Algorithm Configuration Landscapes:

Analysis and Exploitation

by

Yasha Pushak

BSc. (Honours), The University of British Columbia, 2015

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES
(Computer Science)

The University of British Columbia
(Vancouver)

May 2022

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

Algorithm Configuration Landscapes: Analysis and Exploitation

submitted by **Yasha Pushak** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy** in **Computer Science**.

Examining Committee:

Holger H Hoos,
Alexander von Humboldt Professor of Artificial Intelligence,
Department of Computer Science,
Rheinisch-Westfälische Technische Hochschule Aachen University
Co-Supervisor

Mark Schmidt,
Associate Professor,
Department of Computer Science,
The University of British Columbia's Vancouver Campus
Co-Supervisor

Mark Greenstreet,
Professor,
Department of Computer Science,
The University of British Columbia's Vancouver Campus
Supervisory Committee Member

Shawn Xianfu Wang,
Professor,
Department of Computer Science, Mathematics, Physics and Statistics,
The University of British Columbia's Okanagan Campus
University Examiner

Sathish Gopalakrishnan,
Associate Professor,
Department of Electrical and Computer Engineering,
The University of British Columbia's Vancouver Campus
University Examiner

Peter van Beek,
Professor and Associate Director,
Cheriton School of Computer Science,
University of Waterloo
External Examiner

Abstract

Algorithm designers are regularly faced with the tedious task of finding suitable default values for the parameters that impact the performance of algorithms. Thoroughly evaluating even a single parameter *configuration* typically requires running the algorithm on a large number of problem instances, which can make the process very slow. To address this problem, many automated algorithm configuration procedures have been proposed. The vast majority of these are based on powerful meta-heuristics with strong diversification mechanisms, thereby ensuring that they sufficiently explore the parameter configuration space.

However, despite the prominence of automated algorithm configuration, relatively little is known about the *algorithm configuration landscapes* searched by these procedures, which relate parameter values to algorithm performance. As a result, while these strong diversification mechanisms make existing configurators robust, it is unclear whether or not they are actually required or simply increase the running time of the configurators.

One particularly notable early work in the field showed evidence suggesting that the algorithm configuration landscapes of two optimization algorithms are, in fact, close to uni-modal. However, existing fitness landscape analysis techniques are unable to account for the stochasticity in the performance measurements of algorithms in a statistically principled way, which is a major barrier to their application to studying algorithm configuration scenarios. We address this gap by developing the first statistically principled method for detecting significant deviations from uni-modality in a stochastic landscape.

We apply this method, along with other (new and existing) landscape analysis techniques, to a variety of algorithm configuration scenarios arising in auto-

mated machine learning (AutoML) and the minimization of the running time of algorithms for solving $\mathcal{NP}$-hard problems. We show that algorithm configuration landscapes are most often highly structured and relatively simple.

Using the intuition from this analysis, we develop two prototype algorithm configuration procedures designed for AutoML. We show that the methods make assumptions that are too strong, leading to mixed results. However, we build on this intuition and develop another procedure for the configuration of $\mathcal{NP}$-hard algorithms. Compared to state-of-the-art baselines, we show that our new method often finds similar or better configurations in the same or less time.

Lay Summary

Problems on a computer are solved by a series of steps, called *algorithms*. Much like recipes, which can be modified by changing, for example, the baking temperature, algorithms have *parameters* that control their performance. Manually finding suitable values for these parameters by trial and error is a tedious task. To alleviate this burden, algorithm designers designed new algorithms, called *configurators*, that can automatically find high-quality settings for these parameters. These configurators are highly effective; however, they typically require evaluating a very large number of parameter settings to ensure that no high-quality solutions are missed. We hypothesize that there should be patterns in the parameters that make the problem easier to solve. By way of analogy, consider the baking temperature of a cake. Most likely, it is either too low, too high or just right. We develop statistical methods for detecting this structure and show how it can be exploited.

Preface

All of the research presented in this thesis is the original intellectual product of the author, Yasha Pushak, with guidance and mentorship from my primary thesis advisor, Holger Hoos. Most of this work has been published or submitted for publication in various venues.

In 2015 during a class with Holger, I expressed interest in algorithm configuration landscapes and what was known about them, which was, at the time, very little. Two years later when my original research plan was struggling, Holger proposed that I switch topics and begin investigating them. In particular, Holger first suggested that we study individual parameter response slices to check for uni-modality in a statistically principled way. From this idea, I developed early versions of some of the methods proposed in Chapter 3, which were first published in Pushak and Hoos [148] (the rest of the methods in Chapter 3 have been accepted for publication in Pushak and Hoos [151], pending minor revisions). For the same paper [148], I designed the experimental setup and conducted the analysis of the results presented in Chapter 6, with guidance from Holger.

The writing from Pushak and Hoos [148] used in this thesis is largely my own; however, as it was the first paper I co-authored with Holger, he naturally made substantial edits to improve the clarity and presentation of the text, without substantially modifying the content.

Chapter 4 takes a large quantity of text from Pushak and Hoos [151], with only very minor modifications. Pushak and Hoos [151] also introduced several new or improved versions of the methods which are presented in Chapter 3. The writing of these sections, the conception and design of the methods, experimental setup and the analysis of the results were primarily my own contributions, with minor edits,

suggestions and guidance from Holger.

Chapter 5 is also joint work with Holger Hoos that is under review for publication [147]. The design and implementation of the CQA method was done by me, with little guidance from Holger. However, he proposed the second modification using splines. He further substantially improved the work by giving suggestions to improve the analysis of both methods using artificial scenarios with known properties in order to demonstrate why the two methods were too simple to work well on the real scenarios.

An early prototype of GPS (see Chapter 7) was designed as part of a course project for a class taught by Mark Greenstreet. I conceived of one of the key components of GPS, how it makes use of parallelism, based on the guiding principles from his class. My original idea for how the bandit queue should work was substantially more complicated than the one presented in Chapter 7.1.9, which was Holger's idea. Otherwise, the ideas behind the method and its implementation are substantially my own contributions. The design of the experiments, analysis of the results and their presentation in Chapter 7, were done with guidance from Holger.

Much of the writing in Chapter 7 was originally published in Pushak and Hoos [149]. The writing was substantially my own, with moderate editing and guidance from Holger on its structure and presentation.

The writing, research and content of each of the sections in the appendix comes substantially from the papers mentioned above, for each of their respective chapters. Contributions to the ideas, results and authorship are largely the same as for their respective chapters.

Finally, Chapters 1, 2 and 8 all borrow small snippets of text from Pushak and Hoos [148, 149] and Pushak and Hoos [151]. However, nearly all of the writing, the presentation of ideas and synthesis of the related literature is my own contribution.

Table of Contents

List of Tables

List of Figures

Acknowledgments

Over the course of my academic career I have been lucky to have the opportunity to work with and learn from a great many brilliant and wonderful people. I am deeply grateful for the many people who have helped and supported me over these many years.

First and foremost, I am indebted to my primary thesis advisor, Holger Hoos, without whom this thesis would not have been possible. Throughout my PhD you have challenged my work and ideas in precisely the right way to help me mature as a researcher and scholar. In the early years of my graduate studies, I came away from most of our meetings mentally exhausted from trying to understand and learn from all of the feedback you gave me. I have never met anyone else from whom I have been able to learn so much on such a wide variety of topics, from the strategic prioritization of possible research ideas and and how to evaluate them to the presentation of their final results.

Next, I am grateful to Mark Schmidt, who agreed to act as my co-supervisor when Holger accepted a faculty position at Leiden University early into my PhD, thereby allowing me to continue my studies at UBC. Even though I did not have the opportunity to work directly with you on any research projects, your feedback and support – particularly regarding my mental and physical well-being – have been greatly appreciated.

Thank you to Mark Greenstreet, for teaching me about the most effective ways to use parallel resources in your course on parallel algorithms and architectures, and for the opportunity to harness that knowledge to develop an early prototype of GPS as a part of a course project. Ultimately, that early prototype lay the foundation for Chapter 7 of my thesis, which is, perhaps, the strongest practical result of

my PhD.

I am also thankful for the valuable feedback from my PhD supervisory committee, Holger Hoos, Mark Schmidt and Mark Greenstreet, for their valuable feedback on various drafts of my thesis. The feedback from all three of you substantially improved the overall clarity and presentation of the thesis. Finally, thank you for putting up with my rushed timeline at the end of my PhD and helping me complete my thesis within a relatively short period.

I was also very fortunate to receive a great deal of financial support throughout my PhD in the form of scholarships and awards; however, a discussion of those honors necessitates a discussion of the many people without whom I would not have had the opportunity and courage to establish myself as a scholar and a leader.

First, thank you to my undergraduate research supervisors Warren Hare and Yves Lucet, both for the opportunity, mentorship and guidance that set me up for success as a graduate student and later for your help presenting my undergraduate work in the form of an application for the NSERC CGS-M, a scholarship that supported me during the first year of my graduate studies. Second, thank you to Abby Arnold (and the many students and faculty who helped us), who co-founded the Canadian Undergraduate Computer Science Conference with me. Working with you when we were undergraduate students was an amazing experience; it helped me develop confidence in my ability to lead and shaped my years and activities as a PhD student. I also have no doubt that it was a major factor in my recognition as a Vanier Scholar, which provided me with three years of funding during my PhD. Finally, I am also thankful to Patricia Lasserre, Yves Lucet and Holger Hoos, for writing me what must have been fantastic reference letters and helping me put together my Vanier application.

While pursuing my PhD I was also both fortunate and honoured to serve as the first Student Representative on the Board of Directors for CS-Can – a newly-formed national organization that represents and promotes computer science in Canada. First, thank you to Carey Williamson for recruiting me and later providing support as I learned to navigate in a completely new environment. My time on the board was eye-opening – I learned a great deal of valuable insights into the national academic scene and funding opportunities. Second, thank you to Kellogg Booth for your support and advice on my work for the board. Finally, thank you to the

many students who helped me bootstrap the student events and programs run by the student and postdoc advisory committee of CS-Can – in particular: Parastoo Baghaei and Jad Kabbara. It was a pleasure to work with you both and a relief to know I could always count on you for anything that urgently needed to be done.

During my PhD I also had the opportunity to pursue an internship at Oracle Labs, which evolved into a permanent part-time (and then full-time) position. During my time at Oracle Labs I have been fortunate to work with many more great people who helped me learn and grow. I am particularly grateful to Karoon Rashedi Nia and Tayler Hetherington, for teaching me invaluable software engineering skills that substantially reduced the time it took me to implement the methods and algorithms in Chapters 4 and 5. Thank you also to my supervisor Hesam Fathi Moghadam for your support and accommodation of my PhD schedule.

I have also had the pleasure to meet and befriend many people over the course of my PhD. Due to the pandemic and differing graduation timelines it has been a long time since I have had the pleasure of seeing or catching up with many of you. However, you nevertheless brightened my days as a graduate student, for which I will always be grateful. In no particular order, thank you to Clement Fung, Giovanni Viviani, Neil Newman, Kuba Karpierz, Nico Ritschel, Alistair Wick, Louie Dinh, Kimberly Dextras-Romagnino, Chris Cameron, Hayley Guillou, Adam Geller, Anna Scholtz and of course many others for: a) many late nights spent playing board games; b) conversations in the bull pens, algorithms lab or at Tuesday Tea; or c) for our many games of ultimate Frisbee.

I am also grateful to my family and extended family for their love and encouragement throughout my life. I am also thankful for my family in law, who accepted me into their family and provided me with love, encouragement and support as soon as I moved to Vancouver for my PhD – a time at which I missed my own family greatly – even though I had only been dating their daughter/granddaughter/sister for 10 months when I moved.

Finally, I am deeply grateful to my wife, Jacoba. Thank you for understanding my many late nights and long weekends spent working on my PhD (including this one). Thank you for your love, support and encouragement. Thank you for introducing me to the joy that it is to adopt our two dogs, Sage and Iris, who have given me many wonderful reasons to set aside my PhD to go outside to enjoy some

fresh air, exercise, sunshine and (of course) rain. Finally, thank you for putting up with my complete inability to accurately estimate when I would be done my graduate studies, which made it nearly impossible for us to plan out the first few years of our lives as a married couple.

Chapter 1

Introduction

If you were leading a team on a mountaineering expedition, with your goal being to find the highest peak of an unfamiliar mountain range that is covered in a dense fog, what would you want to know?

