is $|\Theta|$. Since by assumption the cutoff time is large enough to allow θ^* to reach the optimum of each training instance with overwhelming probability and, also by assumption, it reaches the optimum of each training instance in less time than all other configurations, also with overwhelming probability, the probability that both of these events occur in all of the at most rt comparisons (and thus a lower bound on the probability that it will win all comparisons in which it is involved) is the product of two overwhelming probabilities raised to the power rt, which is equal to

$$(1 - \exp(-\Omega(n^{\varepsilon})))^{rt},$$

for some constant $\varepsilon > 0$. The second term in the stated probability is the probability that θ^* is sampled at least once in t comparisons.

For the second claim, the probability that θ^* is *not* sampled within $t = \Omega(n^{\varepsilon}/p_{\min})$ steps is at most

$$(1 - p_{\min})^t \leq e^{-tp_{\min}} \leq e^{-\Omega(n^{\varepsilon})}.$$

Hence, with overwhelming probability θ^* is sampled within t steps. Once θ^* is sampled, it is not beaten in any remaining comparison with overwhelming probability: as $rt \in \text{poly}(n)$, a union bound over polynomially many events that all occur with overwhelming probability (as shown in the proof of the first claim) proves that the claimed events occur with overwhelming probability. □

We highlight again that Theorem 38 requires the assumption that the T-optimal configuration reaches the target solution quality for *all* training instances with overwhelming probability, since otherwise runs terminated at the cutoff time may result in it having a larger estimated runtime than some non-T-optimal configurations. This condition is likely to be far stronger than is required in practice (where it is only necessary that this configuration has a smaller mean penalised optimisation time than its competitors over the sampled training instances).

For configurators that sample configurations uniformly at random without replacement, such as ParamILS when configuring single-parameter algorithms, it is possible to derive stronger guarantees on the time required to identify the optimal configuration.

Corollary 39. Consider ParamILS for the configuration of an algorithm $\mathcal{A}$ with a single parameter θ for a training set Π' with instance sizes n, arbitrary polynomial values for R, s and r, and arbitrary p_{restart}. Assume that the parameter space has size ϕ and that ϕ is at most polynomial in n. Assume that the cutoff time is large enough such that the T-optimal configuration, θ^*, reaches the target solution quality for every instance $\pi \in \Pi'$, with overwhelming probability. Assume also that θ^* reaches the optimum of every instance $\pi \in \Pi'$ faster than all other configurations with overwhelming probability. Then $R+\phi$ comparisons suffice for ParamILS to return θ^*, with overwhelming probability.

Proof. After the first $R+1$ comparisons, ParamILS starts the *IterativeFirstImprovement* procedure. In this procedure, since the target algorithm has only a single parameter, the set of unvisited neighbours of a configuration consists of *all* configurations which have not yet been sampled during that call to the procedure. Since there are only ϕ configurations in the scenario which we consider, this implies that θ^* is discovered within $\phi-1$ comparisons with probability 1.

Now, θ^* will be returned by *IterativeFirstImprovement* if, once sampled, it wins every subsequent comparison. By assumption, the cutoff time is large enough such that this happens with overwhelming probability in any given comparison. Taking a union bound over $R+\phi$ comparisons, the probability that, once sampled, θ^* never loses a comparison is overwhelming. □

The above results imply that choosing the optimum as the target solution quality and selecting a sufficiently large cutoff time allows configurators using the Optimisation-Time performance metric to identify the O-optimal configuration after at most a linear number of comparisons. This performance is also achieved by the Best-Fitness performance metric with the same cutoff times and thus these results do not constitute an advantage of the Optimisation-Time performance metric over Best-Fitness. This follows because ties (i.e. cases where both configurations achieve the same fitness within the cutoff time) are broken in favour of the configuration that made progress least recently, and thus if the O-optimal configuration reaches the optimum first for all training instances with overwhelming probability, then it will win any comparison against all non-O-optimal configurations with overwhelming probability, even if the Best-Fitness performance metric is used.

Corollary 40. Consider a configurator using the Best-Fitness performance metric tuning an algorithm $\mathcal{A}$ using a training set Π' with instance sizes n. Assume that there are $|\Theta|$ configurations of $\mathcal{A}$, the configurator uses at most r runs per configuration evaluation. Assume also that the cutoff time is large enough such that the O-optimal configuration θ^* reaches the optimum of every instance $\pi \in \Pi'$ with overwhelming probability and that for each instance it does so first, with overwhelming probability. Assume that the configurator uses a global mutation operator (e.g. samples configurations uniformly at random). Then the claims of Theorem 38 and Corollary 39 hold.

4.4 Conclusions

In this chapter, we have highlighted some important limitations for the general applicability of the Fixed-Target performance metric. In particular, we have shown that any configurator that uses this performance metric is highly dependent on the selection of an appropriate cutoff time, the choice of which may require significant problem knowledge. We provided a formal proof that if the cutoff time is so small that no configuration reaches the target solution quality for any member of the training set, then the configurator will be blind. This implies that if the cutoff time is no greater than $(n \ln n)/2$ then no configurator using the Optimisation-Time performance metric is able to configure any unary unbiased target algorithm when the training set consists of problem instances each with no more than $\exp(\sqrt{n}/\log^2 n)$ optima.

We also showed that if the cutoff time is large enough such that the T-optimal configuration reaches the target solution quality with overwhelming probability, then in expectation, for any configurator that samples configurations uniformly at random, a linear number of samples is sufficient to identify this configuration. Thus we highlighted the importance of setting the cutoff time appropriately when the Fixed-Target performance metric is employed. Hence choosing a large enough cutoff time is necessary, otherwise the configurator will return a random configuration, with overwhelming probability.

The great impact of the choice of cutoff time on the performance of configurators using the Fixed-Target performance metric highlights numerous difficulties that may be encountered when employed for the configuration of algorithms for real-world optimisation and serious potential limitations for its use in practice. For example, it necessitates that the training set consist of problem instances for which the time taken to reach the target solution quality is known for the parameter value ranges under consideration. In the case of the Optimisation-Time performance metric, we must additionally know the value of the optimum itself. These requirements are not necessarily fulfilled in real-world applications, and also imply that the training set must consist of easy problem instances since these are likely to be the ones for which an algorithm that has not yet been configured is able to identify the optimum. Hence 'good generalisation' with the Optimisation-Time performance metric may only be reasonably expected if 'easy' instances are expected to be encountered by the target algorithm after it has been tuned.

The main result of this chapter is that if the configuration that identifies the global optimum fastest is sought, then any configurator using the Fixed-Target performance metric to tune a unary unbiased algorithm requires a cutoff time of at least $\Omega(n \log n)$ for virtually any problem class of interest, or the algorithm configurator will return a random configuration. In the following chapters, we will show that considerably larger cutoff times may be necessary for even extremely simple algorithm configuration problems. On the other hand, we demonstrate that other performance metrics may be far more effective for this purpose. In particular, the Best-Fitness performance metric can identify the O-optimal configuration using considerably smaller cutoff times (i.e. $o(n \log n)$) than the lower bound proven in this chapter for the Optimisation-Time performance metric.

Chapter 5

On the Configuration of the Neighbourhood Size of Randomised Local Search

5.1 Introduction

In the previous chapter, we provided necessary lower bounds and sufficient upper bounds on the cutoff time for algorithm configurators using the Fixed-Target performance metric to be efficient, independent of the particular algorithm configuration scenario. In this chapter, we focus on two specific algorithm configuration scenarios and compare the performance of simple algorithm configurators using the Fixed-Target performance metric to those that use the Best-Fitness performance metric. In particular, we analyse the local search tuners ParamRLS and ParamILS for the configuration of the neighbourhood size k of randomised local search. Our aim is to characterise the impact of the cutoff time on the ability of the algorithm configurators to identify the optimal neighbourhood size (with respect to the performance metric) for the RIDGE* and ONEMAX* standard benchmark problem classes.

In Chapter 4, we proved that, when using the Optimisation-Time performance metric, a cutoff time of at least $(n \ln n)/2$ is required for any configurator to identify an O-optimal configuration of any unary unbiased search algorithm for any problem class where every training instance has at most $\exp(\sqrt{n}/\log^2 n)$ optima and n is the size of the largest instance in the training set. Whilst this lower bound on the required cutoff time holds for virtually any target problem class of interest, we will show that even larger cutoff times may be necessary to prevent a configurator using this performance metric from being blind. In particular, we prove that any configurator that uses the Optimisation-Time performance metric requires cutoff times that are at least quadratic in the problem size when configuring RLS$_k$ for the RIDGE* benchmark problem class. If the cutoff time is smaller, then any configurator using this performance metric will be blind. This result confirms that considerable problem knowledge is required to tune algorithms using the Optimisation-Time performance metric, even for very simple algorithm configuration scenarios.

We then turn our attention to the Best-Fitness performance metric. We first show that ParamRLS-F can efficiently identify that $k = 1$ is the F-optimal neighbourhood size for RIDGE* independent of the cutoff time as long as sufficiently many runs are performed in each evaluation (i.e. r is sufficiently large). We then use the ONEMAX* benchmark problem class to show the power of the Best-Fitness performance metric. When tuning RLS$_k$ for this problem class, and allowing k to take values in $\{1,2,3,4,5\}$, ParamRLS-F identifies that $k = 1$ is the F-optimal neighbourhood size for any cutoff time of at least $0.975n$ whilst for smaller cutoff times of $0.02n \leq \kappa \leq 0.72n$ it will identify that the F-optimal parameter value is $k = 5$. This result highlights how the Best-Fitness performance metric can identify the parameter value that maximises the solution quality for the available time budget. Significantly, unlike when using the Fixed-Target performance metric, no problem-specific information (such as the expected optimisation time or an estimate of the expected solution quality after a given time budget) is required for the tuner to be effective. Our results imply that, remarkably, even in the case where the parameter value that has the smallest expected optimisation time (i.e. the O-optimal configuration) is sought, the Best-Fitness performance metric can do so using considerably smaller cutoff times than those required by configurators using Optimisation-Time, for both RIDGE* and ONEMAX*.

5.2 On the Configuration of RLS$_k$ for RIDGE*

In this section, we analyse the ability of ParamRLS and ParamILS to configure RLS$_k$ for the RIDGE* benchmark problem class. This allows us to characterise the behaviour of these configurators for different performance metrics and ranges of cutoff times for the simplest benchmark problem class used in the theory of evolutionary computation. We allow k to take values up to $\sqrt{n}$, since larger values of k degrade to random search.

We first show that any configurator using the Optimisation-Time performance metric will be blind for any cutoff time $\kappa \leq (1-\varepsilon)n^2$, for constant $\varepsilon > 0$. Note that this negative result for RIDGE* is stronger than the general negative results provided in Chapter 4 since it holds for $\kappa = O(n^2)$ rather than $\kappa = O(n \log n)$.

We then prove that ParamRLS-F, on the other hand, identifies that $k = 1$ is F-optimal for RLS$_k$ and RIDGE* for any cutoff time. If the cutoff time is large enough (i.e. $\kappa = \omega(n)$), then even a single run per configuration evaluation suffices. For smaller cutoff times, ParamRLS-F requires more runs per configuration evaluation to identify that RLS$_1$ performs better than any other RLS$_k$ for $k > 1$. We show this for the extreme case of $\kappa = 1$, where $n^{3/2}$ runs per evaluation suffice for ParamRLS-F to return $k = 1$ with overwhelming probability. Since $k = 1$ is also O-optimal, this implies that *any* cutoff time is sufficient for ParamRLS-F to return the O-optimal parameter value (whilst any configuration using Optimisation-Time requires at least quadratic cutoff times).

Before we can analyse the performance of the configurators tuning RLS$_k$ for RIDGE*, we must first understand how the value of k affects the performance of RLS$_k$ for this problem class. We present this analysis in the following section. Without loss of generality, in the following section and the rest of this thesis, all proofs assume that the target algorithm runs on the canonical instance of the problem class (i.e. the training set only contains copies of the canonical instance). This assumption does not affect any results since for all considered problem classes and target algorithms the performance is independent of the problem instance. We do not make this assumption explicit after this point.

5.2.1 The Optimisation Time of RLS$_k$ for RIDGE* is Tightly Concentrated

In this section, we derive the expected optimisation time of RLS$_k$ with respect to the neighbourhood size k and show that it is minimised for $k = 1$. We then show that the optimisation time is tightly concentrated around its expectation.

Lemma 41. For $k \leq \frac{n}{2}$, when initialised at 0^n the expected optimisation time of RLS$_k$ for RIDGE* is $\left\lceil \frac{n-\sqrt{n}+1}{k} \right\rceil \binom{n}{k}$.

Proof. During a single iteration, it is only possible to increase the fitness of an individual by exactly k since we must flip exactly the first k zeroes in the bit string (any other combination of flips will mean that the string is no longer in the form 1^i0^{n-i} and will be rejected). We call an iteration in which we flip exactly the first k zeroes in the bit string an *improvement*. There are $\binom{n}{k}$ possible ways in which we can flip k bits and exactly one of these combinations flips the first k zeroes. Therefore the probability of making an improvement at any given time is $1/\binom{n}{k}$.

By the waiting time argument, we wait $\binom{n}{k}$ iterations in expectation to make a single improvement. Since the algorithm is initialised at 0^n, we need to make $\lceil (n-\sqrt{n}+1)/k \rceil$ improvements in order to reach the optimum. We therefore wait $\lceil (n-\sqrt{n}+1)/k \rceil \binom{n}{k}$ iterations in expectation until we reach the optimum. □

Corollary 42. A value of $k = 1$ leads to the smallest expected optimisation time for RLS$_k$ for RIDGE* for any $k \leq n/2$.

The following lemma shows that optimisation time of RLS$_k$ on RIDGE* is tightly concentrated around its expectation, i.e. with overwhelming probability deviations from $\binom{n}{k}(n/k)$ by a factor greater than $(1 \pm \varepsilon)$ for every constant $\varepsilon > 0$, do not occur.

Lemma 43. With probability at least $1 - \exp(-\Omega(n/k))$, RLS$_k$ requires at least $(1-\varepsilon)\binom{n}{k}(n/k)$ and at most $(1+\varepsilon)\binom{n}{k}(n/k)$ iterations to optimise RIDGE*, for every constant $\varepsilon > 0$ and large enough n.

Proof. Let X^i equal 1 if RLS$_k$ makes an improvement in iteration i and equal 0 otherwise. Let X_t denote the number of improvements made by RLS$_k$ within the first t iterations. That is, $X_t = \sum_{i=1}^{t} X^i$. Recall that RLS$_k$ can only make improvements by exactly k when optimising RIDGE* and therefore needs to make $\lceil (n - \sqrt{n} + 1)/k \rceil$ improvements to reach the optimum. Since the probability of making an improvement is $1/\binom{n}{k}$ when at any non-optimal point, we have $\mathrm{E}[X_t] \leq t/\binom{n}{k}$, by the linearity of expectation (this is not an equality since progress stops once we reach an optimum).

Let $t = (1-\varepsilon)\binom{n}{k}\frac{n}{k}$, for a constant ε satisfying $\varepsilon > 0$. For $\varepsilon \geq 1$ the claim that the runtime is at least t holds automatically since $t \leq 0$. To prove the claim for $\varepsilon < 1$, we first observe that $\mathrm{E}[X_t] \leq (1-\varepsilon)\frac{n}{k}$. We now apply Chernoff bounds (Theorem 19), treating this upper bound on $\mathrm{E}[X_t]$ as an equality (this Chernoff bound remains correct despite an upper bound on the expectation being used [32]):

$$\Pr\left(X_t \geq \left\lceil \frac{n-\sqrt{n}+1}{k} \right\rceil\right) = \Pr\left(X_t \geq \left(1 + \left(\frac{\left\lceil \frac{n-\sqrt{n}+1}{k} \right\rceil}{(1-\varepsilon)\frac{n}{k}} - 1\right)\right)\mathrm{E}[X_t]\right)$$
$$\leq \exp\left(-\frac{\Theta(1)\cdot\Theta(n)}{3\cdot\Theta(k)}\right) = \exp(-\Omega(n/k)).$$

The above Chernoff bound holds as $\lceil (n-\sqrt{n}+1)/k \rceil / ((1-\varepsilon)(n/k)) - 1 = \Theta(1)$ is positive for large enough n.

We proceed similarly to obtain an upper bound on the runtime. However, since we cannot use the upper bound on $\mathrm{E}[X_t]$ in the Chernoff bound for the lower tail (as we did above), we instead assume that the algorithm is operating on an infinite bit string and thus can make progress at any time, still with probability $1/\binom{n}{k}$. The time required by this modified process to make $\lceil (n-\sqrt{n}+1)/k \rceil$ improvements is identical to that required to do so by RLS$_k$ on RIDGE*. This time, we let $t = (1+\varepsilon)\binom{n}{k}\frac{n}{k}$, which yields $\mathrm{E}[X_t] = (1+\varepsilon)\frac{n}{k}$. Hence, again by

Chernoff bounds:

$$\begin{aligned}
\Pr\left(X_t < \left\lceil \frac{n-\sqrt{n}+1}{k} \right\rceil\right) &\le \Pr\left(X_t \le \left\lceil \frac{n-\sqrt{n}+1}{k} \right\rceil\right) \\
&= \Pr\left(X_t \le \left(1-\left(1-\frac{\left\lceil \frac{n-\sqrt{n}+1}{k} \right\rceil}{(1+\varepsilon)\frac{n}{k}}\right)\right)\mathrm{E}[X_t]\right) \\
&\le \exp\left(-\frac{\Theta(1)\cdot\Theta(n)}{2\cdot\Theta(k)}\right) = \exp(-\Omega(n/k)).
\end{aligned}$$

The above Chernoff bound holds as $1-\lceil (n-\sqrt{n}+1)/k \rceil/((1+\varepsilon)(n/k)) = \Theta(1)$ converges to $1-1/(1+\varepsilon) < 1$ for large enough n. □

5.2.2 Configurators Using the Optimisation-Time Performance Metric Require Quadratic Cutoff Times

In this section, we show that any configurator that uses the Optimisation-Time performance metric is blind for any cutoff time of at most $\kappa \le (1-\varepsilon)n^2$ (for any constant $\varepsilon > 0$). However, larger cutoff times will allow the configurator to be effective.

Theorem 44. Consider any configurator that uses the Optimisation-Time performance metric for the configuration of RLS$_k$ for RIDGE* with $\phi \le \sqrt{n}$. Assume that it runs for a number of comparisons that is polynomial in n, uses cutoff time $\kappa \le (1-\varepsilon)n^2$ (for any constant $\varepsilon > 0$). Then, for any $r \in \mathrm{poly}(n)$, the configurator is blind.

Proof. By Lemma 43, no configuration will have reached the optimum within the cutoff time with overwhelming probability (since this is the case for RLS$_1$, which minimises $\binom{n}{k}(n/k)$). Therefore the claim follows by applying Theorem 36. □

Since the optimisation time of RLS$_k$ for RIDGE* is so tightly concentrated around its expectation, even a small increase in the cutoff time (i.e. from $(1-\varepsilon)n^2$ to $(1+\varepsilon)n^2$) allows configurators using the Optimisation-Time performance metric to be able to identify the O-optimal neighbourhood size. We now prove that ParamRLS-T is able to do so for RLS$_k$ optimising RIDGE* if the cutoff time is at least $(1+\varepsilon)n^2$.

We derive the expected number of comparisons required before the active parameter in ParamRLS-T has been set to $k = 1$. We also bound the probability that the active parameter has not been set to $k = 1$ after a given number of comparisons. Note that it is not sufficient for the active parameter merely to be set to $k = 1$, since it is still possible for it to then change again to a different value. We therefore require that the active parameter remains at this configuration for the remainder of the tuning time. We deal with this requirement in the same theorem. For this result and the others in this section, the $\pm\{1\}$ mutation operator suffices for ParamRLS to be efficient.

Theorem 45. Consider ParamRLS-T for the configuration of RLS_k for RIDGE* with $\phi \leq \sqrt{n}$. Assume that it uses cutoff time $\kappa \geq (1+\varepsilon)n^2$ (for any constant ε satisfying $\varepsilon > 0$), $r \in \text{poly}(n)$ runs per evaluation, and that it uses the local search operator $\pm\{1\}$. Then the expected number of comparisons T before ParamRLS-T sets the active parameter to $k = 1$ for the first time is at most $2\phi(\phi-1)$. After $t \geq 4\phi(\phi-1)$ comparisons, ParamRLS-T returns the parameter value $k = 1$ with probability at least

$$1 - 2^{-\Omega(t/\phi^2)} - t \cdot r \cdot \exp(-\Omega(n/\phi)).$$

Note that this is exponentially small for $t = \Omega(\phi^2 n^\varepsilon)$, for polynomial t.

Proof. According to Lemma 43, RLS_1 has reached the optimum of RIDGE* within $(1+\varepsilon)n^2$ iterations (for any constant $\varepsilon > 0$), with probability $1 - \exp(-\Omega(n))$. Lemma 43 also implies that, with probability $1 - \exp(-\Omega(n/b)) \geq 1 - \exp(-\Omega(n/\phi))$, RLS_a reaches the optimum of RIDGE* before RLS_b, for $a < b$. Thus, in ParamRLS-T with r runs per evaluation and t comparisons tuning RLS_k for RIDGE*, RLS_b never beats RLS_a (with $a < b$) in a comparison, with probability at least $1 - t \cdot r \cdot \exp(-\Omega(n/\phi))$.

Let us assume that the value of the active parameter performs a lazy random walk (see Section 3.2.5) over the possible parameter values, with the parameter value 1 corresponding to an absorbing state. That is, the value of the active parameter either increases or decreases by 1 (assuming that this new value is permitted) with probability 1/4 each, and remains the same with probability 1/2. When it is not possible to increase the value by 1 (i.e. when the active parameter is $k = \phi$), the active parameter decreases with probability 1/4 and otherwise remains the same. This model is pessimistic since it is the case that arises in the scenario in which neither configuration reaches the optimum within the cutoff time. Since, as shown above, RLS_b does not beat RLS_a in a comparison with overwhelming probability if the cutoff time is large enough such that both configurations reach the optimum, it holds that this random walk assumption is therefore a worst-case scenario, and, provided that the cutoff time is large enough, progress towards the state $k = 1$ will in fact be faster. Note that we cannot assume that RLS_a beats RLS_b with overwhelming probability for all a and $b > a$ since, for some values of a, the cutoff time may not be large enough to ensure that RLS_a has reached the optimum with overwhelming probability.

Using standard random walk arguments (Section 3.2.5), the expected first hitting time of state 1 is at most $2\phi(\phi-1)$, where the factor of 2 accounts for the probability of remaining at each state. By Markov's inequality, the probability that state 1 has not been reached in $4\phi(\phi-1)$ steps is at most $1/2$. Hence the probability that state 1 is not reached during $\lfloor t/(4\phi(\phi-1))\rfloor$ periods each consisting of $4\phi(\phi-1)$ steps is $2^{-\lfloor t/(4\phi(\phi-1))\rfloor} = 2^{-\Omega(t/\phi^2)}$.

Once state 1 is reached, the configurator remains there unless RLS$_2$ beats RLS$_1$ in a run. By the above arguments, this event does not happen in a specific comparison with probability at least $1-t\cdot r\cdot \exp(-\Omega(n/\phi))$.

By the union bound, the probability that ParamRLS-T returns $k=1$ after $t \geq 4\phi(\phi-1)$ comparisons is at least $1-2^{-\Omega(t/\phi^2)}-t\cdot r\cdot\exp(-\Omega(n/\phi))$. □

We now prove a similar result for ParamILS tuning RLS$_k$ for RIDGE* when using the Optimisation-Time performance metric.

Theorem 46. Consider ParamILS for the configuration of RLS$_k$ for RIDGE*, with $k \in \{1,\dots,\phi\}$, $\phi \leq \sqrt{n}$. Assume that it uses the Optimisation-Time performance metric, cutoff time $\kappa \geq (1+\varepsilon)n^2$ for some constant $\varepsilon > 0$, and arbitrary values for s, R, and p_{restart}. Then after $R+\phi$ comparisons, ParamILS returns the configuration $k=1$ with overwhelming probability.

Proof. By the same arguments as in the proof of Theorem 45, in a single run RLS$_1$ will reach the optimum of RIDGE* before any other configuration with probability at least $1-\exp(-\Omega(n/\phi))$, which is overwhelming since $\phi \leq \sqrt{n}$. Hence, RLS$_1$ wins a comparison against RLS$_k$ with any $k>1$, with overwhelming probability. The result then follows from the general upper bound in Corollary 39. □

5.2.3 ParamRLS-F can Identify the Optimal Neighbourhood Size Using Arbitrary Cutoff Times

We now turn our attention to ParamRLS-F for the configuration of RLS$_k$ for RIDGE*. We first analyse the relative performance of RLS$_a$ and RLS$_b$ with $a<b$ when optimising RIDGE*. We derive a general bound which can be applied to any two random processes with probabilities of improving that remain the same throughout the process, as is the case for RLS$_k$ optimising the RIDGE* problem class. We then derive an upper bound on the probability that the process with the higher probability of improving is ahead at some time t.

As with the positive result for ParamRLS-T, we derive the expected number of comparisons required before the active parameter in ParamRLS-F has been set to $k=1$ and calculate the probability that this does not occur within a given number of comparisons. In Theorem 49 we prove that, for cutoff times $\kappa=\Omega(n^{1+\varepsilon})$ (for some constant $\varepsilon>0$), one run per configuration evaluation suffices for ParamRLS-F to return the configuration $k=1$.

We then prove that any cutoff time suffices for ParamRLS-F to return this configuration if suitably many runs per evaluation are conducted. Hence ParamRLS-F can identify the O-optimal configuration for any cutoff time.

Lemma 47. Let $\mathcal{A}$ and $\mathcal{B}$ be two random processes which both take values from the non-negative real numbers, and both start with value 0. At each time step, $\mathcal{A}$ increases by some real number $\alpha \geq 0$ with probability p_a, and otherwise stays put. At each time step, $\mathcal{B}$ increases by some real number $\beta \geq 0$ with probability p_b, and otherwise stays put. Let Δ_t^a and Δ_t^b denote the total progress of $\mathcal{A}$ and $\mathcal{B}$ in t steps, respectively. Let $q := p_a(1-p_b)+(1-p_a)p_b$, $q_a := p_a(1-p_b)/q$, and $q_b := p_b(1-p_a)/q$. Then, for all $0 \leq p_b \leq p_a$ and $\alpha, \beta \geq 0$

$$\Pr(\Delta_t^b \geq \Delta_t^a) \leq \exp\left(-qt\left(1-2q_b^{\alpha/(\alpha+\beta)}q_a^{\beta/(\alpha+\beta)}\right)\right).$$

Proof. Let $q := p_a(1-p_b)+(1-p_a)p_b$ be the probability that exactly one process makes progress in a single time step. Let $q_a := p_a(1-p_b)/q$ be the conditional probability of $\mathcal{A}$ making progress, given that one process makes progress, and define q_b likewise. Assume that in t steps we have ℓ progressing steps. Then the probability that $\mathcal{B}$ makes at least as much progress as $\mathcal{A}$ is $\Pr(\text{Bin}(\ell, q_b) \geq \lceil \ell\alpha/(\alpha+\beta)\rceil)$. Then,

$$\Pr(\Delta_t^b \geq \Delta_t^a) = \sum_{\ell=0}^{t} \Pr(\text{Bin}(t,q)=\ell)\cdot\Pr(\text{Bin}(\ell,q_b) \geq \lceil \ell\alpha/(\alpha+\beta)\rceil) \tag{5.1}$$

Note that $p_b \leq p_a$ is equivalent to $q_b \leq q_a$. Thus, $q_b/q_a \leq 1$. Hence

$$\begin{aligned}\Pr(\text{Bin}(\ell,q_b) \geq \lceil \ell\alpha/(\alpha+\beta)\rceil) &= \sum_{i=\lceil \ell\alpha/(\alpha+\beta)\rceil}^{\ell} \binom{\ell}{i} q_b^i q_a^{\ell-i}\\ &= \sum_{i=\lceil \ell\alpha/(\alpha+\beta)\rceil}^{\ell} \binom{\ell}{i} q_b^{\ell\alpha/(\alpha+\beta)} q_a^{\ell-(\ell\alpha/(\alpha+\beta))}(q_b/q_a)^{i-(\ell\alpha/(\alpha+\beta))}\\ &\leq 2^\ell q_b^{\ell\alpha/(\alpha+\beta)} q_a^{\ell-(\ell\alpha/(\alpha+\beta))} = \left(2q_b^{\alpha/(\alpha+\beta)}q_a^{\beta/(\alpha+\beta)}\right)^\ell.\end{aligned}$$

Using the above in (5.1) and $\Pr(\text{Bin}(t,q)=\ell) = \binom{t}{\ell}q^\ell(1-q)^{t-\ell}$ yields

$$\begin{aligned}\Pr(\Delta_t^b \geq \Delta_t^a) &\leq \sum_{\ell=0}^{t}\binom{t}{\ell}q^\ell(1-q)^{t-\ell}\cdot\left(2q_b^{\alpha/(\alpha+\beta)}q_a^{\beta/(\alpha+\beta)}\right)^\ell\\ &= \sum_{\ell=0}^{t}\binom{t}{\ell}(1-q)^{t-\ell}\cdot\left(2q\cdot q_b^{\alpha/(\alpha+\beta)}q_a^{\beta/(\alpha+\beta)}\right)^\ell\end{aligned}$$

(using the binomial theorem)

$$\begin{aligned}
&= \left(1-q+2q\cdot q_b^{\alpha/(\alpha+\beta)}q_a^{\beta/(\alpha+\beta)}\right)^t \\
&= \left(1-q\left(1-2q_b^{\alpha/(\alpha+\beta)}q_a^{\beta/(\alpha+\beta)}\right)\right)^t \\
&\le \exp\left(-qt\left(1-2q_b^{\alpha/(\alpha+\beta)}q_a^{\beta/(\alpha+\beta)}\right)\right). \qquad \square
\end{aligned}$$

This lemma allows us to derive a lower bound on the probability that RLS$_a$ has found a better fitness value than RLS$_b$ after κ steps, and hence that it wins a comparison against RLS$_b$ (with $a < b$) with a cutoff time of κ. Additional arguments for $\kappa < \binom{n}{a}$ allow us to show in the next lemma that the probability that RLS$_a$ wins is always at least $1/2$.