1.1 Automated Algorithm Configuration

When faced with a computational problem, computer scientists, engineers, researchers and programmers write *algorithms* to solve them – that is, a collection of steps that can be followed until a solution to the problem has been found. Algorithm designers are typically faced with a variety of choices that need to be made regarding their algorithm. For example, if the problem is to bake a cake, then two of the decisions that the algorithm (or in this case: recipe) designer will need to make, is to determine for how long the cake should be baked and at which temperature.

These algorithmic design choices are typically referred to as *parameters* of the algorithm. These parameters can be real-valued (for example, the temperature), integer-valued (for example, the number of layers) or categorical (for example, whether or not the cake has icing). The parameters can also be *conditional*, that is, their values are only used if one or more *parent* parameters are set to certain values. For example, it only makes sense to define the flavor of icing on the cake if the cake has icing. Clearly, choosing good values for these parameters can be very important; if set incorrectly, many combinations, or *configurations*, of parameter

values can lead to undesirable results or performance. For example, in our cake baking analogy, if the temperature and baking time are set too high, then the cake will dry out or possibly even burn.

For algorithmic problems, there are numerous methods that can be used to measure how well a particular parameter configuration performs. For example, if the algorithm is a machine learning classifier, then the goal may be to maximize the classification accuracy of the model on a validation data set. Similarly, the goal could be to minimize the root mean squared error of a machine learning regression model. If the problem itself is an optimization problem, then the performance of the algorithm may correspond to the quality of the solution that the algorithm is able to find on a given problem instance. For example, if the algorithm is for the travelling salesperson problem (TSP), then the solution quality, and hence the performance of the algorithm, could correspond to the length of the shortest tour found by the algorithm.

Many algorithms do not return results that can be used as a performance metric. For example, a sorting algorithm produces output that is either correct or incorrect. However, one may still be interested in minimizing the running time, memory usage or number of parallel resources required to run the algorithm (perhaps while also enforcing constraints on the maximum amount of each such resource that can be used).

Finding a good set of parameter values by hand for an algorithm is typically quite tedious. To configure an algorithm by hand, the algorithm designer needs to select a particular candidate parameter configuration and then evaluate it by running the algorithm to decide whether or not it is good enough. In order to obtain good results, this process needs to be repeated many times to sufficiently explore the space of possible parameter configurations. However, this process is often prone to error, as the designer may evaluate the impact of varying each parameter independently, and then, once a suitable value is found, fixing them for the remainder of the trials for the other parameters.

Even if (or perhaps: especially when) an algorithm designer evaluates a large number of candidate configurations to adequately explore the parameter configuration space, the final configuration chosen may still perform poorly. This can be due to over-fitting – that is, two or more parameter configurations may result in com-

pletely different relative performance when the algorithm is evaluated using different instances of a particular problem. As a result, an algorithm designer should evaluate each candidate configuration on a large number of problem instances that are representative of some important and relevant distribution of problem instances. In this case, the performance of the algorithm should then be measured using some population statistic that summarizes the performance of the algorithm on each individual problem instance in the training set; in most applications, the mean performance is a suitable choice.

Naturally, to address the tedium and challenges of the algorithm configuration problem, clever algorithm designers proposed to design new algorithms to automate the process of configuring other algorithms. For clarity, throughout this thesis, we will refer to:

- *target algorithms*, as the algorithms that are being optimized; and
- *configuration procedures* or *configurators*, as the algorithms that configure the parameters of the target algorithms.

Similarly, we will use the terms:

- *problem instances*, to refer to the problems that are solved by the target algorithms; and
- *problem scenarios*, to refer to the problems that are solved by the algorithm configuration procedures.

Over the last two decades, algorithm configuration procedures have been independently proposed to automate the configuration of algorithms for solving a wide variety of different computational problems; for example, automated machine learning (AutoML – for example, see Bergstra and Bengio [21], Feurer et al. [62], Jamieson and Talwalkar [95], Li et al. [113] or Maron and Moore [124]), the configuration of meta-heuristics for $\mathcal{NP}$-hard and $\mathcal{NP}$-complete problems (for example, see Ansótegui et al. [6], Balaprakash et al. [15], Birattari et al. [26] or Hutter et al. [87, 89]), numerical optimization (for example, see Audet and Orban [12], Audet et al. [13] or Hutter et al. [87, 89]), sorting (for example, see Audet

et al. [14]), compiler optimization (for example, see Balaprakash et al. [16], Cavazos and O'Boyle [37], Gschwandtner et al. [74], Tiwari and Hollingsworth [176] or Tiwari et al. [177]) or even the design parameters of field-programmable gate array (FPGA) circuits (for example, see Mametjanov et al. [123]).

1.2 Algorithm Configuration Landscapes

Despite the considerable interest and substantial practical relevance of automated algorithm configuration, to the best of our knowledge, very little is known about the nature of the *landscapes* that are searched by automated algorithm configuration procedures. Informally, an *algorithm configuration landscape* relates the changes that we make to a parameter's value to the changes that we observe in the algorithm's performance. Note that these landscapes are intrinsic to the target algorithms and their parameters. Hence, they are distinct from the landscapes that arise from different initial guesses to solutions of optimization problems, which can sometimes be quite irregular in structure (for example, see Figure 2 of Van Den Doel and Ascher [182]).

Landscape analysis is by no means a new concept in the field of optimization. Indeed, the credit for the analogy that optimization algorithms can be viewed as procedures that explore landscapes in order to find a minimum (valley) or maximum (peak) can be tracked back to the seminal work by Wright [191] in 1932 on evolutionary biology. Since then, the analysis of landscapes searched by optimization algorithms (typically referred to as *fitness landscape analysis*) has attracted substantial attention in a wide variety of optimization applications (for an overview of the field, for example, see Hoos and Stützle [84], Malan [120], Malan and Engelbrecht [121], Pitzer and Affenzeller [143] or Watson [186]). The reason for this is that knowing problem structure is of great practical interest. Derivative free, black-box optimization algorithms [11] are widely applicable, and could be used to solve a broad range of optimization problems, for example, convex optimization [32], linear programming [71] and integer programming [43]. However, in practice, black box optimization is rarely used to solve any of these problems. Why? Because they are all highly-structured problems that can be solved more efficiently with specialized methods.

Indeed, even though the well-known *no free lunch* theorem proves that no optimization technique performs better than any other on average over all possible objective functions [189, 190], it has been shown that very general classes of optimization problems – for example, those with solutions that can be evaluated in polynomial time – contain structure that can be exploited [51]. In fact, it has even been shown that fitness landscape analysis can be exploited to select an optimization algorithm (or even a configuration of a given algorithm) that can efficiently solve a given problem instance (for example, see Abell et al. [2], Belkhir et al. [18], Kerschke et al. [101] or Malan [119]).

Intuitively, this makes sense. For example, suppose you were leading a team on a mountaineering expedition, with your goal being to find the highest peak of an unfamiliar mountain range that is covered in a dense fog. Each member of your team is equipped with a single altimeter, which they can use to determine their relative elevations.

If you knew that the mountain range contained only a single mountain with a single peak, then you would instruct your team to focus purely on exploitation. That is, the team could stick together, with each person simply exploring a small distance away from the current highest known position, calling out to each other what they have found. In contrast, if you knew that there were multiple mountains, each highly rugged with numerous peaks at varying elevations, your best bet would be to divide your team in several different groups. Each group would conduct their own exploration starting from numerous different locations, hoping that one of them would find the highest peak in the mountain range.

In fact, this very type of structure is one which seems quite natural to assume should arise frequently in automated algorithm configuration scenarios. For example, consider again our analogy to cake baking. In practice, changing the baking temperature likely simply trades off between *too little* and *too much*. If it is set too low, the cake will not bake, but if it is set too high, then the cake will burn. Our hypothesis is that similar structure exists for most algorithmic parameters. For example, consider a hyper-parameter that controls the depth of a decision tree. If set too small, the tree will be unable to learn all but the simplest of trends, thus frequently leading to under-fitting. However, if set too large, the tree will be able to memorize the training data, thus leading to over-fitting.

Alternatively, suppose you knew that each parameter (in our example: latitude and longitude) produced changes in the performance of the algorithm that were completely independent of each other. For example, suppose the mountain range, $f(x,y)$, could be perfectly described by two functions, $f(x,y) = f_x(x) + f_y(y)$, where x and y correspond to the latitude and longitude. In this case, you would only need a team of two people to find the highest peak in the mountain range. Both would traverse the mountain range in straight lines, one parallel to the lines of latitude and one parallel to the lines of longitude. Once each had reported the highest location they observed, they could report back and you would know exactly where to find the highest peak in the mountain.

Of course, we would not expect any natural mountain range to contain this sort of structure. However, should we expect algorithm configuration landscapes to appear this way? In fact, our hypothesis is that there should be at least some cases where this is true. For example, in our cake baking analogy, it seems quite natural to assume that the flavour of the cake will have a negligible affect on the optimal baking temperature or time (see the left pane of Figure 1.1). However, we would not expect this to be true for the temperature and time parameters. In particular, if the baking temperature is somewhat larger than optimal, it should be possible to compensate by slightly decreasing the baking time (see the right pane of Figure 1.1). Therefore, the question of interest is to determine how frequently the parameters of algorithm configuration landscapes interact and to what extent these interactions increase the complexity of algorithm configuration scenarios.

Besides developing new automated algorithm configuration techniques that can better exploit landscape structure, an equally valuable contribution from landscape analysis are the insights that can be gained. Without a better understanding of algorithm configuration landscape structure, it can be challenging or impossible to develop intuitive explanations for why one algorithm configuration procedure works a) better than another, b) similarly to another completely different method, or c) worse than anticipated. Without such insights, algorithm configuration researchers are only able to speculate about the relative performance of different approaches, and it is often unclear without additional costly study of the configurators (for example, see ablation analysis [23, 61]), whether or not their guesses align with reality.

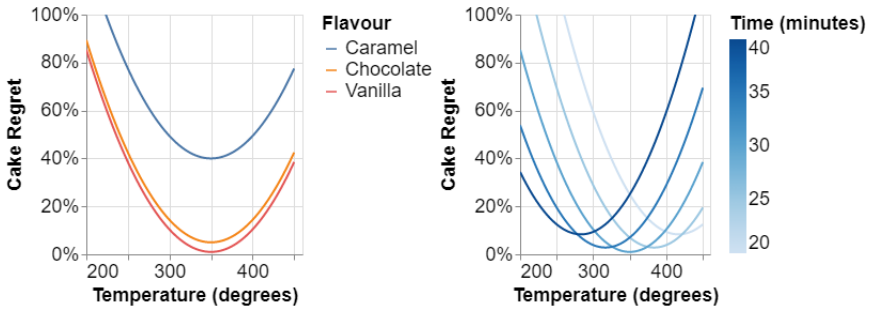

Figure 1.1: Two examples of algorithm (or in this case: recipe) configuration landscapes for our cake baking analogy. We show the quality of the cake as a function of the baking temperature, baking time and cake flavour. We measure quality in terms of regret to make this a minimization problem, which is consistent with the remainder of the scenarios in this thesis.

In cases where a researcher's intuition is incorrect, they (and future researchers) can get stuck at local minima (or plateaus) of the design space of possible algorithm configuration procedures. We hope that the insights we shed in this thesis into algorithm configuration landscapes will help future researchers a) build intuition, b) perform retrospective analyses of existing algorithm configuration procedures and c) gain inspiration for new and better configuration techniques.

1.3 Preliminaries and Notation

The goal in automated algorithm configuration is to optimize the performance of a target algorithm, A, on a distribution of problem instances, $\mathcal{I}$, thereby improving the performance of A when it is used to solve future problem instances from $\mathcal{I}$. Formally, let P be the set of configurable parameters for A, which we can modify to adjust the performance of A. Furthermore, let $m_{\mathcal{I}}$ be the performance metric for which we wish to optimize A on distribution $\mathcal{I}$.

Since it is typically intractable to measure m on an entire distribution of problem instances, $\mathcal{I}$, we typically approximate it with m_I, where $I \subset \mathcal{I}$ is a random sample of problem instances from distribution $\mathcal{I}$. Hence, we denote by $\underline{m}_I(c)$ and $\overline{m}_I(c)$ the lower and upper bounds, respectively, of a confidence interval for $m_{\mathcal{I}}(c)$

calculated using I. Throughout the following, we will sometimes write m rather than m_I in cases where I is clear in a given context.