Lemma 48. For every $1 \le a < b = o(n)$, in a comparison in ParamRLS-F with a single run on RIDGE* with cutoff time κ, RLS$_a$ wins the comparison against RLS$_b$ with probability at least

$$\max\left\{\frac{1}{2},\ 1-\exp\left(-\kappa/\binom{n}{a}\cdot(1-o(1))\right)-\exp(-\Omega(n/b))\right\}.$$

Proof. Using the notation from Lemma 47, we have $p_a = 1/\binom{n}{a}$ and $p_b = 1/\binom{n}{b}$, which implies $p_b = o(p_a)$ since $b = o(n)$. Furthermore, $q \ge 1/\binom{n}{a}\cdot(1-o(1))$, $q_a = 1-o(1)$ and $q_b = p_b(1-p_a)/q \le p_b(1-p_a)/(p_a(1-p_b)) \le p_b/p_a = \frac{b!(n-b)!}{a!(n-a)!} \le (b/(n-b))^{b-a}$. This implies $q_b^{a/(a+b)} \le (b/(n-b))^{a(b-a)/(a+b)}$. Using $b/(n-b) = o(n)/n = o(1)$ and $a(b-a)/(a+b) \ge a/(2a+1) \ge 1/3$, we obtain $q_b^{a/(a+b)} = o(1)$. By Lemma 47, RLS$_a$ is ahead of RLS$_b$ with probability at least

$$1-\exp\left(-\kappa/\binom{n}{a}\cdot(1-o(1))\right).$$

The above argument ignores that progress stops once a global optimum is reached. If RLS$_a$ reaches a global optimum and RLS$_b$ does not, RLS$_a$ still wins. We use the union bound to include a term reflecting the possibility that RLS$_b$ finds the global optimum. By Lemma 43, if $\kappa \le (1-\varepsilon)\binom{n}{b}\lfloor n/b\rfloor$, for some constant $\varepsilon > 0$, the probability that RLS$_b$ does find the optimum is at most $\exp(-\Omega(n/b))$. For $\kappa \le (1-\varepsilon)\binom{n}{b}\lfloor n/b\rfloor$ this proves a lower bound of

$$1-\exp\left(-\kappa/\binom{n}{a}\cdot(1-o(1))\right)-\exp(-\Omega(n/b)). \tag{5.2}$$

For larger κ we argue that by Lemma 43, the probability that RLS$_a$ finishes within the first $(1-\varepsilon)\binom{n}{b}\lfloor n/b\rfloor \ge (1+\varepsilon)\binom{n}{a}\lfloor n/a\rfloor$ steps is $1-\exp(-\Omega(n/a)) \ge 1-\exp(-\Omega(n/b))$. Along with the fact that RLS$_b$ with probability $1-\exp(-\Omega(n/b))$ needs more than $(1-\varepsilon)\binom{n}{b}\lfloor n/b\rfloor$

steps, this proves that RLS$_a$ wins with probability at least $1-\exp(-\Omega(n/b))$ for $\kappa > (1-\varepsilon)\binom{n}{b}\lfloor n/b \rfloor$.

We have proved the claim for all $\kappa \geq \binom{n}{a}$, assuming n is large enough to make (5.2) at least as large as $1/2$. For $\kappa < \binom{n}{a}$ we additionally have to show that the probability of RLS$_a$ winning a comparison against RLS$_b$ is at least $1/2$. To this end, we argue that RLS$_b$ can only win if it makes progress in κ steps. The probability for this is at most $\kappa/\binom{n}{b}$, by the union bound. RLS$_a$ wins for sure if it does make progress in κ steps and RLS$_b$ does not make progress. The probabilities for these events are at least $1-\left(1-1/\binom{n}{a}\right)^{\kappa} \geq \kappa/(\kappa+\binom{n}{a})$ (using Theorem 16) and $1-\kappa/\binom{n}{b} = 1-o(1)$, respectively. So the probability that they both occur is at least

$$\frac{\kappa}{\kappa+\binom{n}{a}} \cdot (1-o(1)) \geq \frac{\kappa}{2\binom{n}{a}} \cdot (1-o(1)) > \frac{\kappa}{\binom{n}{b}}$$

for large enough n. Hence, in all cases where at least one algorithm makes progress, RLS$_a$ is more likely to win than RLS$_b$. In all other cases there is a tie and the probability that RLS$_a$ is declared winner is $1/2$. This proves a lower bound of $1/2$ for the probability that RLS$_a$ wins. □

Note that Lemma 48 implies that, for cutoff times $\kappa = \Omega(n^{\varepsilon}\binom{n}{\phi})$ (for some constant $\varepsilon > 0$), the parameter landscape seen by ParamRLS-F for the configuration of RLS$_k$ for RIDGE* is unimodal.

We now use the lower bound on the probability that RLS$_a$ beats RLS$_b$ (where $1 \leq a < b$) within κ steps given by Lemma 48 to analyse the time sufficient for ParamRLS-F to return the (both F-optimal and O-optimal) neighbourhood size $k=1$ of RLS$_k$ for RIDGE*.

Theorem 49. Consider ParamRLS-F for the configuration of RLS$_k$ for RIDGE* with $\phi \leq \sqrt{n}$. Assume that it uses cutoff time κ, a single run per evaluation (i.e. $r=1$) and that it uses the local search operator $\pm\{1\}$. Then the expected number of comparisons T before ParamRLS-F sets the active parameter to $k=1$ for the first time is at most $2\phi(\phi-1)$. After $t \geq 4\phi(\phi-1)$ comparisons, ParamRLS-F returns the parameter value $k=1$ with probability at least

$$1-2^{-\Omega(t/\phi^2)}-t\cdot(2^{-\Omega(\kappa/n)}+2^{-\Omega(n)}).$$

Note that this is overwhelming for $t=\Omega(\phi^2 n^{\varepsilon})$, where t is polynomial, and $\kappa = \Omega(n^{1+\varepsilon})$, for a positive constant ε.

Proof. By Lemma 48, the probability that RLS$_a$ beats RLS$_b$ in a comparison with any cutoff time is at least $1/2$. We can therefore model the tuning process as the value of the active parameter performing a lazy random walk over the states $1,\ldots,\phi$ (where state i corresponds to the active parameter being set to the configuration $k=i$). We pessimistically assume that the active parameter decreases and increases by 1 with respective probabilities $1/4$ and that it stays the same with probability $1/2$.

Using standard random walk arguments (Section 3.2.5) as in the proof of Theorem 45, the expected first hitting time of state 1 is at most $2\phi(\phi-1)$. By Markov's inequality, the probability that state 1 has not been reached in $4\phi(\phi-1)$ steps is at most $1/2$. Hence the probability that state 1 is not reached during $\lfloor t/4\phi(\phi-1)\rfloor$ periods each consisting of $4\phi(\phi-1)$ steps is $2^{-\lfloor t/4\phi(\phi-1)\rfloor} = 2^{-\Omega(t/\phi^2)}$.

Once state 1 is reached, the configurator remains there unless RLS$_2$ beats RLS$_1$ in a run. By Lemma 48, this event happens in a specific comparison with probability at most $2^{-\Omega(\kappa/n)} + 2^{-\Omega(n)}$. By a union bound over at most t comparisons, the probability that this ever happens is at most $t \cdot (2^{-\Omega(\kappa/n)} + 2^{-\Omega(n)})$. □

We remark that the probability bound from Theorem 49, and similarly for other later results, can be refined for a superpolynomial number of comparisons t by considering only the last n^c steps, for some polynomial $n^c \leq t$, yielding a probability bound of $1 - 2^{-\Omega(\min\{t,n^c\}/\phi^2)} - \min\{t,n^c\} \cdot (2^{-\Omega(\kappa/n)} + 2^{-\Omega(n)})$. We do not use this refined version in order to simplify the statements.

We now show that even the smallest possible cutoff time of $\kappa = 1$ (i.e. where each configuration is only run for a single iteration) is sufficient for ParamRLS-F to return $k = 1$ as long as there are sufficiently many runs per configuration evaluation.

Theorem 50. Consider ParamRLS-F for the configuration of RLS$_k$ for RIDGE* with $\phi \leq \sqrt{n}$. Assume that it uses cutoff time $\kappa = 1$, $n^{3/2}$ runs per configuration evaluation (i.e. $r = n^{3/2}$) and that it uses the local search operator $\pm\{1\}$. Then the expected number of comparisons T before ParamRLS-F sets the active parameter to $k = 1$ for the first time is at most $2\phi(\phi-1)$. After $t \geq 4\phi(\phi-1)$ comparisons, ParamRLS-F returns the value $k = 1$ with probability at least

$$1 - 2^{-\Omega(t/\phi^2)} - t \cdot \exp(-\Omega(\sqrt{n})).$$

Note that this is exponentially small for $t = \Omega(\phi^2 n^{\varepsilon})$, for any positive constant ε, and polynomial t.

Proof. We begin by showing that the active parameter value remains at $k = 1$ with overwhelming probability once it has been set to this value for the first time. Define X as the number of runs out of $n^{3/2}$ runs, each with cutoff time $\kappa = 1$, in which RLS$_1$ makes progress. Define Y as the corresponding variable for RLS$_2$. By standard Chernoff bounds (Theorem 19) we can show that $\Pr(X > \sqrt{n}/2) \geq 1 - \exp(-\Omega(\sqrt{n}))$. We can also show that, again by Chernoff bounds, $\Pr(Y < \sqrt{n}/2) \geq 1 - \exp(-\Omega(\sqrt{n}))$. Therefore, with overwhelming probability, RLS_1 has made progress in more of these $n^{3/2}$ runs than RLS_2. That is, with overwhelming probability, RLS_1 wins the comparison.

We can analyse this tuning process as a whole in the same way in which we analyse the tuning process in the proof of Theorem 49. We first observe that, in order for RLS_a to beat RLS_b (with $a < b$) in a run with cutoff time $\kappa = 1$, it is sufficient for it to have made an improvement and for RLS_b to have failed to do so. Letting A be the event that RLS_a beats RLS_b in a run with cutoff time $\kappa = 1$, we have

$$\Pr(A) \geq \frac{1}{\binom{n}{a}} \left(1 - \frac{1}{\binom{n}{b}}\right).$$

Let B denote the event that RLS_b beats RLS_a in a run with cutoff time $\kappa = 1$. Since RLS_b making progress is a necessary condition for event B to take place, we have $\Pr(B) \leq 1/\binom{n}{b}$. For large enough n, we have that

$$\frac{1}{\binom{n}{a}} \left(1 - \frac{1}{\binom{n}{b}}\right) \geq 1/\binom{n}{b}$$

which implies that $\Pr(A) \geq \Pr(B)$. This means that, for any $1 \leq x \leq r$ the probability that RLS_a wins x runs in a comparison is at least the probability that RLS_b wins x runs. Observing that if a comparison does not end in a draw then the winner must have won more runs than its competitor, we see that, since $\Pr(A) \geq \Pr(B)$, the winner must be RLS_a with probability at least $1/2$. This means that we can make the same pessimistic assumption as we do in the proof of Theorem 49, i.e. that the value of the active parameter decreases and increases by 1 with respective probabilities $1/4$ and that it stays the same with probability $1/2$. By the same arguments as in the proof of Theorem 49, we derive that the expected number of comparisons required to reach state $k = 1$ is at most $2\phi(\phi - 1)$. Again by the same technique as the proof of Theorem 49, we obtain that the probability that state $k = 1$ has not been reached after t comparisons is $2^{-\Omega(t/\phi^2)}$. Thus the probability that the tuner returns $k = 1$ after t comparisons is $1 - 2^{-\Omega(t/\phi^2)} - t \cdot \exp(-\Omega(\sqrt{n}))$. □

5.3 On the Configuration of RLS_k for OneMax*

For the Ridge* problem class analysed in the previous section, the configuration $k = 1$ is F-optimal regardless of the available time budget. Naturally, this is not necessarily the case for other problem classes. In this section, we consider one such case and analyse the performance of ParamRLS for the configuration of RLS_k for OneMax*. If RLS_k runs for few fitness function evaluations, then an algorithm with a larger neighbourhood size performs better (i.e. identifies better solutions) than one with a smaller one. On the other hand, if more fitness evaluations (i.e. larger time budgets) are permitted, then RLS$_1$ will identify better solutions than configurations that use larger values of k. Hence, $k = 1$ is O-optimal.

Since, $k = 2c + 1$ often (but not always: see Table 5.1, page 121) constitutes a local optimum of the parameter landscape, it is necessary to use the local search operator $\pm\{1,2\}$ to enable the active parameter to move from $k = 2c + 1$ to $k = 2c - 1$.

As ONEMAX* has at most polynomially many optima, by Corollary 37 any configurator that uses the Optimisation-Time performance metric is blind when configuring RLS$_k$ using a cutoff time of $\kappa \leq (n \ln n)/2$.

In this section, we show that ParamRLS-F, on the other hand, can identify whether $k = 1$ is F-optimal or whether a larger value for k performs better according to the time budget specified by the chosen cutoff time. To prove our point it suffices to consider ParamRLS-F with the parameter space $k \in \{1,2,3,4,5\}$, which also simplifies the analysis. We will prove that, even for a single run per configuration evaluation, ParamRLS-F identifies that $k = 1$ is F-optimal for any $\kappa \geq 0.975n$. Surprisingly, this time is considerably shorter than the expected time required by any configuration to optimise ONEMAX* (i.e. $\Theta(n \ln n)$), and hence ParamRLS-F is able to return the O-optimal configuration a logarithmic factor faster than any configurator using the Optimisation-Time performance metric. If, however, the cutoff time κ satisfies $0.02n \leq \kappa \leq 0.72n$, then ParamRLS-F will identify that $k = 5$ is F-optimal, as desired.

Before analysing ParamRLS-F, we first need to characterise the performance of RLS$_k$ for ONEMAX* with each considered neighbourhood size with respect to the cutoff time. We do so in the following section.

5.3.1 Fixed-Budget Analysis of RLS$_k$ and ONEMAX*

In this section, we show that RLS$_k$ with smaller neighbourhood sizes performs best (i.e. achieves the highest solution quality) for large cutoff times, whilst larger neighbourhood sizes are preferable for smaller cutoff times. This result is supported by previous analysis showing that the drift-maximising number of bit flips decreases as the distance to the optimum decreases [36]. We first derive precise bounds on the fitness values identified by different configurations after a given cutoff time. Afterwards, we will use these bounds to identify which configurations outperform others for different cutoff times. We begin by deriving bounds on the drift of RLS$_k$ when optimising ONEMAX*.

Lemma 51. The drift $\Delta_k(s)$ of RLS_k with current distance s to the optimum is

$$\Delta_k(s) = \sum_{i=\lfloor k/2 \rfloor+1}^{k} (2i-k) \cdot \binom{s}{i}\binom{n-s}{k-i} / \binom{n}{k}.$$

In particular, for $s \geq k$,

$$\begin{aligned}
\Delta_1(s) &= \frac{s}{n} \\
\Delta_2(s) &= \frac{2s(s-1)}{n(n-1)} \leq 2\left(\frac{s}{n}\right)^2 \\
\Delta_3(s) &= \frac{3s(s-1)}{n(n-1)} \leq 3\left(\frac{s}{n}\right)^2 \\
\Delta_4(s) &= \frac{8s(s-1)(s-2)(n-s/2-3/2)}{n(n-1)(n-2)(n-3)} \leq 8\left(\frac{s}{n}\right)^3 \\
\Delta_5(s) &= \frac{10s(s-1)(s-2)(n-s/2-3/2)}{n(n-1)(n-2)(n-3)} \leq 10\left(\frac{s}{n}\right)^3.
\end{aligned}$$

Proof. We first calculate the probability of flipping a certain number of bits in a bit string using RLS_k. If the bit string currently has Hamming distance s to the optimum, then the probability that a k-bit mutation flips exactly i bits that disagree with the optimum and $k-i$ bits that agree with the optimum is

$$\binom{s}{i}\binom{n-s}{k-i} / \binom{n}{k} \tag{5.3}$$

This corresponds to a hypergeometric distribution with parameters s and n.

If a k-bit mutation flips i disagreeing bits and $k-i$ agreeing bits, the distance to the optimum decreases by $i-(k-i) = 2i-k$. This is only accepted if $2i-k \geq 0$, and progress is only made if $2i-k > 0$ or, equivalently, $i > \lfloor k/2 \rfloor$. The claim then follows from (5.3) and the definition of the expectation.

By [36, Lemma 28] we have $\Delta_2(s) = 2\Delta_3(s)/3$ and $\Delta_4(s) = 4\Delta_5(s)/5$, hence we only need to show the claims for $\Delta_1(s), \Delta_3(s)$, and $\Delta_5(s)$. The formula $\Delta_1(s) = s/n$ follows immediately. For $\Delta_3(s)$ we have

$$\begin{aligned}\Delta_3(s) &= \left(\binom{s}{2}\binom{n-s}{1} + 3\binom{s}{3}\binom{n-s}{0}\right) / \binom{n}{3} \\ &= \left(\frac{s(s-1)(n-s)}{2} + \frac{3s(s-1)(s-2)}{6}\right) / \binom{n}{3} \\ &= \left(\frac{s(s-1)(n-2)}{2}\right) / \binom{n}{3} = \frac{3s(s-1)}{n(n-1)}.\end{aligned}$$

In order to bound $\Delta_3(s)$ we calculate that

$$\begin{aligned}\frac{3s(s-1)}{n(n-1)} \leq 3\left(\frac{s}{n}\right)^2 \iff \frac{s-1}{n-1} \leq \frac{s}{n} \iff \frac{s-1}{s} \leq \frac{n-1}{n} \\ \iff 1 - \frac{1}{s} \leq 1 - \frac{1}{n} \iff s \leq n\end{aligned}$$

which is trivially true. We can use a nearly identical argument to bound $\Delta_2(s)$. For $\Delta_5(s)$ we have

$$\begin{aligned}\Delta_5(s) &= \left(\binom{s}{3}\binom{n-s}{2} + 3\binom{s}{4}\binom{n-s}{1} + 5\binom{s}{5}\binom{n-s}{0}\right) / \binom{n}{5} \\ &= \left[\left(\frac{s(s-1)(s-2)}{6}\right)\left(\frac{(n-s)(n-s-1)}{2}\right) + \frac{3s(s-1)(s-2)(s-3)(n-s)}{24}\right. \\ &\quad \left. + \frac{5s(s-1)(s-2)(s-3)(s-4)}{120}\right] / \left(\frac{n(n-1)(n-2)(n-3)(n-4)}{120}\right) \\ &= \frac{s(s-1)(s-2)(10n^2 - 5ns - 55n + 20s + 60)}{n(n-1)(n-2)(n-3)(n-4)} \\ &= \frac{5s(s-1)(s-2)(2n-s-3)(n-4)}{n(n-1)(n-2)(n-3)(n-4)} = \frac{10s(s-1)(s-2)(n-s/2-3/2)}{n(n-1)(n-2)(n-3)}.\end{aligned}$$

By similar arguments to those used to bound $\Delta_3(s)$ we can see that

$$\frac{10s(s-1)(s-2)(n-s/2-3/2)}{n(n-1)(n-2)(n-3)} \leq 10\left(\frac{s}{n}\right)^3 \frac{n-s/2-3/2}{n-3}.$$

We therefore need to show that

$$n - \frac{s}{2} - \frac{3}{2} \leq n - 3$$

which holds if and only if $s \geq 3$. The stated bound holds if $s < 3$ since $\Delta_5(s)$ will equal 0. These two facts therefore prove the claim for all s. As above, we can use a nearly identical argument to prove the bound on $\Delta_4(s)$. □

In order to derive tight bounds on the fitness of the individual after a given number of iterations, we divide a run into periods of a fixed length and show that, with overwhelming probability, at the end of each period the fitness is always contained within a narrow interval of fitness values (Lemma 53). The location of these intervals depends on k. When the cutoff time is large enough, these intervals become non-overlapping, which allows us to prove that one algorithm is ahead of the other, with overwhelming probability.

In the proof of Lemma 53, we make an assumption about the current distance to the optimum at the start of a new period to simplify the analysis. In brief, we assume that we start the period at the smallest distance contained in the interval.

This assumption is in some sense pessimistic because it minimises the drift. However, we now give a rigorous argument to show that the assumption also generally decreases the distance to the optimum at the end of the period and thus is arguably better thought of as an optimistic assumption[17].

The following lemma states that, for any two distances $i < j$, the distance after t generations is generally smaller when starting close to distance i (according to a specific probability distribution), compared to when starting at distance j.

Lemma 52. Consider RLS$_k$ optimising ONEMAX*, for an arbitrary value of $k \geq 1$, during t generations, for an arbitrary value of t. For every two integers $i \leq j$ there is a probability distribution $\alpha_{i,j}$ over distances to the optimum in $[i-k+1, i]$ such that the distance to the optimum of RLS$_k$ after t generations, when initialised according to $\alpha_{i,j}$, is stochastically dominated by that of RLS$_k$ after t generations, when initialised with distance j.

Proof. If $j = i$, the statement is trivial if we take $\alpha_{i,j}$ as the point distribution at $i = j$.

Assume $i < j$ and note that RLS$_k$ has to pass through the distance interval of $I = [i-k+1, i]$ in order to progress from an initial distance j to a distance smaller than $i-k+1$. For $\ell \in I$, let p_ℓ be the probability that the first fitness reached in I is ℓ. The distance of RLS$_k$ after t generations, when initialised at distance ℓ, is stochastically dominated by the distance of RLS$_k$ after t generations, when initialised at distance j, if we condition on traversing distance ℓ as the first distance in I; this is because the former algorithm runs for

[17]A universal statement such as "starting closer to the optimum is always better" is not true. For instance, when considering RLS$_2$ running on the unmodified ONEMAX function, starting at distance 1 to the optimum means that the algorithm cannot reach the optimum as it will always flip at least one bit incorrectly. However, RLS$_2$ starting at distance 2 will eventually reach the optimum, given enough time. Hence, given enough time, the algorithm achieves a better final distance when starting further away from the optimum.

all t generations from distance ℓ and the latter algorithm runs for fewer than t generations from distance ℓ.

Note that we may have $\sum_{\ell \in I} p_\ell < 1$ since there may be a positive probability that RLS$_k$ does not reach the interval I at all. Conditional on not reaching I, the distance when starting anywhere in I will be smaller than the distance of RLS$_k$ starting at distance j, with probability 1. We therefore may construct $\alpha_{i,j}$ by first assigning $\alpha_{i,j}(\ell) := p_\ell$ and then distributing the remaining probability mass $1 - \sum_{\ell \in I} p_\ell$ arbitrarily to states in I. □

We now split the run of RLS$_k$ into periods of linear length and establish intervals $[\ell_i, u_i]$ that with overwhelming probability contain the distance to the optimum at the end of period i.

Lemma 53. Consider RLS$_k$ on OneMax* with $k = O(1)$ and a cutoff time $\kappa \geq 3.225n$. Divide the first $3.225n$ generations into 645 periods of length $n/200$ each. Define $\ell_0 = n/2 - n^{3/4}$ and $u_0 = n/2 + n^{3/4}$ and, for all $1 \leq i \leq 645$,

$$\ell_i = \ell_{i-1} - \frac{n}{200}\Delta_k(\ell_{i-1}) - o(n) \quad \text{and} \quad u_i = u_{i-1} - \frac{n}{200}\Delta_k(\ell_i) + o(n).$$

Then, with overwhelming probability at the end of period i for $0 \leq i \leq 645$, the current distance to the optimum is in the interval $[\ell_i, u_i]$ and throughout period i, $1 \leq i \leq 645$, it is in the interval $[\ell_{i-1}, u_i]$.

Proof. We prove the statement by induction. We first show that, at time 0, the current distance to the optimum is in $[n/2 - n^{3/4}, n/2 + n^{3/4}]$ with overwhelming probability. Let $X_i = 1$ if bit i is equal to 1, and let it equal 0 otherwise. Then let $X = \sum_{i=1}^{n} = X_i$ be the number of 1-bits at time 0. Since the bit string is initialised uniformly at random, $\mathrm{E}[X] = n/2$. By applying additive Chernoff bounds (Theorem 20) with $\delta = n^{3/4}$ we calculate that the number of 1-bits at initialisation is in $[n/2 - n^{3/4}, n/2 + n^{3/4}]$ with probability at least $1 - \exp(-\Omega(\sqrt{n}))$. The same applies to the distance to the optimum at this time as this is given by n minus the number of 1-bits.

Assume that at the end of period $i-1$ the current distance to the optimum is $d^*_{i-1} \in [\ell_{i-1}, u_{i-1}]$. We now derive a lower bound on the distance to the optimum at the end of period i (i.e. ℓ_i). We do so by first arguing that the assumption that $d^*_{i-1} \in [\ell_{i-1}, u_{i-1}]$ can be replaced by another assumption under which the current distance to the optimum is in the interval $[\ell_{i-1} - k + 1, \ell_{i-1}]$. According to Lemma 52, there is a probability distribution $\alpha_{\ell_{i-1}, d^*_{i-1}}$ over distances in $[\ell_{i-1} - k + 1, \ell_{i-1}]$ such that the distance to the optimum after period i when starting from $\alpha_{\ell_{i-1}, d^*_{i-1}}$ is stochastically dominated by the distance after period i when starting this period from distance d^*_{i-1}. In other words, starting in the region $[\ell_{i-1} -$

$k+1, \ell_{i-1}]$ is generally preferable to starting at distance d^*_{i-1}. Hence, in order to determine the next lower bound ℓ_i on the distance, we may optimistically assume that at the end of period $i-1$, we are at some distance in the interval $[\ell_{i-1}-k+1, \ell_{i-1}]$ and the probability that we are at each distance in this interval is given by the distribution $\alpha_{\ell_{i-1}, d^*_{i-1}}$. The precise distribution is immaterial: we will only use the fact that at the end of period $i-1$ the distance to the optimum is in the interval $[\ell_{i-1}-k+1, \ell_{i-1}]$. When bounding ℓ_i, this replaces the original assumption that the distance to the optimum is in $[\ell_{i-1}, u_{i-1}]$.

Since the current distance to the optimum can only decrease and the expected progress is increasing in the distance to the optimum, this new assumption implies that, during period i, the expected progress in each step is at most $\Delta_k(\ell_{i-1})$. We now use the method of bounded martingale differences (Theorem 22, Section 3.2.4) to bound the total progress in $n/200$ steps. Let us optimistically assume that the expected progress is always $\Delta_k(\ell_{i-1})$ throughout this period. Let $f(X_1, \ldots, X_{\frac{n}{200}})$ be a function yielding the progress over the period given the amount of progress X_j at each iteration j. Then $f = \sum_{j=1}^{n/200} X_j$ and by the linearity of expectation $\mathrm{E}[f] = n\Delta_k(\ell_{i-1})/200$. Also, $\mathrm{E}[f \mid X_1, \ldots, X_j] = \sum_{m=1}^{j} X_m + \sum_{m=j+1}^{n/200} \Delta_k(\ell_{i-1})$. Notice that, for any i, once the variable X_i is observed it subtracts a term of $\Delta_k(\ell_{i-1})$ from the expectation of f and can contribute a term of at most k. If no progress is made at time j then $X_j = 0$ and the change in the expectation of f is $-\Delta_k(\ell_{i-1})$. If progress of k is made (i.e. $X_j = k$), then the change in the expectation of f is $k - \Delta_k(\ell_{i-1})$. Thus

$$\begin{aligned} |\mathrm{E}[f \mid X_1, \ldots, X_j] - \mathrm{E}[f \mid X_1, \ldots, X_{j-1}]| &\le \max\{|-\Delta_k(\ell_{i-1})|, |k - \Delta_k(\ell_{i-1})|\} \\ &\le k =: c_j, \end{aligned}$$

since $0 \le \Delta_k(\ell_{i-1}) \le k$.

We are now ready to apply the method of bounded martingale differences. Applying said method with $\delta = (n/200)^{3/4} - k + 1$ yields

$$\begin{aligned} \Pr(f \ge n/200 \cdot \Delta_k(\ell_{i-1}) + (n/200)^{3/4} - k + 1) &\le \exp\left(-\frac{((n/200)^{3/4} - k + 1)^2}{2nk^2/200}\right) \\ &= \exp(-\Omega(\sqrt{n})). \end{aligned}$$

That is, the total progress in $n/200$ steps is thus at most $n/200 \cdot \Delta_k(\ell_{i-1}) + (n/200)^{3/4} - k + 1 = n/200 \cdot \Delta_k(\ell_{i-1}) + o(n)$ with overwhelming probability. Hence we obtain $\ell_i = \ell_{i-1} - k + 1 - \frac{n}{200}\Delta_k(\ell_{i-1}) - o(n) = \ell_{i-1} - \frac{n}{200}\Delta_k(\ell_{i-1}) - o(n)$ as a lower bound on the distance at the end of period i, with overwhelming probability.

Since the distance in period i is at least ℓ_i, the expected progress in every step is at least $\Delta_k(\ell_i)$. Again using the method of bounded martingale differences, by the same calculations as above, the progress is at least $n/200 \cdot \Delta_k(\ell_i) - o(n)$ with overwhelming probability. This establishes $u_i = u_{i-1} - n/200 \cdot \Delta_k(\ell_i) + o(n)$ as an upper bound on the distance at the end of period i. Taking the union bound over all failure probabilities proves the claim. □

Iterating the recurrent formulas from Lemma 53 demonstrates that with overwhelming probability the fitness values reached by different configurations differ by a linear amount after $3.225n$ iterations.

Lemma 54. Define $\ell_{i,k}$ as the value ℓ_i in Lemma 53 that corresponds to RLS$_k$. Define $u_{i,k}$ similarly. After $3.225n$ steps, w. o. p. RLS$_1$ is ahead of RLS$_2$ and RLS$_3$ by a linear distance: $u_{645,1} \leq \ell_{645,2} - \Omega(n)$ and $u_{645,1} \leq \ell_{645,3} - \Omega(n)$ respectively. Furthermore, w. o. p. RLS$_3$ is ahead of RLS$_4$ and RLS$_5$ by a linear distance: $u_{645,3} \leq \ell_{645,4} - \Omega(n)$ and $u_{645,3} \leq \ell_{645,5} - \Omega(n)$ respectively. And w. o. p. the distance to the optimum is at most $0.13n$ for RLS$_1$, RLS$_3$ and RLS$_5$.

In order to prove Lemma 54, however, we must first show the following result.

Lemma 55. Define $\ell_{i,k}$ and $u_{i,k}$ as in Lemma 54. Then $\ell_{i,2} \geq \ell_{i,3}$ as well as $\ell_{i,4} \geq \ell_{i,5}$ and

$$\begin{aligned}
u_{i,1} &= u_{i-1,1} - \frac{\ell_{i,1}}{200} + o(n)\\
\ell_{i,3} &\geq \ell_{i-1,3} - \frac{3\ell_{i-1,3}^2}{200n} - o(n)\\
u_{i,3} &\leq u_{i-1,3} - \frac{3\ell_{i,3}^2}{200n} + o(n)\\
\ell_{i,5} &\geq \ell_{i-1,5} - \frac{10\ell_{i-1,5}^3}{200n^2} - o(n).
\end{aligned}$$

Proof. The inequalities $\ell_{i,2} \geq \ell_{i,3}$ and $\ell_{i,4} \geq \ell_{i,5}$ follow from the fact that for even k, $\Delta_k(s) \leq \Delta_{k+1}(s)$ for all distances s [36, Lemma 28].

The other results essentially follow from Lemma 53 along with the drift bounds from Lemma 51. The equality for $u_{i,1}$ follows immediately from $\Delta_1(\ell_{i-1,1}) = \ell_{i-1,1}/n$. The lower bound for $\ell_{i,3}$ follows from $\Delta_3(\ell_{i-1,3}) \leq \frac{3\ell_{i-1,3}^2}{n^2}$ and, likewise, the lower bound for $\ell_{i,5}$ follows from $\Delta_5(\ell_{i-1,5}) \leq \frac{10\ell_{i-1,5}^3}{n^3}$. The upper bound for $u_{i,3}$ follows from $\Delta_3(\ell_{i,3}) = \frac{3\ell_{i,3}(\ell_{i,3}-1)}{n(n-1)} \geq \frac{3\ell_{i,3}^2}{n^2} - O(1/n)$. Along with a factor of $n/200$, the term $-O(1/n)$ leads to an error term of $-O(1)$ that is absorbed in the $-o(n)$ term. □

We can now prove Lemma 54.

Proof of Lemma 54. We first argue that it is safe to focus on the leading constants in the recurrences given in Lemma 55, i.e. that the terms of $o(n)$ can essentially be neglected. Since the drift $\Delta_k(s)$ is increasing in s, we have $\Delta_k(s-o(n)) \leq \Delta_k(s)$ and thus any negative small order terms in $\ell_{i,1}/200$, $3\ell_{i,3}/(200n)$, and $10\ell_{i,5}/(200n)$ can be ignored since the lower bounds on the distance obtained by ignoring the negative small order terms will be smaller (i.e. looser) than those which could be obtained by considering them. Every application of a recurrence formula from Lemma 55 subtracts another term of $-o(n)$. However, since we only consider a constant number of applications, the total error term is still $-o(n)$.