The performance metric, m, can correspond to any measurable attribute of an algorithm, for example a resource consumed by the algorithm (for example the running time or memory consumption) to solve a problem instance. For an optimization problem instance, solving a problem instance may be defined as finding the optimal solution, whereas for a boolean satisfiability (SAT) problem instance, it may correspond to a certification of whether or not the boolean formula can be satisfied by an assignment of the variables. In these cases, the set I would contain a collection of problem instances representative of $\mathcal{I}$.

Alternatively, m may measure the quality of the solution returned by the algorithm. For example, this could be the objective function value for the final incumbent in an optimization problem, or the accuracy or loss of a machine learning classifier on a set of validation data. In the latter case, I typically corresponds to a set of random training and validation splits of the data set. Depending on the time required to train a machine learning model in these cases, it is not uncommon for I to contain only a single training and validation split.

Throughout the following, we will always choose m such that a smaller value corresponds to a better configuration, hence our algorithm configuration scenarios all correspond to minimization problems; however, a metric could, of course, be chosen such that the algorithm configuration scenario may correspond to a maximization problem.

Let $p \in P$ be a parameter (or hyper-parameter, in the context of a machine learning scenario) of algorithm A with values $v \in V_p$. The parameter range V_p may either be a range of continuous, real values, for example, $v_{\texttt{sample\_fraction}} \in [0,1)$; an ordered set of discrete (often integer) values, for example, $v_{\texttt{population\_size}} \in \{10,11,...,1\,000\}$; or a discrete set of un-ordered, categorical values, for example, $v_{\texttt{subsampling}} \in \{\texttt{ON},\texttt{OFF}\}$. We denote by $c = (v_1,...,v_n) \in C \subseteq V_{p_1} \times ... \times V_{p_n}$ a configuration for algorithm A.

Some parameters may be *conditional*, that is, their values are either considered to be undefined or are simply ignored, depending on the value(s) of one or more parent or ancestor parameters. For example, $v_{\texttt{sample\_fraction}}$ may always be ignored when $v_{\texttt{subsampling}} = \texttt{OFF}$.

We can now formally define an instance of an algorithm configuration scenario.

Definition 1 (Algorithm Configuration Problem Scenario)**.** *An instance of an* algorithm configuration scenario *is defined by the tuple* (A, P, C, I, m)*, where the goal is to find a configuration* $c^* \in C$ *such that* $m_I(c^*) \leq m_I(c)$ *for all* $c \in C$*, and where*

- A *is a parameterized target algorithm;*
- P *is the set of parameters of* A*;*
- C *is the configuration space of the parameters* P*;*
- I *is a set of problem instances representative of a distribution of problem instances* $\mathcal{I}$ *on which* A *is to be applied in the future; and,*
- m_I *is a performance metric that can be used to evaluate a particular configuration* $c \in C$ *of* A *on instance set* I*.*

Given an instance of an algorithm configuration scenario, we can define its corresponding *algorithm configuration landscape*. However, the properties of any search landscape also depend upon the definition of the underlying neighbourhood – in our case, on the relation that specifies which parameter configurations are neighbours and on the metric used to quantify distance between configurations.

We define a graph $G = (C, E)$ over a grid that discretizes[1] the parameter configuration space, where the vertices C are the parameter configurations and edge $e = (c_a, c_b)$ is in E if c_a and c_b are adjacent in the grid. When the landscape contains categorical parameters, we define all of the values of a given categorical parameter to be adjacent, and thus there exists an edge between them in the graph. Each edge $e \in E$ is assigned a weight that represents the distance between the two configurations.

Definition 2 (Algorithm Configuration Landscape)**.** *An* algorithm configuration landscape *is defined by the tuple* (A, P, C, I, m, G)*, where*

[1] While fitness landscape analysis can be extended to continuous domains, in this thesis, we simplify the analysis by discretizing all real-valued parameters to a grid of configurations. This allows us to easily treat both numerical and categorical parameters uniformly and makes evaluating the landscapes computationally tractable.

- *A is a parameterized target algorithm;*
- *P is the set of parameters of A;*
- *C is the configuration space of the parameters P, such that any numerical parameters have been discretized;*
- *I is a set of problem instances representative of a distribution of problem instances $\mathcal{I}$ on which A is to be applied in the future;*
- *m_I is a performance metric that can be used to evaluate a particular configuration $c \in C$ of A on instance set I, which corresponds to an estimate of the performance of A on the instance distribution $\mathcal{I}$; and,*
- *$G = (C, E)$ is a graph that defines the neighbourhood relation of the configuration space.*

Throughout this thesis, we will typically refer to candidate solutions to the algorithm configuration problem as *configurations*; however, in some contexts, we may also use the word *solution* interchangeably. In most cases, when we refer to candidate solutions instead of configurations, it is because we are speaking about solutions to optimization problems more broadly, which could either be algorithm configuration scenarios or classic numeric optimization problems (for example, see our discussion of existing fitness landscape analysis techniques in Chapter 2.2). Similarly, we will often use the terms *solution quality* and *performance* interchangeably when speaking in broad terms about methods that are applicable for any optimization problem, and we will typically refer specifically to *performance* when discussing only automated algorithm configuration scenarios, in order to help distinguish between the solution quality of a particular problem instance in an algorithm configuration scenario which may, or may not, be the metric used for the target algorithm's performance.

Furthermore, we will use the following notation related to configurations and parameter settings: $c[p]$ denotes the value of parameter p in configuration c; $c[p] := v$ modifies c by setting parameter p to value v; and $c|_{p=v}$ denotes configuration c with parameter p (temporarily) set to v.

1.4 Thesis Contributions and Structure

The primary contribution of this thesis is that we are the first to take a rigorous and statistically principled approach to analyzing the landscapes arising from automated algorithm configuration scenarios. We provide new insights into the typical structure of these landscapes for two prominent and important applications of automated algorithm configuration: AutoML and the running time minimization of algorithms for solving $\mathcal{NP}$-hard and $\mathcal{NP}$-complete problems. Using these insights, we develop three new automated algorithm configuration procedures, which we evaluate to show how well they work and the conditions under which we can expect them to work well. In each case, we provide a discussion of what we believe were the key components responsible for the success (or failure) of the newly proposed configuration procedures.

Chapter 2: Existing landscape analysis and automated algorithm configuration procedures. In Chapter 2, we discuss what is already known about algorithm configuration landscapes, and we provide a detailed survey of the fitness landscape analysis methods that have been applied to study algorithm configuration landscapes and related problems. In Chapter 2, we also provide a summary of a diverse set of the most prominent algorithm configuration procedures that have been applied to a variety of algorithm configuration scenarios.

Chapter 3: New, statistically principled algorithm configuration landscape analysis methods. In Chapter 3, we introduce the landscape analysis methods that we use throughout this thesis, both old and new. For each of the new methods, we place a particular emphasis on developing statistically sound methods. In particular, as discussed throughout Chapter 2.2, existing fitness landscape analysis techniques are all poorly suited to handling the stochasticity of algorithm configuration landscapes, which casts doubts about the accuracy of many of the conclusions drawn by the authors of the work discussed in Chapter 2.1.

In particular, from our analysis in Chapter 2.1, we have concluded that there is sufficient evidence to suggest that algorithm configuration landscapes can sometimes contain more than a single mode (see Chapter 5.3.6 of Pedersen [140]). How-

ever, without careful treatment of the noisy performance objective values, it is unclear whether or not the observation that these landscapes also frequently contain dozens of local minima is warranted. We therefore introduce two novel methods. The first detects statistically significant deviations from uni-modality. The second investigates an even stricter form of structure; it looks for statistically significant deviations from convexity in the responses of the numerical parameters of algorithms.

Furthermore, while there is also clear evidence that some algorithm configuration landscapes can have strong parameter interactions (for example, see Yuan et al. [196]), it is unclear how frequently this arises, how much of a given landscape is typically affected by strong interactions, and whether or not the presence of these interactions makes algorithm configuration scenarios substantially harder to optimize. To answer these questions, we propose to measure the fraction of partial derivatives in the landscape that are significantly different from zero, and we use a simplistic algorithm configuration procedure that assumes independence of the parameters in the landscape, thereby allowing us to quantify how frequently it can obtain high-quality final results.

Chapter 4: Analysis of AutoML loss landscapes. In Chapter 4, we perform an analysis of a broad range of the landscapes that arise in AutoML, an application of automated algorithm configuration for which relatively little is known about their landscapes. Unlike those discussed in Chapters 2.1.6 and 2.1.7, we focus primarily on the numerical hyper-parameters of the landscapes, with some limited analysis of categorical hyper-parameters. We believe that these are the natural starting place for research in the field, as they are the most likely candidates to contain exploitable structure (which was, in fact, also observed by Rodrigues et al. [157]).

In this chapter, we show that other than a few particular exceptions, most AutoML loss landscapes appear to be highly structured and rather simple. In particular, they are often both uni-modal (but not convex) and, despite their presence, hyper-parameter interactions do not typically increase the complexity of the algorithm configuration scenario.

Chapter 5: Attempts to exploit AutoML loss landscape structure. In Chapter 5, we propose two new variations of an existing, state-of-the-art hyper-parameter configuration procedure. Each method was designed to exploit a specific property that we observed common among most AutoML loss landscapes; however, we demonstrate on a set of problem scenarios that neither are competitive with the original algorithm configuration procedure. In light of this result, we construct artificial algorithm configuration scenarios that allow us to show when the two new methods can be expected to outperform the original configurator and when they cannot, thereby revealing why these methods performed poorly on the realistic application scenarios.

Chapter 6: Analysis of running time minimization landscapes for $\mathcal{NP}$-hard and $\mathcal{NP}$-complete scenarios. In Chapter 6, we provide what is, to the best of our knowledge, the very first empirical analysis of algorithm configuration landscapes that arise when minimizing the running time of algorithms for $\mathcal{NP}$-hard and $\mathcal{NP}$-complete problems. We show that many of these landscapes contain similar structure to their counterparts that arise in AutoML, albeit with substantially noisier performance estimates.

Chapter 7: Exploiting structure in running time minimization landscapes. In Chapter 7, we draw on insights from all previous chapters in this thesis to motivate the design of a new algorithm configuration procedure, golden parameter search (GPS). We show that GPS often finds similar or better configurations than a variety of other state-of-the-art algorithm configuration procedures, while requiring a similar or smaller configuration budget.

Chapter 8: Conclusions and future work. Finally, in Chapter 8, we summarize the core contributions and primary results of this thesis, and we contrast them with what was previously known about algorithm configuration landscapes. We also discuss several promising directions for future work, and comment on the next generation of automated algorithm configuration procedures.

1.5 Chapter Summary

In this chapter, we motivated and formally defined the *algorithm configuration* problem. Put simply, the goal of automated algorithm configuration is to optimize the performance of an algorithm by automatically finding high-quality values for parameters that control the behaviour of the algorithm. We also introduced a formal definition for *algorithm configuration landscapes*, which provides an analogy between the relationship of parameter values and algorithm performance to physical landscapes.

To provide more intuition about our hypotheses regarding the structure of algorithm configuration landscapes – which we expect are often quite simple – we made an analogy between computational algorithms and cake-baking recipes. In particular, we hypothesize that many parameters of algorithms frequently simply trade off between *too little* and *too much* (much like the baking temperature of a cake). Similarly, while some parameters likely interact in a compensatory way (for example, the baking time and temperature of a cake), we suspect that many interactions are negligible (for example, the baking time and the cake flavour).

With the intuition of our hypotheses in mind, we made yet another analogy between the optimization of algorithm configuration landscapes to a team of mountaineers trying to find the highest peak of an unfamiliar mountain range that is cloaked in a dense fog. We observed that knowing whether or not the mountain range is uni-modal or covered in numerous sub-optimal peaks (and similarly whether or not the parameters – latitude and longitude – interact strongly or weakly), has substantial ramifications for the search procedure the team of mountaineers would use.

Finally, we summarized the main contributions of this thesis and outlined the thesis structure, revealing where more details of each contribution can be found.

Chapter 2

Related Work

We are by no means the first to wish to automatically improve the performance of algorithms by means of configuring their parameters, nor are we the first to study the structure of optimization problems by way of an analogy to physical landscapes. Indeed, both topics have been extensively studied in a variety of different application scenarios.

In Chapter 2.1, to put our questions and hypotheses regarding algorithm configuration landscapes in perspective, we provide an overview of all of the main empirical *findings* on algorithm configuration landscapes (both old and new), while only mentioning the methods used to gain those insights in passing. We include a more detailed discussion of the many different methods for performing fitness landscape analysis in Chapter 2.2, which focuses more broadly on existing fitness landscape analysis methods that have been applied to algorithm configuration landscapes and other optimization problems.

Similarly, in Chapter 2.3, rather than focusing on the application scenarios from which each configuration procedure arose, we instead provide an overview of the various methods, while only mentioning the application areas in passing. However, we do make note of any methods that can only be applied in application areas that contain specific properties (for example, see Chapter 2.3.2).

2.1 What is known about algorithm configuration landscapes?

To the best of our knowledge, when we began work on this thesis in 2015, very little was known about the landscapes that arise in various algorithm configuration scenarios. In particular, we are only aware of three studies [140, 195, 196] (that were conducted prior to the research that makes up this thesis) that present empirical results regarding algorithm configuration landscapes, and one that shows theoretically that a particular parameter, the noise probability, of a stochastic local search algorithm should yield a uni-modal response in performance [126]. However, since our first publication on algorithm configuration landscapes in 2018 [148], there has been substantially more interest in the topic, leading to several new publications in the field (as well as one on a related topic [173]).

2.1.1 Particle Swarm Optimization

To the best of our knowledge, particle swarm optimization (PSO) [54] configuration landscapes have been studied more extensively than any landscapes arising for any other algorithm. Particle swarm optimization is an optimization algorithm that maintains a population of "particles", that is, candidate solutions, that are iteratively updated in a way that was inspired by the behaviour of birds when swarming prey. The velocity of each particle is controlled by three factors, each of which are typically associated with a single parameter: the inertia, cognitive and social weights. The inertia parameter controls the tendency of a particle to continue moving in the same direction, whereas the cognitive and social parameters control the strength with which a particle is attracted to the best solution it has visited so far and the best solution visited by any of the particles in its vicinity, respectively.

To the best of our knowledge, Pedersen [140] was the first to study the landscapes induced by two of these parameters (they studied a PSO variant with the cognitive parameter set to 0, thereby removing the attraction of a particle to its personal previous best solution). In particular, the objective, m, of their algorithm configuration scenario was the solution quality found by their PSO algorithm, and they looked at grids of values for the two remaining parameters when evaluated on 12 different well-known optimization problems; for example, the widely-used

Rastrigin and Rosenbrock benchmark functions. These functions are frequently used as benchmarks for optimization methods because they are known to be challenging to optimize – in the case of Rastrigin, it is challenging because it combines the globally uni-modal structure of a quadratic function with the local ruggedness of a sinusoidal function.

While they did not make use of any landscape analysis methods other than to plot the landscapes with 3D visualizations, they showed that the two parameters yielded a landscape with two close, but distinct regions, or *modes*, that contained high-quality configurations. While the exact location and shape of these modes varied between problem instances, the rough global structure appeared to be consistent, as it also showed up in their plots of the landscape induced by the mean solution quality obtained over all 12 problem instances.

Later, Yuan et al. [196] performed a simple analysis of five parameters of PSO, in which they recorded the performance of the algorithm as the mean solution quality of a single run of the algorithm over 25 randomly selected problem instances. They showed contour plots of two sets of parameters for PSO, which showed that the structure of these parameters was globally uni-modal shaped, something which was corroborated by their analysis of these (and the full) landscapes with fitness-distance correlation (FDC) analysis. They pointed out that while there was a substantial amount of local variability that caused there to be numerous local minima, it was unclear whether or not these local minima were in fact artifacts introduced by the relatively small number of problem instances and target algorithm runs performed in their analysis. Another notable observation made by Yuan et al. [196], is that many of the parameters of PSO appeared to have high degrees of correlation between the high-quality values of various pairs of parameters.

Several years later, it was shown that PSO algorithms exhibit *order-2 stability*, when their parameters satisfy a specific parabolic-shaped condition [40, 41]. That is, when the inertia and the sum of the cognitive and social parameters statisfy this condition, then the expected value and variance of a particle's step sizes will tend towards a constant value.

Recently, three additional studies have been performed on the landscapes of PSO [42, 79, 81]. In each case, they study the landscapes of the algorithm on 20-26 individual problem instances, which again correspond to well known functions

like Rastrigin and Rosenbrock. They all study the three main parameters of PSO; however, in two cases they fix the cognitive and social weights to have the same values, as suggested by the order-2 stability conditions.

The three studies use a variety of techniques to perform their analyses, but overall they all conclude that the landscapes are roughly globally uni-modal in shape with some local ruggedness leading to a configuration problem with "mid-ranged" difficulty. However, one of the pitfalls of their methods, is that they do not properly account for the variability in the performance of the algorithm. In particular, they all assume that taking the mean over 30 independent runs of the algorithm is sufficient to produce an accurate representation of the landscape. However, since none of them include any additional information about this variance, for example, the standard deviation of the performance metric, it is impossible to determine whether or not any of their observations regarding the local features of the landscape are merely random artifacts.

Nevertheless, between them they show several examples of two-dimensional slices of the PSO configuration landscape on various problem instances. In each case, there appears to be a single basin of high-quality values, which roughly corresponds to the parabolic order-2 stability conditions. However, curiously, these particular landscapes appear to often be shaped somewhat-similarly to the bottom half of the inside of a human mouth. That is, the configurations close to the boundary of the parabola yield high-quality performance, the configurations inside of the parabola often yield mediocre-quality performance (the "tongue"), and the configurations outside of the parabola yield very poor performance (the "teeth").

2.1.2 Differential Evolution

Differential evolution is another anytime optimization algorithm that iteratively updates a population of candidate solutions [168]. Using the DE/rand/1/bin strategy [80], there are two main parameters that are typically used to control how the candidate solutions are updated. First, the crossover rate, which determines the frequency with which candidates are updated or remain the same. And second, the differential weight, which controls how much elements of different candidate solutions in the population are "mixed" when crossover occurs.

Pedersen [140] also studied the landscapes induced by the population size and the differential weight, while keeping the crossover rate fixed. Similar to their analysis for PSO, they measured the performance of a configuration as the mean solution quality obtained over 50 independent runs of the algorithm. They showed that the landscapes tend to be uni-modal, both on the 12 individual problem instances as well as when taking the mean over all of the problem instances. They point out that while there are some minor differences between the landscapes on the individual problem instances, they are all relatively similar. Furthermore, they show that there exists a single valley of high-quality configurations, which happens to be diagonally oriented in the landscape, with some degree of curvature.

Belkhir et al. [18] conducted a study of how effectively random forests can be used to predict which configurations can be expected to do well based on features of given problem instances and the particular parameter configuration of the landscape. While the focus of their work was not on any form of landscape analysis, it did include heatmaps of the landscapes induced by the crossover rate and differential weight parameters on four separate problem instances, in order to compare them with the predictions from the random forest model. In each case, there again appears to be a single region that contains high-quality configurations, with some speckling around the boundaries between colours, indicating some degree of local ruggedness. In some cases, one or two corners of the landscapes yield substantially worse performance than the rest, suggesting that there is some degree of interactions between the parameter values.

More recently, Harrison et al. [80] performed an analysis of the landscapes for the differential weight and crossover rate parameters for 20 different individual problem instances. They recorded the performance of the algorithm as the mean solution quality over 30 independent runs of the algorithm, and they considered two configurations the same if their performance scores were within 10^{-8} of each other. Overall, their results were similar to the previous ones for differential evolution. They concluded that the global structure of the landscapes is uni-modal. From looking at their visualizations, we can again see some evidence of parameter interactions via corners of the landscapes that yield better or worse performance. They also note that some landscapes, particularly those that correspond to higher-dimensional optimization problems, appear to be more rugged than others.

2.1.3 Genetic Algorithms

Genetic algorithms are another widely-studied family of optimization algorithms that maintain a population of candidate solutions [72]. These solutions are iteratively randomly mutated and combined (for example, with different components of two candidate solutions swapped) and then subjected to a selective pressure, which typically only allows the highest-quality candidates to remain in the population.

Treimun-Costa et al. [179] studied a generic genetic algorithm that contained three parameters: the population size, the crossover rate and the mutation rate. Like the previous work studying the landscapes of anytime optimization algorithms, they measured the performance of a configuration as the mean solution quality the algorithm obtains over multiple independent runs of the algorithm on a particular problem instance, where individual problem instances correspond to standard objective functions, including the Rastrigin and Rosenbrock benchmark functions (see Chapter 2.1.1).

From their analysis, they concluded that the landscapes contained several local minima and, in some cases, even a small number of sub-optimal funnels (that is, collections of local minima from which it is difficult to escape, even when using strong perturbation operations). However, the mechanism they used to identify local minima may incorrectly classify sub-optimal plateaus as local minima (for more details, see Chapter 2.2.7). Furthermore, because, like much of the work before them, they simply take the mean solution quality over multiple independent runs, it is unclear how many of the local minima they observed are simply random artifacts. Nevertheless, it seems unlikely that the sub-optimal funnels they observed in some of the landscapes were simply spurious features, and hence we surmise that there are at least some sub-optimal local minima in the landscapes corresponding to certain individual problem instances. Nevertheless, because we are primarily interested in the landscapes of algorithms over distributions of problem instances, $\mathcal{I}$, instead of the landscapes of algorithms on individual problem instances $i \in \mathcal{I}$, this result should be considered orthogonal from the contributions of this thesis.

Curiously, they also show that problem instances with more complex objective functions tend to yield algorithm configuration landscapes with more sub-optimal funnels, suggesting a correlation between the difficulty of the underlying problem

instances and the corresponding algorithm configuration scenarios.

2.1.4 Ant Colony Optimization

The $\mathcal{MAX}$–$\mathcal{MIN}$ ant system ($\mathcal{MM}$AS) [170] is another iterative anytime optimization algorithm that maintains a population of candidate solutions, called "ants". At each iteration, there are two parameters, α and β, which trade off between the strength of attraction that an ant has towards making locally greedy steps compared with making moves that correspond to high pheromone trails. A particular step is assigned a high pheromone trail if it is a step that was taken before that proved to be helpful. A third parameter is also used to control the speed with which pheromone trails "evaporate" over time.

Yuan et al. [196] studied the landscape induced by seven parameters of the $\mathcal{MM}$AS for solving TSP instances, as well as various subsets of those parameters. Overall, similar to their results for PSO, they showed that several of the parameter values that yield good-quality configurations are highly correlated, which suggests that some strong parameter interactions exist. In contrast to the landscape for PSO, they showed that the fitness distance correlation is relatively high, suggesting that the scenario should be easier for configurators.

Interestingly, they also showed that the α and β parameters are typically set to integer values by the configurators, which yields their optimal performance. They observe that the reason for this is because the two parameters are used as exponents when determining the relative strength of attraction for their two corresponding factors. Since exponentiation with integers is typically handled by compilers with multiplication, this is much more efficient than the Taylor series expansions that are used to perform exponentiation with non-integers.

In some earlier work on ant colony optimization algorithm configuration scenarios, Yuan et al. [195] studied the effect of disabling or enabling a local search heuristic in the ant colony optimization algorithm on the landscapes for the remaining parameters. They observed that when the local search heuristic is enabled, the landscape for the remaining parameters is substantially more flat, which indicates that categorical parameters can cause substantial changes in the landscapes of the remaining numerical parameters.

2.1.5 Gradient Descent Step Size

Gradient descent and its many variants are another class of optimization algorithms [105]; however, unlike those discussed in Chapters 2.1.1–2.1.4, they typically only track a single candidate solution, which is iteratively updated by taking small steps in a gradient-based descent direction.

Correctly setting the (initial) choice of this step size parameter is important. In fact, it is a well-known example of a parameter that frequently yields a uni-modal response in performance – if set too large, the optimization algorithm overshoots and diverges; if set too small, the optimization algorithm converges to the optimal solution very slowly. To the best of our knowledge, even though this is common knowledge among those familiar with (stochastic) gradient descent, this behaviour was only recently demonstrated empirically in a statistically principled way [9].

Asi and Duchi [9] studied the sensitivity of several variants of stochastic gradient descent on the initial step size parameter. In particular, they evaluated several different initial steps sizes for each method when applied to four different optimization problem instances. They recorded the performance of the algorithm as both the number of iterations to convergence and the solution quality after a small, fixed number of iterations. The main focus of their study was to show that stochastic gradient descent methods could be designed to be more robust to the initial choice of this parameter value. However, because they performed multiple independent runs of each algorithm and included 90% confidence intervals for the performance of the algorithms, we can easily examine their plotted results to see that in every case the initial step size parameter yields a uni-modal response in performance.