For the upper bounds, it is also not hard to show that $\Delta_k(s+o(n)) \leq \Delta_k(s)+o(1)$ for $k \in \{1,3,5\}$, which introduces an additional $+o(n)$ term in each application of a recurrence. By the previous arguments, the total error in a constant number of applications sums up to $+o(n)$.

This implies that, modulo small order terms, the distance to the optimum in any period can be bounded by considering the leading constants $c_{\ell,i,k}$ in $\ell_{i,k}$ and $c_{u,i,k}$ in $u_{i,k}$, when taking the inequalities as equalities. Then $c_{\ell,0,k} = c_{u,0,k} = 1/2$ for all k and $c_{u,i,1} = c_{u,i-1,1} - c_{\ell,i,1}/200; c_{\ell,i,3} = c_{\ell,i-1,3} - 3c_{\ell,i-1,3}^2/200; c_{\ell,i,5} = c_{\ell,i-1,5} - 10c_{\ell,i-1,5}^3/200$; $c_{u,i,3} = c_{u,i-1,3} - 3c_{\ell,i,3}^2/200$.

We solved these recurrences numerically[18]. After 645 periods of length $n/200$ (i.e. after $3.225n$ iterations), we observe that all distance intervals are non-overlapping and are in what we would assume will be their final ordering (i.e. RLS$_1$ is the closest to the optimum, followed by RLS$_3$, then RLS$_5$: see Figure 5.1 for a plot of these intervals, including ones not relevant until the proof of Theorem 63). We show that this is indeed the final ordering in the proof of Lemma 57. The resulting leading constants were (we also show $c_{\ell,645,1}$ and $c_{u,645,5}$ defined similarly, though we do not need them):

$$\begin{aligned}
[c_{\ell,645,1}, c_{u,645,1}] &= [0.019717738, 0.022119149]\\
[c_{\ell,645,3}, c_{u,645,3}] &= [0.085458797, 0.089099249]\\
[c_{\ell,645,5}, c_{u,645,5}] &= [0.120636109, 0.126798327].
\end{aligned}$$

Noticing that these intervals are non-overlapping, with gaps of order $\Omega(1)$, implies the claim for the stated comparisons of bounds for RLS$_1$, RLS$_3$, and RLS$_5$, even when taking into account error terms of $o(n)$. The claims for RLS$_2$ and RLS$_4$ follow immediately from these results along with Lemma 55.

[18]The code we used to do so and the data generated is available at `https://george-hall-sheff.github.io/rlsk_om_recurrences`.

The additional statement about the distance being at most $0.13n$ follows since all $c_{u,645,k}$ values are less than $0.13 - \Omega(1)$. □

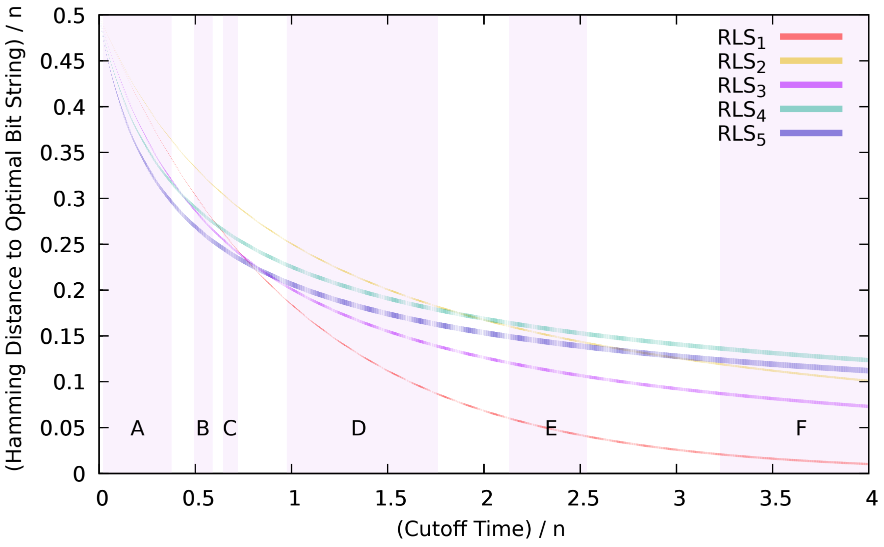


Fig. 5.1 Intervals within which the fitness of the individual in RLS$_k$, with $1 \le k \le 5$, is contained w. o. p. Calculated using periods of length $n/200$. The distance intervals for each configuration at the end of period i for $0 \le i \le 800$ (corresponding to linear cutoff times $\kappa \le 4n$) are displayed (note that each curve consists of 801 vertical lines indicating the interval at the end of each considered period).

We point out that the techniques from the proof of Lemma 54 yield closed-form bounds that with overwhelming probability contain the distance to the optimal bit string for RLS$_1$ after a linear number of iterations. These statements are not used further in this thesis, but we nevertheless state them here since they significantly improve on the state-of-the-art fixed-budget bounds for RLS$_1$ optimising ONEMAX (reviewed in Section 3.2.8).

Corollary 56. For constant $i \ge 0$, RLS$_1$ after $\kappa = (n \cdot i)/200$ iterations running on ONEMAX has Hamming distance s_κ to the optimal bit string that satisfies

$$\frac{n}{2}\left(\frac{199}{200}\right)^{i} \le s_\kappa \le n \cdot \left(\frac{1}{2}\left(\frac{199}{200}\right)^{i+1} + \frac{1}{400}\right),$$

with overwhelming probability. This implies that

$$\frac{n}{2}\left(\frac{199}{200}\right)^{200\kappa/n} \le s_\kappa \le n \cdot \left(\frac{1}{2}\left(\frac{199}{200}\right)^{(200\kappa/n)+1} + \frac{1}{400}\right),$$

with overwhelming probability.

Proof. As analysed in the proof of Lemma 54, the upper and lower bounds on the coefficient of the linear term in the distance to the optimal bit string after i periods of $n/200$ iterations are $c_{\ell,i,1} = c_{\ell,i-1,1} - c_{\ell,i-1,1}/200$ and $c_{u,i,1} = c_{u,i-1,1} - c_{\ell,i,1}/200$, respectively. Recall that $c_{\ell,0} = c_{u,0} = 1/2$. These recurrence relations have solutions

$$c_{\ell,i,1} = \frac{1}{2}\left(\frac{199}{200}\right)^{i}$$

and

$$c_{u,i,1} = \frac{1}{2}\left(\frac{199}{200}\right)^{i+1} + \frac{1}{400}.$$

The first claims follow by multiplying these solutions by n, and the second follows by replacing i with $200\kappa/n$, the number of iterations that corresponds to the end of period i. □

Lemma 54 states that, for $\kappa = 3.225n$, smaller odd parameter values win comparisons using the Best-Fitness metric with overwhelming probability. The following lemma proves that this is in fact the case for all cutoff times $\kappa \geq 3.225n$.

Lemma 57. Let (a,b) be a member of the set $\{(1,2),(1,3),(3,4),(3,5)\}$, let $\kappa \geq 3.225n$, and let RLS$_a$ and RLS$_b$ both be run on ONEMAX* for κ iterations. Then with overwhelming probability RLS$_a$ either has a higher fitness than RLS$_b$ or, if both algorithms have reached the optimum, RLS$_a$ did so first.

Proof. Lemma 54 proves the claim for a cutoff time of $\kappa = 3.225n$. For cutoff times larger than $3.225n$, it is possible for the algorithms that lag behind to catch up after time $3.225n$. To this end, we define the distance between two algorithms RLS$_a$ and RLS$_b$ (with $a < b$) as $D_t^{a,b} := s_{t,b} - s_{t,a}$, where $s_{t,a}$ and $s_{t,b}$ refer to the respective distances to the optimum at time t. Initially, by Lemma 54, we have $D_t^{a,b} = \Omega(n)$ for all considered algorithm pairs. We will apply the negative drift theorem for self-loops (Theorem 24, page 69) to show that with overwhelming probability $D_t^{a,b}$ does not drop to 0 until RLS$_a$ has found an optimum ($s_{t,a} < a$).

Consider the situation where $D_t^{a,b}$ has decreased to a value of at most $n^{1/4}$. We then argue that

$$\mathrm{E}[D_{t+1}^{a,b} - D_t^{a,b} \mid 0 \leq D_t^{a,b} \leq n^{1/4}, s_{t,a} \geq a, s_{t,b}] = \Omega(\Delta_a(s_{t,a})).$$

For RLS$_1$ and RLS$_3$ the above expectation is at least (using Lemma 51 and $s_{t,1} \leq 0.13n$)

$$\begin{aligned}
\Delta_1(s_{t,1}) - \Delta_3(s_{t,3}) &\geq \frac{s_{t,1}}{n} - \frac{3(s_{t,1}+n^{1/4})^2}{n^2} \\
= \frac{s_{t,1}}{n}\left(1 - \frac{3s_{t,1}}{n} - o(1)\right) &\geq \frac{s_{t,1}}{n}(1 - 3\cdot 0.13 - o(1)) = \Omega(\Delta_1(s_{t,1})).
\end{aligned}$$

For RLS$_3$ and RLS$_5$ the above expectation is at least (using Lemma 51 and $s_{t,3} \leq 0.13n$)

$$\begin{aligned}
\Delta_3(s_{t,3}) - \Delta_5(s_{t,5}) &\geq \frac{3s_{t,3}(s_{t,3}-1)}{n(n-1)} - \frac{10(s_{t,3}+n^{1/4})^3}{n^3} \\
&= \frac{3s_{t,3}(s_{t,3}-1)}{n(n-1)} - \frac{3s_{t,3}^2}{n^2}\left(\frac{10s_{t,3}}{3n} + o(1)\right) \\
&= \Omega(\Delta_3(s_{t,3})).
\end{aligned}$$

The statement also follows for even b as $\Delta_b(s) < \Delta_{b+1}(s)$.

We also have $\Delta_k(s)/k \leq \Pr(s_{t+1,k} < s_{t,k}) \leq \Delta_k(s)$ for all k, s. The above calculations have further established $\Delta_b(s_{t,b}) = O(\Delta_a(s_{t,a}))$. Hence $\Pr(D_{t+1}^{a,b} \neq D_t^{a,b}) = \Theta(\Delta_a(s_{t,a}))$.

This implies that the first condition of the negative drift theorem with self-loops is satisfied with respect to $D_t^{a,b}$ and the interval $[0, n^{1/4}]$. The second condition is trivial as the jump length is bounded by $b = O(1)$. Applying said theorem yields that the probability of RLS$_b$ catching up to RLS$_a$ before RLS$_a$ finds an optimum in $2^{\Omega(n^{1/4})}$ generations is $e^{-\Omega(n^{1/4})}$. By Markov's inequality, the probability that RLS$_a$ has not found an optimum within this time is $\Theta(n \log n) \cdot 2^{-\Omega(n^{1/4})} = e^{-\Omega(n^{1/4})}$. Summing up all failure probabilities proves the claim. □

Lemma 57 implies that with overwhelming probability RLS$_1$ has a smaller optimisation time than any rival configuration and RLS$_3$ has a smaller optimisation time than RLS$_4$ and RLS$_5$. We prove this in the following corollary.

Corollary 58. The following statements hold when optimising ONEMAX*:

- RLS$_3$ has a smaller optimisation time than RLS$_4$ and RLS$_5$, with overwhelming probability.
- RLS$_1$ has a smaller optimisation time than RLS$_k$ with $k \in \{2, 3, 4, 5\}$, with overwhelming probability.

Proof. We prove the claim for RLS$_3$ in a comparison against RLS$_5$. The same technique can be used to prove the result for RLS$_3$ vs. RLS$_4$, RLS$_1$ vs. RLS$_2$, and RLS$_1$ vs. RLS$_3$. The remaining claims for RLS$_1$ hold by transitivity.

By Lemma 57, for cutoff times $\kappa \geq 3.225n$, RLS$_3$ beats RLS$_5$ in a comparison in ParamRLS-F w. o. p. Recall that, in ParamRLS-F, RLS$_a$ beats RLS$_b$ in a comparison if and only if: (1) RLS$_a$ has a higher fitness than RLS$_b$ at the cutoff time; or (2) both configurations have the same fitness at the cutoff time, but RLS$_a$ reached this fitness first.

Let τ_3 and τ_5 be the expected optimisation times of RLS$_3$ and RLS$_5$, respectively, and let $\tau = \max\{\tau_3, \tau_5\}$. Then, by Markov's inequality, both algorithms have reached an optimum within $e^n \cdot \tau$ iterations, w. o. p. We can therefore apply Lemma 57 since $\kappa \geq 3.225n$. Lemma 57 states that, if both configurations have reached an optimum (and therefore the same fitness) w. o. p., then RLS$_3$ did so first, w. o. p. Hence, in a run with $\kappa = e^n \cdot \tau$, RLS$_3$ takes less time than RLS$_5$ to reach an optimum, w. o. p. Taking a union bound over all exponentially small failure probabilities proves the claim.

Proving the claim for a run with $\kappa = e^n \cdot \tau$ implies the claim for runs with any other κ. □

From Lemma 57 it follows that with overwhelming probability configurations with smaller odd values of k will beat their opponents in a ParamRLS-F comparison with cutoff time $\kappa \geq 3.225n$. We now use the techniques introduced in its proof to extend the analysis to ParamRLS-F comparisons with cutoff times $\kappa \geq 0.02n$. The following lemma proves that, for ParamRLS-F using cutoff times in this range, the parameter landscape is not unimodal (unlike when configuring RIDGE*) but there is nevertheless an underlying gradient towards the F-optimal configuration.

Lemma 59. For ParamRLS-F for the configuration of RLS$_k$ for ONEMAX*, with cutoff time $\kappa \geq 0.02n$, the parameter landscape has the structure given in Table 5.1.

Proof. We use a similar approach to that used in the proof of Lemma 54. We again use periods of length $n/200$ to determine the cutoff times at which the fitness of the individuals in different configurations of RLS$_k$ are distinct by a linear amount. In addition to the quantities

Region	Cutoff Time	Ordering of Configurations
A	$\kappa \in [0.020n, 0.375n]$	RLS$_5$ > RLS$_4$ > RLS$_3$ > RLS$_1$ > RLS$_2$
	$\kappa \in (0.375n, 0.495n)$	RLS$_5$ > {RLS$_3$, RLS$_4$} > RLS$_1$ > RLS$_2$
B	$\kappa \in [0.495n, 0.590n]$	RLS$_5$ > RLS$_3$ > RLS$_4$ > RLS$_1$ > RLS$_2$
	$\kappa \in (0.590n, 0.645n)$	RLS$_5$ > RLS$_3$ > {RLS$_1$, RLS$_4$} > RLS$_2$
C	$\kappa \in [0.645n, 0.720n]$	RLS$_5$ > RLS$_3$ > RLS$_1$ > RLS$_4$ > RLS$_2$
	$\kappa \in (0.720n, 0.975n)$	{RLS$_1$, RLS$_3$, RLS$_5$} > RLS$_4$ > RLS$_2$
D	$\kappa \in [0.975n, 1.760n]$	RLS$_1$ > RLS$_3$ > RLS$_5$ > RLS$_4$ > RLS$_2$
	$\kappa \in (1.760n, 2.130n)$	RLS$_1$ > RLS$_3$ > RLS$_5$ > {RLS$_2$, RLS$_4$}
E	$\kappa \in [2.130n, 2.535n]$	RLS$_1$ > RLS$_3$ > RLS$_5$ > RLS$_2$ > RLS$_4$
	$\kappa \in (2.535n, 3.225n)$	RLS$_1$ > RLS$_3$ > {RLS$_2$, RLS$_5$} > RLS$_4$
F	$\kappa \geq 3.225n$	RLS$_1$ > RLS$_3$ > RLS$_2$ > RLS$_5$ > RLS$_4$

Table 5.1 Ordering of configurations for all cutoff times $\kappa \geq 0.02n$. "RLS$_a$ > RLS$_b$" indicates that RLS$_a$ has a higher fitness than RLS$_b$ at the cutoff time w. o. p. and "{RLS$_a$, RLS$_b$}" indicates that we cannot draw any conclusions about which configuration will have the higher fitness at this cutoff time. The region names correspond to Figure 5.1.

defined in the proof of Lemma 54, we similarly define

$$\begin{aligned}
\ell_{i,1} =& \ell_{i-1,1} - \frac{\ell_{i-1,1}}{200} + o(n), \\
\ell_{i,2} \geq& \ell_{i-1,2} - \frac{2\ell_{i-1,2}^2}{200n} - o(n), \\
u_{i,2} \leq& u_{i-1,2} - \frac{2\ell_{i,2}^2}{200n} + o(n), \\
\ell_{i,4} \geq& \ell_{i-1,4} - \frac{8\ell_{i-1,4}^3}{200n^2} - o(n), \\
u_{i,4} \leq& u_{i-1,4} - \frac{8\ell_{i,4}^3}{200n^2} - o(n), \\
u_{i,5} \leq& u_{i-1,5} - \frac{10\ell_{i,5}^3}{200n^2} - o(n).
\end{aligned}$$

We derive that the coefficients of their $\Theta(n)$ terms are $c_{\ell,i,1} = c_{\ell,i-1,1} - c_{\ell,i-1,1}/200$, $c_{\ell,i,2} = c_{\ell,i-1,2} - 2c_{\ell,i-1,2}^2/200$, $c_{u,i,2} = c_{u,i-1,2} - 2c_{\ell,i,2}^2/200$, $c_{\ell,i,4} = c_{\ell,i-1,4} - 8c_{\ell,i-1,4}^3/200$, $c_{u,i,4} = c_{u,i-1,4} - 8c_{\ell,i,4}^3/200$, $c_{u,i,5} = c_{u,i-1,5} - 10c_{\ell,i,5}^3/200$.

Iterating these recurrences in the same way as in the proof of Lemma 54, we observe that, for the named ranges of cutoff times in Table 5.1, the configurations are ordered in the stated way (we illustrate these intervals in Figure 5.1 on page 117). Consider, for instance, the range of cutoff times $\kappa \in [0.975n, 1.760n]$ (we only prove the claim for this range of cutoff times: the claim for other ranges of cutoff times names in Table 5.1 can be proved in the same way). We prove that $\text{RLS}_1 > \text{RLS}_3 > \text{RLS}_5 > \text{RLS}_4 > \text{RLS}_2$ for all cutoff times in this range (where "$\text{RLS}_a > \text{RLS}_b$" indicates that RLS_a has a higher fitness than RLS_b w. o. p.) by recalling that by definition (Lemma 53) the interval $[\ell_{i-1,k}, u_{i,k}]$ w. o. p. contains the distance to the optimum of the individual in RLS_k throughout period i and observing that, for all periods i with $195 \leq i \leq 352$ we have $c_{\ell,i-1,1} < c_{u,i,1} < c_{\ell,i-1,3} < c_{u,i,3} < c_{\ell,i-1,5} < c_{u,i,5} < c_{\ell,i-1,4} < c_{u,i,4} < c_{\ell,i-1,2} < c_{u,i,2}$.

For cutoff times $\kappa \geq 3.225n$, the result follows from Lemma 57. For the unnamed ranges in Table 5.1 it is the case that the distance intervals for almost all configurations are non-overlapping, but they are overlapping for the sets of configurations for which no ordering is stated. □

5.3.2 Optimisation-Time Requires Superlinear Cutoff Times

According to Corollary 37, any configurator that uses the Optimisation-Time performance metric is blind for any cutoff time that is at most $(n \ln n)/2$ when configuring RLS_k for OneMax*. For larger cutoff times we now prove that ParamRLS-T and ParamILS are able to identify the O-optimal neighbourhood size (i.e. $k = 1$) for RLS_k for OneMax*. To do so, we first prove that RLS_1 has reached the optimum of OneMax* within $n^{1+\varepsilon}$ iterations, for any positive constant ε, with overwhelming probability.

Lemma 60. For every initial search point, RLS_1 reaches the optimum of OneMax* within $n^{1+\varepsilon}$ iterations, for any positive constant ε, with probability at least $1 - n\exp(-n^{\varepsilon})$.

Proof. A sufficient condition for RLS_1 having found the optimum is that every bit has been mutated at least once. The probability that a fixed bit i is not mutated in $n^{1+\varepsilon}$ steps is $(1-1/n)^{n^{1+\varepsilon}} \leq \exp(-n^{\varepsilon})$. By the union bound, the probability that there is a bit that has not been mutated in $n^{1+\varepsilon}$ steps is at most $n \cdot \exp(-n^{\varepsilon})$. □

We now use this result to show that ParamRLS-T is able to configure RLS_k for OneMax* given sufficiently large cutoff times.

Theorem 61. Consider ParamRLS-T for the configuration of RLS_k for OneMax* with $\phi = 5$. Assume that it uses cutoff time $\kappa \geq n^{1+\varepsilon}$ for any positive constant ε, a single run per configuration evaluation (i.e. $r = 1$), and that it uses local search operator $\pm\{1,2\}$. Then the expected number of comparisons T before ParamRLS-T sets the active parameter to $k = 1$ for the first time is at most $32 + 2^{-\Omega(n^{\varepsilon})}$ (for some constant $\varepsilon > 0$). After $n^{\varepsilon'}$ comparisons, for some constant $\varepsilon' > 0$, ParamRLS-T returns the parameter value $k = 1$ with overwhelming probability.

Proof. By Corollary 58, RLS$_1$ has a lower optimisation time than RLS$_k$, with $k \in \{2,3,4,5\}$, w. o. p., and RLS$_3$ has a lower optimisation time than RLS$_4$ and RLS$_5$, w. o. p.

Lemma 60 implies that RLS$_1$ reaches the optimum within $n^{1+\varepsilon}$ iterations, with overwhelming probability. At any point, the active parameter can reach $k = 1$ within two comparisons: in the worst case it is at $k = 5$ and can move to $k = 3$, which happens with probability at least $1/8 - 2^{-\Omega(n^{\varepsilon})}$ (the probability is larger than this lower bound if the cutoff time is large enough to allow RLS$_3$ to reach the optimum) and then to move to $k = 1$, which happens with probability $1/4 - 2^{-\Omega(n^{\varepsilon})}$. The probability that in two consecutive comparisons the active parameter reaches $k = 1$ is hence $1/32 - 2^{-\Omega(n^{\varepsilon})}$. Therefore the active parameter is set to $k = 1$ within $32 + 2^{-\Omega(n^{\varepsilon})}$ comparisons, in expectation.

By Markov's inequality we have that the probability that the active parameter has not been set to $k = 1$ after 64 comparisons is at most $1/2 + 2^{-\Omega(n^{\varepsilon})}$. Hence the probability that it has not been set to $k = 1$ after $64n^{\varepsilon'}$ comparisons is at most $2^{-\Omega(n^{\varepsilon'})}$, for some positive constant ε'. Combining this with the fact that, in a single run, by Lemma 57, RLS$_1$ beats both RLS$_2$ and RLS$_3$ with overwhelming probability, a union bound over the polynomially many comparisons proves the claim. □

Similar arguments allow us to prove a similar result for ParamILS.

Theorem 62. Consider ParamILS for the configuration of RLS$_k$ for ONEMAX*, with $k \in \{1,\ldots,5\}$, using the Optimisation-Time performance metric, cutoff time $\kappa \geq n^{1+\varepsilon}$ for any positive constant ε, and arbitrary values for s, R, and p_{restart}. Then after $R+4$ comparisons, the configuration $k = 1$ is returned by ParamILS with overwhelming probability.

Proof. As shown in the proof of Theorem 61, if $\kappa \geq n^{1+\varepsilon}$ then, with overwhelming probability, RLS$_1$ reaches the optimum of ONEMAX* before any other RLS$_k$ with $2 \leq k \leq 5$. Then the result follows from the general upper bound in Corollary 39. □

5.3.3 ParamRLS-F Identifies the Optimal Neighbourhood Size with Arbitrary Cutoff Times

In Section 5.3.2, we proved that any configurator that uses the Optimisation-Time performance metric requires superlinear cutoff times to configure RLS$_k$ for ONEMAX*. In this section we will show that, for cutoff times that are at least linear, ParamRLS-F is able to identify the neighbourhood size that achieves the highest solution quality within the given time budget (i.e. the F-optimal configuration) and for large enough (but still linear) cutoff times it is able to return the O-optimal configuration. In particular, for cutoff times satisfying $0.02n \leq \kappa \leq 0.72n$ ParamRLS-F identifies that the F-optimal neighbourhood size is

$k = 5$, whilst cutoff times of $\kappa \geq 0.975n$ suffice for it to return the O-optimal and F-optimal configuration $k = 1$.

We first derive an upper bound on the expected number of comparisons before the tuner first sets the active parameter to $k = 1$ for cutoff times $\kappa \geq 3.225n$. We also derive the number of comparisons that are sufficient for it to return $k = 1$ with overwhelming probability.

Theorem 63. Consider ParamRLS-F for the configuration of RLS$_k$ for ONEMAX* with $\phi = 5$. Assume that it uses cutoff time $\kappa \geq 0.975n$, a single run per configuration evaluation (i.e. $r = 1$), and that it uses the local search operator $\pm\{1,2\}$. Then the expected number of comparisons T before ParamRLS-F sets the active parameter to $k = 1$ for the first time is at most $16 + 2^{-\Omega(n^{\varepsilon})}$ (for some constant $\varepsilon > 0$). After $n^{\varepsilon'}$ comparisons, for some constant $\varepsilon' > 0$, ParamRLS-F returns the parameter value $k = 1$ with overwhelming probability.

Proof. By Lemma 59, for all cutoff times $\kappa \geq 0.975n$, no configuration is more than two configurations away from one that with overwhelming probability will beat it in a ParamRLS-F comparison. This implies that the $\pm\{1,2\}$ operator is sufficient to escape the local optima caused by even values of k for some ranges of cutoff times.

In the gaps between the named ranges of cutoff times given in Table 5.1, exactly one pair of distance intervals is overlapping (i.e. where the outcome of a comparison between them is ambiguous), meaning that the number of comparisons required to locate $k = 1$ will be no larger than that required before or after the region with the overlap. For all cutoff times $\kappa \geq 0.975n$, the interval corresponding to RLS$_1$ is non-overlapping with all others and is smaller than that corresponding to all other considered configurations (i.e. RLS$_1$ has a higher fitness than by a distance of $\Omega(n)$, w. o. p.). Since for each ambiguity-free ordering of configurations given in the table the $\pm\{1,2\}$ operator is sufficient to reach RLS$_1$, we can conclude that it is also sufficient at all points during the gap.

Since the $\pm\{1,2\}$ operator is sufficient to reach RLS$_1$ for all cutoff times $\kappa \geq 0.975n$, the claims follow using the same arguments as in the proof of Theorem 61. The bounds derived here are half as large as the values derived in that proof since, in this case, the configuration closer to the optimum wins a comparison w. o. p., whereas in the proof of Theorem 61 it is assumed that this only occurs with probability at least $1/2 - 2^{-\Omega(n^{\varepsilon})}$. □

We now prove that, for cutoff times $0.02n \leq \kappa \leq 0.72n$, ParamRLS-F returns the configuration $k = 5$, which is F-optimal.

Theorem 64. Consider ParamRLS-F for the configuration of RLS$_k$ for ONEMAX* with $\phi = 5$. Assume that ParamRLS-F uses cutoff time $0.02n \leq \kappa \leq 0.72n$, a single run per configuration evaluation (i.e. $r = 1$), and that it uses the local search operator $\pm\{1,2\}$. Then the expected number of comparisons T before ParamRLS-F sets the active parameter to $k = 5$ for the first time is at most $16 + 2^{-\Omega(n^{\varepsilon})}$. After n^{ε} comparisons, for some constant $\varepsilon > 0$, the tuner returns the parameter $k = 5$ with overwhelming probability.

Proof. As in the proof of Theorem 63, we observe that for $0.02n \leq \kappa \leq 0.72n$ no configuration is more than two away from one that, with overwhelming probability, will beat it in a ParamRLS-F comparison. This implies that, for all cutoff times in these ranges, the $\pm\{1,2\}$ operator is again sufficient to reach the F-optimal parameter value of $k = 5$.

In the gaps between the named ranges of cutoff times given in Table 5.1 it is again the case that exactly one pair of distance intervals is overlapping. However, for all cutoff times satisfying $0.02n \leq \kappa \leq 0.72n$ the interval corresponding to RLS$_5$ is non-overlapping with all others and is smaller than that corresponding to all other considered configurations (i.e. RLS$_5$ has a higher fitness than all other considered configurations by a distance of $\Omega(n)$). Since in each named range of cutoff times given in Table 5.1 the $\pm\{1,2\}$ operator is sufficient to reach $k = 5$, we can conclude that it is also sufficient at all points during the gap.

The bounds on the number of comparisons required by the configurator to set the active parameter to $k = 5$ for the first time therefore hold by the same reasoning as in the proof of Theorem 63. □

We conclude this section by pointing out that the above analyses reveal that for all cutoff times $\kappa \geq 0.02n$ either RLS$_1$ or RLS$_5$ (or both) identify higher fitness values than RLS$_3$ with overwhelming probability (RLS$_2$ and RLS$_4$ are similarly outperformed, but this is expected). This can be observed in Figure 5.1 for almost all linear cutoff times $\kappa \geq 0.02n$. However, there is one region of ambiguity (for cutoff times $\kappa \in [0.72n, 0.975n]$) in which the fitness interval for RLS$_3$ is not distinct from those corresponding to RLS$_1$ and RLS$_5$. We resolve this ambiguity using a period length of $n/2000$ (instead of $n/200$ as otherwise used in this section), and observe that the fitness interval for RLS$_3$ becomes distinct for all cutoff times in this range (see Figure 5.2). Therefore, for all cutoff times $\kappa \geq 0.02n$, either $k = 1$ or $k = 5$ is F-optimal, but never $k = 3$.

Whilst, for a range of fitness values, RLS$_3$ has a higher drift than both of these configurations, it is too far behind RLS$_5$ when entering this region of the search space (failing to overtake it before leaving the region) and not far enough ahead of RLS$_1$ when leaving this region (being overtaken before taking advantage of its momentarily higher drift) to become the F-optimal configuration at any point. It is unlikely that using $k = 3$ is F-optimal for smaller cutoff times, however we do not prove such a result.

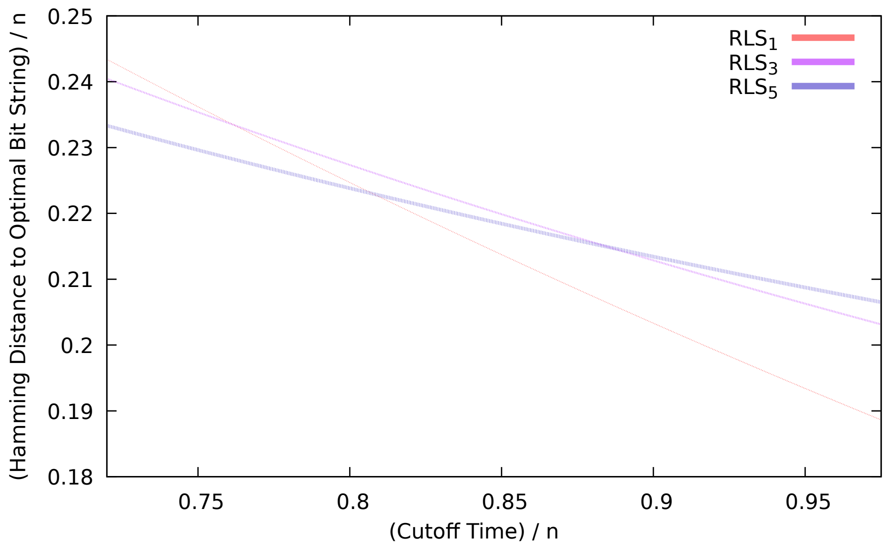


Fig. 5.2 Intervals that w. o. p. contain the distance to the optimum of the individual in RLS$_k$ ($k \in \{1,3,5\}$) for cutoff times $0.72n\kappa \leq 0.975n$, calculated using periods of length $n/2000$.