2.1.6 AutoML Pipelines

Machine learning has made many impressive contributions to a broad range of applications through the development of a diverse set of models and training algorithms [5, 33, 45, 108, 110]. However, no single technique dominates for all applications; therefore, for many applications, there are benefits in evaluating and comparing many approaches. Furthermore, it is well known that hyper-parameter settings can strongly impact model quality [22]. Therefore, significant attention has recently been devoted to developing automated machine learning (*AutoML*) meth-

ods to address this problem (for example, see Bergstra and Bengio [21], Bergstra et al. [22], Falkner et al. [59] or Li et al. [113]). However, we are only aware of two recent publications that study the loss landscapes of AutoML pipelines.

In the first, Garciarena et al. [69] make several weakly substantiated claims, for example: that AutoML methods typically optimize the *training loss* of a model, which is trivially false (see for example, Bergstra et al. [22], Eggensperger et al. [55] or Falkner et al. [59]). They also claim that multiple distinct local optima exist in the landscapes they studied. From the context of this statement, we surmise that they substantiate this claim by the fact that multiple distinct machine learning models each contain hyper-parameter configurations that yield similarly-good solutions. However, given the neighbourhood operator *that they define*, each of these "local" minima would be directly connected.

Independently and in parallel to a major part of our work, Pimenta et al. [142] studied the landscapes of AutoML pipelines that arise from the combination of 18 possible preprocessors and 23 classification algorithms from scikit-learn [141], when applied to six different datasets. They measured the performance of the algorithm using the weighted F-score, which they approximate using 5-fold cross-validation. They concluded that AutoML pipeline landscapes tend to be flatter close to high-quality solutions and that FDC [97] (one of the most widely-used metrics for fitness landscape analysis, see Chapter 2.2.2) is a poor metric for characterizing AutoML loss landscapes, necessitating further study.

2.1.7 Neural Architecture Search

Clearly, the work in this field is closely related to that on developing AutoML pipelines; however, NAS provides the unique additional challenge that neural networks have architectures that are graphical in nature, thereby imposing additional constraints on the parameterization of the target algorithms. Consequently, the algorithm configuration landscapes that arise in NAS scenarios may (or may not) share similar properties with those of traditional AutoML pipelines.

Recently, neural architecture search landscapes have also been studied with fitness landscape analysis techniques. For example, Rodrigues et al. [157] studied the landscapes of a neuroevolution procedure for optimizing the performance of con-

volutional neural networks (CNNs). In particular, they compared the performance responses induced by three different types of neuroevolution mutation operations. The first modifies the hyper-parameters of individual neurons, for example, the choice of the activation function. The second modifies the hyper-parameters of the optimization procedure used to train the CNNs, for example, the learning rate. Finally, the third modifies the topology of the network itself.

They measured the performance of a given configuration as the categorical cross-entropy loss when calculated on a held-out test set. They studied the landscapes on four different well-known datasets, including MNIST [109] and CIFAR-10 [107]. From their analysis, they concluded that it is easiest to configure the hyper-parameters of the optimization procedure, because they yield the smoothest responses, and hardest to configure the topology-related hyper-parameters, because they yield the most rugged responses.

Nunes et al. [134] applied fitness landscape analysis methods to the validation accuracy obtained by various graph neural network (GNN) architectures. They estimate the validation accuracy using a single run for a given configuration with a given train/validation split. Unlike Rodrigues et al. [157], they do not consider the hyper-parameters of the optimization procedure in the landscape, instead they focus only on parameters that relate specifically to the architecture of the GNN, for example, the activation functions used.

They conclude from a fitness distance correlation analysis that the landscapes that arise for neural architecture search on three different datasets should be relatively simple to optimize, with most high-quality configurations grouped together in the search space. They also speculate from their results that the landscapes do not contain a substantial amount of neutrality (or "flatness", see Chapter 2.2.10), but call for further study on the matter.

2.1.8 Double Descent Curves

Somewhat related to our work is the recent discovery of the so-called "double descent curve" that appears for many neural networks (see for example, Belkin et al. [19] and references therein), which shows empirical evidence that there are two modes in the AutoML loss landscapes of neural networks. As model com-

plexity increases, you pass from an "under-parameterized" regime to an "over-parameterized" regime – that is, in the under-parameterized regime there is insufficient flexibility in the model to perfectly memorize the training data, hence modifying model complexity corresponds to trading off between under- and over-fitting. However, somewhat surprisingly, it appears that a second mode of even-better models can exist in the over-parameterized regime.

Belkin et al. [19] speculate that this is an example of Occam's razor, that is, that a large number of *smooth* models exist in the over-parameterized regime that can perfectly learn the training data and generalize well to unseen data. They show evidence of double descent curves for random forests and xgboost, and hypothesize that this phenomenon is ubiquitous among all sufficiently expressive models. Our results do not contradict theirs; most (if not all) of the landscapes we study (see Chapter 4.1) are likely restricted to being entirely within one of the two regimes. Furthermore, Belkin et al. [19] conjecture that the peak between the two modes is very narrow and may be easily missed by discretization of hyper-parameters (as must be done for empirical landscape analysis).

2.2 Landscape Analysis Methods

The analysis of the landscapes induced by optimization problems is a well established topic, that can be traced back to the seminal work by Wright [191] on the study of evolutionary biology. Since then, a vast number of techniques have been proposed for what is typically referred to as *fitness landscape analysis*. In this thesis, we do not attempt to fully enumerate or characterize all of these methods; instead, we focus only on those methods which have been previously applied to study algorithm configuration landscapes and closely related problems. We also include the more influential and well-known methods that have been prominently applied in the study of other problems. For a summary of the methods we discuss, see Table 2.1; for a more exhaustive overview of the work on fitness landscape analysis in related fields, see Hoos and Stützle [84], Malan [120], Malan and Engelbrecht [121], Pitzer and Affenzeller [143] or Watson [186].

Landscape Method	Global	Local	RS	RW	SW
Uni-Modality	✓				
FDC	✓		✓		
Dispersion	✓		✓		
Information landscapes	✓		✓		
Optima Statistics	✓	✓			✓
Regional Structures		✓			✓
Local Optima Networks	✓	✓			✓
Geomorphons		✓	✓		
Ruggedness	✓	✓		✓	
Neutrality	✓	✓			✓
Parameter Importance	✓	✓	✓		✓

Table 2.1: Summary of the most closely related work on landscape analysis methods. Includes whether or not the method provides insights into the global or local shape of a landscape (or both). Clearly, all methods can be applied if the full landscape is evaluated; however, some can also be efficiently approximated with a random sample (RS) of solutions, a random walk (RW) or a selective walk (SW).

2.2.1 Modality

Perhaps the simplest and most obvious question to ask about a landscape, is whether or not it is uni-modal. Here, the term *modality* comes from the statistical analysis of distributions, where a mode corresponds to a single, connected region with a high probability density. In an optimization problem, a mode similarly corresponds to one or more neighbouring solutions which all exhibit a locally optimal solution quality. If an optimization problem is uni-modal, then there is only a single such local optimum, which is also the global optimum.

Uni-modality is perhaps most intuitively defined for numerical optimization problems, for which candidate solutions have a natural ordering. For uni-modal numerical landscapes taking small steps in the direction of the gradient (that is, using gradient descent) will result in converging to the same region of the solution space, regardless of the initial starting position.

However, uni-modality can also be defined for combinatorial optimization problems. In our application, the combinatorial nature of algorithm configuration sce-

narios arises due to categorical parameters. By definition, categorical parameters have no natural ordering. As a result, a landscape that contains only a single categorical parameter must always be uni-modal. However, for landscapes with two or more categorical parameters, the distance between two candidate solutions can be defined as the number of categorical parameters whose values are different between the two solutions, thereby inducing an ordering and hence the possibility that the landscape may not be uni-modal. For uni-modal combinatorial landscapes taking small descent steps in a suitably defined neighbourhood (for example, single parameter flips) also leads to the same final region of the solution space, regardless of the initial starting position.

Without additional knowledge about the nature of the optimization problem (for example, that it is a linear program [163]), the only way to identify whether or not the landscape of a given problem instance is uni-modal is to exhaustively verify it by examining every solution. However, for any combinatorial problem instance of a non-trivial size, this can be prohibitively expensive.

2.2.2 Fitness Distance Correlation (FDC)

Arguably the most widely-known and commonly-used method for analyzing the global structure of a landscape is known as *fitness distance correlation* (FDC) analysis [97]. FDC analysis does exactly as it suggests; it analyses the correlation between the fitness of a solution and its distance to the closest globally optimal solution. Most often FDC is calculated using Pearson's linear correlation coefficient; however, other variants use, for example, Spearman's rank-based correlation coefficient (for example, see Pimenta et al. [142]).

Obviously, exactly calculating the fitness distance correlation coefficient for non-trivially-sized problem instances is also computationally prohibitive, and thus it (and in fact, most landscape analysis methods) is typically computed on a random sample of candidate solutions [84]. In cases where a globally-optimal solution is unknown, an approximate version of FDC can also be computed where the distance to the nearest best known solution is used instead [143].

FDC coefficient values range between -1 and 1. For minimization problems large values correspond to problem instances with positive correlation between

solution quality and distance to the nearest global optimum. Negative values correspond to deceptive landscapes, where the best solution is hidden among poor quality solutions. Usually, values near 0 are taken to indicate that the problem instance has solutions of high and low quality spread throughout the landscape. Intuitively, if a landscape has high FDC, then it must be globally "funnel-shaped" and therefore the highest-quality solutions must be relatively close together in the landscape. However, high FDC does not necessarily imply uni-modality, nor do all uni-modal landscapes have high FDC. For example, the very simple uni-modal function $f(x) = \frac{1}{x} + \sqrt{x}$ over the domain $0.0001 < x < 10$ has a (linear) FDC coefficient of approximately -0.006, because there are very poor solutions very close to the left of the optimal value and medium-quality solutions much farther to the right of the optimal value. Nevertheless, since its introduction, FDC has been prominently used to study and characterize the difficulty of a wide variety of landscapes, including those arising in automated algorithm configuration (for example, see Harrison et al. [79], Nunes et al. [134], Pimenta et al. [142], Pitzer and Affenzeller [143], Tanabe [173], Treimun-Costa et al. [179] or Yuan et al. [196]).

2.2.3 Dispersion

In a similar vein to FDC, *dispersion* [118] seeks to quantify the global structure of a landscape by identifying whether or not the set of high-quality solutions are all densely clustered together. Unlike FDC, dispersion was originally designed to be computationally efficient, and hence it only requires the evaluation of a set of randomly sampled candidate solutions. In particular, given a threshold, p, and a set of random candidate solutions, s, dispersion is calculated as the mean pair-wise distance between the best p percentage of the solutions in s.

In general, if the difference between the dispersion with a low value of p and a high value of p is negative, then this corresponds to a *low dispersion* landscape, whereas a high value corresponds to a *high dispersion* landscape [118]. Any uni-modal landscape must have low dispersion; however, a low-dispersion landscape can also contain an arbitrary number of distinct local minima, provided that the highest-quality local minima are all clustered within a small region of the solution space. Similar to FDC, dispersion has been prominently applied to study a vari-

ety of landscapes, including algorithm configuration landscapes (for example, see Harrison et al. [79], Nunes et al. [134] or Tanabe [173]).

2.2.4 Information Landscapes

Borenstein and Poli [30] introduced *information landscapes* as an alternate means to study a landscape. In particular, an information landscape is constructed by creating a matrix where each row and column corresponds to one of the candidate solutions in the original landscape. They then fill in the matrix with a 1 in locations for which the solution corresponding to the row is greater than that corresponding to the column, a 0.5 if they are the same and a 0 everywhere else.

Borenstein and Poli [30] argue that the information landscape can be a more informative way of studying a landscape, because it more closely aligns with how search procedures view the problem. That is, because a search procedure has no direct means of determining how far away from the optimal solution a candidate solution is, they effectively use a pairwise comparison between two candidate solutions as a proxy to determine which solution is *assumed* to be closer to the optimal solution. In this light, the *information* present in an information landscape can either be: *positive*, that is, the comparison of the solution qualities accurately reflects which candidate solution is closer to the optimal solution; *negative*, that is, the comparison of the solution qualities is misleading, because the candidate solution with better quality is farther from the optimal solution; or *absent*, because the two candidate solutions have equal quality. Borenstein and Poli [30] then define the *degree* of information that is present in a landscape as the fraction of elements in the matrix that are not 0.5.

Furthermore, Borenstein and Poli [30] propose to measure the distance between two information landscapes as the normalized sum of the absolute value of the differences between each element in the two information landscape matrices. Using this distance metric, Malan and Engelbrecht [122] proposed the *information landscape negative searchability* (IL_{ns}) metric, which measures the distance between a given information landscape and the D-dimensional sphere function, $f(x) = \sum_{i=1}^{D} x_i^2$. To do this, they first shift the global optimum of the sphere function to align it with global optimum of the original landscape. Malan and Engel-

brecht [122] proposed the term *negative searchability*, because large values of the metric correspond to landscapes that should be hard to optimize. The information landscape negative searchability metric provides a completely different take on analyzing the global structure of a landscape compared to FDC and dispersion. However, care needs to be taken with its interpretation. In particular, other functions that may be easy to optimize for many search procedures (for example, $f(x,y) = |x-y|$) may end up receiving relatively large IL_{ns} scores due to their dissimilarity from a quadratic function.

In their original application, information landscapes were proposed to study optimization problems with deterministic objective functions, hence it was trivial to determine whether or not the quality of two candidate solutions are equal. However, to apply them to algorithm configuration landscapes, care needs to be taken when determining whether or not two configurations are considered to have equal performance, otherwise the outcome of the results may differ substantially for some problem instances. Borenstein and Poli [30] suggested that it may be possible to make use of other intermediate values between 0 and 1 in some applications. However, even though Harrison et al. [79] used the IL_{ns} measure to study PSO configuration landscapes (see Chapter 2.1.1), they did not mention how they determined configuration equality nor whether or not they used any additional intermediate values, hence we surmise they treated means of the 30 independent runs of the target algorithm as deterministic values.

2.2.5 Optima Statistics

For multi-modal problems, a variety of statistics computed on a set of local optima have been proposed as features that can be used to estimate the difficulty of a landscape for a given optimization algorithm [2, 125, 186]. For example, Abell et al. [2] used a local hill-climbing procedure initialized at random candidate solutions to identify a set of local optima. They then measured several statistics, including the fraction of local optima with equal solution quality to the best found optimum, the mean and standard deviation of the pairwise distance between the local optima, and the mean and standard deviation of the distance between each local optima and the nearest best known optimum.

Many of the features used by Abell et al. [2] are somewhat similar, because they were chosen to be computationally efficient to obtain. This efficiency allowed them to be used to predict which configuration of an optimization algorithm should be used to attempt to solve a given problem instance. This application requires the ability to compute the statistics in substantially less time than the problem instance requires to be solved. This also means that their set of features could be suitable for studying algorithm configuration landscapes, where the cost to evaluate a single candidate solution is typically rather high. However, they would need to be adapted to handle the non-determinism that arises in the objective functions of algorithm configuration landscapes, which makes it more difficult and costly to determine whether or not a particular configuration is a local optimum.

2.2.6 Regional Structures

Several local or regional structures can be defined that comprise a landscape, for example, plateaus, basins and barriers [84]. Informally, a *plateau* is a flat region of a landscape. Formally, a plateau is simply a connected set of candidate solutions that all yield equal solution quality. Similarly, a *basin*, for a given solution and solution quality, is defined to be the set of all solutions that can be reached from the solution by a path through the solution space that does not exceed the given solution quality. In contrast, a *basin of attraction* for a given local minimum is the set of all solutions that are connected to the local minimum by a path through the solution space for which the quality of the solutions decreases monotonically. Furthermore, both *strict* and *non-strict* basins of attraction can be defined, which corresponds to excluding or including those solutions that can reach more than one local minimum via such a path. A closely related concept is the *barrier level* or *barrier height* between two candidate solutions, which is the smallest solution quality at which a path can be constructed between the two solutions such that none of the solutions on said path yield a solution quality larger than the barrier height.

Throughout this thesis we will sometimes refer to these concepts; however, we do not discuss them in more depth here, since the primary focus of the landscape analysis in this thesis is instead on the global structure of algorithm configuration landscapes. For a more technical and thorough discussion of each of these and

other related structures, see Hoos and Stützle [84]. Furthermore, to the best of our knowledge, relatively little work has been done on the study of the presence of these structures in algorithm configuration landscapes. The only exception to this is the application of local optima networks, which is a closely related concept that we discuss in more detail in the following section.

2.2.7 Local Optima Networks

Local optima networks were first introduced by Ochoa et al. [135]. They provide a method for efficiently summarizing the global (and, to some extent, regional) structure of a landscape without boiling it down to a single summary statistic, which allows them to provide deeper insights into the structure of the landscape. In their original formulation, each vertex in the local optima network graph corresponded to a basin of attraction. Edges were placed between two vertices if their corresponding basins of attractions were adjacent. Later, a variant was introduced that includes weighted directional edges, where the weight of an edge corresponds to the probability of transitioning from one basin of attraction to another with a given movement operator [136].

This variation works particularly well with, for example, iterated local search procedures such as BasicILS [87] (a variant of ParamILS, see Chapter 2.3.4), which can be adapted to extract local optima networks while solving a particular problem instance [179]. In particular, if we assume that the termination of each local search procedure performed with BasicILS corresponds to a local optimum, then it can be recorded as a vertex in the local optima network. Edges can then be introduced any time a perturbation operation causes the configuration procedure to move from one local optimum to another. By performing many independent runs of the iterated local search procedure and counting the fraction of times that each edge is taken to move from one local optimum to another, the weights can be approximated.

While local optima networks provide a very useful overview of a landscape, this method of extracting local optima networks is not without flaws. For example, the local search procedure used by BasicILS may not terminate at a local optimum, instead, it may terminate at a plateau. However, even more problematic with the method used by Treimun-Costa et al. [179] is that they do not properly account for

the variability in the performance measurements for their randomized algorithm. Instead they simply take the mean solution quality over 30 independent runs of their algorithm and treat this as if it were the exact solution quality. As a result, it is unclear whether or not any of the "local optima" in the network correspond to features of the underlying landscape or merely to random fluctuations in the performance measurements.

2.2.8 Geomorphons

Recently, *geomorphons* were proposed as a method to classify the physical landforms present in a digital elevation model of Poland [96]. Inspired by this work, Harrison et al. [80, 81] proposed to use geomorphons to classify the landforms present in algorithm configuration landscapes.

Geomorphons are landscape structures that often occur at a smaller scale than the plateaus, barriers and basins discussed in Chapter 2.2.6. In particular, geomorphon classification is performed by first constructing a three-by-three grid around a particular candidate solution and counting the number of neighbors of that solution that have better, worse or equal solution quality. Based on these numbers, the solution can be classified as being one of eight different geomorphons. For example, if all eight neighbours have worse solution quality than the particular candidate solution, then the solution is considered a *pit*; if all of them have better solution qualities, then it is considered a *peak*; or, if all them are the same, then it is considered a *valley*. By counting the total number of each type of geomorphon present in a landscape, one can gain some insights into the types of structures present.

However, one major limitation of geomorphons is that they are only well-defined for landscapes that contain two numerical parameters. While it may be possible to extend the classification schema to higher dimensions, the intuitive interpretation of many of the landform types would become challenging.

Geomorphons also heavily aggregate information, which can abstract away some important details. For example, Harrison et al. [80] show an example of a "shoulder" geomorphon that contains three adjacent neighbours that move downwards, with all of the other neighbours at equal solution quality to the solution at the center of the three-by-three grid. Intuitively, this seems like a reasonable de-

scription for this landform shape. However, the shoulder geomorphon also includes all three-by-three grids such that any three neighbours move down with the remaining six flat. In particular, this includes the case where the two bottom corners are moving downwards, along with the top middle neighbour, which is a landform that does not intuitively represent a shoulder.

Furthermore, like most studies of algorithm configuration landscapes, Harrison et al. [80, 81] define two neighbours to be of equal solution quality if their performance scores are within a small absolute tolerance, in this case, 10^{-8}. However, they do not report any measure of the variability in the performance measurement between independent runs of the algorithm, so it is unclear if this was a suitable choice or if it will introduce spurious artifacts into their analysis.

2.2.9 Ruggedness

Intuitively, the ruggedness of a landscape is typically considered to relate to the tendency of neighbouring candidate solutions to obtain similar solution quality. This is often closely related to the number of local optima in a landscape, since if a landscape is highly rugged then it is likely that many solutions will be local optima. Landscapes with low degrees of ruggedness are generally assumed to be easier to optimize, because the quality of a given candidate solution will provide more information about the quality of its neighbours; from the perspective of information landscapes (see Chapter 2.2.4), low degrees of ruggedness should typically correspond to higher degrees of positive information.

Perhaps the most common definition for ruggedness in a landscape is the *landscape correlation function*, which is the correlation in solution quality between all pairs of candidate solutions at a fixed distance from each other in the landscape [187]. If the correlation in solution quality is large for short distances, then this implies that the landscape must be relatively smooth, with high-quality solutions grouped together in the solution space. As a result, low degrees of ruggedness should correspond to landscapes that are easier to optimize.

Of course, exactly calculating the correlation between all pairs of candidate solutions at a given length is typically infeasible. Instead, this quantity is typically estimated by means of a random walk through the landscape [187], a measure

which is typically referred to as the (empirical) *autocorrelation function* or simply *autocorrelation*. However, in order for the autocorrelation of a landscape to be representative of the landscape's correlation function, the landscape must be *isotropic* [84], which, in this context, means that the starting location of the random walk does not bias the estimation of the autocorrelation. In practice, we do not expect this to be true for algorithm configuration landscapes. In these landscapes we expect to observe small changes in parameters that yield very large changes in algorithm performance in some regions of a landscape and very small changes in other regions of the landscape. Hence we believe that any autocorrelation measurements for algorithm configuration landscapes can only be interpreted as local measures of ruggedness. For this reason, we do not further discuss the *correlation length*, which is another common and closely related measure of ruggedness [167].

Other related measures of ruggedness have also been proposed, for example, Vassilev et al. [184] proposed to perform a random walk on the landscape and then measure, for each step, whether or not the solution quality increases, decreases or remains within some small threshold from zero. Then, they calculated the *information content* of the landscape by measuring the Shannon entropy [161] of the pairs of adjacent solution quality transition types. Vassilev et al. [184] also proposed to calculate the *information stability* as the smallest value for the tolerance such that the landscape is flat. They further defined the *partial information content* to be the number of local optima encountered along the walk divided by the length of the walk. Despite the fact that Vassilev et al. [184] originally positioned these measures as quantifying something other than ruggedness, the information content of a landscape has come to be referred to as an *entropic measure of ruggedness* by some (for example, see Rodrigues et al. [157]). However, like autocorrelation, all of these measures should be viewed as local measures of ruggedness rather than global ones, since they are also based on random walks.

Another related measure to the information content, is a geomorphon-based entropic measure of ruggedness that was proposed by Harrison et al. [80]. In particular, they proposed to measure the ruggedness by calculating the entropy of the various landform classification geomorphons in the landscape.

However, one of the problems with all of these measures when applied to algorithm configuration landscapes is that none of them take into account the variabil-

ity between independent estimates of the performance of the algorithm. Without proper treatment this variability can increase the apparent ruggedness of the landscape – that is, a landscape that appears smooth on a large number of problem instances may appear rugged if it is not evaluated on a large enough instance set.

2.2.10 Neutrality

Closely related to the ruggedness of a landscape is its neutrality [154]. Informally, landscapes that are highly *neutral* are those that contain a large number of neighbouring solutions with equal solution quality – that is, a landscape with high neutrality is one that contains many plateaus (or perhaps one large plateau). While clearly related to ruggedness, low degrees of neutrality do not necessarily imply high degrees of ruggedness and *vice versa* [154]. Similarly, high neutrality does not necessarily make a landscape easier to optimize – in fact, the opposite can be true, as an optimization procedure may have difficulty knowing in which direction it should search if it finds itself at a sub-optimal plateau.

Nevertheless, neutrality has been extensively used to study a variety of different types of landscapes (for example, see Nunes et al. [134], Pimenta et al. [142], Reidys and Stadler [154] or van Aardt et al. [181]). Similar to measures of ruggedness, neutrality can be calculated using random walks. For example, van Aardt et al. [181] measured the neutrality in neural network loss landscapes by measuring the percentage of a walk that was neutral, as well as the longest segment of the walk that was neutral. However, neutrality is also frequently defined in other ways; for example, Vanneschi et al. [183] defined the *neutrality ratio* of a candidate solution as the percentage of its neighbours which have the same solution quality as the original candidate solution. This method was later adopted by Pimenta et al. [142] when studying the loss landscapes of AutoML pipelines.