5.4 Conclusions

In this chapter, we have highlighted the superiority of the Best-Fitness performance metric over Optimisation-Time for simple configuration scenarios. In particular, we have presented an analysis of the impact of the cutoff time on the ability of algorithm configurators to configure the neighbourhood size of randomised local search for the RIDGE* and ONEMAX* problem classes. When using the Optimisation-Time performance metric, a quadratic cutoff time is required to configure RLS$_k$ for the RIDGE* problem class and for the ONEMAX* problem class a cutoff time of $\Omega(n \ln n)$ is required.

Matters are considerably different if the Best-Fitness performance metric is used instead. In that case, the cutoff time only slightly impacts the performance of ParamRLS. When configuring RLS$_k$ for RIDGE* using any cutoff time, a quadratic number of comparisons suffice in expectation for ParamRLS-F to return $k = 1$, provided that sufficiently many runs per evaluation are used. For cutoff times of $\kappa = \omega(n)$, a single run per evaluation suffices, whilst in the extreme case of $\kappa = 1$, $n^{3/2}$ runs suffice. Note that $n^{3/2}$ runs, each lasting a single iteration, is still considerably less time than the quadratic time required in expectation for any configuration to identify the optimum of RIDGE*. These results imply that ParamRLS-F is able to return the O-optimal configuration using any cutoff time, rather than requiring a quadratic one, as is the case for any Optimisation-Time-based configurator. Since $k = 1$ is F-optimal for any cutoff time, ParamRLS-F also successfully configures RLS$_k$ for RIDGE* with respect to the Best-Fitness performance metric using any cutoff time.

For OneMax* (with k taking values in $\{1,2,3,4,5\}$), ParamRLS-F returns the F-optimal configuration $k = 5$ for cutoff times satisfying $0.02n \leq \kappa \leq 0.72n$, whilst for cutoff times $\kappa \geq 0.975n$, it returns the F-optimal parameter value $k = 1$. In both cases, 17 comparisons suffice, in expectation, for it to do so. Since $k = 1$ is the O-optimal parameter value, ParamRLS-F is able to configure RLS$_k$ with respect to the Optimisation-Time performance metric using cutoff times that are by a logarithmic factor smaller than those required by any configurator that actually uses this metric.

Chapter 6

On the Configuration of the Mutation Rate of a Simple Evolutionary Algorithm

6.1 Introduction

In this chapter, we focus on a more complex algorithm configuration scenario than the one analysed in Chapter 5. In particular, we consider the problem of tuning the mutation rate χ of a simple evolutionary algorithm called the (1+1)$_\chi$ EA that uses a global mutation operator (Section 2.2, Algorithm 15). This configuration scenario is considerably more complex than the configuration of the neighbourhood size of RLS$_k$ analysed in the previous chapter since χ may take any real value whereas the neighbourhood size of RLS$_k$ is discrete. In each mutation, any number of bits may be flipped by the operator. Small differences in the value of χ (e. g. $\chi = 1$ vs. $\chi = 1.1$) are hardly visible as in most mutations the number of flipped bits is identical. In stark contrast, RLS$_k$ behaves very differently when always flipping, say, one bit compared to always flipping two bits. Hence tuning the mutation rate is much more challenging than configuring the neighbourhood size of RLS$_k$.

We analyse the configuration of the (1+1)$_\chi$ EA for the RIDGE and LEADINGONES benchmark problem classes. As in Chapter 5, we want to characterise the impact of the cutoff time on the ability of algorithm configurators to identify the optimal parameter value of the (1+1)$_\chi$ EA (with respect to the performance metric) for the considered problem classes, in addition to the time required to do so.

Our aim is to show that also in this more sophisticated algorithm configuration scenario the impact of the cutoff time is highly dependent on the choice of performance metric. In particular, we will show that the insights from the analysis of Chapter 5 also apply here: for large ranges of cutoff times, the Best-Fitness performance metric allows the configurator to return a configuration that is F-optimal and, for cutoff times that are smaller than the time required by *any* configuration to reach the optimum, to return a configuration that is O-optimal. Again, for cutoff times smaller than the expected optimisation time of the O-optimal configuration, all configurators using the Optimisation-Time performance metric are blind.

We prove that, when configuring the (1+1)$_\chi$ EA for RIDGE, any configurator using the Optimisation-Time performance metric is blind when using a cutoff time of $\kappa \leq (1-\varepsilon)en^2$. Matters change considerably for algorithm configurators that use the Best-Fitness performance metric. ParamRLS-F efficiently returns the F-optimal (and O-optimal) parameter value of $\chi = 1$ for the (1+1)$_\chi$ EA optimising RIDGE for any cutoff time that is at least linear in the problem size. This implies that ParamRLS-F can return the O-optimal configuration for cutoff times that are a linear factor smaller than the expected optimisation time of the (1+1)$_\chi$ EA with the O-optimal $\chi = 1$ mutation rate.

When configuring the (1+1)$_\chi$ EA for LEADINGONES, we prove that any configurator that uses the Optimisation-Time performance metric is blind for all $\kappa \leq 0.772075n^2$. On the other hand, ParamRLS-F is able to return the O-optimal parameter value $\chi = 1.6$ (where χ is allowed to take values from the set $\{0.1, 0.2, \ldots, 2.9, 3.0\}$) with overwhelming probability using any cutoff time $\kappa \geq 0.721118n^2$. Note that this is $\approx 0.05n^2$ smaller than the expected optimisation time of the (1+1)$_\chi$ EA for LEADINGONES using any configuration (including $\chi = 1.6$). Furthermore, for over 99% of cutoff times between $0.000001n^2$ and $0.720843n^2$, we prove that ParamRLS-F returns the F-optimal parameter value within 61 configuration comparisons, in expectation. Thus ParamRLS-F is capable of identifying that the smaller the cutoff time the higher the F-optimal mutation rate.

We point out that, for both RIDGE and LEADINGONES, our analyses show that for the considered ranges of cutoff times the parameter landscape seen by ParamRLS-F is unimodal. This implies that all algorithm configurators capable of hillclimbing can efficiently tune the mutation rate of the (1+1)$_\chi$ EA for the same cutoff time values if they use the Best-Fitness performance metric.

6.2 On the Configuration of the (1+1)$_\chi$ EA for RIDGE

In this section, we consider the impact of the cutoff time required by algorithm configurators to identify the optimal mutation rate (with respect to the performance metric) for the (1+1)$_\chi$ EA optimising RIDGE. We show in Section 6.2.2 that any algorithm configurator that uses Optimisation-Time as the performance metric is blind for sub-quadratic cutoff times. On the other hand, we show in Section 6.2.3 that cutoff times that are a linear factor smaller than the expected optimisation time of any configuration are sufficient for algorithm configurators that use the Best-Fitness performance metric to identify that $\chi = 1$ achieves the highest solution quality for any time budget. To demonstrate this point, we will prove that linear cutoff times suffice for ParamRLS-F to return $\chi = 1$, and that the expected number of comparisons before the active parameter is set to this configuration for the first time is at most linear in the size of the parameter space. Since $\chi = 1$ is also O-optimal, this proves that ParamRLS-F is able to return the O-optimal parameter value using cutoff times that are a linear factor smaller than those required by Optimisation-Time-based configurators.

To prove these results, we must first analyse the behaviour of the (1+1)$_\chi$ EA for the optimisation of RIDGE for all permitted parameter values. We do so in Section 6.2.1.

6.2.1 Analysis of the (1+1)$_\chi$ EA for RIDGE

In this section, we analyse the effect of the mutation rate χ on the performance of the (1+1)$_\chi$ EA for the optimisation of RIDGE. We first derive bounds on the drift and the expected optimisation time. The bounds on the drift are tight up to an additive term of $O(1/n^2)$ and show that it is independent of the current fitness of the individual. The drift is maximised for $\chi = 1$, and hence this configuration minimises the expected optimisation time.

Lemma 65. For the (1+1)$_\chi$ EA optimising RIDGE, initialised at 0^n:

(i) The drift, $\Delta_\chi(x_t) := \textsc{Ridge}(x_{t+1}) - \textsc{Ridge}(x_t)$, is bounded as follows:

$$\frac{\chi}{n}\left(1-\frac{\chi}{n}\right)^{n-1} \le \mathrm{E}[\Delta_\chi(x_t) \mid x_t, x_t \neq 1^n]$$
$$\mathrm{E}[\Delta_\chi(x_t) \mid x_t] \le \frac{\chi}{n}\left(1-\frac{\chi}{n}\right)^{n-1} + O\left(\frac{1}{n^2}\right).$$

(ii) Setting $\chi = 1$ yields the smallest expected optimisation time for any constant χ, which is at most en^2, assuming n is large enough.

Proof. We split this proof into a section for either claim of the lemma.

Proof of Lemma 65(i).
By definition,

$$\mathrm{E}[\Delta_\chi(x_t) \mid x_t, \textsc{Ridge}(x_t) = j < n] = \sum_{i=1}^{n-j} (i \cdot \Pr(\textsc{Ridge}(x_{t+1}) - \textsc{Ridge}(x_t) = i)).$$

Due to the nature of RIDGE, we know that the current best-found individual will be in the form $1^j 0^{n-j}$. This means that, in order to improve by exactly i in a single iteration, we must flip exactly the first i leading zeroes in the bit string. The probability of doing this is $(\chi/n)^i(1-\chi/n)^{n-i}$. This implies that

$$\mathrm{E}[\Delta_\chi(x_t) \mid x_t, \textsc{Ridge}(x_t) = j < n] = \sum_{i=1}^{n-j}\left(i \cdot \left(\frac{\chi}{n}\right)^i \left(1-\frac{\chi}{n}\right)^{n-i}\right).$$

We can trivially bound this sum from below by considering only the first term (as all terms are non-negative):

$$\mathrm{E}[\Delta_\chi(x_t) \mid x_t, \textsc{Ridge}(x_t) = j < n] \geq \frac{\chi}{n}\left(1-\frac{\chi}{n}\right)^{n-1}.$$

We bound $\mathrm{E}[\Delta_\chi(x_t) \mid x_t, \textsc{Ridge}(x_t) = j < n]$ from above by observing that

$$\begin{aligned}\sum_{i=1}^{n-j}\left(i \cdot \left(\frac{\chi}{n}\right)^i \left(1-\frac{\chi}{n}\right)^{n-i}\right) &= \left(1-\frac{\chi}{n}\right)^n \sum_{i=1}^{n-j}\left(i \cdot \left(\frac{\chi}{n-\chi}\right)^i\right)\\ &\leq \left(1-\frac{\chi}{n}\right)^n \sum_{i=1}^{\infty}\left(i \cdot \left(\frac{\chi}{n-\chi}\right)^i\right) = \left(1-\frac{\chi}{n}\right)^n \cdot \frac{\chi(n-\chi)}{(n-2\chi)^2}\end{aligned}$$

and then observing that

$$\begin{aligned}\left(1-\frac{\chi}{n}\right)^n \cdot \frac{\chi(n-\chi)}{(n-2\chi)^2} &= \frac{\chi}{n}\left(1-\frac{\chi}{n}\right)^{n-1} \cdot \left(\frac{n(n-\chi)}{(n-2\chi)^2}\left(1-\frac{\chi}{n}\right)\right)\\ &= \frac{\chi}{n}\left(1-\frac{\chi}{n}\right)^{n-1} \cdot \frac{(n-\chi)^2}{(n-2\chi)^2} = \frac{\chi}{n}\left(1-\frac{\chi}{n}\right)^{n-1} + O\left(\frac{1}{n^2}\right).\end{aligned}$$

Proof of Lemma 65(ii).
By Lemma 65(i), the drift of the $(1+1)_1$ EA optimising RIDGE, for every $x_t \neq 1^n$, is

$$\mathrm{E}[\Delta_1(x_t) \mid x_t, x_t \neq 1^n] \geq \frac{1}{n}\left(1-\frac{1}{n}\right)^{n-1}.$$

By the same lemma we have that

$$\mathrm{E}[\Delta_\chi(x_t) \mid x_t] \leq \frac{\chi}{n}\left(1-\frac{\chi}{n}\right)^{n-1} + \Theta\left(\frac{1}{n^2}\right).$$

Since the expression $\frac{\chi}{n}\left(1-\frac{\chi}{n}\right)^{n-1}$ is maximised for $\chi = 1$ and is $\Theta(1/n)$ for all positive constants χ, we have that, for sufficiently large n, the lower bound on the drift of the

(1+1)$_1$ EA is larger than the upper bound on the drift of the (1+1)$_\chi$ EA with $\chi \neq 1$. As these expressions for the drift do not depend on the fitness of the individual, we can determine the expected optimisation time of the (1+1)$_\chi$ EA on RIDGE using additive drift analysis. Since all algorithms are initialised at 0^n, a distance of n has to be bridged. Theorem 23 (Section 3.2.6, page 68) states that the expected optimisation time is at most n divided by a lower bound on the drift, and it is at least n divided by an upper bound on the drift. Along with the first statement of this lemma, this implies that the expected optimisation time is at most en^2 for $\chi = 1$ and at least $e^\chi/\chi \cdot n^2 - O(n)$, which is larger than en^2 for all $\chi \neq 1$ if n is large enough. Hence, for large enough n, the expected optimisation time for the (1+1)$_1$ EA is smaller than the expected optimisation time for all other configurations with constant $\chi \neq 1$. □

The following lemma bounds the probability that one configuration attains a higher fitness than another within the cutoff time or states that one of the configurations has reached the optimum within this time. For $\kappa = \Omega(n^{1+\varepsilon})$, for a positive constant ε, this claim holds with overwhelming probability.

Lemma 66. Assume that the (1+1)$_a$ EA and the (1+1)$_b$ EA, with a and b two distinct non-negative constants such that $ae^{-a} > be^{-b}$, are both initialised at 0^n. Then with probability at least

$$1 - 3\exp(-\Omega(\kappa/n))$$

the (1+1)$_a$ EA has a higher fitness on RIDGE than the (1+1)$_b$ EA after κ iterations or one of the considered algorithms has found a global optimum within the first $\kappa - 1$ iterations.

Proof. Define A_t and B_t to be the fitness values of the (1+1)$_a$ EA and the (1+1)$_b$ EA, respectively, after t iterations on RIDGE. Note that the probability of an improvement for the (1+1)$_a$ EA is $\Pr(A_{t+1} \geq A_t + 1 \mid A_t < n) \geq a/n \cdot (1 - a/n)^{n-1}$. We may drop the condition $A_t < n$ as if it is violated then an optimum has been found within the first t steps. This means we assume that the number of improvements is not bounded. The fitness A_κ is obviously at least as large as the number of improvements in the first κ generations. Applying Chernoff bounds to the latter, for every constant $0 < \varepsilon < 1$, we get

$$\Pr\left(A_\kappa \leq (1-\varepsilon)\kappa \cdot \frac{a}{n}\left(1 - \frac{a}{n}\right)^{n-1}\right) \leq \exp(-\Omega(\kappa/n)).$$

Using $\left(1-\frac{a}{n}\right)^{n-1} = \left(1-\frac{a}{n}\right)^{n-a}\left(1-\frac{a}{n}\right)^{a-1} \geq e^{-a}\left(1-\frac{a}{n}\right)^{a-1} \geq e^{-a}\left(1-O\left(\frac{1}{n}\right)\right)$ where the last step is trivial for $a \leq 1$ and for $a > 1$ follows from Bernoulli's inequality, we obtain

$$A_\kappa \geq (1-\varepsilon)\kappa \cdot \frac{ae^{-a}}{n}\left(1-O\left(\frac{1}{n}\right)\right) \tag{6.1}$$

with probability $1-\exp(-\Omega(\kappa/n))$.

We bound B_κ from above in a similar fashion. However, we need to take into account the possibility of jumps that increase the fitness by more than 1. Note that, for all $i \in \mathbb{N}$,

$$\begin{aligned}\Pr(B_{t+1} = B_t + i \mid B_t) \leq \left(\frac{b}{n}\right)^i \left(1-\frac{b}{n}\right)^{n-i} &= \left(\frac{b}{n-b}\right)^i \left(1-\frac{b}{n}\right)^n \\ &= \left(\frac{b}{n-b}\right)^{i-1}\left(1-\frac{b}{n-b}\right)\cdot\left(1-\frac{b}{n}\right)^n \cdot \frac{b}{n-2b}.\end{aligned}$$

Hence $(B_{t+1} - B_t \mid B_t)$ has the same distribution as the convolution $X_{t+1}Y_{t+1}$ where X_{t+1} is a Bernoulli random variable with parameter $p_X := \left(1-\frac{b}{n}\right)^n \cdot \frac{b}{n-2b}$ and Y_{t+1} is a geometric random variable with parameter $1-\frac{b}{n-b}$; in other words, $\Pr(Y_{t+1} = i) = \left(\frac{b}{n-b}\right)^{i-1}\left(1-\frac{b}{n-b}\right)$. Intuitively, the X-variables can be seen as indicator variables signalling whether an improvement happens and the Y-variables correspond to the jump length in an improving generation.

Applying Chernoff bounds to the variables $X_1,\dots,X_\kappa$, for every constant $0 < \varepsilon < 1$,

$$\Pr\left(\sum_{t=1}^{\kappa} X_t \geq (1+\varepsilon)\kappa p_X\right) \leq \exp(-\Omega(\kappa/n)).$$

Assuming $\sum_{t=1}^{\kappa} X_t \leq n_X := (1+\varepsilon)\kappa p_X$, we bound the contribution of up to this number of Y-variables with a corresponding X-variable with value 1 using Chernoff bounds for geometric random variables. For ease of notation, we rename these variables $Y_1,\dots,Y_{n_X}$. This yields

$$\Pr\left(\sum_{t=1}^{n_X} Y_t \geq (1+\varepsilon)n_X \cdot \frac{n-b}{n-2b}\right) \leq \exp(-\Omega(n_X)) = \exp(-\Omega(\kappa/n)).$$

Together, with probability at least $1-2\exp(-\Omega(\kappa/n))$,

$$B_\kappa \le \sum_{t=1}^{\kappa} X_t Y_t \le (1+\varepsilon) n_X \cdot \frac{n-b}{n-2b} = (1+\varepsilon)^2 \kappa \left(1-\frac{b}{n}\right)^n \cdot \frac{b}{n-2b} \cdot \frac{n-b}{n-2b}$$
$$\le (1+\varepsilon)^2 \kappa \frac{be^{-b}}{n-2b} \cdot \frac{n-b}{n-2b} \le (1+\varepsilon)^2 \kappa \frac{be^{-b}}{n} \cdot \left(1+O\left(\frac{1}{n}\right)\right). \tag{6.2}$$

Since $ae^{-a} > be^{-b}$ we can choose ε small enough such that $(1-\varepsilon)ae^{-a}(1-O(1/n)) > (1+\varepsilon)^2 be^{-b}(1+O(1/n))$, for large enough n. Then the lower bound for A_κ from (6.1) and the lower bound for B_κ from (6.2) imply that $A_\kappa > B_\kappa$ with probability at least $1-3\exp(-\Omega(\kappa/n))$. □

6.2.2 Optimisation-Time Requires at Least Quadratic Cutoff Times

In this section, we show that all algorithm configurators that use the Optimisation-Time performance metric are blind if the cutoff time is at most $\kappa \le (1-\varepsilon)en^2$. We first show that for this cutoff time no configuration has reached the optimum of RIDGE with overwhelming probability.

Lemma 67. For all constants $\chi, \varepsilon > 0$, the (1+1)$_\chi$ EA requires more than $(1-\varepsilon)en^2$ iterations to reach the optimum of RIDGE, with probability $1-\exp(-\Omega(n))$.

Proof. By the progress bounds established in the proof of Lemma 66, the fitness of the individual in the (1+1)$_\chi$ EA after $\kappa := (1-\varepsilon')en^2$ iterations on RIDGE is at most

$$(1+\varepsilon'')^2(1-\varepsilon)n \cdot \left(1+O\left(\frac{1}{n}\right)\right) \le (1+\varepsilon'')^3(1-\varepsilon)n \le (1-\varepsilon')n,$$

with probability at least $1-2\exp(-\Omega(n))$, for suitably chosen positive constants ε' and ε'' and large enough n. Hence, for cutoff times of $\kappa \le (1-\varepsilon)n^2$, the (1+1)$_\chi$ EA has not reached the optimum of RIDGE, with probability at least $1-2\exp(-\Omega(n)) = 1-\exp(-\Omega(n))$. □

This implies that all configurators that use the Optimisation-Time performance metric are blind if the cutoff time is at most $\kappa \le (1-\varepsilon)en^2$.

Theorem 68. Consider any configurator using the Optimisation-Time performance metric for the configuration of the (1+1)$_\chi$ EA for RIDGE for any positive constant ϕ and discretisation factor d. For cutoff times $\kappa \le (1-\varepsilon)en^2$, for some constant $\varepsilon > 0$, the configurator remains blind for any polynomial number of comparisons and runs per evaluation.

Proof. By Lemma 67, for cutoff times $\kappa \leq (1-\varepsilon)en^2$ and for every constant choice of χ, all configurations of the (1+1)$_\chi$ EA fail to reach the optimum of RIDGE, with overwhelming probability. The result therefore follows from Theorem 36. □

6.2.3 Linear Cutoff Times Suffice for Best-Fitness

In this section, we prove that for any discretisation factor d (Section 3.3.1), ParamRLS-F is able to identify the F-optimal and O-optimal configuration of (1+1)$_\chi$ EA for RIDGE (i.e. $\chi = 1$) using linear cutoff times. In particular, we show that for any cutoff time $\kappa \geq \varepsilon n$ and a sufficiently large constant $\varepsilon > 0$, the expected number of comparisons required by ParamRLS-F before the active parameter is set to $\chi = 1$ is at most $6d\phi$ (i.e. the expected configuration time is linear in the number of parameter values). Moreover, after dn^ε comparisons with cutoff time $\kappa \geq n^{1+\varepsilon}$, for any positive constant ε, ParamRLS-F returns $\chi = 1$ with overwhelming probability. Note that this implies that ParamRLS-F is able to identify the O-optimal configuration using cutoff times that are a linear factor smaller than those required by any configurator using the Optimisation-Time performance metric.

We prove the above claims by bounding the probability that one configuration has a higher fitness than another after κ iterations. The following lemma shows that, for large enough cutoff times, in a comparison between two configurations where either both have $\chi \leq 1$ or both have $\chi \geq 1$, the configuration with χ closer to 1 wins the comparison with overwhelming probability. This implies that the parameter landscape seen by ParamRLS-F is unimodal.

Lemma 69. Assume that the (1+1)$_a$ EA and the (1+1)$_b$ EA, with a and b two positive constants such that $ae^{-a} > be^{-b}$, are both initialised at 0^n. Then with probability at least

$$1 - 3\exp(-\Omega(\kappa/n)) - \kappa\exp(-\Omega(n))$$

the (1+1)$_a$ EA wins in a comparison in ParamRLS-F against the (1+1)$_b$ EA on RIDGE with cutoff time κ. Note that if a and b satisfy either $0 < b < a \leq 1$ or $1 \leq a < b \leq \phi$ then the condition $ae^{-a} > be^{-b}$ is implied.

Proof. We can show that the probability that either algorithm reaches the optimum within the first n^2 iterations is $\exp(-\Omega(n))$ by setting $\varepsilon = 1 - 1/e \approx 0.632$ in Lemma 67. If $\kappa \leq n^2$ then the statement follows from Lemma 66 and a union bound over the failure probabilities from Lemma 66 and the aforementioned term of $\exp(-\Omega(n))$.

If $\kappa > n^2$ we argue that the (1+1)$_b$ EA only wins the comparison if either it finds the optimum before the (1+1)$_a$ EA or if it is ahead of the (1+1)$_a$ EA after κ iterations. A necessary condition for this union of events is that the (1+1)$_b$ EA is ahead of the (1+1)$_a$ EA during some point in time t with $n^2 \leq t \leq \kappa$. Applying Lemma 66 for all such values of t and taking a union bound, the probability that this happens is at most

$$\sum_{t=n^2}^{\kappa} 3\exp(-\Omega(t/n)) \leq (\kappa-1)3\exp(-\Omega(n)).$$

The claim follows as the sum of all failure probabilities is at most $(\kappa-1)3\exp(-\Omega(n)) + \exp(-\Omega(n)) \leq \kappa\exp(-\Omega(n))$.

For the remark at the end of the lemma, note that the expression $\chi e^{-\chi}$ is maximised for $\chi = 1$ and decreases monotonically on either side of this point. The claim therefore holds since a is closer to 1 than b by assumption. □

We now use the unimodality of the parameter landscape derived in Lemma 69 to prove that ParamRLS-F can efficiently identify that $\chi = 1$ achieves the highest solution quality for any polynomial cutoff time.

Theorem 70. Consider ParamRLS-F for the configuration of the (1+1)$_\chi$ EA for RIDGE with discretisation constant d and $\phi = \Theta(1)$. Assume that it uses $\kappa \geq \varepsilon n$ for a sufficiently large constant $\varepsilon > 0$, a single run per configuration evaluation (i.e. $r = 1$), and the local search operator $\pm\{1\}$. Then:

- Using a cutoff time $\kappa \geq \varepsilon n$ for a sufficiently large constant $\varepsilon > 0$, the expected number of comparisons T before ParamRLS-F sets the active parameter to $\chi = 1$ for the first time is at most $6d\phi$.
- Using a cutoff time $\kappa \geq n^{1+\varepsilon}$ for some constant $\varepsilon > 0$, after $dn^{\varepsilon'}$ for some constant $\varepsilon' > 0$, ParamRLS-F returns $\chi = 1$ with overwhelming probability.

Proof. Given a pair of configurations, let us call the configuration with a value of χ closer to 1 the 'better' configuration and the other configuration the 'worse' configuration.

By Lemma 69, using $\kappa \geq \varepsilon n$, the probability that the better configuration wins a comparison with cutoff time κ is at least $1 - \exp(-\Omega(\kappa/n)) - \kappa\exp(-\Omega(n)) \geq 2/3$, the inequality holding since we can choose the constant $\varepsilon > 0$ appropriately and $\kappa \in \mathrm{poly}(n)$.

The current configuration is compared against a better one with probability at least $1/2$ and it is compared against a worse one with probability at most $1/2$. Hence the distance to the optimal parameter value decreases in expectation by at least $(1/d) \cdot (1/2 \cdot 2/3 - 1/2 \cdot 1/3) = 1/(6d)$. The initial distance is at most ϕ. By additive drift arguments (Theorem 23), the expected time to reach the optimal parameter value for the first time is at most $6d\phi$.

For the second statement, we use that, if $\kappa \geq n^{1+\varepsilon}$, then the probability of accepting the worse configuration is exponentially small. Hence, w. o. p. within any polynomial number of comparisons we never experience the event that the worse configuration wins a comparison. This implies that $\max\{1, \phi - 1\}(d+1)$ steps decreasing the distance towards the optimal parameter value are sufficient. By Chernoff bounds, the probability of not seeing this many steps in dn^{ε} iterations is exponentially small. Finally, once the optimal parameter value is reached, it is never left w. o. p. Thus, after dn^{ε} iterations, the optimal parameter value is returned with overwhelming probability. □

6.3 On the Configuration of the (1+1)$_\chi$ EA for LEADINGONES

We now turn our attention to the configuration of the mutation rate of the (1+1)$_\chi$ EA for the LEADINGONES problem class. The analysis is considerably more complicated than for RIDGE since the progress depends (mildly, but not insignificantly) on the current fitness value. For a search point with m leading ones, the probability of improving the fitness is exactly $\chi/n \cdot (1-\chi/n)^m$ as it is necessary and sufficient to flip the first 0-bit while not flipping the m leading ones. This probability decreases over the course of a run, from $\chi/n \cdot (1-\chi/n)^0 = \chi/n$ for $m = 0$ to $\chi/n \cdot (1-\chi/n)^{n-1} \approx \chi/(en)$ for $m = n-1$. The mutation rate that maximises the expected progress decreases as the distance to the optimum decreases, similarly to how the neighbourhood size of RLS$_k$ that maximises the expected progress decreases as the distance to the optimum of ONEMAX* decreases, as observed in Chapter 5.

Due to the increased complexity of the analysis, we focus on one specific discretisation factor d and choice of ϕ as a proof of concept. We choose a discretisation factor of $d = 10$ and $\phi = 3$, which implies that χ may take values from the set $\{0.1, 0.2, \ldots, 2.9, 3.0\}$. We are confident that our method generalises to any constant discretisation factor by increasing the precision of our analytical results in relation to the granularity of the parameter space (given by d) and we provide a tool to do so[19].

It is easily verified that for the chosen granularity the parameter value with the smallest expected optimisation time is $\chi = 1.6$, which is closest to the mutation rate with the smallest expected optimisation time for any real value of χ ($\chi \approx 1.59$). We therefore expect the tuner

[19]This tool (available at `https://github.com/george-hall-sheff/leading_ones_recurrences_tool`) applies our proof technique for any parameter space with constant ϕ and d and any period length. Therefore the user can repeatedly decrease the period length until this tool is able to prove the desired results for their chosen parameter space.

to return $\chi = 1.6$ when the cutoff time is large enough, since it is O-optimal. For smaller cutoff times, larger values of χ are F-optimal.

We prove that all configurators that use the Optimisation-Time performance metric are blind for cutoff times $\kappa \leq 0.772n^2$. We then prove that ParamRLS-F is once again able to identify the F-optimal configuration for arbitrary cutoff times. For almost all quadratic cutoff times $\kappa \leq 0.720843n^2$, we prove that ParamRLS-F can determine which configuration maximises the solution quality for that time budget with overwhelming probability, and returns the F-optimal configuration within 65 comparisons, in expectation. Furthermore, for cutoff times $\kappa \geq 0.721118n^2$ (i.e. approximately $0.05n$ smaller than the required cutoff time for all Optimisation-Time-based configurators), it returns the O-optimal configuration $\chi = 1.6$, again within 65 comparisons in expectation.

In order to prove the above claims it is necessary to conduct a precise fixed-budget analysis of the (1+1)$_\chi$ EA for LEADINGONES. We do so in Section 6.3.1, and prove along the way that the parameter landscape seen by ParamRLS-F is unimodal.

6.3.1 The Parameter Landscape of the (1+1)$_\chi$ EA for LEADINGONES is Unimodal Under Best-Fitness

Due to the complexity of the configuration scenario we follow the approach we used when analysing the configuration of RLS$_k$ for ONEMAX* in Section 5.3 and establish intervals that bound the fitness at various stages of a run. This allows us to locate the final fitness after κ iterations with the desired precision and with overwhelming probability. For almost all cutoff times, our fitness intervals reveal that the configuration closer to the F-optimal one leads to a higher solution quality with overwhelming probability. This implies that the parameter landscape seen by ParamRLS-F is unimodal.

We prove this claim first for over 99% of cutoff times satisfying $0.000001n^2 \leq \kappa \leq 0.720843n^2$ and then for all cutoff times $\kappa \geq 0.721118n^2$, with the mutation rate $\chi = 1.6$ being returned for this second range of cutoff times, as expected.