2.2.11 Parameter Importance

While somewhat orthogonal to the work done most commonly in the fitness landscape analysis community, some of the most common work done to study algorithm configuration landscapes is the identification of the importance of each parameter of an algorithm. Many such methods exist to quantify the importance of

parameters and, in some cases, their interactions.

Many of these methods are local, for example, *ablation analysis* [61], which attempts to determine which parameters contribute the most to the difference in performance between two given configurations. Starting with one of the configurations, it first evaluates the change in performance associated with individually changing the value of each parameter from the first configuration to the value of the second configuration. Whichever parameter value change brings the performance of the original configuration closest to the second configuration is considered the most important. The first configuration is then updated such that its value is equal to that from the second configuration, and the process repeats to find the next most important parameter.

Clearly, a brute-force implementation of ablation analysis can be computationally expensive, as it requires evaluating a potentially large number of configurations of an algorithm on a potentially large number of problem instances. To reduce this cost, Fawcett and Hoos [61] proposed to use the F-Race procedure [26] to speed up the comparisons between configurations. Later, Biedenkapp et al. [23] proposed to replace the evaluation of the configurations' performances with the predicted performance obtained by a surrogate model (a random forest) fit to data collected by one or more previous runs of a configuration procedure.

In a separate line of work, Biedenkapp et al. [24] propose an alternative formulation of *local parameter importance (LPI)*. It evaluates the importance of the parameters around a particular configuration. For each parameter, they independently evaluate the performance of modified copies of the configuration using a grid of values for the parameter. They then measure the importance of each parameter based on the fraction of the variance in the performance changes that can be explained by each of the individual parameters. Similar to their previous work on ablation analysis [23], they proposed to speed up this process by replacing the actual performance of the algorithm with the predicted performance from a surrogate model.

While local parameter importance methods can be very useful, they are less helpful for our particular goal of identifying the global structure present in algorithm configuration landscapes. For example, they cannot help us determine whether or not parameters can safely be configured independently.

As a global parameter importance method, Hutter et al. [92] proposed *forward selection*, which attempts to first identify the most important set of parameters. To do this, it trains a machine learning model to predict the performance of the algorithm based on the parameter values (and, in their case, problem instance features). They start with an initially-empty set of parameters and iteratively test how much the accuracy of the model increases as the parameters are added one at a time. The parameter that yields the largest increase in validation loss is added to the set and considered an important parameter. This is repeated until a desired number of parameters have been selected. Then, to identify their relative performance, they use the same mechanism as Friedman [66] and Leyton-Brown et al. [112], and measure the decrease in RMSE obtained by individually omitting each of those features from the model. The largest decrease in RMSE is normalized to have an importance score of 100.

Perhaps the most relevant global parameter importance technique is *functional analysis of variance (ANOVA)* or *fANOVA* [93]. Functional ANOVA is somewhat similar to LPI, in that it assesses the importance of each parameter by quantifying the degree of variance in performance that is explained by each of the parameters. However, it does so on a global scale, and it can be used to quantify the importance of the interactions between all possible subsets of the parameters. To make the computation more efficient when applied to algorithm configuration scenarios, fANOVA is typically calculated using the performance estimates from a surrogate model; however, it can also be calculated using a grid of pre-evaluated parameter configurations [103].

One property of functional ANOVA is that it is particularly sensitive to parameters for which a small fraction of their values yield extremely bad performance. Furthermore, while it returns technically sound results, they are nevertheless unintuitive, which can obscure a lay-person's understanding when parameters interact strongly. For example, the function $f(x,y) = |x-y|$ over the domain $x,y \in [-1,1]$ would be assigned no importance to each of x and y individually, whereas the interaction effect would be assigned 100% of the importance. While this is accurate in a mathematical sense, it obscures the fact that any independent change to either x or y will result in a change in the function's value.

Probst et al. [146] propose a very different take on the importance of parame-

ters, in which they focus on the so-called tunability of the parameters. In particular, they define the *tunability* of a given parameter to be the performance gain that can be achieved by configuring it independently compared to some reference default configuration. Similarly, they define the joint tunability of two parameters to be the performance gain that can be obtained by configuring both of those parameters relative to the best performance that can be obtained by configuring either one of them independently. They also define the tunability of the algorithm itself, for a given problem instance or set of problem instances, as the maximum performance gain that can be obtained by configuring the parameters of the algorithm, again relative to some default configuration. Clearly, like many other methods for calculating parameter importance, it would be prohibitively expensive to exactly calculate all of these quantities. Therefore, like much of the work before them, Probst et al. [146] proposed to use a random forest model as a surrogate, which is then queried to obtain performance estimates that are used in place of the true performance of the algorithm.

2.3 Automated Algorithm Configuration

The most basic strategy for algorithm configuration is simply manual trial and error. Typically, when algorithm configuration is performed by hand in this way, the algorithm is only tested on a limited number of problem instances. Slightly more advanced techniques automate this process with a grid or random search over the parameters [21]. While even simple techniques like random search are known to substantially outperform manual or grid search [21], there still remains significant room for improvement by more intelligent optimization and exploitation strategies. In the following, we provide an overview of many such possible improvements for the automated configuration of algorithms.

2.3.1 Racing

Birattari et al. [26] proposed racing procedures for general purpose algorithm configuration. The key idea is to begin with an initial set of candidate configurations and then iteratively evaluate them on new problem instances interleaved with statistical tests to determine as quickly as possible when challengers can be discarded

with confidence. Specifically, Birattari et al. [26] recommended to use the Friedman test. However, the initial version of F-Race is often prohibitively expensive to apply in practice, since they began the race by evaluating all possible configurations of an algorithm.

A family of methods known as iterated racing procedures overcomes this limitation, whereby racing procedures are iteratively applied to a set of randomly generated configurations. In these methods, the winners of each race are used to bias the random sampling procedure towards high-quality regions of the configuration space. Balaprakash et al. [15] introduce such a method for numerical parameters, which Birattari et al. [27] later extended to handle categorical and conditional parameters.

By their nature, racing procedures are an embarrassingly parallel method. This fact was first exploited in an implementation by López-Ibáñez et al. [117], which was later improved to include an adaptive capping procedure [35] (the variant we study in Chapter 7). In a separate line of work, Styles and Hoos [171] proposed a permutation test to determine the outcome of individual races, arguing that the rank-based F-Test only indirectly optimizes mean running time. It was this work that inspired the use of permutation tests in our work (see Chapters 7.1.1, 7.1.2 and 7.1.6).

Yuan et al. [195, 196] also proposed to combine racing procedures with various existing continuous black-box optimization procedures, including MADS [10], CMAES [78] and BOBYQA [144]. Notably, when F-RACE was combined with CMAES, an optimization procedure that is known to work relatively well on objective functions that are globally uni-modal, it was found to be competitive with the other variants on all of the benchmarks they studied. Their benchmarks configured the parameters of particle swarm optimization and ant colony optimization algorithms to improve the final solution qualities found by the target algorithms.

2.3.2 Exploiting Low Fidelity Approximations

A closely related concept to the one underlying racing procedures is to make use of "low-fidelity" approximations of an algorithm's performance to quickly eliminate poorly performing configurations. This terminology and these methods arise

frequently in AutoML. These are typically used in place of racing procedures in AutoML because the number of problem instances (here: training and validation splits) is typically kept to a very small number, for example 1–5. However, Jamieson and Talwalkar [95] point out that many machine learning methods are iterative by nature, and thus the final validation loss of a machine learning method can often be cheaply approximated by examining the validation loss of the method after an intermediate number of training iterations.

With this insight, Jamieson and Talwalkar [95], introduced the pioneering bandit algorithm known as successive halving. The procedure is straight-forward. First, an initial set of configurations are trained using a small training budget. Next, each configuration is compared on a validation set and the worst half are discarded. The training for the remaining configurations is resumed until they have exhausted twice their original training budget. This process iterates until a single configuration remains or the configuration budget is otherwise exhausted.

Numerous variants of successive halving have since been proposed. For example, Hyperband [113] performs several successive halving brackets with different values for the initial training budget, thereby reducing the dependence of the original method on the choice of this parameter value. Later, Li et al. [114] proposed an asynchronous version of the successive halving algorithm, ASHA, designed to scale well to very large degrees of parallelism.

Besides varying the number of training iterations or epochs to obtain low-fidelity estimates of performance, it has also been proposed to use, for example, the number of estimators in an ensemble or a subsample of the training set (for example, see Kandasamy et al. [99] or Klein et al. [104]). Other methods have also been proposed that combine multiple different types of low-fidelity approximations to further speedup the evaluation of cheap approximations [99].

While numerous methods have been proposed that combine advanced search strategies with methods for exploiting low-fidelity approximations (for example, see Kandasamy et al. [99], Klein et al. [104] or Falkner et al. [59], the latter of which we discuss further in Chapter 2.3.3), even the methods that rely strictly on random search for generating candidate configurations have been shown to perform very well in practice (see for example, Li et al. [113, 114]).

To the best of our knowledge, methods that exploit low-fidelity approximations

(not including racing procedures, which effectively exploit a specific type of low-fidelity approximation) have only been proposed for use in algorithm configuration scenarios other than AutoML scenarios a single time, by Audet et al. [14]. However, instead of proposing to exploit these approximations with a bandit method as is typically done in AutoML, they suggest that a surrogate function for an anytime optimization algorithm's [46] performance could be obtained by relaxing the optimization procedure's termination condition. (We discuss the remaining components of their proposed method further in Chapter 2.3.6.)

2.3.3 Bayesian Optimization

Bartz-Beielstein et al. [17] introduce using concepts from Bayesian Optimization for general-purpose automated algorithm configuration. This work was extended by Hutter et al. [88] to use training budgets based on running time (target algorithm + configurator time) instead of only the number of target algorithm runs. Hutter et al. [88] also argued that the large variance in the running time distributions of many algorithms posed a unique challenge to algorithm configuration procedures that are used for running time minimization (a key insight that informed many components crucial to one of the configuration procedures we propose, see for example, Chapters 7.1.1, 7.1.7 and 7.1.8). They proposed to address this in two ways. First, with a projected process model that incorporated this uncertainty and which could be built more quickly than the previous Gaussian process model). Second, via an intensification mechanism, which slowly increases the number of problem instances on which candidate configuration are evaluated. However, all of this early work was limited to configuring only numerical parameters of algorithms applied to individual instances rather than instance sets. Hutter et al. [89] later proposed a configuration procedure capable of working with instance sets called SMAC, which uses a random forest model, thereby also extending the method to configuring categorical parameters.

Since then SMAC has continued to be refined [58, 90, 116] and used as the engine underlying Auto Scikit-learn [62, 64]. In another recent extension, Anastacio and Hoos [3, 4] showed that SMAC's performance can be further improved by exploiting prior knowledge regarding high-quality configurations – that is, by biasing

the search process to spend more time evaluating configurations near to the default configuration.

Another variant of SMAC was proposed by Hutter et al. [91] that evaluated multiple candidate configurations in parallel. However, to the best of our knowledge this version of SMAC has never been made publicly available. Instead, the so-called *standard protocol* when using SMAC is to perform 10–25 independent runs of SMAC in parallel and then validate the resulting configurations on the entire training set and then to return the best one [172].

Bayesian optimization is also by far the most common approach used to configure the hyper-parameters of machine learning methods (for example, see Bergstra et al. [22], Feurer et al. [62], Kandasamy et al. [99], Snoek et al. [164] or Falkner et al. [59]). In particular, Bergstra et al. [22] proposed two of the first and most influential methods in this space. The first models the machine learning method's performance directly using Gaussian processes, and the second uses tree-structured parzen estimators (TPEs) to instead model regions of the configuration space believed to be good or bad, thereby only indirectly modelling the machine learning method's performance as a function of the hyper-parameters.

A notable extension of the work by Bergstra et al. [22] was done by Falkner et al. [59], which combined the TPE-based method with Hyperband [113] (discussed in more detail in Chapter 2.3.2), thereby yielding a method capable of quickly finding reasonable-quality configurations via low fidelity approximations, while still obtaining strong final performance by making use of Bayesian optimization. They dubbed their method BOHB, short for Bayesian optimization with Hyperband. This is the method that forms the basis of the two configuration procedures we propose in Chapter 5.