We first derive bounds on the progress made by the (1+1)$_\chi$ EA in a *period* of n^2/ψ iterations (for a positive constant ψ) that hold with overwhelming probability. We define progress as the difference between the fitness distance to the optimum at the beginning of the period and at the end of the period. This allows us to sum the progress made in a constant number of periods to create intervals that with overwhelming probability contain the fitness of the (1+1)$_\chi$ EA after a specified quadratic number of iterations. Given two configurations, we can then calculate for which cutoff times the intervals corresponding to these configurations do not overlap. This tells us which configuration will win a comparison under the Best-Fitness performance metric with that cutoff time with overwhelming probability.

Lemma 71. Consider the (1+1)$_\chi$ EA for LEADINGONES and an optimisation period of n^2/ψ iterations, for some positive constant ψ, that starts with a fitness of j. Let Z be the amount of progress made by the algorithm over the period. Then, with overwhelming probability:

(i) $Z \leq \frac{2\chi n}{\psi \cdot \exp\left(\frac{\chi j}{n}\right)} + o(n)$

(ii) For every i with $j \leq i < n$, $Z \geq \frac{2\chi n}{\psi \cdot \exp\left(\frac{\chi i}{n}\right)} - o(n)$, or the algorithm exceeds fitness i at the end of the period.

The intuition behind these bounds is that the probability of improving the fitness of a search point with m leading ones is $\chi/n \cdot (1-\chi/n)^m$, which is at least $\chi/n \cdot (1-\chi/n)^i \approx \chi/n \cdot \exp(-\chi i/n)$ and at most $\chi/n \cdot (1-\chi/n)^j \approx \chi/n \cdot \exp(-\chi j/n)$ if $j \leq m \leq i$. The factor of 2 stems from the fact that when the first 0-bit is flipped the fitness increases by 2 in expectation as the following bits may be set to 1 with probability 0.5.

We use several helper lemmas to prove Lemma 71. We begin by deriving an upper bound on the number of iterations in which the (1+1)$_\chi$ EA increases the fitness of the individual (iterations that we call *improvements*) in a period of n^2/ψ iterations, for some positive constant ψ.

Lemma 72. With overwhelming probability the (1+1)$_\chi$ EA optimising LEADINGONES starting at a fitness of i makes at most

$$(1+n^{-1/4})\frac{\chi n}{\psi \cdot \exp\left(\frac{\chi i}{n}\right)}$$

improvements in a period of length n^2/ψ.

Proof. The probability of the (1+1)$_\chi$ EA making an improvement is equal to the probability that it flips the first leading zero and does not flip any of the leading ones. Let us assume that the current fitness of the individual is j. Then the probability of making an improvement is therefore

$$\Pr(\mathrm{LO}(x_{t+1}) > \mathrm{LO}(x_t) \mid \mathrm{LO}(x_t) = j) = \frac{\chi}{n}\left(1-\frac{\chi}{n}\right)^j.$$

Since $j \geq i$ by assumption, we have

$$\Pr(\mathrm{LO}(x_{t+1}) > \mathrm{LO}(x_t) \mid \mathrm{LO}(x_t) = j) \leq \frac{\chi}{n}\left(1-\frac{\chi}{n}\right)^i.$$

Using Theorem 15 (Section 3.2.3, page 62), the quantity $(1-(\chi/n))^i$ can be bounded from above by $\exp(-\chi i/n)$. Hence

$$\Pr(\mathrm{LO}(x_{t+1}) > \mathrm{LO}(x_t) \mid \mathrm{LO}(x_t) = j) \leq \frac{\chi}{n} \exp\left(-\frac{\chi i}{n}\right).$$

We now use a Chernoff bound to obtain a bound on the number of improvements made in a single period of length n^2/ψ, for some positive constant ψ. Let the random variable X_k equal 1 if and only if an improvement occurs in iteration k. Otherwise let it equal 0. Now define the random variable $Y^\psi := \sum_{k=0}^{(n^2/\psi)-1} X_k$ that counts the number of improvements that occur in a period of length n^2/ψ. Since X_k is an indicator variable we have that

$$\mathrm{E}[Y^\psi] \leq \frac{\chi n}{\psi \exp\left(\frac{\chi i}{n}\right)}.$$

Let us now optimistically assume that $\mathrm{E}[Y^\psi]$ is equal to this upper bound. Then by a standard Chernoff bound we derive that

$$\Pr\left(Y^\psi \geq (1+n^{-1/4})\frac{\chi n}{\psi \exp\left(\frac{\chi i}{n}\right)}\right) \leq \exp\left(-\frac{n^{-1/2}\chi n}{3\psi \exp\left(\frac{\chi i}{n}\right)}\right)$$
$$\leq \exp\left(-\frac{n^{-1/2}\chi n}{3\psi e^\chi}\right) = \exp(-\Omega(n^{1/2})). \qquad \square$$

We now derive a corresponding lower bound on the number of improvements in a period of the same length.

Lemma 73. Assume that the (1+1)$_\chi$ EA optimising LEADINGONES currently has a fitness of j. Fix a value of $i \leq n$ and consider a period of length n^2/ψ (for any positive constant ψ). Then with overwhelming probability the (1+1)$_\chi$ EA makes at least

$$(1-n^{-1/4})\frac{\chi n}{\psi \cdot \exp\left(\frac{\chi i}{n-\chi}\right)}$$

improvements during the period, or the (1+1)$_\chi$ EA exceeds a fitness of i.

Proof. Assuming that the current fitness of the individual is j, the probability of making an improvement is

$$\Pr(\mathrm{LO}(x_{t+1}) > \mathrm{LO}(x_t) \mid \mathrm{LO}(x_t) = j) = \frac{\chi}{n}\left(1-\frac{\chi}{n}\right)^j.$$

Since $j \leq i$ by assumption, we have

$$\Pr(\mathrm{LO}(x_{t+1}) > \mathrm{LO}(x_t) \mid \mathrm{LO}(x_t) = j) \geq \frac{\chi}{n}\left(1 - \frac{\chi}{n}\right)^i.$$

Using Theorem 15, the quantity $(1-(\chi/n))^i$ can be bounded from below by $\exp(-\chi i/(n-\chi))$. This implies that

$$\Pr(\mathrm{LO}(x_{t+1}) > \mathrm{LO}(x_t) \mid \mathrm{LO}(x_t) \leq i) \geq \frac{\chi}{n}\exp\left(-\frac{\chi i}{n-\chi}\right).$$

We now use a Chernoff bound to derive a lower bound on the number of improvements made in a single period of length n^2/ψ, for some positive constant ψ. Let the indicator random variable X_k attain value 1 with probability $\frac{\chi}{n}\exp\left(-\frac{\chi i}{n-\chi}\right)$. We now define the random variable $Y^\psi := \sum_{k=0}^{(n^2/\psi)-1} X_k$, which is stochastically dominated by the number of improvements that occur in a period of length n^2/ψ, unless a fitness larger than i is reached. Note that the event of exceeding fitness i is included in the event for which we aim to derive a lower bound on the probability. Since X_k is an indicator variable we have that

$$\mathrm{E}[Y^\psi] \geq \frac{\chi n}{\psi \cdot \exp\left(\frac{\chi i}{n-\chi}\right)}.$$

By a standard Chernoff bound we derive that

$$\Pr\left(Y^\psi \leq (1-n^{-1/4})\frac{\chi n}{\psi \cdot \exp\left(\frac{\chi i}{n-\chi}\right)}\right) \leq \exp\left(-\frac{n^{-1/2}\chi n}{2\psi \cdot \exp\left(\frac{\chi i}{n-\chi}\right)}\right) = \exp(-\Omega(n^{1/2})).$$

□

Associated with each improvement are a number of *free riders*. These are the consecutive 1-bits immediately following the former leading 0 which has just been flipped. Therefore the total fitness gained in each improvement is one more than the number of associated free riders. We first derive an upper bound on the number of free riders encountered during a period of length n^2/ψ.

Lemma 74. Consider ℓ improving steps of the (1+1)$_\chi$ EA optimising LEADINGONES. Then with probability $1-\exp(-\Omega(\ell^{1/2}))$ it gains at most $(1+\ell^{-1/4})\ell$ in fitness from free riders in total.

Proof. For this proof, we assume that the algorithm is operating on an infinite bit string. This is a valid assumption since we are deriving an upper bound on the number of free riders that occur during the optimisation process and the number that occurs in reality is strictly smaller than the number that is possible on an infinite bit string. It is well known that the distribution of the ones and zeroes following the leading zero is uniformly at random, since this section of the bit string has no effect on the fitness of the individual (see Lemma 1 in [85] for a formal proof). This fact and our infinite bit string assumption allow us to use a geometric random variable X_k with parameter $1/2$ to count the number of free riders in the k-th improvement. We define $X^\ell := \sum_{k=1}^{\ell} X_k$ as the total number of free riders encountered over the ℓ improvements. Using a Chernoff bound for sums of geometric random variables (Theorem 21(i), page 64) we have that

$$\Pr(X^\ell \geq (1+\ell^{-1/4})\cdot\ell) \leq \exp\left(-\frac{\ell^{-1/2}(\ell-1)}{2(1+\ell^{-1/4})}\right) = \exp(-\Omega(\ell^{1/2})). \qquad \square$$

We now derive a corresponding lower bound on the number of free riders encountered during a period of the same length. The lemma below proves that with overwhelming probability the (1+1)$_\chi$ EA gains at least $(1-\ell^{-1/4})\ell$ in fitness from free riders, assuming there are ℓ improvements in a period, or the (1+1)$_\chi$ EA reaches the optimum during the period.

Lemma 75. Consider a period with ℓ improving steps of the (1+1)$_\chi$ EA optimising LEADING-ONES. Then, with overwhelming probability, the (1+1)$_\chi$ EA gains at least $(1-\ell^{-1/4})\ell$ in fitness from free riders in total during this period, or the (1+1)$_\chi$ EA reaches the optimum during the period.

Proof. We define the random variables X_k and X^ℓ as in the proof of Lemma 74. We may assume in the following that the free riders are effectively drawn from an infinite bit string. This assumption is only false when the optimum is reached and this event is contained in the event for which we aim to derive a lower bound on the probability.

By Theorem 21(ii) we have that

$$\Pr(X^\ell \leq (1-\ell^{-1/4})\cdot\ell) \leq \exp\left(-\frac{\ell^{-1/2}\ell}{2-4\ell^{-1/4}/3}\right) = \exp(-\Omega(\ell^{1/2})). \qquad \square$$

Combining Lemmas 72 and 74 allows us to derive an upper bound on the total progress made by the algorithm in a period of length n^2/ψ which holds with overwhelming probability. This allows us to prove Lemma 71.

Proof of Lemma 71. We split this proof into a section for each claim of the lemma.

Proof of Lemma 71(i):

Note that the progress of the algorithm stops abruptly if the global optimum is reached. Hence we assume pessimistically that progress is not bounded. By Lemma 72, w. o. p. the algorithm makes at most $\ell := (1+n^{-1/4})\frac{\chi n}{\psi \cdot \exp\left(\frac{\chi j}{n}\right)}$ improvements. By Lemma 74, the number of free riders is at most $(1+\ell^{-1/4})\ell$ with probability $1-\exp(-\Omega(\ell^{1/2})) = 1-\exp(-\Omega(n^{1/2}))$. Together, the fitness increases by at most

$$(2+\ell^{-1/4})\ell = \frac{2\chi n}{\psi \cdot \exp\left(\frac{\chi j}{n}\right)} + o(n),$$

w. o. p., as claimed.

Proof of Lemma 71(ii):

By Lemma 73, with overwhelming probability, the algorithm makes at least $\ell := (1-n^{-1/4})\frac{\chi n}{\psi \cdot \exp\left(\frac{\chi i}{n-\chi}\right)}$ improvements within a period of length n^2/ψ or exceeds some fitness i as defined in the statement of this theorem. If the algorithm exceeds fitness i at the end of the period then we are done since this event is contained in the event for which we aim to derive a lower bound on the probability. Assuming at least ℓ improvements are made, by Lemma 75 the total gain through free riders is at least $(1-\ell^{-1/4})\ell$ with probability $1-\exp(-\Omega(\ell^{1/2})) = 1-\exp(-\Omega(n^{1/2}))$. Hence, w. o. p. the (1+1)$_\chi$ EA makes at least

$$(2-\ell^{-1/4})\ell = \frac{2\chi n}{\psi \cdot \exp\left(\frac{\chi i}{n-\chi}\right)} - o(n)$$

progress in total during the period, or the (1+1)$_\chi$ EA exceeds a fitness of i at the end of the period. Finally, we argue that the term $n-\chi$ in the exp-term can be replaced by n since

$$\exp\left(\frac{\chi i}{n-\chi}\right) = \exp\left(\frac{\chi i}{n}\cdot\frac{n}{n-\chi}\right) = \exp\left(\frac{\chi i}{n} + \frac{\chi^2 i}{n(n-\chi)}\right) = \exp\left(\frac{\chi i}{n}\right)\cdot\exp\left(\frac{\chi^2 i}{n(n-\chi)}\right)$$
$$\leq \exp\left(\frac{\chi i}{n}\right)\cdot\exp\left(\frac{\chi^2}{n-\chi}\right) \leq \exp\left(\frac{\chi i}{n}\right)\cdot\frac{1}{1-\frac{\chi^2}{n-\chi}},$$

where in the last step we used $e^x \leq \frac{1}{1-x}$ for $x<1$. Together,

$$\frac{2\chi n}{\psi \cdot \exp\left(\frac{\chi i}{n-\chi}\right)} - o(n) \geq \frac{2\chi n}{\psi \cdot \exp\left(\frac{\chi i}{n}\right)}\left(1-\frac{\chi^2}{n-\chi}\right) - o(n) = \frac{2\chi n}{\psi \cdot \exp\left(\frac{\chi i}{n}\right)} - O(1) - o(n),$$

and the $O(1)$ term is absorbed in the $o(n)$ term. □

We apply the progress bounds from Lemma 71 to derive bounds on the fitness of the (1+1)$_\chi$ EA after an arbitrary number of periods that hold with overwhelming probability.

Lemma 76. Consider the (1+1)$_\chi$ EA optimising LEADINGONES. Let a run of length αn^2 be split into $\alpha\psi$ periods of length n^2/ψ (for positive constants α and ψ). Define $\ell_{\chi,0} := 0$ and $u_{\chi,0} := \sqrt{n}$. Then for $i \le \alpha$ there exist $u_{\chi,i+1}$ and $\ell_{\chi,i+1}$ with

$$u_{\chi,i+1} = u_{\chi,i} + \frac{2\chi n}{\psi \exp\left(\frac{\chi u_{\chi,i}}{n}\right)} + o(n)$$
$$\ell_{\chi,i+1} = \ell_{\chi,i} + \frac{2\chi n}{\psi \exp\left(\frac{\chi u_{\chi,i+1}}{n}\right)} - o(n)$$

such that, with overwhelming probability, the following holds. At the end of period i, for $0 \le i \le \alpha$, the current fitness is in the interval $[\ell_{\chi,i}, u_{\chi,i}]$ or an optimum has been found, and throughout period i the fitness is in $[\ell_{\chi,i-1}, u_{\chi,i}]$ or an optimum has been found.

Proof. We prove the statement by induction. If an optimum has been reached at the end of period i then there is nothing to prove. We may therefore assume that $\ell_{\chi,i} < n$. With overwhelming probability, the initial fitness will be in $[\ell_{\chi,0}, u_{\chi,0}] := [0, \sqrt{n}]$, since the fitness of the initial search point is at most $\sqrt{n}$ with probability $1 - 2^{-\sqrt{n}}$.

By Lemma 71(i) we have that if the individual begins with a fitness of $u_{\chi,i}$ then, with overwhelming probability, after a period of n^2/ψ iterations it has a fitness of at most

$$u_{\chi,i+1} := u_{\chi,i} + \frac{2\chi n}{\psi \exp\left(\frac{\chi u_{\chi,i}}{n}\right)} + o(n).$$

Note that the upper bound $u_{\chi,i+1}$ still holds (trivially) if $u_{\chi,i+1} \ge n$.

We now apply Lemma 71(ii) to show that the fitness of the individual at the end of period $i+1$ is at least $\ell_{\chi,i+1}$ given that its fitness at the end of period i is at least $\ell_{\chi,i}$. By Lemma 71(ii), with overwhelming probability the fitness of the individual in the (1+1)$_\chi$ EA is at least

$$\ell_{\chi,i+1} := \ell_{\chi,i} + \frac{2\chi n}{\psi \cdot \exp\left(\frac{\chi u_{\chi,i+1}}{n}\right)} - o(n)$$

or it exceeds $u_{\chi,i+1}$. Since we do not exceed fitness $u_{\chi,i+1}$ with overwhelming probability (and we certainly do not exceed it if $u_{\chi,i+1} \ge n$), we have that the fitness is at least $\ell_{\chi,i+1}$, w. o. p.

Since at the beginning of period $i+1$ the individual has a fitness in the interval $[\ell_{\chi,i}, u_{\chi,i}]$ and at the end of the period it has a fitness in the interval $[\ell_{\chi,i+1}, u_{\chi,i+1}]$, and since both $\ell_{\chi,j}$

and $u_{\chi,j}$ are monotonically increasing for all j we can conclude that, w. o. p., the fitness of the individual remains in the interval $[\ell_{\chi,i}, u_{\chi,i+1}]$ throughout the period. Taking a union bound over all failure probabilities concludes the proof of both claims. □

Since we do not have a closed form for the intervals derived in Lemma 76, we follow the approach used in Section 5.3 and iterate them computationally in order to derive bounds on the fitness after a given constant number of iterations. We observe that in expectation the (1+1)$_\chi$ EA makes a linear amount of progress during a period that consists of a quadratic number of iterations. This fact implies that we can verify whether one configuration is ahead of another by computing the leading constant of the $\Theta(n)$ term in the fitness bounds from Lemma 76 and checking whether the intervals are overlapping. If they are not overlapping, then one configuration is ahead of another by a linear amount, with overwhelming probability. If n is large enough, then the $o(n)$ terms from Lemma 76 can be ignored as the fitness is determined exclusively by the coefficients of the linear terms. We extract the relevant coefficients from the fitness bounds in the following lemma.

Lemma 77. Let $c_{\ell,\chi,i}$ and $c_{u,\chi,i}$ denote the leading constants in the definition of $\ell_{\chi,i}$ and $u_{\chi,i}$ from Lemma 76, respectively (i. e., $\ell_{\chi,i} = c_{\ell,\chi,i} \cdot n - o(n)$ and $u_{\chi,i} = c_{u,\chi,i} \cdot n + o(n)$). Then $c_{u,\chi,i+1}$ and $c_{\ell,\chi,i+1}$ can be expressed using the recurrences $c_{\ell,\chi,0} = c_{u,\chi,0} = 0$,

$$\begin{aligned} c_{u,\chi,i+1} &= c_{u,\chi,i} + \frac{2\chi}{\psi \exp\left(\chi \cdot c_{u,\chi,i}\right)} \\ c_{\ell,\chi,i+1} &= c_{\ell,\chi,i} + \frac{2\chi}{\psi \exp\left(\chi \cdot c_{u,\chi,i+1}\right)}. \end{aligned}$$

Proof. The statement about $c_{\ell,\chi,0}$ and $c_{u,\chi,0}$ is obvious as $\ell_{\chi,0} = 0$ and $u_{\chi,0} = \sqrt{n}$.

By definition, $u_{\chi,i+1}$ can be written as

$$c_{u,\chi,i} \cdot n + \frac{2\chi n}{\psi \exp\left(\frac{\chi \cdot (c_{u,\chi,i} \cdot n + o(n))}{n}\right)} + o(n) \leq c_{u,\chi,i} \cdot n + \frac{2\chi n}{\psi \exp\left(\chi \cdot c_{u,\chi,i}\right)} + o(n)$$

and the leading constant is

$$c_{u,\chi,i+1} := c_{u,\chi,i} + \frac{2\chi}{\psi \exp\left(\chi \cdot c_{u,\chi,i}\right)}.$$

By definition, $\ell_{\chi,i+1}$ can be written as

$$c_{\ell,\chi,i} \cdot n + \frac{2\chi n}{\psi \exp\left(\frac{\chi(u_{\chi,i+1}+o(n))}{n}\right)} - o(n) = c_{\ell,\chi,i} \cdot n + \frac{2\chi n}{\psi \exp\left(\chi \cdot c_{u,\chi,i+1} + o(1)\right)} - o(n).$$

This is at least

$$c_{\ell,\chi,i} \cdot n + \frac{2\chi n}{\psi \exp\left(\chi \cdot c_{u,\chi,i+1}\right)} - o(n)$$

since $e^{-o(1)} \geq 1 - o(1)$ by Theorem 13(i). The leading constant is thus

$$c_{\ell,\chi,i+1} := c_{\ell,\chi,i} + \frac{2\chi}{\psi \exp\left(\chi \cdot c_{u,\chi,i+1}\right)}. \qquad \square$$

We can now prove that the parameter landscape seen by ParamRLS-F is unimodal. We do so first for the cutoff times given in Table 6.1 with $\kappa \leq 0.772075n^2$.

Optimal χ	lower bound on κ	upper bound on κ
3.0	$0.000030n^2$	$0.225138n^2$
2.9	$0.225628n^2$	$0.241246n^2$
2.8	$0.241720n^2$	$0.259143n^2$
2.7	$0.259600n^2$	$0.279105n^2$
2.6	$0.279545n^2$	$0.301461n^2$
2.5	$0.301885n^2$	$0.326611n^2$
2.4	$0.327018n^2$	$0.355040n^2$
2.3	$0.355431n^2$	$0.387346n^2$
2.2	$0.387720n^2$	$0.424266n^2$
2.1	$0.424623n^2$	$0.466723n^2$
2.0	$0.467064n^2$	$0.515884n^2$
1.9	$0.516208n^2$	$0.573238n^2$
1.8	$0.573546n^2$	$0.640714n^2$
1.7	$0.641006n^2$	$0.720843n^2$
1.6	$0.721118n^2$	∞

Table 6.1 Ranges of cutoff times κ for which, for ParamRLS-F configuring the (1+1)$_\chi$ EA for LEADINGONES, the parameter is unimodal with the optimum at χ.

Lemma 78. Consider ParamRLS-F for the configuration of the (1+1)$_\chi$ EA for LEADING-ONES with $\phi = 3$ and discretisation constant $d = 10$. For the ranges of κ given in Table 6.1, with $\kappa \leq 0.772075n^2$ and for large enough n, the parameter landscape is unimodal with the stated optimum.

Proof. In order to derive bounds on the fitness of the individual in the (1+1)$_\chi$ EA after αn^2 iterations, we simply iterate the recurrences given by Lemma 77 $\psi \cdot \alpha$ times. We do so for all cutoff times in the set $\{0.000001n^2, 0.000002n^2, \dots, 0.772074n^2, 0.772075n^2\}$, setting $\psi = 10^6$. For all configurations and cutoff times that we consider here, the upper bound on the leading constant of the fitness is strictly less than 1. Then Lemma 77 implies that, w. o. p., no configuration reaches the optimum within any of these cutoff times, and hence we can ignore the case in Lemma 76 that the optimum is reached by the end of a period and simply assume that the fitness is contained in the interval given by the lemma. By Lemma 77, the fitness of the individual in the (1+1)$_\chi$ EA is in the range $[c_{\ell,\chi,i}n - o(n), c_{u,\chi,i+1}n + o(n)]$ w. o. p. Hence, for two parameter values a, b, if the interval $[c_{\ell,a,i}, c_{u,a,i+1}]$ is non-overlapping with the interval $[c_{\ell,b,i}, c_{u,b,i+1}]$ and $c_{\ell,a,i} > c_{u,b,i+1}$ then we can conclude that, for all times t satisfying $\tau_i \le t \le \tau_{i+1}$, where τ_j is the number of iterations corresponding to the end of period j, the (1+1)$_a$ EA is ahead of the (1+1)$_b$ EA with overwhelming probability.

We conducted the above verification for all pairs of neighbouring configurations (i.e. for all configurations between which it is possible to transition in a single mutation using the $\pm\{1\}$ local search operator) for all cutoff times up to $0.772075n^2$. We discovered that each value of χ that is at least 1.6 is F-optimal for some range of quadratic cutoff times (bounds on which are given in Table 6.1), and also that for these ranges of cutoff times, for ParamRLS-F, the parameter landscape is unimodal. □

Having proved that the parameter landscape seen by ParamRLS-F is unimodal for the cutoff times given in Table 6.1 with $\kappa \le 0.772075n^2$, we now turn our attention to cutoff times $\kappa \ge 0.772075n^2$. The following lemma proves that the parameter landscape seen by ParamRLS-F is also unimodal for this range of cutoff times.

Lemma 79. Consider ParamRLS-F for the configuration of the (1+1)$_\chi$ EA for LEADINGONES with $\phi = 3$ and discretisation constant $d = 10$. For $\kappa \ge 0.772075n^2$ and for large enough n, the parameter landscape is unimodal with the optimum at $\chi = 1.6$.

Before proving this result, we derive a helper lemma. Given two configurations and the distance between them, the lemma provides a sufficient condition for the configuration that is closer to the optimum to reach the optimum before the other one covers the initial distance between them with overwhelming probability.

Lemma 80. Let the fitness of the individuals in the (1+1)$_a$ EA and the (1+1)$_b$ EA optimising LEADINGONES be contained in the intervals $[c_{\ell,a,i} \cdot n - o(n), c_{u,a,i} \cdot n + o(n)]$ and $[c_{\ell,b,i} \cdot n - o(n), c_{u,b,i} \cdot n + o(n)]$, respectively. Assume that $c_{\ell,a,i} > c_{u,b,i}$ (that is, at the end of period i

the (1+1)$_a$ EA is ahead of the (1+1)$_b$ EA by some linear distance). If

$$\frac{\left(\frac{2b}{(c_{\ell,a,i}-c_{u,b,i})\cdot\exp\left(bc_{\ell,b,i}\right)}+\varepsilon\right)}{\left(\frac{2a}{(1-c_{\ell,a,i})\cdot\exp(a)}\right)}\leq 1,$$

for some positive constant ε, then the (1+1)$_a$ EA reaches the optimum before the (1+1)$_b$ EA has covered the initial distance between the two algorithms, with overwhelming probability.

Proof. We know by Lemma 71(ii) that, with overwhelming probability, for any i greater than the current fitness, the (1+1)$_\chi$ EA makes progress of at least

$$\frac{2\chi n}{\psi\cdot\exp\left(\frac{\chi i}{n-\chi}\right)}-o(n)$$

in a period of length n^2/ψ or exceeds a fitness of i at the end of the period. We know by Lemma 71(i) that, with overwhelming probability, the (1+1)$_\chi$ EA makes progress of at most

$$\frac{2\chi n}{\psi\cdot\exp\left(\frac{\chi j}{n}\right)}+o(n)$$

in a period of length n^2/ψ, given that it begins the period at fitness j.

By setting i in Lemma 71(ii) to $n-1$ we derive that with overwhelming probability the (1+1)$_a$ EA either makes progress of at least

$$\frac{2an}{\psi\cdot\exp\left(\frac{a(n-1)}{n-a}\right)}-o(n)\geq\frac{2an}{\psi\cdot\exp(a)}-o(n)$$

in a period of length n^2/ψ (where the inequality holds due to Theorem 13(i)) or exceeds a fitness of $n-1$ at the end of the period (i.e. reaches the optimum). The quantity $n-(c_{\ell,a,i}\cdot n-o(n))$ is an upper bound on the distance to the optimum of the (1+1)$_a$ EA and therefore if the algorithm makes more progress than this then it has reached the optimum. We have that

$$\begin{aligned}\frac{2an}{\psi\cdot\exp(a)}-o(n)\geq n-(c_{\ell,a,i}\cdot n-o(n)) &\iff \frac{2a}{\psi\cdot\exp(a)}\geq 1-c_{\ell,a,i}+o(1)\\ &\iff \psi\leq\frac{2a}{(1-c_{\ell,a,i}+o(1))\cdot\exp(a)},\end{aligned}$$

for which a sufficient condition is

$$\psi \leq \frac{2a}{(1-c_{\ell,a,i}) \cdot \exp(a)}.$$

Recalling that the length of a period is n^2/ψ, we derive that, when beginning at a fitness of at least $c_{\ell,a,i} \cdot n - o(n)$, the (1+1)$_a$ EA reaches the optimum w. o. p. for all periods of length at least

$$\frac{n^2}{\left(\frac{2a}{(1-c_{\ell,a,i}) \cdot \exp(a)}\right)}. \tag{6.3}$$

We now derive a lower bound on the time required for the (1+1)$_b$ EA to make progress of at least the initial distance between the (1+1)$_a$ EA and itself. Since by assumption the fitness of the (1+1)$_a$ EA is at least $c_{\ell,a,i} \cdot n - o(n)$ and the fitness of the (1+1)$_b$ EA is at most $c_{u,b,i} \cdot n + o(n)$ we have a lower bound on this distance of $(c_{\ell,a,i} - c_{u,b,i}) \cdot n - o(n)$. By Lemma 76 we have that w. o. p. the (1+1)$_b$ EA makes progress of at most

$$\frac{2bn}{\psi \cdot \exp\left(\frac{b(c_{\ell,b,i} \cdot n - o(n))}{n}\right)} + o(n) \leq \frac{2bn}{\psi \cdot \exp\left(bc_{\ell,b,i}\right)} + o(n)$$

in a period of length n^2/ψ, where the inequality holds due to Theorem 13(ii). Therefore the (1+1)$_b$ EA does not cover the initial distance between the two algorithms if

$$\frac{2bn}{\psi \cdot \exp\left(bc_{\ell,b,i}\right)} + o(n) \leq (c_{\ell,a,i} - c_{u,b,i}) \cdot n - o(n)$$

$$\iff \psi \geq \frac{2b}{(c_{\ell,a,i} - c_{u,b,i}) \cdot \exp\left(bc_{\ell,b,i}\right)} + o(1),$$

for which a sufficient condition is

$$\psi \geq \frac{2b}{(c_{\ell,a,i} - c_{u,b,i}) \cdot \exp\left(bc_{\ell,b,i}\right)} + \varepsilon$$

for some positive constant ε. Recalling that the length of the period is n^2/ψ, we see that, with overwhelming probability, the (1+1)$_b$ EA requires at least

$$\frac{n^2}{\left(\frac{2b}{(c_{\ell,a,i} - c_{u,b,i}) \cdot \exp\left(bc_{\ell,b,i}\right)} + \varepsilon\right)} \tag{6.4}$$

iterations before the probability that it has covered the initial distance between the two algorithms is not overwhelmingly small. Combining the bounds on ψ in Equations (6.3) and (6.4) we conclude that the (1+1)$_a$ EA reaches the optimum of LEADINGONES before the (1+1)$_b$ EA catches up if

$$\frac{n^2}{\left(\frac{2b}{(c_{\ell,a,i}-c_{u,b,i})\cdot\exp\left(bc_{\ell,b,i}\right)}+\varepsilon\right)} \geq \frac{n^2}{\left(\frac{2a}{(1-c_{\ell,a,i})\cdot\exp(a)}\right)}$$

which holds if and only if

$$\frac{\left(\frac{2b}{(c_{\ell,a,i}-c_{u,b,i})\cdot\exp\left(bc_{\ell,b,i}\right)}+\varepsilon\right)}{\left(\frac{2a}{(1-c_{\ell,a,i})\cdot\exp(a)}\right)} \leq 1. \qquad \square$$

We can now prove Lemma 79.

Proof of Lemma 79. Consider a comparison between two configurations, $\chi = a$ and $\chi = b$, with either $0.1 \leq b < a \leq 1.6$ or $1.6 \leq a < b \leq 3.0$. By Lemma 78, the (1+1)$_a$ EA is ahead of the (1+1)$_b$ EA at time $0.772075n^2$, w. o. p. In this proof, we first show that, from time $0.772075n^2$ until some time t_a, the (1+1)$_a$ EA remains ahead of the (1+1)$_b$ EA w. o. p. We then prove that, for cutoff times larger than t_a, the (1+1)$_a$ EA will find the optimum before the (1+1)$_b$ EA reaches where the (1+1)$_a$ EA began the period, with overwhelming probability. These two cases together imply the claim for all cutoff times $\kappa \geq 0.772075n^2$.

We first verify that the (1+1)$_a$ EA is ahead of the (1+1)$_b$ EA at all times between $0.772075n^2$ and some time t_a. By Lemma 77, from the end of period i to the end of period $i+1$, the leading constant of the fitness of the individual in the (1+1)$_\chi$ EA is in the range $[c_{\ell,\chi,i}, c_{u,\chi,i+1}]$ with overwhelming probability, where $c_{\ell,\chi,i}$ and $c_{u,\chi,i+1}$ are as defined in Lemma 77. Hence if (for $\psi = 10^6$) we verify for all periods i satisfying $772076 \leq i \leq \psi \cdot t_a$ that the intervals $[c_{\ell,a,i}, c_{u,a,i+1}]$ and $[c_{\ell,b,i}, c_{u,b,i+1}]$ are non-overlapping then we can conclude that, w. o. p., the (1+1)$_a$ EA is ahead of the (1+1)$_b$ EA for all times t satisfying $0.772075n^2 \leq t \leq t_a$. For each b, the time t_a is chosen to be the end of the final period for which $c_{u,b,i} < 1$. For these values of t_a, we verified that the above intervals are non-overlapping and hence that the (1+1)$_a$ EA is ahead of each worse configuration for all time t satisfying $0.772075n^2 \leq t \leq t_a$.

If $a \leq 1.6$ and for all times t satisfying $0.772075n^2 \leq t \leq t_a$ the condition in Lemma 80 holds for all $b < a$ then we can conclude that, with overwhelming probability, the (1+1)$_a$ EA reaches the optimum before any (1+1)$_b$ EA catches it. Therefore the claim holds for any cutoff time $\kappa \geq t_a$. Similarly, if $a \geq 1.6$ and the condition in Lemma 80 holds for all $b > a$

then we can conclude that w. o. p. the $(1+1)_a$ EA reaches the optimum before any $(1+1)_b$ EA catches it. Therefore again the claim holds for any cutoff time $\kappa \geq t_a$.

We can therefore prove the claim by verifying that, for all $a \leq 1.6$ and $b < a$ the inequality in Lemma 80 holds, and also for all $a \geq 1.6$ and $b > a$ the same inequality is true. We do so for the specific configuration scenario given by this theorem by iterating over all cases where we require this inequality to hold and verifying that it does so in each case. We used a value of $\varepsilon = 10^{-11}$. Table B.1 (Appendix B, page 194) contains the values of the quantity from Lemma 80, scaled up by a factor of 100,000 for readability, for all required comparisons for values of $\chi \leq 1.6$. It is easily verified that all values are much smaller than $100,000$, as required. This proves our claim. □

6.3.2 Any Configurator Using the Optimisation-Time Performance Metric is Blind for $\kappa \leq 0.772075n^2$

In this section, we prove that with overwhelming probability any configurator that uses the Optimisation-Time performance metric is blind when configuring the $(1+1)_\chi$ EA for LeadingOnes if the cutoff time is smaller than $\kappa = 0.772075n^2$.

Lemma 81. For all configurations $\chi \in \{0.1, 0.2, \dots, 2.9, 3.0\}$, the $(1+1)_\chi$ EA does not reach the optimum of LeadingOnes within $0.772075n^2$ iterations, with overwhelming probability.

Proof. After $772,075$ periods of length $n^2/10^6$ (that is, with $\psi = 10^6$) we observe that the value $c_{u,\chi,i}$ (defined as in Lemma 77) for all $\chi \in \{0.1, 0.2, \dots, 2.9, 3.0\}$ is less than 1. This implies that, with overwhelming probability, after $0.772075n^2$ iterations, no configuration has found the optimum of LeadingOnes. □

Using Lemma 81, we are now able to prove that all configurators that use the Optimisation-Time performance metric are blind for any cutoff time $\kappa \leq 0.772075n^2$.

Theorem 82. Consider any configurator using the Optimisation-Time performance metric for the configuration of the $(1+1)_\chi$ EA for LeadingOnes with $\chi \in \{0.1, 0.2, \dots, 2.9, 3.0\}$ (i.e. $d = 10, \phi = 3$). If the cutoff time for each run is never allowed to exceed $0.772075n^2$ then, after any polynomial number of comparisons and runs per evaluation, the configurator remains blind.

Proof. Since by Lemma 81 we know that, with overwhelming probability, no configuration finds the optimum of LeadingOnes within $0.772075n^2$ iterations, the result follows by Theorem 36. □

6.3.3 Best-Fitness Identifies the Optimal Mutation Rate Independent of the Cutoff Time

Lemma 78 proves that the parameter landscape seen by ParamRLS-F is unimodal for the ranges of cutoff times given in Table 6.1 (page 147). Similarly, Lemma 79 proves that, for ParamRLS-F, for all cutoff times $\kappa \geq 0.772075n^2$ the parameter landscape is unimodal with the optimum at $\chi = 1.6$.

We now use these facts to prove that ParamRLS-F is able to identify the F-optimal mutation rate for almost all quadratic cutoff times (and all cutoff times that are larger than quadratic). Furthermore, for all cutoff times of at least $0.721118n^2$ it returns the O-optimal mutation rate (i.e. $\chi = 1.6$).

Theorem 83. Consider ParamRLS-F for the configuration of the (1+1)$_\chi$ EA for LEADING-ONES with $\chi \in \{0.1, 0.2, \ldots, 2.9, 3.0\}$ (i.e. $d = 10, \phi = 3$) using the local search operator $\pm\{1\}$. For all cutoff times in one of the ranges listed in Table 6.1 it holds that, for any positive constant ε and for large enough n:

(i) The expected number of comparisons T before ParamRLS-F sets the active parameter to the F-optimal value (given in Table 6.1) for the first time is at most $2d\phi + \exp(-\Omega(n^\varepsilon)) = 60 + \exp(-\Omega(n^\varepsilon))$.

(ii) After a number of comparisons that is both polynomial and at least n^ε, ParamRLS-F returns the F-optimal parameter value with overwhelming probability.

Proof. We proceed in the same manner as in the proof of Theorem 70. We pessimistically assume that the active parameter value is initialised as far away from the F-optimal parameter value as possible. The initial distance is clearly bounded by $d\phi$.

Given a comparison between two configurations that are both on the same side of the F-optimal configuration, let us call the configuration with a value of χ closer to the optimum the 'better' configuration, and the other configuration the 'worse' configuration. By Lemma 78 or 79 (depending on the cutoff time), in a comparison between any pair of configurations that are both on the same side of the optimum, the better configuration wins w. o. p. Let us assume that the better configuration always beats the worse configuration. Since when using the local search operator $\pm\{1\}$ the tuner will mutate the active parameter to one closer to the optimum with probability $1/2$, the tuner takes a step towards the optimum with probability $1/2$. With the remaining probability, the active parameter will remain the same. The expected time to move closer to the optimum is thus 2. Since the tuner needs to take at most $d\phi$ steps towards the F-optimal configuration, this implies that $\mathrm{E}[T] \leq 2d\phi$. In the overwhelmingly unlikely event that the worse configuration wins a comparison then we restart the argument. Therefore, $\mathrm{E}[T] \leq 2d\phi + \exp(-\Omega(n^{\varepsilon}))$ for some positive constant ε from the definition of overwhelming probabilities.

Using a Chernoff bound to count the number of times that the tuner takes a step towards the F-optimal configuration proves that, with overwhelming probability, n^{ε} comparisons, for any positive constant ε, suffice for ParamRLS-F to set the active parameter value to the optimum. By the union bound, the value of the active parameter remains at the optimum w. o. p. once it has been found, since there are polynomially many comparisons. This implies that, w. o. p., the tuner returns the F-optimal configuration for the cutoff time if run for at least this many comparisons. □

6.4 Conclusions

In Chapter 5, we showed that, when configuring the neighbourhood size of randomised local search for two benchmark functions, using the Best-Fitness performance metric allows a configurator to identify the O-optimal parameter value with considerably smaller cutoff times (i.e. more efficiently) than the Optimisation-Time performance metric designed for this purpose. The analysis presented in this chapter reveals that this insight is also true for the more complex task of tuning the continuously-valued standard bit mutation global mutation operator. In particular, we proved that, when configuring the (1+1)$_\chi$ EA for RIDGE, any configurator using the Optimisation-Time performance metric is blind for cutoff times of at most $(1-\varepsilon)en^2$ for any constant $\varepsilon > 0$, and when configuring the (1+1)$_\chi$ EA for LEADINGONES, any configurator using the Optimisation-Time performance metric is blind for cutoff times of at most $0.772075n^2$.

On the other hand, ParamRLS-F identifies the F-optimal and O-optimal mutation rate $\chi = 1$ for the (1+1)$_\chi$ EA optimising RIDGE using only linear cutoff times. When configuring for LEADINGONES, ParamRLS-F identifies the F-optimal mutation rate for over 99% of quadratic cutoff times of at most $0.772075n^2$. For all cutoff times greater than this, ParamRLS-F returns the F-optimal and O-optimal parameter value $\chi = 1.6$, whilst any configurator using Optimisation-Time requires a cutoff time of at least $0.772075n^2$. In each case, ParamRLS-F identifies the desired mutation rate efficiently: in expectation a linear number of comparisons are sufficient to do so.

The efficiency of ParamRLS-F is due to the fact that, for both problem classes, the parameter landscape induced by the Best-Fitness performance metric is unimodal. This allows gradient-following algorithm configurators to efficiently identify optimal mutation rates for both problem classes, as we have proven using ParamRLS-F.

Chapter 7

Faster Algorithm Configuration by Exploiting Parameter Space Unimodality

7.1 Introduction

In this chapter, we build upon the insights gained from the previous chapters regarding the performance of ParamRLS and ParamILS in simple algorithm configuration scenarios and propose a new mutation operator to achieve faster algorithm configuration.

In Chapters 5 and 6, we proved that ParamRLS and ParamILS can identify the optimal parameter values for RLS$_k$ and the (1+1)$_\chi$ EA in expected linear (or in the worst case quadratic) time in the size of the parameter space. An insight gained from these analyses is that, for the considered configuration scenarios, the parameter landscape seen by ParamRLS-F is of the simplest kind: either unimodal (i.e. RIDGE and LEADINGONES) or nearly unimodal (i.e. it is rugged but with an underlying gradient towards the optimum, as seen when configuring RLS$_k$ for ONEMAX). We begin this chapter by complementing these upper bounds with linear lower bounds on the expected runtimes of ParamRLS and ParamILS for any algorithm configuration scenario.

Recent experimental work by Pushak and Hoos suggests that also in practice, when configuring sophisticated algorithms for the optimisation of NP-Hard problems, the parameter landscape is more benign than one might expect (i.e. they are largely unimodal or approximately so) [104].

The search operators used by ParamRLS and ParamILS fail to exploit unimodal or approximately unimodal characteristics of parameter landscapes since they either perform only small perturbations (the $\pm\{1,\dots,m\}$ operator used by ParamRLS) or are unable to efficiently fine-tune configurations by searching locally near good candidates (the *random-*

step mutation operator employed by ParamILS). In both cases, this leads to a linear number of configuration comparisons being required in expectation to identify the optimal configuration, even when the parameter landscape is favourable (i.e. unimodal or approximately unimodal).

We propose the usage of a "harmonic" mutation operator that is provably considerably faster for (approximately) unimodal parameter landscapes than the $\pm\{1,\dots,m\}$ and random-step operators employed by ParamRLS and ParamILS, respectively, when configuring target algorithms with a single parameter. In particular, for ParamRLS using the harmonic operator for the configuration of a single-parameter algorithm with ϕ permitted parameter values, we prove an expected runtime of $O(\log^2 \phi)$ if the parameter landscape is approximately unimodal, compared to the linear expected runtime required by the standard mutation operators of ParamRLS and ParamILS for any configuration scenario.

The harmonic mutation operator enables much faster configuration when the parameter landscape is unimodal. This speed-up is achieved at the expense of being slower in worst-case scenarios (e.g. deceptive landscapes where the gradient leads away from the optimum), which the empirical analyses of Pushak and Hoos suggest to be unlikely in practice.

We proceed by quantifying the worst-case slow-down to be paid in exchange for the speed-up when the parameter landscape is (approximately) unimodal and prove that the harmonic operator is at most a logarithmic factor slower than the standard mutation operators in the worst case. We conclude the chapter with an experimental analysis that validates the theoretical insights for more sophisticated algorithm configuration scenarios where two parameters have to be tuned for the NP-Hard MAX-SAT problem.

7.2 The Harmonic Mutation Operator

We investigate the improvement in performance that can be gained by using a mutation operator that samples a step size according to the harmonic distribution, and which has previously been shown to have good performance when optimising unimodal functions [31, 34]. We apply it for the first time to the task of algorithm configuration. This operator allows small perturbations with sufficiently high probability to fine-tune good parameter values efficiently whilst, at the same time, allowing larger mutations that can help follow the general gradient from a macro perspective, e.g. by tunnelling through local optima. It is an example of the recent trend towards *fast* mutation operators that aim to balance the proportion of small and large mutations in evolutionary computation, where operators differ in the choice of the probability distribution from which the size of the mutation is drawn [26, 27, 28, 40]. In particular, this search operator can easily be incorporated into any "perturbative" algorithm configurator that maintains a set of best-found configurations and mutates them in search for better ones, as in ParamRLS and ParamILS.

The *harmonic-step* mutation operator analysed in this chapter selects a parameter uniformly at random and samples a step size δ according to the harmonic distribution that dictates by how much it may perturb the value of the chosen parameter. The probability of selecting a step size δ is $1/(\delta \cdot H_{\phi-1})$, where H_m is the m-th harmonic number (i.e. $H_m = \sum_{k=1}^{m} \frac{1}{k}$) and ϕ is the number of permitted values of the parameter. It returns the best parameter value at distance[20] $\pm\delta$. Formally, assuming that the value of the selected parameter θ has index x in the ordered set of possible values of θ, the harmonic-step search operator returns the best configuration among those where the index of the value of θ is in the set $\{x-\delta, x, x+\delta\}$. It executes a good amount of exploration and can deal robustly with non-unimodal functions. We call the variant of ParamRLS that uses the harmonic-step operator *ParamHS* and give its pseudocode in Algorithm 16.

```
Algorithm 16: ParamHS ($\mathcal{A}, \Theta, \Pi, \kappa, r$)
Input : target algorithm ($\mathcal{A}$), parameter space ($\Theta$), training instances ($\Pi$), cutoff time
        ($\kappa$), number of runs per evaluation ($r$).
1  $\theta \leftarrow$ initial configuration chosen uniformly at random
2  while termination condition not satisfied do
3      $\theta_i \leftarrow$ parameter chosen u.a.r.
4      $\phi \leftarrow$ number of values of $\theta_i$
5      $x \leftarrow$ index of value of $\theta_i$ in $\theta$
       // Choose step size $\delta$:
6      $\delta \leftarrow 1$
7      rand $\leftarrow$ chosen uniformly at random from range $[0,1]$
8      while $\sum_{j=1}^{\delta} 1/(j \cdot H_{\phi-1}) \leq$ rand do
9          $\delta \leftarrow \delta + 1$
10     $\theta^{\text{dec}} \leftarrow \theta$ with value for $\theta_i$ with index $x - \delta$
11     $\theta^{\text{inc}} \leftarrow \theta$ with value for $\theta_i$ with index $x + \delta$
12     $\theta' \leftarrow$ better($\mathcal{A}, \theta^{\text{dec}}, \theta^{\text{inc}}, \Pi, \kappa, r$)
13     $\theta \leftarrow$ better($\mathcal{A}, \theta, \theta', \Pi, \kappa, r$)
14 return $\theta$
```

This operator is inspired by one designed to perform fast greedy random walks in one-dimensional domains [31] and was shown to perform better than the $\pm\{1\}$ operator (i.e. as used in ParamRLS with $m = 1$) and the random local search (as used in ParamILS) operators for optimising the multi-valued ONEMAX problem [34].

[20]Throughout this chapter, we define the distance of two parameter values as the absolute difference between the indices of the two values in the ordered set of the possible values of this parameter.

For simplicity, in both the theoretical and experimental analyses in this chapter we optimistically assume that all mutation operators can only generate valid parameter values.

7.3 General Lower Bounds for Default Mutation Operators

To set a baseline for the performance gains obtained by using the harmonic-step operator, we first show general lower bounds on the expected configuration time of a wide class of algorithm configurators, including ParamRLS and ParamILS. Our results apply to all configurators either using a mutation operator that changes one parameter by a small amount, such as the $\pm\{1,\ldots,m\}$ operator with constant m, or an operator that chooses a value uniformly at random (with or without replacement), as is done in the random-step operator in ParamILS. We use a general framework to show that the poor performance of the default mutation operators is not limited to particular configurators, and to identify which algorithm design aspects are the cause of this poor performance.

We first show that mutation operators that only change one parameter by a small amount lead to expected times to identify the optimal configuration that are at least linear in the number of parameter values (i.e. the sum of all ranges of parameter values).

Theorem 84. Consider an algorithm configurator $\mathcal{C}$ for the configuration of a target algorithm with D parameters with ranges $\phi_1,\ldots,\phi_D \geq 2$ such that there is a unique optimal configuration, θ^*. Let $M = \sum_{i=1}^{D} \phi_i$. Assume that in each mutation $\mathcal{C}$ only changes a single parameter, doing so by at most a constant absolute value (e.g. ParamRLS with local search operator $\pm\{1,\ldots,m\}$ for constant m). Then the expected number of comparisons T before $\mathcal{C}$ sets the active parameter to θ^* for the first time is $\Omega(M)$.

Proof. Consider the L_1 distance of the current configuration $x = (x_1,\ldots,x_D)$ from the optimal one $\text{opt} = (\text{opt}_1,\ldots,\text{opt}_D)$: $\sum_{i=1}^{D} |x_i - \text{opt}_i|$. For every parameter i, the expected distance between the uniform random initial configuration and opt_i is minimised if opt_i is at the centre of the parameter range. Then, for odd ϕ, there are two configurations at distances $1,2,\ldots,(\phi_i-1)/2$ from opt_i, each being chosen with probability $1/\phi_i$. The expected distance is thus at least $1/\phi_i \cdot \sum_{j=1}^{(\phi_i-1)/2} 2j = (\phi_i-1)(\phi_i+1)/(4\phi_i) = (\phi_i - 1/\phi_i)/4 \geq \phi_i/8$. For even ϕ_i, the expectation is at least $\phi_i/4$. By linearity of expectation, the expected initial distance is at least $\sum_{i=1}^{D} \phi_i/8 \geq M/8$. Every mutation can only decrease the distance by $O(1)$, hence the expected time is bounded by $(M/8)/O(1) = \Omega(M)$. □

The same lower bound also applies if the mutation operator chooses a value uniformly at random (with or without replacement), as is done in ParamILS.

Theorem 85. Consider an algorithm configurator $\mathcal{C}$ for the configuration of a target algorithm with D parameters with ranges $\phi_1, \dots, \phi_D \geq 2$ such that there is a unique optimal configuration, θ^*. Let $M = \sum_{i=1}^{D} \phi_i$. Assume that in each mutation $\mathcal{C}$ only changes a single parameter and does so by choosing a new value uniformly at random (possibly excluding values previously evaluated). Then the expected number of comparisons T before $\mathcal{C}$ sets the active parameter to θ^* for the first time is $\Omega(M)$.

Proof. Let T_i be the number of times that parameter i is mutated (including the initial step) before it attains its value in the optimal configuration. After $j-1$ steps in which parameter i is mutated, at most j parameter values have been evaluated (including the initial value). The best case is that $\mathcal{C}$ always excludes previous values, which corresponds to a complete enumeration of the ϕ_i possible values in a random order. Since every step of this enumeration has a probability of $1/\phi_i$ of finding the optimal value, the expected time spent on parameter i is $\mathrm{E}[T_i] \geq \sum_{j=0}^{\phi_i} j/\phi_i = (\phi_i + 1)/2$. The total expected time is at least $\sum_{i=1}^{D} \mathrm{E}[T_i] - D + 1$ as the initial step contributes to all T_i and each following step only contributes to one value T_i. Noting $\sum_{i=1}^{D} \mathrm{E}[T_i] - D + 1 \geq \sum_{i=1}^{D} \phi_i/2 - D/2 + 1 \geq M/4$ (as $\phi_i \geq 2$) proves the claim. $\square$

ParamILS is not covered directly by Theorem 85 as it uses random sampling during the initialisation that affects all parameters. However, it is straightforward to show that the same lower bound also applies to ParamILS.

Theorem 86. Consider ParamILS for the configuration of a target algorithm with D parameters with ranges $\phi_1, \dots, \phi_D \geq 2$ such that there is a unique optimal configuration, θ^*. Let $M = \sum_{i=1}^{D} \phi_i$. Then the expected number of comparisons T before ParamILS sets the active parameter to θ^* for the first time is $\Omega(M)$.

Proof. Recall that ParamILS first evaluates R random configurations. If $R \geq M/2$ then the probability of finding the optimum during the first $M/2$ random samples is at most $M/2 \cdot \prod_{i=1}^{D} 1/\phi_i \leq 1/2$ since $M = \sum_{i=1}^{D} \phi_i \leq \prod_{i=1}^{D} \phi_i$. Hence the expected time is at least $1/2 \cdot M/2 = M/4$. If $R < M/2$, then with probability at least 0.5 ParamILS does not find the optimum during the R random steps and starts the IterativeFirstImprovement procedure with a configuration θ_0. This procedure scans the neighbourhood of θ_0, which contains all configurations that differ in one parameter. The number of such configurations is $\sum_{i=1}^{D}(\phi_i - 1) = M - D$. If the global optimum is not among these, then it is not found in these $M - D$ steps. Otherwise, the neighbourhood is scanned in a random order and the expected number of steps is $(M - D + 1)/2$ as in the proof of Theorem 85. In both cases, the expected time is at least $(M - D)/4 \geq M/8$ (as $M \geq 2D$). $\square$

7.4 Performance of the Harmonic Mutation Operator

In the setting of Theorem 84, mutation lacks the ability to explore the parameter space quickly, whilst in the setting of Theorems 85 and 86 it lacks the ability to search locally. The harmonic search operator is designed to be able to do both: it uses larger steps to explore the search space, but smaller steps are made with a higher probability, enabling the search to exploit gradients in the parameter landscape. We now prove that this is the case.

For simplicity, in this section we only consider the configuration of target algorithms with a single parameter, which is assumed to have ϕ permitted values (thus the bounds from Theorems 84–86 simplify to $\Omega(\phi)$). However, we have no reason to believe that the harmonic operator would not improve performance in settings with multiple parameters in the same way. We show that ParamHS is robust in the sense that it performs well on all parameter landscapes (with only a small overhead in the worst case, compared to the lower bounds from Theorem 84–86), and it performs extremely well on functions that are unimodal or have an underlying gradient that is close to being unimodal.

To capture the existence of underlying gradients and functions that are unimodal to some degree, we introduce a notion of approximate unimodality.

Definition 11. A function f on $\{1,\dots,m\}$ is (α,β)*-approximately unimodal* for parameters $\alpha \geq 1$ and $1 \leq \beta \leq m$ if for all positions x with distance $\beta \leq i \leq m$ from the optimum and all positions y with distance $j > \alpha i$ to the optimum we have $f(x) < f(y)$.

In terms of algorithm configuration, this implies that only configurations with a distance to the optimal one that is by a factor of α larger than that of the current configuration can be better. To account for landscapes that do not show a clear gradient close to the optimum, this property need only hold within states with a distance to the optimum of at least β. We give an example of a (2,5)-approximately unimodal function in Figure 7.1.

Note that a $(1,1)$-approximately unimodal function is unimodal in the traditional sense and a $(1,\beta)$-approximately unimodal function is unimodal within the states $\{\beta,\dots,m\}$. Also note that all functions are $(1,m)$-approximately unimodal.

The following performance guarantees for ParamHS show that it is efficient on functions that are close to being unimodal and even in the worst case it is only slower than the default operators used by ParamRLS and ParamILS by a logarithmic amount in expectation.

Theorem 87. Consider ParamHS for the configuration of a target algorithm with a single parameter with ϕ values and a unique globally optimal configuration θ^*. If the parameter landscape is (α,β)-approximately unimodal then the expected number of comparisons T

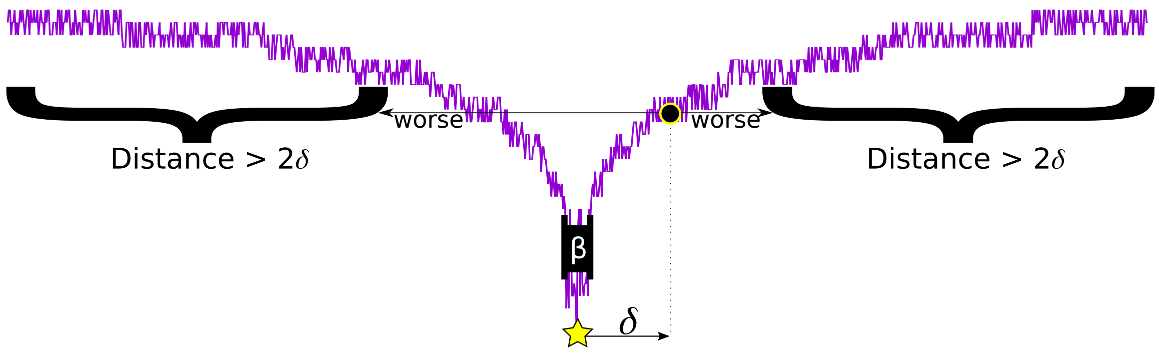


Fig. 7.1 A (2,5)-approximately unimodal function. The point marked by a black circle, for instance, has distance $\delta \geq 5$ from the optimum, and thus all points with a distance from the optimum of at least 2δ correspond to worse (i.e. larger) function values than it.

before the active parameter is set to θ^* for the first time is at most

$$2\alpha H_{\phi-1}\log(\phi) + 2\alpha\beta H_{\phi-1} = O(\alpha\log^2(\phi) + \alpha\beta\log\phi).$$

Proof. Let $f(i)$ describe the performance of the configuration with the i-th largest parameter value. Then the function f is (α,β)-approximately unimodal and we are interested in the time required to locate its minimum.

Let d_t denote the current distance to the optimum and note that $d_0 \leq \phi$. Let d_t^* denote the smallest distance to the optimum seen so far, that is, $d_t^* = \min_{t' \leq t} d_{t'}$. Note that d_t^* is non-increasing over time. Since ParamHS does not accept any worsenings, $f(d_t) \leq f(d_t^*)$.

If $d_t^* \geq \beta$, then by the approximate unimodality assumption, for all $j > \alpha d_t^*$, $f(j) > f(d_t^*) \geq f(d_t)$ (i.e. all points at distance larger than αd_t^* have a worse fitness than the current position and will never be visited).

Now assume that $d_t^* \geq 2\beta$. We estimate the expected time to reach a position with distance at most $\lfloor d_t^*/2 \rfloor$ to the optimum. This includes all points that have distance i to the global optimum, for $0 \leq i \leq \lfloor d_t^*/2 \rfloor$, and distance $d_t - i$ to the current position. The probability of jumping to one of these positions is at least

$$\sum_{i=0}^{\lfloor d_t^*/2 \rfloor} \frac{1}{2(d_t - i)H_{\phi-1}} \geq \sum_{i=0}^{\lfloor d_t^*/2 \rfloor} \frac{1}{2d_t H_{\phi-1}} \geq \frac{d_t^*}{4d_t H_{\phi-1}} \geq \frac{d_t^*}{4\alpha d_t^* H_{\phi-1}} = \frac{1}{4\alpha H_{\phi-1}},$$

where the factor of 1/2 accounts for the possibility of jumping further away from the optimum. Hence, the expected half time for d_t^* is at most $4\alpha H_{\phi-1}$ and the expected time to reach $d_t^* < 2\beta$ is at most $4\alpha H_{\phi-1}\log\phi$.

Once $d_t^* < 2\beta$, the probability of jumping directly to the optimum is at least $\frac{1}{2d_t H_{\phi-1}} \geq \frac{1}{2\alpha d_t^* H_{\phi-1}} \geq \frac{1}{4\alpha\beta H_{\phi-1}}$ and the expected time to reach the optimum is at most $4\alpha\beta H_{\phi-1}$. Adding the above two times and using the well-known fact that $H_{\phi-1} = O(\log\phi)$ yields the claim. □

Corollary 88. In the setting of Theorem 87,

(a) for every unimodal parameter landscape, this bound is $O(\log^2\phi)$.

(b) for every parameter landscape, this bound is $O(\phi\log\phi)$.

Hence ParamHS is far more efficient than the $\Omega(\phi)$ lower bound on the expected time required by tuners using the standard mutation operators (Theorems 84–86) on approximately unimodal parameter landscapes and it is guaranteed never to be slower than the default operators by more than a small $\log\phi$ factor.

We now prove that, if the parameter landscape is (α,β)-approximately unimodal, then ParamHS identifies the optimal parameter value within $\lceil 24\alpha\ln^2\phi\rceil + \lceil 12\alpha\beta\ln^2\phi\rceil$ comparisons, with high probability.

Lemma 89. Consider ParamHS for the configuration of an algorithm with a single parameter having ϕ values and a unique globally optimal configuration θ^*. If the parameter landscape is (α,β)-approximately unimodal, then after

$$\lceil 24\alpha\ln^2\phi\rceil + \lceil 12\alpha\beta\ln^2\phi\rceil$$

comparisons, ParamHS returns θ^* with probability at least $1-2/\phi$.

Proof. Let d_t^* be the smallest distance to the optimum seen so far. Assume initially that $d_t^* \geq 2\beta$, and let $X_t := d_t^* - 2\beta$. We will first bound $T_1 := \inf\{t \geq 0 \mid X_t = 0\}$, the time taken for d_t^* to drop below 2β. As in the proof of Theorem 87, the drift of X_t is

$$\begin{aligned} \mathrm{E}[X_t - X_{t+1} \mid X_t \geq 0] = \mathrm{E}[d_t^* - d_{t+1}^* \mid d_t^* \geq 2\beta] &\geq \frac{d_t^*}{2}\cdot \Pr(d_t^* - d_{t+1}^* \geq d_t^*/2 \mid d_t^* \geq 2\beta) \\ &= \frac{d_t^*}{8\alpha H_{\phi-1}}. \end{aligned}$$

By the tail bound for multiplicative drift given by Theorem 25 (Section 3.2.6, page 69), we derive that

$$\Pr\left(T_1 > \lceil (8\alpha H_{\phi-1})\cdot(r+\ln X_0)\rceil\right) = \Pr\left(T_1 > \left\lceil \frac{r+\ln(X_0/1)}{1/(8\alpha H_{\phi-1})}\right\rceil\right) \leq e^{-r}.$$

Letting $r := \ln\phi$ we obtain $\Pr\left(T_1 > \lceil(8\alpha H_{\phi-1})\cdot(\ln\phi+\ln\phi)\rceil\right) \le 1/\phi$, as $X_0 \le \phi$. Since $H_{\phi-1} < 1.5\ln\phi$, we derive that $\Pr\left(T_1 > \lceil 24\alpha\ln^2\phi\rceil\right) \le 1/\phi$.

We follow a similar approach to derive a bound on the time taken to reach the optimum when $d_t^* < 2\beta$. We denote this time $T_2 := \inf\{t \ge 0 \mid d_t^* = 0, d_0^* < 2\beta\}$. Since, again by the proof of Theorem 87, the drift of d_t^* in this region can be bounded as

$$\mathrm{E}[d_t^* - d_{t+1}^* \mid d_t^* < 2\beta] \ge d_t^* \cdot \Pr(d_t^* - d_{t+1}^* = d_t^* \mid d_t^* < 2\beta) = \frac{d_t^*}{4\alpha\beta H_{\phi-1}},$$

we can again apply Theorem 25. Thus

$$\Pr\left(T_2 > \lceil(4\alpha\beta H_{\phi-1})\cdot(r+\ln d_{T_1}^*)\rceil\right) = \Pr\left(T_2 > \left\lceil\frac{r+\ln(d_{T_1}^*/1)}{1/(4\alpha\beta H_{\phi-1})}\right\rceil\right) \le e^{-r}.$$

Letting $r := \ln\phi$ and observing that $d_{T_1}^* \le \phi$ and $H_{\phi-1} < 1.5\ln\phi$, we obtain

$$\Pr\left(T_2 > \lceil 12\alpha\beta\ln^2\phi\rceil\right) \le \frac{1}{\phi}.$$

Therefore, by the union bound, the total number of comparisons sufficient for ParamHS to first set the active parameter to θ^*, for any (α,β)-approximately unimodal parameter landscape, is at most

$$\lceil 24\alpha\ln^2\phi\rceil + \lceil 12\alpha\beta\ln^2\phi\rceil,$$

with probability at least $1-2/\phi$. Since θ^* is a unique global optimum, it will beat any other configuration in a comparison with probability 1, and thus will be returned by ParamHS once it has been sampled for the first time. □

7.5 Experimental Analysis

In the previous section we proved that, for parameter landscapes that are unimodal or approximately unimodal, it is beneficial to use the harmonic-step operator instead of the default operators used in ParamRLS and ParamILS. In this section, we verify experimentally that this operator yields performance improvements in practice on parameter landscapes that either have or appear to have these characteristics. We investigate the impact of replacing the default search operators in ParamRLS and ParamILS with the harmonic-step operator on the time required to identify the optimal configuration (or in one case a set of near-optimal configurations) in different configuration scenarios. We also investigate the performance of ParamRLS using the random-step operator both with and without replacement.

We analysed the number of configuration comparisons required for each configurator to identify the optimal neighbourhood size k for RLS$_k$ optimising ONEMAX (as analysed in Chapter 5) and the optimal mutation rate χ for the (1+1)$_\chi$ EA optimising RIDGE and the (1+1)$_\chi$ EA optimising LEADINGONES (as analysed in Chapter 6). Finally, we tuned two parameters of the SAT solver SAPS (Section 3.4.3) for the MAX-SAT problem class. Due to the size of the parameter space in this final configuration scenario, we analysed the time taken to identify one of the five best-performing configurations found during an exhaustive search of the parameter space.

In the first two configuration scenarios, the parameter landscape induced by Best-Fitness for the given cutoff time is unimodal (see Sections 6.2 and 6.3 for theoretical proofs and Figures 7.2a and 7.2b on page 168 for empirical verification). In such landscapes we expect the harmonic-step operator to perform well. In the third scenario, the parameter landscape induced by Best-Fitness for the given cutoff time is *not* unimodal ($k = 2c+1$ outperforms $k = 2c$: see Section 5.3 for theoretical proof and Figure 7.2c on page 168 for empirical verification), but it *is* (2,1)-approximately unimodal (i.e. all parameter values k outperform all parameter values $k' > 2k$ [36]). In the fourth scenario, the parameter landscape induced by Best-Fitness for the given cutoff time is more complex since two parameters have to be configured. However, it appears to be approximately unimodal (Figure 7.2d on page 168).

7.5.1 Experimental Setup

We varied the size of the parameter space to investigate its impact on the performance of the different mutation operators (i.e. $\pm\{1,\ldots,m\}$, random-step operator with and without replacement, and harmonic-step). In all configuration scenarios, we measured the number of configuration comparisons before the O-optimal configuration (or, when configuring SAPS, a member of a set of near-F-optimal configurations) is first sampled. We used the Best-Fitness performance metric since it allows smaller cutoff times to be used to identify the O-optimal configuration than if we had used Optimisation-Time, reducing the overall time required for the experiments. Furthermore, for the cutoff times used, Best-Fitness induces a unimodal parameter landscape, whilst Optimisation-Time would have failed to do so. This further reduces the time required for the experiments. For the experiment involving SAPS, using Best-Fitness also meant that we did not require additional experiments to determine the time required to reach a target fitness value. Note that, in the other three experiments, we set the cutoff time large enough that the F-optimal configuration is the same as the O-optimal configuration.

For each parameter space size, the experiment for single-parameter target algorithms was repeated 200 times and the mean number of configuration comparisons was recorded. For the MAX-SAT scenario, the experiment was repeated 500 times for each parameter space size to account for the increased complexity of the configuration scenario.

The cutoff time κ was varied with the problem class. All configurators were initialised uniformly at random. For ParamILS, we used the BasicILS variant (see Section 2.8.1) and set $R = 0$ (i.e. we did not use repeated random initialisation) as preliminary experiments indicated that doing so was harmful for the configuration scenarios considered here.

In order to simplify the implementation, when configuring an algorithm with multiple parameters we modified ParamILS to select a parameter to perturb uniformly at random, as is done with the $\pm\{1,\dots,m\}$ operator, instead of selecting a new configuration uniformly at random from all that differ in a single parameter, as in its default behaviour. This increases the probability that ParamILS perturbs parameters with few values relative to others. In our experiments, this modification only affects the behaviour of ParamILS when configuring SAPS as this is the only target algorithm with multiple parameters.

Setup for Benchmark functions When configuring for RIDGE, LEADINGONES and ONEMAX, we used $n = 50$ and 1500 runs per configuration comparison (i.e. $r = 1500$). When configuring the mutation rate of the (1+1)$_\chi$ EA For RIDGE, we used a quadratic cutoff time of $\kappa = n^2 = 2500$, which our theoretical analyses indicate is more than sufficient for the identification of the O-optimal parameter value using Best-Fitness (Theorem 70, Section 6.2.3). The value of m in the $\pm\{1,\dots,m\}$ operator was set to $m = 1$. The first parameter space that we considered was $\chi \in \{0.5, 1.0, \dots, 4.5, 5.0\}$, where χ/n is the mutation rate and $\chi = 1$ is optimal for RIDGE (see Chapter 6). We increased the size of the parameter space by adding the next five largest parameter values (each increasing by 0.5) until the parameter space $\{0.5, \dots, 25.0\}$ was reached. Following Chapter 6, the (1+1) EA for RIDGE was initialised at the start of the ridge (i.e. at 0^n).

When configuring the mutation rate of the (1+1)$_\chi$ EA for LEADINGONES, we initialised the individual uniformly at random and used $m = 1$ and a quadratic cutoff time of $\kappa = n^2 = 2500$, which our theoretical analyses indicate is more than sufficient for the identification of the O-optimal parameter value using Best-Fitness (Theorem 83, Section 6.3.3). The size of the parameter space was increased in the same way as in the RIDGE experiments, and the initial parameter space was $\chi \in \{0.6, 1.1, \dots, 4.6, 5.1\}$ as the optimal value for χ is approximately 1.6 [18]. The final parameter space was $\{0.6, \dots, 25.1\}$

When configuring the neighbourhood size of RLS$_k$ for ONEMAX, we initialised the individual uniformly at random and used a linear cutoff time of $\kappa = 4n = 200$, which our theoretical analyses indicate is more than sufficient for the identification of the O-optimal parameter value using Best-Fitness (Theorem 63, Section 5.3.3, indicates that any cutoff time of $\kappa \geq 0.975n$ would suffice). The initial parameter space was $\{1, 2, \ldots, 9, 10\}$, where $k = 1$ is the optimal parameter (see Chapter 5), and the next five largest integers were added until $\{1, 2, \ldots, 49, 50\}$ was reached. Since this parameter landscape is only approximately unimodal, we used $m = 2$ (as recommended in Chapter 5).

Setup for SAPS and MAX-SAT We considered tuning two parameters of SAPS – α and ρ – for ten instances[21] of the SAT instance set `circuit-fuzz` (available in AClib [66]). Due to the complexity of the MAX-SAT problem class, it was no longer obvious which configurations should be considered optimal. Therefore we conducted an exhaustive search of the parameter space to identify which configurations perform best (within the time budget). We did so by running the validation procedure in ParamILS for each configuration with $\alpha \in \{\frac{16}{15}, \frac{17}{15}, \ldots, \frac{44}{15}, \frac{45}{15}\}$ and $\rho \in \{0, \frac{1}{15}, \ldots, \frac{14}{15}, 1\}$. Each configuration was evaluated 2000 times on each of the ten considered `circuit-fuzz` problem instances. In each evaluation, the cutoff time was $10,000$ iterations and the quality of a configuration was taken to be the number of satisfied clauses. We selected the five configurations that identify the best solutions within the time budget to be the target and analysed the time taken by a configurator to identify any one of these configurations.

Since it was not feasible to compute the quality of a configuration each time it was evaluated in a tuner, we instead took the average fitness values generated during the initial evaluation of the parameter landscape (i.e. when determining the set of optimal configurations) to be the fitness of each configuration. As these runs were repeated many times, we believe that they provide an accurate approximation of the fitness values of the configurations. It is likely that the results presented in this section would also hold if we determined the fitness of a configuration each time we evaluate it.

In this experiment, we set the range of values of ρ to $\{0, \frac{1}{15}, \ldots, \frac{14}{15}, 1\}$ and the value of the two other parameters of SAPS as $P_{smooth} = 0.05$ and $wp = 0.01$ (their default values). We then increased the size of the set of possible values of α. The initial set of values for α was $\{\frac{16}{15}, \frac{17}{15}, \frac{18}{15}\}$, which contains the values of α in all five configurations with the best performance. We then generated larger parameter spaces by adding a new value to the set of values for α until the set $\{\frac{16}{15}, \ldots, \frac{45}{15}\}$ was reached.

For this scenario, we do not include results for the performance of ParamRLS using the random-step search operator without replacement since this combination appeared to often get stuck at local optima and thus fail to identify any near-optimal configurations.

[21]Problem instances number 78, 535, 581, 582, 6593, 6965, 8669, 9659, 16905, 16079.

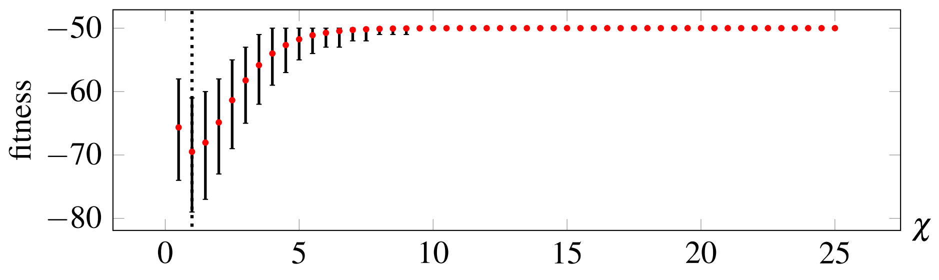
7.5.2 Results

The results from configuring the algorithms for the three benchmark functions are shown in Figures 7.3a, 7.3b, and 7.3c. Solid lines correspond to variants of ParamRLS using different search operators and dotted lines correspond to variants of ParamILS. Green lines correspond to configurators using the random-step operator (without replacement), black lines to configurators using the random-step operator (with replacement), blue lines to configurators using the $\pm\{1,\ldots,m\}$ operator, and red lines to configurators using the harmonic-step operator.

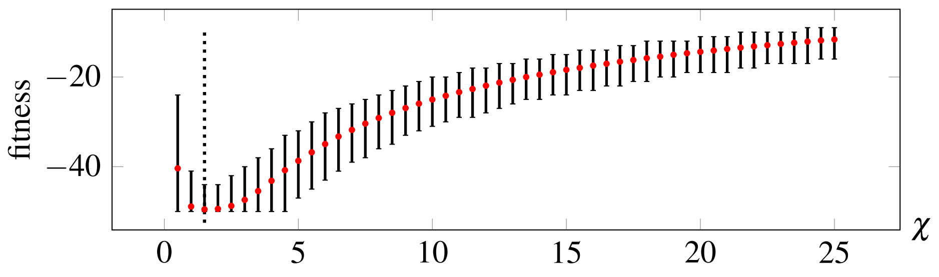
In each configuration scenario, and for both configurators, the harmonic-step operator located the optimal configuration faster than both the $\pm\{1,\ldots,m\}$ and random-step operators. For both configurators, the polylogarithmic growth of the time taken by the harmonic-step operator to locate the optimal configuration can be seen, compared to the linear growth of the time taken by the $\pm\{1,\ldots,m\}$ and random-step search operators. The difference between the performance of the operators is more pronounced when there is a plateau of neighbouring configurations all exhibiting the same performance (as is the case for RIDGE). We also verified that these improvements in performance also occur if few runs per evaluation are used, but we do not include these results.

Similar performance improvements from using the harmonic-step operator can be seen in the results for configuring SAPS for MAX-SAT. Figure 7.3d shows that the harmonic-step operator locates a near-optimal configuration faster than the others.

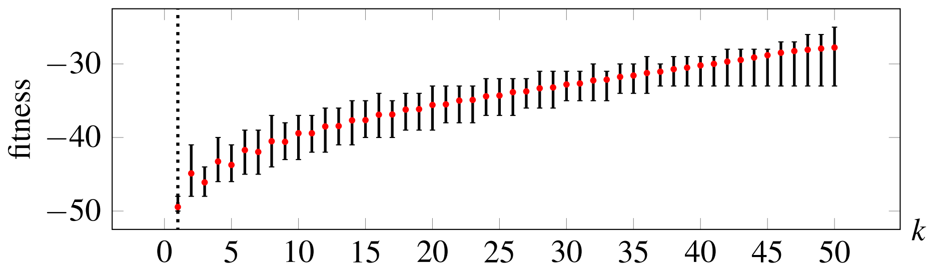
In Figure 7.3, crosses indicate where the difference between the performance of the harmonic-step operator and the other operators is statistically significant at a significance level of 0.95 (according to a two-tailed Mann-Whitney U test [94]). Their position reflects the effect size of this comparison, calculated in terms of Cliff's delta [23]. An effect size of 0 would signify that there is no difference between the two outcomes and effect sizes closer to 1 indicate a greater benefit to using the harmonic operator over the alternative (negative effect sizes would indicate that using the harmonic operator was detrimental, but this never occurred to an extent that was statistically significant). Orange crosses show the effect size of the difference between ParamILS using harmonic-step and using random-step (without replacement). The effect sizes of the differences between ParamHS and ParamRLS using $\pm\{1,\ldots,m\}$, random-step (without replacement) and random-step (with replacement) are shown by blue, green, and black crosses, respectively. In each configuration scenario, for larger parameter space sizes almost all comparisons with all other operators were statistically significant.



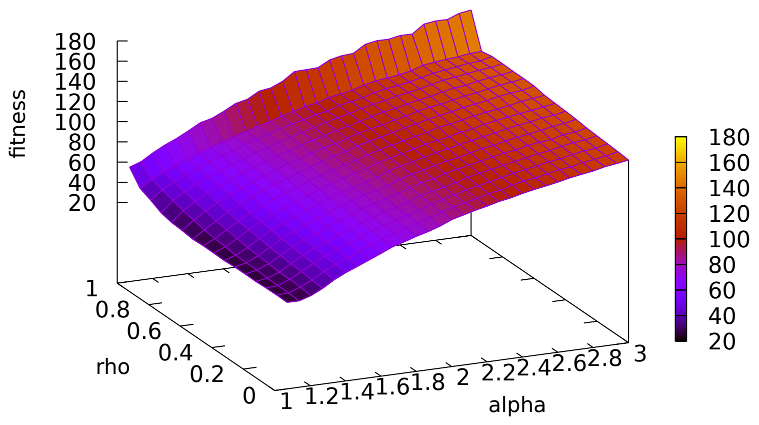
(a) $(1+1)_\chi$ EA and RIDGE, $\kappa = 2500$



(b) $(1+1)_\chi$ EA and LEADINGONES, $\kappa = 2500$

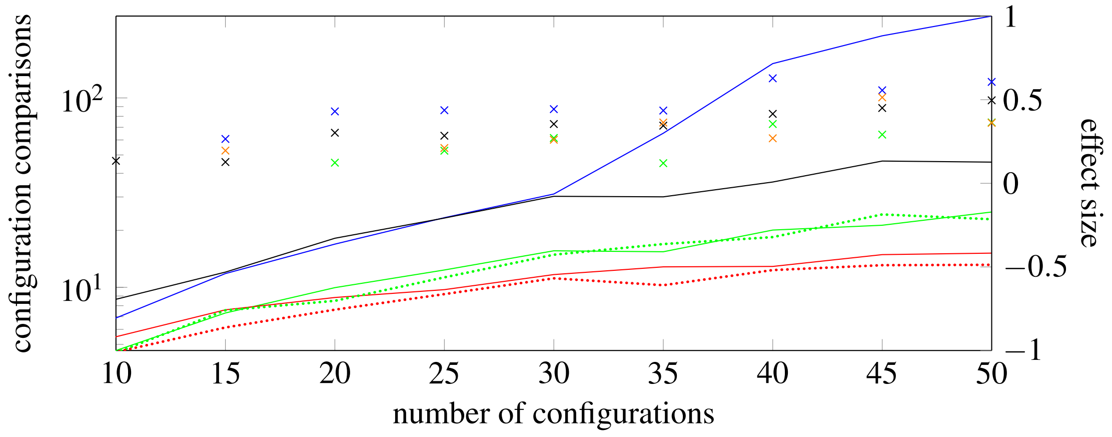


(c) RLS$_k$ and ONEMAX, $\kappa = 200$

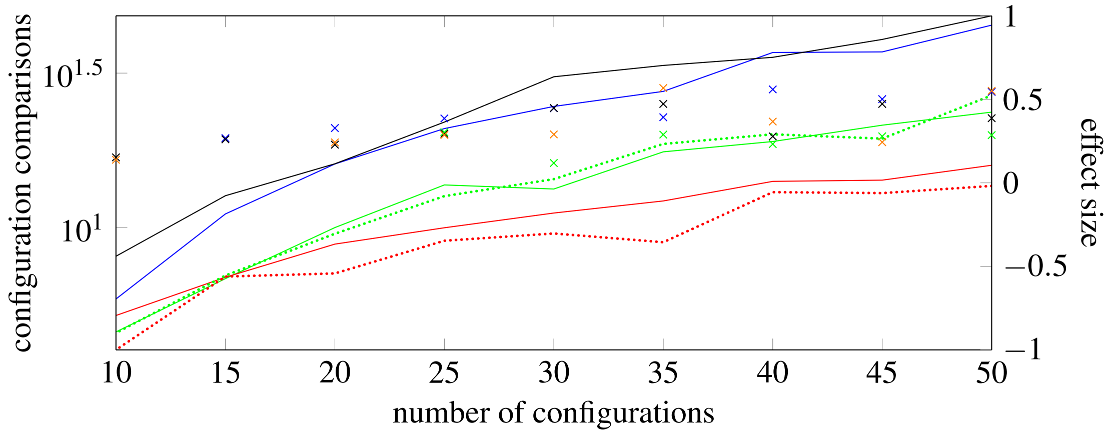


(d) SAPS and MAX-SAT, $\kappa = 10000$

Fig. 7.2 (a),(b),(c): Mean fitness of the individual in the algorithms with $n = 50$, averaged over 10,000 runs for each parameter value, multiplied by -1 to obtain a minimisation problem (as is conventional in algorithm configuration). Error bars show the range of the median 95% of fitness values. The dotted line indicates the optimal configuration for each scenario. (d): The parameter landscape for SAPS in terms of α and ρ computed for a set of ten SAT instances from the `circuit-fuzz` dataset. In all figures, lower values are better.



(a) Configuring the $(1+1)_\chi$ EA for RIDGE with $\kappa = 2500$ and $r = 1500$.



(b) Configuring the $(1+1)_\chi$ EA for LEADINGONES with $\kappa = 2500$ and $r = 1500$.

Configurator variants:

—— ParamRLS variants ······ ParamILS variants

Search operator used:

—— random-step (without replacement) —— $\pm\{1,\ldots,m\}$

—— random-step (with replacement) —— harmonic-step

Effect size indicators:

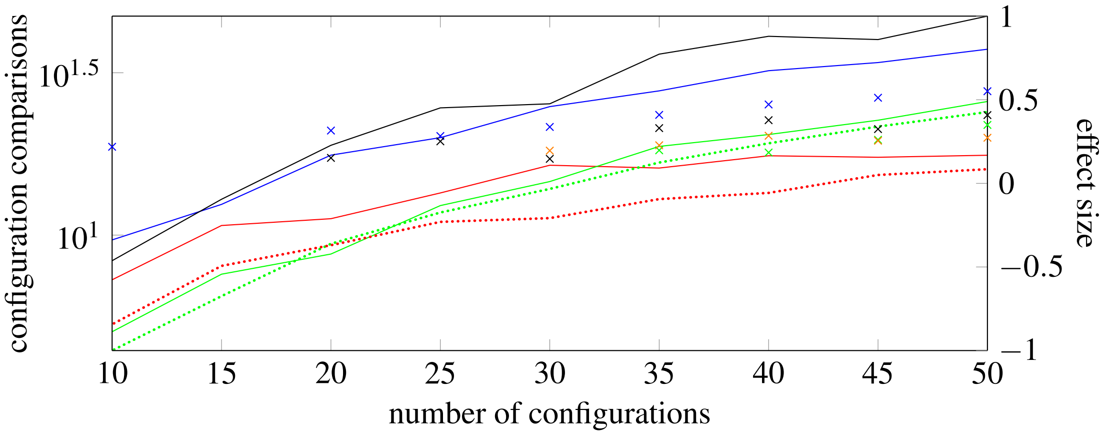
× ParamHS vs. random-step ParamRLS (without replacement)

× ParamHS vs. $\pm\{1,\ldots,m\}$ ParamRLS

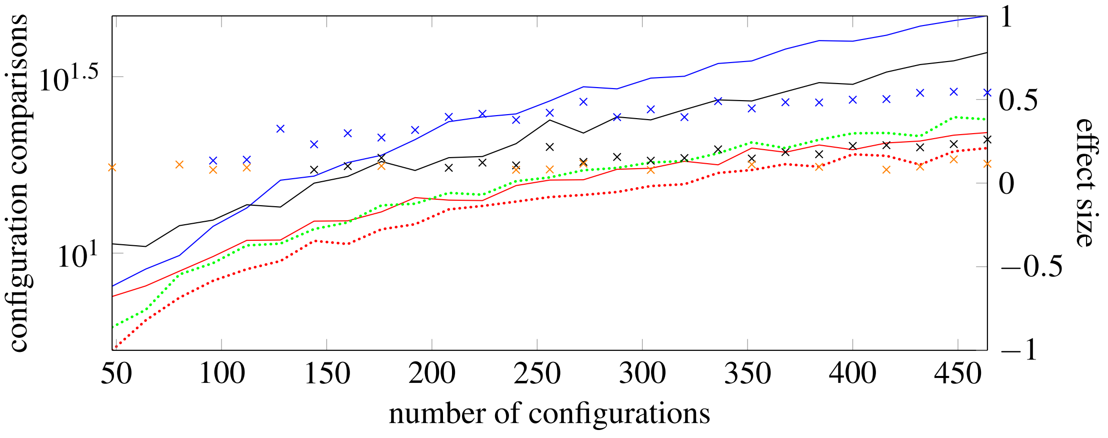
× ParamHS vs. random-step ParamRLS (with replacement)

× harmonic ParamILS vs. default ParamILS

Fig. 7.3 Mean number of configuration comparisons before sampling an optimal configuration.



(c) Configuring RLS_k for ONEMAX with $\kappa = 200$ and $r = 1500$.



(d) Configuring the α and ρ parameters of SAPS for the `circuit-fuzz` MAX-SAT instance set, using cached evaluations with $\kappa = 10{,}000$ and $r = 20{,}000$.

Configurator variants:

—— ParamRLS variants ⋯⋯ ParamILS variants

Search operator used:

—— random-step (without replacement) —— $\pm\{1,\ldots,m\}$

—— random-step (with replacement) —— harmonic-step

Effect size indicators:

× ParamHS vs. random-step ParamRLS (without replacement)

× ParamHS vs. $\pm\{1,\ldots,m\}$ ParamRLS

× ParamHS vs. random-step ParamRLS (with replacement)

× harmonic ParamILS vs. default ParamILS

Fig. 7.3 Mean number of configuration comparisons before sampling an optimal configuration (or a near-optimal configuration in the case of tuning SAPS).

7.6 Conclusions

In this chapter, we demonstrated that ParamRLS and ParamILS benefit if their default mutation operators are replaced with one that uses a harmonic distribution to exploit unimodal parameter landscapes. We proved considerable asymptotic speed-ups for unimodal and approximately unimodal parameter landscapes, whilst in the worst case (e.g. for deceptive parameter landscapes) the proposed modification will only slow down the configurator by at most logarithmic factor. We verified experimentally that this speed-up occurs in practice for benchmark parameter landscapes that are known to be unimodal and approximately unimodal, as well as for tuning a MAX-SAT solver for a well-studied benchmark set.

In parallel independent work, Pushak and Hoos developed a configurator called Golden Parameter Search (GPS) that is also designed to exploit gradients in the parameter landscape [105]. It uses golden section search [76], a method that is optimal for finding the minimum of a unimodal function in the worst-case [103]. They show experimentally that GPS outperforms *irace*, SMAC, and GGA for tuning travelling salesperson problem, SAT, and mixed integer programming solvers. However, GPS assumes that the parameter landscape is unimodal, and therefore, unlike our method, may fail if the parameter landscape violates this assumption too strongly.

Part III

Conclusions

Chapter 8

Conclusions and Outlook

8.1 Summary of Work

In this thesis, we have created a framework that lays the foundations of the rigorous runtime analysis of algorithm configurators. We have used it to analyse the impact of the settings of an algorithm configurator (namely the cutoff time and performance metric) on its ability to tune algorithms and the time taken to do so. Such analyses provide statements regarding how long a configurator must be run to identify good parameter values and the quality of the parameter values that it is expected to return. Furthermore, they allow for a deeper understanding of which components of a configurator are beneficial and which are detrimental. We then used these theoretical insights to a design mutation operator with better performance for configuration scenarios with approximately unimodal gradients towards optimal parameter values.

We began by analysing the impact of the cutoff time in the general case, proving a lower bound on the cutoff time necessary for any configurator that uses the Fixed-Target performance metric. We proved that if the cutoff time is too small to allow any configuration to reach the target solution quality then the configurator is "blind" (i.e. returns a random configuration). This implies that one must be careful when choosing the cutoff time when using this performance metric since the negative impact of choosing too small a cutoff time is drastic. This reliance on an appropriate choice of cutoff time implies that significant problem-specific knowledge is required, namely the time required to reach the target solution qualities (at least for the best configuration in the neighbourhood of the incumbent). Furthermore, if the Optimisation-Time performance metric is employed then only training instances for which optimal values are known can be used. By definition, this is not possible when configuring algorithms in cases where information about the problem at hand is not available. Such so-called black-box settings are common use cases of algorithm configuration, i.e. the configuration of general-purpose algorithms for real-world scenarios. We then proved that, when configuring unary unbiased algorithms with a training set where each instance contains at most $\exp(\sqrt{n}/\log^2 n)$ optima (where n is the largest instance size), a cutoff time of at least $(n \ln n)/2$ is necessary, otherwise any Optimisation-Time-based configurator will be blind. We subsequently showed that, when using the Fixed-Target performance metric in a setting where the optimal configuration has a smaller runtime than every other configuration with overwhelming probability, a cutoff time large enough to allow this configuration to reach the target solution quality for every problem instance with overwhelming probability is sufficient for it to be returned by the configurator. However, if the target solution quality is chosen as the optimum, then this cutoff time is also sufficient for Best-Fitness-based configurators to return the same configuration.

In Chapter 5, we considered the configuration of the neighbourhood size of randomised local search (RLS$_k$) for two benchmark problem classes, RIDGE and ONEMAX. We proved that for any cutoff time ParamRLS using the Best-Fitness performance metric is able to identify that $k = 1$ achieves the highest solution quality within the time budget if sufficiently many runs are used in each configuration evaluation. A single run is sufficient if the cutoff time is $\Omega(n^{1+\varepsilon})$, for constant $\varepsilon > 0$. Any configurator using the Optimisation-Time performance metric is blind if the cutoff time is at most $(1-\varepsilon)n^2$. This implies that not only is ParamRLS-F able to identify the optimal configuration with respect to the Best-Fitness performance metric for all cutoff times, but that it is also able to do so with respect to the Optimisation-Time metric for ranges of cutoff times for which all configurators using Optimisation-Time are blind.

The situation is more complex when configuring for ONEMAX, since different neighbourhood sizes maximise the expected progress depending on the distance to the optimum. As is often the case with randomised search heuristics, it is optimal to decrease the neighbourhood size as the optimum is approached. When the set of permitted values for k is $\{1,2,3,4,5\}$, we showed that, for a cutoff time κ satisfying $0.02n \le \kappa \le 0.72n$, ParamRLS-F is able to identify that $k = 5$ is F-optimal, whilst for cutoff times satisfying $\kappa \ge 0.975n$ it is able to identify that $k = 1$ is F-optimal. As already shown, configurators using the Optimisation-Time performance metric, on the other hand, are blind for cutoff times of $(n \ln n)/2$. Thus, ParamRLS-F is again able to identify the configuration that is optimal under the Optimisation-Time performance metric (i.e. $k = 1$) for cutoff times that are a logarithmic factor smaller than those required by configurators using that performance metric.

In order to investigate whether our findings presented in Chapter 5 also apply to more complex configuration scenarios, in Chapter 6 we analysed the configuration of the mutation rate χ of the (1+1) EA, which uses the continuously-valued standard bit mutation operator.

For RIDGE, the optimal mutation rate for the (1+1) EA is always $\chi = 1$, independent of the cutoff time and performance metric, since this value maximises expected progress. We proved that ParamRLS-F is able to identify that this mutation rate achieves the highest solution quality for any cutoff time $\kappa \geq \varepsilon n$, for constant $\varepsilon > 0$. On the other hand, configurators using the Optimisation-Time performance metric are blind for cutoff times $\kappa \leq (1-\varepsilon)en^2$, for constant $0 < \varepsilon < 1$, and hence ParamRLS-F can identify the configuration that is optimal under the Optimisation-Time metric in cutoff times that are by a linear factor smaller than those required by configurators using Optimisation-Time itself.

As with ONEMAX, for LEADINGONES the optimal amount by which to perturb a solution (in order to maximise expected progress) decreases as the optimum is approached. We proved that ParamRLS-F is once again able to identify the F-optimal mutation rate. In order to prove this result, it was again necessary to consider a specific instantiation of the parameter space. We considered the parameter space $\{0.1, 0.2, \ldots, 2.9, 3.0\}$, for which the parameter value that minimises the expected optimisation time is $\chi = 1.6$ [18]. For almost all quadratic (and all super-quadratic) cutoff times, we proved that the parameter landscape induced by the Best-Fitness performance metric is unimodal. This implies that ParamRLS-F can simply follow the gradient towards the optimal parameter value for that cutoff time. We then proved that any configurator using the Optimisation-Time performance metric is blind for any cutoff time $\kappa \leq 0.772n^2$, whilst ParamRLS-F successfully identifies the optimal configuration under this performance metric using cutoff times that are smaller by $\approx 0.05n^2$.

Our analyses revealed that, for each of the four configuration scenarios considered in this work, the parameter landscape seen by ParamRLS-F is approximately unimodal. This is the first time that such a result has been rigorously proven, although previous experimental work has suggested this to be the case, even for some complex configuration scenarios such as configuring SAT, TSP, and MIP solvers [104] as well as particle swarm algorithms for 20 standard benchmark problems [55]. In Chapter 7, we investigated the performance improvements that can be gained from our theoretical insights by designing mutation operators that provably allow configurators to perform well on such landscapes. We proved that the default mutation operators of ParamRLS (i.e. using small step sizes and hence being slow to exploit gradients, whilst also being liable to get stuck in local optima) and ParamILS (i.e. using random step sizes and therefore being unable to exploit the gradient of parameter landscapes) both require a linear expected number of evaluations. We then proved that the configurators using our proposed harmonic mutation operator identify the optimal configuration of any single-parameter algorithm in polylogarithmic expected time if the parameter landscape is approximately unimodal. Moreover, if the landscape is unimodal then the expected configuration time is reduced to logarithmic. We also proved that in the worst case ParamHS is slower than ParamRLS and ParamILS by only a logarithmic factor in expectation. We verified these speed-ups experimentally, showing that, in the configuration scenarios considered in earlier chapters, both ParamRLS and ParamILS modified to use this new operator identify the optimal configuration faster than their default versions. Furthermore, we also observed these performance improvements when configuring a SAT solver with more than one parameter.

8.2 Future Work

In this thesis, we have laid the foundations for the rigorous runtime analysis of algorithm configurators and we have derived the first results in this direction. However, there remain many open questions that will give further insights into the effects of design choices of algorithm configurators on their ability to identify good parameter configurations and the time required to do so. In this section, we identify potential directions of future research.

1. *Analysis of state-of-the-art configurators.* We have provided bounds on the runtimes of ParamRLS and ParamILS for a range of configuration scenarios. Our work should be built upon to derive similar statements regarding state-of-the-art configurators such as SMAC and *irace*. Whilst such configurators are more challenging to analyse, given the numerous successful applications reported it is crucial to derive a theoretical understanding of their performance for an informed development of even better performing algorithm configurators.

2. *Analysis of more complex configuration scenarios.* The problem classes considered in this thesis are standard benchmark functions used in the analysis of randomised search heuristics. Our work should be extended to the analysis of classical problems from combinatorial optimisation for which configurators are often used in practice. For instance, SAT is probably the most commonly considered problem in experimental algorithm configuration research. The target algorithms that we have considered all have only a single parameter, whereas in practice it is common to tune algorithms with many parameters. It would be of great impact to derive rigorous statements regarding the structure of the parameter landscapes in these more complex scenarios. In particular, configuration scenarios that require the more sophisticated features of state-of-the-art configurators should be identified in order to justify their need and use. However, conducting the necessary fixed-budget analyses to enable the analysis of more complex configuration scenarios is highly non-trivial, as even for the relatively

simple benchmark problem classes analysed in this work the required fixed-budget analysis was surprisingly challenging.

3. *Identify configuration scenarios where the Best-Fitness performance metric is harmful.* In this thesis, we have frequently shown that the Best-Fitness performance metric can identify the same parameter values as Optimisation-Time whilst using smaller cutoff times and that it is not adversely affected by using cutoff times that are too small. However, it is likely that it is not always the case that it is beneficial to use Best-Fitness instead of Fixed-Target. An interesting line of future work would be to identify scenarios where this is the case.

4. *Modify other configurators to exploit unimodal parameter landscapes.* We have shown the benefits that can be gained by modifying ParamRLS and ParamILS to use a search operator tailored to perform well on unimodal fitness landscapes. A natural direction of future work is therefore to identify whether other configurators, such as *irace* and SMAC, would also benefit from being tailored to perform well on such landscapes.

References

[1] Belarmino Adenso-Díaz and Manuel Laguna. Fine-tuning of algorithms using fractional experimental designs and local search. *Operations research*, 54(1):99–114, 2006.

[2] Carlos Ansótegui, Meinolf Sellmann, and Kevin Tierney. A gender-based genetic algorithm for the automatic configuration of algorithms. In *International Conference on Principles and Practice of Constraint Programming (CP '09)*, pages 142–157. Springer, 2009.

[3] Denis Antipov and Benjamin Doerr. A tight runtime analysis for the $(\mu + \lambda)$ EA. *Algorithmica*, 2020.

[4] Charles Audet and John E. Dennis, Jr. Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on optimization*, 17(1):188–217, 2006.

[5] Anne Auger and Benjamin Doerr, editors. *Theory of Randomized Search Heuristics: Foundations and Recent Developments*. World Scientific, 2011.

[6] Golnaz Badkobeh, Per Kristian Lehre, and Dirk Sudholt. Black-box complexity of parallel search with distributed populations. In *Proceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII (FOGA '15)*, pages 3–15. ACM, 2015.

[7] Prasanna Balaprakash, Mauro Birattari, and Thomas Stützle. Improvement strategies for the F-Race algorithm: Sampling design and iterative refinement. In *International workshop on hybrid metaheuristics*, pages 108–122. Springer, 2007.

[8] Maria-Florina Balcan, Dan DeBlasio, Travis Dick, Carl Kingsford, Tuomas Sandholm, and Ellen Vitercik. How much data is sufficient to learn high-performing algorithms? generalization guarantees for data-driven algorithm design. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC '21)*, pages 919–932, 2021.

[9] Maria-Florina Balcan, Tuomas Sandholm, and Ellen Vitercik. Refined bounds for algorithm configuration: The knife-edge of dual class approximability. In *International Conference on Machine Learning (ICML '20)*, pages 580–590. PMLR, 2020.

[10] Thomas Bartz-Beielstein, Christian W. G. Lasarczyk, and Mike Preuß. Sequential parameter optimization. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation (CEC '05)*, volume 1, pages 773–780. IEEE, 2005.

[11] Thomas Bartz-Beielstein, Christian W. G. Lasarczyk, and Mike Preuß. The sequential parameter optimization toolbox. In *Experimental Methods for the Analysis of Optimization Algorithms*, pages 337–362. Springer, 2010.

[12] Thomas Bartz-Beielstein, Martin Zaefferer, and Frederik Rehbach. In a nutshell – the sequential parameter optimization toolbox, 2021.

[13] Mauro Birattari. On the estimation of the expected performance of a metaheuristic on a class of instances. Technical report, 2004.

[14] Mauro Birattari. *Tuning metaheuristics: a machine learning perspective*. Springer, 2009.

[15] Mauro Birattari, Thomas Stützle, Luis Paquete, and Klaus Varrentrapp. A racing algorithm for configuring metaheuristics. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation (GECCO '02)*, pages 11–18. Morgan Kaufmann Publishers Inc., 2002.

[16] Jakob Bossek, Pascal Kerschke, and Heike Trautmann. A multi-objective perspective on performance assessment and automated selection of single-objective optimization algorithms. *Applied Soft Computing*, 88:105901, 2020.

[17] Jakob Bossek and Dirk Sudholt. Time complexity analysis of RLS and (1+1) EA for the edge coloring problem. In *Proceedings of the 15th ACM/SIGEVO Conference on Foundations of Genetic Algorithms (FOGA '19)*, pages 102–115, 2019.

[18] Süntje Böttcher, Benjamin Doerr, and Frank Neumann. Optimal fixed and adaptive mutation rates for the LeadingOnes problem. pages 1–10. Springer Berlin Heidelberg, 2010.

[19] Edmund K. Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.

[20] Nathan Buskulic and Carola Doerr. Maximizing drift is not optimal for solving OneMax. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '19)*, pages 425–426, 2019.

[21] Leslie Pérez Cáceres, Manuel López-Ibáñez, Holger Hoos, and Thomas Stützle. An experimental study of adaptive capping in irace. In *International Conference on Learning and Intelligent Optimization*, pages 235–250. Springer, 2017.

[22] Laura Calvet, Angel A. Juan, Carles Serrat, and Jana Ries. A statistical learning based approach for parameter fine-tuning of metaheuristics. *SORT-Statistics and Operations Research Transactions*, pages 201–224, 2016.

[23] Norman Cliff. Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological bulletin*, 114(3):494, 1993.

[24] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT press, 2009.

[25] Dogan Corus, Duc-Cuong Dang, Anton V. Eremeev, and Per Kristian Lehre. Level-based analysis of genetic algorithms and other search processes. *IEEE Transactions on Evolutionary Computation*, 22(5):707–719, 2017.

[26] Dogan Corus, Pietro S. Oliveto, and Donya Yazdani. Fast artificial immune systems. In *Parallel Problem Solving from Nature (PPSN '18)*, pages 67–78, 2018.

[27] Dogan Corus, Pietro S. Oliveto, and Donya Yazdani. Artificial immune systems can find arbitrarily good approximations for the NP-hard number partitioning problem. *Artificial Intelligence*, 247:180–196, 2019.

[28] Dogan Corus, Pietro S. Oliveto, and Donya Yazdani. When hypermutations and ageing enable artificial immune systems to outperform evolutionary algorithms. *Theoretical Computer Science*, 832:166–185, 2020.

[29] Karel Crombecq, Eric Laermans, and Tom Dhaene. Efficient space-filling and non-collapsing sequential design strategies for simulation-based modeling. *European Journal of Operational Research*, 214(3):683–696, 2011.

[30] Kenneth A. de Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* PhD thesis, 1975.

[31] Martin Dietzfelbinger, Jonathan E. Rowe, Ingo Wegener, and Philipp Woelfel. Precision, local search and unimodal functions. *Algorithmica*, 59(3):301–322, 2011.

[32] Benjamin Doerr. Probabilistic tools for the analysis of randomized optimization heuristics. In Benjamin Doerr and Frank Neumann, editors, *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, pages 1–87. Springer International Publishing, 2020.

[33] Benjamin Doerr, Carola Doerr, and Timo Kötzing. The right mutation strength for multi-valued decision variables. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016 (GECCO '16)*, pages 1115–1122. ACM, 2016.

[34] Benjamin Doerr, Carola Doerr, and Timo Kötzing. Static and self-adjusting mutation strengths for multi-valued decision variables. *Algorithmica*, 80:1732–1768, 2018.

[35] Benjamin Doerr, Carola Doerr, and Jing Yang. Optimal parameter choices via precise black-box analysis. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016 (GECCO '16)*, pages 1123–1130. ACM, 2016.

[36] Benjamin Doerr, Carola Doerr, and Jing Yang. Optimal parameter choices via precise black-box analysis. *Theoretical Computer Science*, 801:1–34, 2020.

[37] Benjamin Doerr and Leslie Ann Goldberg. Adaptive drift analysis. *Algorithmica*, 65(1):224–250, 2013.

[38] Benjamin Doerr, Thomas Jansen, Carsten Witt, and Christine Zarges. A method to derive fixed budget results from expected optimisation times. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 1581–1588. ACM, 2013.

[39] Benjamin Doerr, Daniel Johannsen, and Carola Winzen. Multiplicative drift analysis. *Algorithmica*, 64(4):673–697, 2012.

[40] Benjamin Doerr, Huu Phuoc Le, Régis Makhmara, and Ta Duy Nguyen. Fast genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '17)*, pages 777–784. ACM, 2017.

[41] Benjamin Doerr and Frank Neumann, editors. *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*. Springer Nature, 2019.

[42] Stefan Droste, Thomas Jansen, Karsten Tinnefeld, and Ingo Wegener. A new framework for the valuation of algorithms for black-box optimization. In *Proceedings of the Seventh Workshop on Foundations of Genetic Algorithms (FOGA '02)*, pages 253–270. Morgan Kaufmann Publishers Inc., 2002.

[43] Stefan Droste, Thomas Jansen, and Ingo Wegener. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 276(1-2):51–81, 2002.

[44] Katharina Eggensperger, Marius Lindauer, Holger H. Hoos, Frank Hutter, and Kevin Leyton-Brown. Efficient benchmarking of algorithm configurators via model-based surrogates. *Machine Learning*, 107(1):15–41, 2018.

[45] Ágoston E. Eiben, Robert Hinterding, and Zbigniew Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, 1999.

[46] Ágoston E. Eiben and James E. Smith. *Introduction to Evolutionary Computing*. Springer, 2015.

[47] William Feller. *An introduction to probability theory and its applications: volume I*. John Wiley & Sons New York, 1968.

[48] Ronald A. Fisher. *The Design of Experiments*. Oliver and Boyd, 1935.

[49] John J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 16(1):122–128, 1986.

[50] Bruce Hajek. Hitting-time and occupation-time bounds implied by drift analysis with applications. *Advances in Applied probability*, 14(3):502–525, 1982.

[51] George T. Hall, Pietro S. Oliveto, and Dirk Sudholt. 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. ACM, 2019.

[52] George T. Hall, Pietro S. Oliveto, and Dirk Sudholt. Analysis of the performance of algorithm configurators for search heuristics with global mutation operators. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '20)*, pages 823–831. ACM, 2020.

[53] George T. Hall, Pietro S. Oliveto, and Dirk Sudholt. Fast perturbative algorithm configurators. In *Proceedings of the International Conference on Parallel Problem Solving from Nature (PPSN '20)*, pages 19–32. Springer, 2020.

[54] Nikolaus Hansen. The CMA evolution strategy: a comparing review. In *Towards a new evolutionary computation*, pages 75–102. Springer, 2006.

[55] Kyle R. Harrison, Beatrice M. Ombuki-Berman, and Andries P. Engelbrecht. The parameter configuration landscape: A case study on particle swarm optimization. In *2019 IEEE Congress on Evolutionary Computation (CEC '19)*, pages 808–814. IEEE, 2019.

[56] Jun He and Xin Yao. Drift analysis and average time complexity of evolutionary algorithms. *Artificial Intelligence*, 127(1):57–85, 2001.

[57] Jun He and Xin Yao. A study of drift analysis for estimating computation time of evolutionary algorithms. *Natural Computing*, 3(1):21–35, 2004.

[58] Jorge Pérez Heredia. Modelling evolutionary algorithms with stochastic differential equations. *Evolutionary computation*, (26.4):1–30.

[59] Holger H. Hoos. Automated algorithm configuration and parameter tuning. In *Autonomous search*, pages 37–71. Springer, 2011.

[60] Holger H. Hoos and Thomas Stützle. *Stochastic local search: Foundations and applications*. Elsevier, 2004.

[61] Changwu Huang, Yuanxiang Li, and Xin Yao. A survey of automatic parameter tuning methods for metaheuristics. *IEEE Transactions on Evolutionary Computation*, 24(2):201–216, 2020.

[62] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36(1):267–306, 2009.

[63] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. Technical report, 2010. Technical Report TR-2010-10, University of British Columbia Computer Science.

[64] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization (LION '11)*, pages 507–523. Springer, 2011.

[65] Frank Hutter, Marius Lindauer, Adrian Balint, Sam Bayless, Holger H. Hoos, and Kevin Leyton-Brown. The configurable sat solver challenge (CSSC). *Artificial Intelligence*, 243:1–25, 2017.

[66] Frank Hutter, Manuel López-Ibáñez, Chris Fawcett, Marius Lindauer, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. AClib: A benchmark library for algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pages 36–40. Springer, 2014.

[67] Frank Hutter, Dave A. D. Tompkins, and Holger H. Hoos. Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In *International Conference on Principles and Practice of Constraint Programming*, pages 233–248. Springer, 2002.

[68] Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *Artificial Intelligence and Statistics*, pages 240–248, 2016.

[69] Thomas Jansen. *Analyzing evolutionary algorithms: The computer science perspective*. Springer Science & Business Media, 2013.

[70] Thomas Jansen. Analysing stochastic search heuristics operating on a fixed budget. In Benjamin Doerr and Frank Neumann, editors, *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, pages 249–270. Springer International Publishing, 2020.

[71] Thomas Jansen and Ingo Wegener. A comparison of simulated annealing with a simple evolutionary algorithm on pseudo-boolean functions of unitation. *Theoretical Computer Science*, 386(1-2):73–93, 2007.

[72] Thomas Jansen and Christine Zarges. Fixed budget computations: A different perspective on run time analysis. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, pages 1325–1332. ACM, 2012.

[73] Thomas Jansen and Christine Zarges. Performance analysis of randomised search heuristics operating with a fixed budget. *Theoretical Computer Science*, 545:39–58, 2014.

[74] Daniel Johannsen. *Random Combinatorial Structures and Randomized Search Heuristics*. PhD thesis, 2010.

[75] Giorgos Karafotias, Mark Hoogendoorn, and Ágoston E. Eiben. Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Transactions on Evolutionary Computation*, 19(2):167–187, 2014.

[76] Jack Kiefer. Sequential minimax search for a maximum. *Proceedings of the American mathematical society*, 4(3):502–506, 1953.

[77] Scott Kirkpatrick, C. Daniel Gelatt, and Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[78] Robert Kleinberg, Kevin Leyton-Brown, and Brendan Lucier. Efficiency through procrastination: Approximately optimal algorithm configuration with runtime guarantees. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI '17)*, pages 2023–2031. AAAI Press, 2017.

[79] Robert Kleinberg, Kevin Leyton-Brown, Brendan Lucier, and Devon Graham. Procrastinating with confidence: Near-optimal, anytime, adaptive algorithm configuration. In *Advances in Neural Information Processing Systems (NeurIPS '19)*, pages 8881–8891, 2019.

[80] Donald E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 6: Satisfiability*. Addison-Wesley Professional, 2015.

[81] Vladimir Koltchinskii. Rademacher penalties and structural risk minimization. *IEEE Transactions on Information Theory*, 47(5):1902–1914, 2001.

[82] Timo Kötzing and Carsten Witt. Improved fixed-budget results via drift analysis. In *Proceedings of the International Conference on Parallel Problem Solving from Nature (PPSN '20)*, pages 648–660. Springer International Publishing, 2020.

[83] Pedro Larrañaga and Jose A. Lozano. *Estimation of distribution algorithms: A new tool for evolutionary computation*. Springer, 2001.

[84] Per Kristian Lehre and Dirk Sudholt. Parallel black-box complexity with tail bounds. *IEEE Transactions on Evolutionary Computation*, 2019.

[85] Per Kristian Lehre and Carsten Witt. Black-box search by unbiased variation. *Algorithmica*, 64(4):623–642, 2012.

[86] Johannes Lengler. Drift analysis. In Benjamin Doerr and Frank Neumann, editors, *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, pages 89–131. Springer International Publishing, 2020.

[87] Johannes Lengler and Nicholas Spooner. Fixed budget performance of the (1+1) EA on linear functions. In *Proceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII (FOGA '15)*, pages 52–61. ACM, 2015.

[88] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

[89] Andrei Lissovoi, Pietro S. Oliveto, and John Alasdair Warwicker. How the duration of the learning period affects the performance of random gradient selection hyper-heuristics. In *Proceeding of the Thirty-Fourth AAAI Conference on Artificial Intelligence*, pages 2376–2383. AAAI Press, 2020.

[90] Shengcai Liu, Ke Tang, Yunwei Lei, and Xin Yao. On performance estimation in automatic algorithm configuration. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence*, pages 2384–2391. AAAI Press, 2020.

[91] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[92] Manuel López-Ibáñez and Thomas Stützle. Automatically improving the anytime behaviour of optimisation algorithms. *European Journal of Operational Research*, 235(3):569–582, 2014.

[93] Helena R. Lourenço, Olivier C. Martin, and Thomas Stützle. Iterated local search. In *Handbook of Metaheuristics*, pages 320–353. Springer, 2003.

[94] Henry B. Mann and Donald R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, pages 50–60, 1947.

[95] Oded Maron and Andrew W. Moore. Hoeffding races: Accelerating model selection search for classification and function approximation. In *Advances in Neural Information Processing Systems 6 (NIPS '93)*, pages 59–66. Morgan Kaufmann Publishers Inc., 1993.

[96] Volodymyr Mnih, Csaba Szepesvári, and Jean-Yves Audibert. Empirical bernstein stopping. In *Proceedings of the 25th International Conference on Machine learning (ICML '08)*, pages 672–679, 2008.

[97] Frank Neumann and Carsten Witt. *Bioinspired Computation in Combinatorial Optimization: Algorithms and Their Computational Complexity*. Springer, 2010.

[98] Pietro S. Oliveto, Jun He, and Xin Yao. Time complexity of evolutionary algorithms for combinatorial optimization: A decade of results. *International Journal of Automation and Computing*, 4(3):281–293, 2007.

[99] Pietro S. Oliveto and Carsten Witt. Simplified drift analysis for proving lower bounds in evolutionary computation. *Algorithmica*, 59(3):369–386, 2011.

[100] Pietro S. Oliveto and Carsten Witt. Erratum: Simplified drift analysis for proving lower bounds in evolutionary computation. *arXiv preprint arXiv:1211.7184*, 2012.

[101] David Pollard. *Convergence of Stochastic Processes*. Springer, 1984.

[102] Michael J. D. Powell. The BOBYQA algorithm for bound constrained optimization without derivatives. Technical Report DAMTP2009/NA06, University of Cambridge, 2009.

[103] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C (2nd Ed.): The Art of Scientific Computing*. Cambridge University Press, 1992.

[104] Yasha Pushak and Holger H. Hoos. Algorithm configuration landscapes: - more benign than expected? In *Proceedings of the International Conference on Parallel Problem Solving from Nature (PPSN '18)*, pages 271–283. Springer, 2018.

[105] Yasha Pushak and Holger H. Hoos. Golden parameter search: exploiting structure to quickly configure parameters in parallel. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference (GECCO '20)*, pages 245–253, 2020.

[106] Jonathan E. Rowe and Dirk Sudholt. The choice of the offspring population size in the $(1,\lambda)$ evolutionary algorithm. *Theoretical Computer Science*, 545:20–38, 2014.

[107] Ranjit K. Roy. *A primer on the Taguchi method*. Society of Manufacturing Engineers, 2010.

[108] Christian Scheideler. *Probabilistic Methods for Coordination Problems*. HNI-Verlagsschriftenreihe 78, University of Paderborn, 2000. Habilitation Thesis.

[109] Selmar K. Smit and Ágoston E. Eiben. Comparing parameter tuning methods for evolutionary algorithms. In *2009 IEEE Congress on Evolutionary Computation (CEC '09)*, pages 399–406. IEEE, 2009.

[110] Selmar K. Smit and Ágoston E. Eiben. Beating the 'world champion'evolutionary algorithm via REVAC tuning. In *IEEE Congress on Evolutionary Computation (CEC '10)*, pages 1–8. IEEE, 2010.

[111] Thomas Stützle and Manuel López-Ibáñez. *Automated Design of Metaheuristic Algorithms*, pages 541–579. Springer International Publishing, 2019.

[112] Ryoji Tanabe. Analyzing adaptive parameter landscapes in parameter adaptation methods for differential evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '20)*, page 645–653. Association for Computing Machinery, 2020.

[113] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.

[114] Ingo Wegener. Methods for the analysis of evolutionary algorithms on pseudo-boolean functions. In *Evolutionary optimization*, pages 349–369. Springer, 2003.

[115] Gellért Weisz, András György, and Csaba Szepesvári. LeapsAndBounds: A method for approximately optimal algorithm configuration. In *Proceedings of the 35th International Conference on Machine Learning (ICML '18)*, pages 5254–5262. PMLR, 2018.

[116] Gellért Weisz, András György, and Csaba Szepesvári. CapsAndRuns: An improved method for approximately optimal algorithm configuration. In *Proceedings of the 36th International Conference on Machine Learning (ICML '19)*, pages 6707–6715. PMLR, 2019.

[117] Carsten Witt. Tight bounds on the optimization time of a randomized search heuristic on linear functions. *Combinatorics, Probability & Computing*, 22(2):294–318, 2013.

[118] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[119] Zhi Yuan, Marco A. Montes De Oca, Mauro Birattari, and Thomas Stützle. Continuous optimization algorithms for tuning real and integer parameters of swarm intelligence algorithms. *Swarm Intelligence*, 6(1):49–75, 2012.

[120] Eckart Zitzler and Lothar Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, 1999.

Appendices

Appendix A: Learning Theory Tools

The results in Section 2.9.1 employ tools commonly used in learning theory to quantify the complexity of a class of functions. For completeness, we define these in this section. We define three complexity measures, *pseudo-dimension* [101], *VC-dimension* [113], and *empirical Rademacher complexity* [81].

We first define pseudo-dimension, following [8]. However, before doing so it is first necessary to define the notion of shattering.

Definition 12 (Shattering)**.** Let $\mathcal{H} \subseteq [0,\mathcal{H}]^{\mathcal{Y}}$ be a set of functions mapping an abstract domain $\mathcal{Y}$ to an interval $[0,H]$. Let $\mathcal{S} = \{y_1,\dots,y_m\}$ be a subset of $\mathcal{Y}$ and let $z_1,\dots,z_m \in \mathbb{R}$ be a set of targets. We say that $z_1,\dots,z_m$ witness the shattering of $\mathcal{S}$ by $\mathcal{H}$ if for all subsets $T \subseteq \mathcal{S}$, there exists some function $h \in \mathcal{H}$ such that for all elements $y_i \in T$, $h(y_i) \leq z_i$ and for all $x_i \notin T$, $h(y_i) > z_i$.

Definition 13 (Pseudo-dimension)**.** Let $\mathcal{H} \subseteq [0,H]^{\mathcal{Y}}$ be a set of functions mapping an abstract domain $\mathcal{Y}$ to an interval $[0,H]$. Let $S \subseteq \mathcal{Y}$ be a largest set that can be shattered by $\mathcal{H}$. Then the pseudo-dimension of $\mathcal{H}$ is $\text{Pdim}(\mathcal{H}) = |\mathcal{S}|$.

We now define the VC-dimension of a class of functions, following [113].

Definition 14 (VC-dimension)**.** Let $A \leq F(z,\alpha) \leq B, \alpha \in \Lambda$, be a set of real functions bounded by constants A and B (where A can be $-\infty$ and B can be ∞). Then, for $\alpha \in \Lambda$ and $\beta \in (A,B)$, VC-dimension of $F(z,\alpha)$ is the maximum number of vectors shattered by

$$I(z,\alpha,\beta) = \begin{cases} 0 & \text{if } F(z,\alpha) < \beta \\ 1 & \text{if } F(z,\alpha) \geq \beta. \end{cases}$$

Finally, we define empirical Rademacher complexity. Intuitively, this is a measure of the extent to which functions in the class are correlated with a set of random noise vectors. An empirical Rademacher complexity of 0 indicates that a function class is of the lowest possible complexity, whereas a value of 1/2 indicates that the class has the highest possible complexity. We follow the definition of empirical Rademacher complexity given in [9].

Definition 15 (Empirical Rademacher Complexity)**.** The empirical Rademacher complexity of a function class $\mathcal{F} = \{f_r \mid r \in \mathcal{R}\}$ given a set $\mathcal{S} = \{x_1,\dots,x_N\} \subseteq \mathcal{X}$ is

$$\mathfrak{R}_{\mathcal{S}}(\mathcal{F}) = \frac{1}{N} \mathrm{E}_{\sigma \sim 1,-1^N} \left[\sup_{\{r \in \mathcal{R}\}} \sum_{i=1}^{N} \sigma_i f_r(x_i) \right],$$

where each σ_i equals -1 or 1 with equal probability.

Appendix B: Inequality Values Used in Proof of Lemma 79

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2	1.3	1.4	1.5
0.2	0.0	–	–	–	–	–	–	–	–	–	–	–	–	–	–
0.3	0.0	0.2	–	–	–	–	–	–	–	–	–	–	–	–	–
0.4	0.0	0.1	0.4	–	–	–	–	–	–	–	–	–	–	–	–
0.5	0.0	0.1	0.2	0.4	–	–	–	–	–	–	–	–	–	–	–
0.6	0.0	0.1	0.2	0.3	0.8	–	–	–	–	–	–	–	–	–	–
0.7	0.0	0.1	0.1	0.3	0.5	1.1	–	–	–	–	–	–	–	–	–
0.8	0.0	0.1	0.2	0.3	0.5	0.9	2.2	–	–	–	–	–	–	–	–
0.9	0.0	0.1	0.2	0.3	0.4	0.7	1.2	2.9	–	–	–	–	–	–	–
1.0	0.1	0.1	0.2	0.3	0.5	0.8	1.3	2.2	5.2	–	–	–	–	–	–
1.1	0.1	0.1	0.2	0.3	0.5	0.8	1.1	1.8	3.1	7.3	–	–	–	–	–
1.2	0.1	0.2	0.3	0.4	0.6	0.9	1.3	2.0	3.1	5.5	12.9	–	–	–	–
1.3	0.1	0.1	0.3	0.4	0.6	0.8	1.2	1.7	2.6	4.1	7.3	17.5	–	–	–
1.4	0.1	0.2	0.3	0.4	0.7	0.9	1.3	1.8	2.7	4.0	6.5	12.0	30.1	–	–
1.5	0.1	0.2	0.4	0.6	0.9	1.3	1.8	2.5	3.6	5.3	8.3	14.0	27.8	78.0	–
1.6	0.1	0.3	0.4	0.7	1.0	1.4	2.0	2.7	3.9	5.7	8.8	14.5	27.4	65.9	294.9
	1.7	1.8	1.9	2.0	2.1	2.2	2.3	2.4	2.5	2.6	2.7	2.8	2.9	3.0	
1.6	245.3	67.0	31.3	18.3	12.1	8.7	6.6	5.2	4.2	3.5	2.9	2.5	2.2	1.9	
1.7	–	95.3	37.1	20.5	13.2	9.3	7.0	5.5	4.4	3.6	3.1	2.6	2.3	2.0	
1.8	–	–	69.8	29.9	17.6	11.8	8.6	6.6	5.3	4.3	3.6	3.1	2.7	2.4	
1.9	–	–	–	52.0	23.3	14.2	9.8	7.3	5.7	4.6	3.8	3.2	2.8	2.4	
2.0	–	–	–	–	46.6	21.4	13.3	9.3	7.0	5.6	4.5	3.8	3.2	2.8	
2.1	–	–	–	–	–	49.0	22.9	14.4	10.2	7.8	6.2	5.1	4.3	3.7	
2.2	–	–	–	–	–	–	43.4	20.5	13.0	9.3	7.2	5.7	4.7	4.0	
2.3	–	–	–	–	–	–	–	42.8	20.4	13.0	9.4	7.2	5.8	4.8	
2.4	–	–	–	–	–	–	–	–	42.2	20.3	13.0	9.4	7.3	5.9	
2.5	–	–	–	–	–	–	–	–	–	42.6	20.5	13.3	9.6	7.5	
2.6	–	–	–	–	–	–	–	–	–	–	44.8	21.7	14.1	10.3	
2.7	–	–	–	–	–	–	–	–	–	–	–	43.6	21.2	13.8	
2.8	–	–	–	–	–	–	–	–	–	–	–	–	45.9	22.4	
2.9	–	–	–	–	–	–	–	–	–	–	–	–	–	46.0	

Table B.1 The number in row a column b is 100,000 times the value of the quantity given in Lemma 80 (to one decimal place) for the $(1{+}1)_a$ EA ahead of the $(1{+}1)_b$ EA by some linear distance. Hence if it is no greater than 100,000 then, w. o. p., the $(1{+}1)_a$ EA remains ahead of the $(1{+}1)_b$ EA. The values have been displayed in this way to give an idea of their relative size, since all values are so small this relationship was otherwise lost when reducing the size of the table. It is easily verified that all values are several orders of magnitude smaller than we require.