2.3.4 Iterated Local Search

In a separate line of work, Hutter et al. [86] introduced ParamILS, an iterated local search procedure for automated algorithm configuration. ParamILS was later improved with a novel *adaptive capping mechanism* [87], which is used to pick running time cutoffs when optimizing algorithms in terms of running time. The key observation behind adaptive capping is that when a challenging configuration

is being evaluated, we can limit the running time cutoff of the algorithm with a bound that reflects the total running times already used to evaluate the challenger and incumbent, thereby avoiding spending a very long time evaluating configurations that perform very poorly. Since for many $\mathcal{NP}$-Hard problems the running time of algorithms can vary substantially between problem instances, the use of an adaptive capping mechanism can substantially reduce the computational budget required for configuration.

Independently and in parallel to the work in this thesis, Hall et al. [75, 76, 77] proposed a few variants of ParamILS, and rigorously proved various properties about the configurators' performance. For example, they modified ParamILS to use a harmonic mutation operator and proved that this variant obtains optimal worst-case performance for one-dimensional uni-modal or approximately uni-modal functions [77]. However, one of the downsides of ParamILS and its variants is that they all require a discretization of the parameter configuration space.

Blot et al. [29] proposed a multi-objective variant of ParamILS, dubbed MO-ParamILS, that is suitable for automated algorithm configuration procedures in which the user needs to optimize the performance of an algorithm in terms of multiple performance metrics.

2.3.5 Genetic Algorithms

Another approach for automated algorithm configuration is the use of a gender-based genetic algorithm, GGA, proposed by Ansótegui et al. [6]. GGA applies a different selection pressure to each gender of the population. For one gender only the top X% of the population are kept, whereas the other gender is not subjected to pressure and is instead used merely to store diversity in the population. While this avoids the computation time needed to evaluate half of the population, it is also a departure from classical gender-based genetic algorithms where both populations are normally evaluated according to a fitness criteria.

Ansótegui et al. [7] later proposed a modified method, GGA++, which uses a tailor-purposed random forest model trained on the competitive gender to predict the performance of the configurations in the non-competitive gender. These predictions are then used to sample from the non-competitive population when select-

ing mates. This random forest model is also used when determining which genes to keep from each parent when mating. Surprisingly, neither GGA nor GGA++ makes use of several of the crucial components commonly used by state-of-the-art algorithm configurators, for example an adaptive capping mechanism [35, 87], an intensification mechanism [88] or a racing mechanism [26]. However, both methods are implemented in such a way that the evaluation of candidate configurations can be performed in parallel.

While less common than Bayesian optimization, genetic algorithms have also been used in various AutoML applications to configure the hyper-parameters of machine learning methods. For example, Lessmann et al. [111] showed that genetic algorithms can be used to find improved kernel functions for SVMs, albeit at a substantial computational cost. More recently, Di Francescomarino et al. [49], showed that genetic algorithms are particularly suitable in AutoML applications, where the goal is to optimize the performance of a machine learning method in terms of multiple performance metrics. This is because genetic algorithms are known to contain diversity among their populations, which often makes them competitive when solving multi-objective optimization problems.

In a separate line of work, Olson et al. [139] proposed TPOT, which uses genetic programming to automatically configure and construct tree-based AutoML pipelines. In their early work on TPOT, they only supported random forests and decision trees as binary classifiers. In fact, they even proposed to use these same classifiers for feature construction and selection. For feature construction, they used the output of classifiers built in previous stages as potential new features, and for selection they used decision trees to rank pairs of features based on the classification accuracy that could be obtained when using only those two features. They then used the genetic programming package DEAP [65] to generate pipelines that composed feature construction, selection and classification. These pipeline were tree-shaped, in that multiple copies of the data set could be fed into different preprocessors, which would eventually be merged by the time they reached the final binary classifier.

While the initial version of TPOT showed promise, it sometimes produced complex pipelines that were slow to run. To address this, Olson et al. [138] later proposed to use the highly successful NSGA-II [47] selection strategy, which is

designed for multi-objective optimization problems. In addition to optimizing for classification accuracy, they sought to minimize the complexity (that is, number of stages) of the pipeline. They then used copies of the pipelines on this Pareto front to seed the next generation of each population. Olson and Moore [137] further extended this line of research to show that this strategy can effectively find simple pipelines that obtain high accuracy across a wide variety of classification settings.

2.3.6 Pattern and Direct Search

Audet and Orban [12] proposed to use mesh adaptive direct search (MADS) algorithms to configure numerical parameters of algorithms. MADS algorithms are derivative-free optimization procedures that generalize coordinate descent algorithms. However, in contrast to coordinate descent, parameters are searched along iteratively updated meshes that may not be axis-aligned, and which grow dense in the unit sphere [14]. Unlike many of the other state-of-the-art algorithm configurators, the method by Audet and Orban [12] required candidate configurations to be evaluated on one or both of two fixed sets of problem instances. In particular, rather than evaluating each configuration on the full set of problem instances, they treated a subset of the problem instances as a surrogate function used to speed up the optimization process.

In their initial work, Audet and Orban [12] noted small amounts of stochasticity in performance measurements between repeated runs of the target algorithm; however, they left this unaddressed. This restriction was later removed by Yuan et al. [195], who proposed to hybridize MADS with F-Race [26], which yielded an improvement to both methods and compared favourably to iterated racing [15] (see Chapter 2.3.1).

Later, Audet et al. [14] further generalized the use of MADS algorithms in a framework for the optimization of algorithms (dubbed OPAL), by adding support for categorical parameters. At the same time, they proposed to improve the use of a subset of the problem instances as a surrogate via clustering. In particular, by clustering the full set of problem instances they identified groups of instances with similar characteristics from each of which a single instance could be sampled to generate a set of representative problem instances. However, to the best

of our knowledge, MADS algorithms for algorithm configuration have never been combined with adaptive capping mechanisms, which are known to be crucial for obtaining state of the art performance when minimizing an algorithm's running time [35, 87].

In a separate line of work Pedersen [140] proposed an algorithm configuration procedure called local unimodal sampling (LUS), which is essentially a pattern search algorithm. Their method was inspired by the observation that two particular meta-heuristic optimization algorithms, differential evolution and particle swarm optimization, yield algorithm configuration landscapes that are close to uni-modal. The algorithm is very simple, at each iteration it samples a new configuration uniformly around the incumbent and checks to see if it is an improvement. If it is, it is accepted, otherwise the challenging configuration is rejected and the sampling radius is decreased. Pedregosa et al. [141] showed that LUS was able to effectively configure the parameters of particle swarm optimization and differential evolution; however, to the best of our knowledge it has not been compared against, nor made available for comparison against, other existing state-of-the-art algorithm configuration procedures.

2.3.7 Gradient-Based Optimization

Despite their widespread success in a variety of other optimization problems (for example, see Bottou and LeCun [31], Cauchy et al. [36], Nesterov [133], Robbins and Monro [156] or Roux et al. [159]), very few methods exist for performing automated algorithm configuration that explicitly exploit gradient-based information. The most obvious reason for this is that gradients cannot be easily obtained through differentiation, nor is the objective "function" guaranteed to be either smooth or continuous [95].

Nevertheless, Yakovlev et al. [193] recently showed that gradient approximation via finite differences can be used to create an effective hyper-parameter configuration procedure used as a part of an AutoML pipeline. In particular, they configure each hyper-parameter semi-independently in parallel, and then use pairs of similar hyper-parameter values to estimate gradients. They use these gradient approximations to make educated guesses about which regions of the hyper-

parameter configuration space should be evaluated next.

2.3.8 Bandit-based Bi-level Optimization

Thornton et al. [175] formally defined the *combined algorithm selection and hyper-parameter configuration* (CASH) problem, which, despite its name, is most often reformulated as a single-level optimization problem that contains hierarchically-structured parameters (for example, see Feurer et al. [62, 64], Kotthoff et al. [106] or Thornton et al. [175]). However, Li et al. [115] instead argued that this unnecessarily complicates the problem for the hyper-parameter configuration procedure. They instead treat the CASH problem as a bi-level one, in which the outer level selects which machine learning method is to be used, and in which the inner level then seeks to configure the hyper-parameters of said machine learning method.

For the outer optimization problem, they propose a new bandit-based algorithm, dubbed rising bandits. Each "arm" in their formulation is a single run of a hyper-parameter configuration procedure applied to a particular machine learning method. They track the anytime performance of each arm, linearize the most recent performance improvements and use this linear model to predict the final performance of each arm. As soon as the predicted final performance of a particular arm falls below the current performance of another arm, the particular arm is removed from consideration and its hyper-parameter configurator is stopped.

In their implementation, they used SMAC as their inner hyper-parameter configuration procedure and compared against methods that configure the entire bi-level optimization process. They used a variety of benchmarks to show that their method compared favourably to other state-of-the-art Bayesian optimization configurators, for example, SMAC [89] and BOHB [59].

2.4 Chapter Summary

In this chapter, we thoroughly reviewed what little is known about the landscapes of algorithm configuration scenarios (most of which was, in fact, obtained from research performed subsequently to the publication of our first paper on algorithm configuration landscapes in 2018) and we presented an overview of the most closely related work in the fields of fitness landscape analysis and automated algo-

rithm configuration.

We observed that nearly all studies of algorithm configuration landscapes pertained only to the landscapes of optimization algorithms, wherein the performance of the algorithm was measured almost exclusively in terms of the objective function value of the final incumbent solution for the optimization algorithm. (Although there is also a small amount of recent research on AutoML loss landscapes as well.)

Much of this work analyzed the landscapes of algorithms only on individual problem instances instead of on instance sets, which, while interesting, is much less useful, since in practice one is rarely interested in optimizing the performance of an algorithm for a single problem instance. Indeed, the purpose behind automated algorithm configuration is to improve the performance of the algorithm on a distribution of problem instances, thereby realizing performance gains on future problem instances that need to be solved.

Furthermore, we showed that all existing algorithm configuration landscape analysis (both old and new) lacks the statistical sophistication necessary to handle the stochastic nature of algorithm performance. Therefore, contrary to the conclusions of some of the authors of work on algorithm configuration landscapes, we believe that there has been insufficient evidence collected so far to conclude with confidence that algorithm configuration landscapes frequently contain more than even a single mode. In addition, while we believe that clear evidence exists that the parameters of an algorithm can interact strongly, the extent to which these interactions complicate the algorithm configuration problem remains unclear.

Next, we reviewed methods for performing fitness landscape analysis, which studies the structure of objective functions that arise in various optimization problems by way of an analogy to physical landscapes. We discussed a broad range of these techniques, primarily targeting those that are most relevant for the study of the global structure of algorithm configuration landscapes.

In particular, we discussed the *modality* of a landscape as well as several local optima summary statistics that can be computed for multi-modal landscapes in order to characterize the difficulty of an optimization problem. We reviewed the widely-used *fitness distance correlation* coefficient and the *dispersion* metric, which each measure the degree to which high-quality solutions are grouped together. We also discussed *information landscapes*, which provide a different lens

for the analysis of optimization landscapes that can be useful for quantifying the degree to which a given problem instance's landscape resembles a simple landscape that should be easy to optimize.

Next, we briefly reviewed the definitions of several different kinds of regional structures that can arise in a landscape. For example, *basins of attraction* and *barriers*, which can be jointly analyzed through the use of *local optima networks* to gain a view of an optimization landscape that is both detailed and broad. We also reviewed the definition of *geomorphons*, which are used to classify small-scale features that are present in physical landforms.

Next, we reviewed several of the most common methods for quantifying the *ruggedness* of a landscape, which include the *autocorrelation* function and *information stability*, an entropic measure of landscape ruggedness. In a similar vein, we discussed some of the methods used to study the *neutrality* of a landscape, which measure the extent to which many neighbouring solutions within a landscape yield equal or similar solution quality.

Finally, we discussed the most commonly used methods for quantifying the importance of an algorithm's parameters and their interactions, for example, *functional ANOVA*, *ablation analysis* and *tunability*.

In our discussion of the related work on automated algorithm configuration, we reviewed both *racing* and the exploitation of *low-fidelity approximations* of solution quality. We observed that both of these some-what related concepts can be used to substantially reduce the running time required to perform automated algorithm configuration by using early feedback about the performance of low-quality configurations to quickly eliminate them. In each case, it has been proposed to combine the techniques with various optimization search procedures.

We further presented a summary of the most well-known applications of various derivative-free optimization techniques (that is, *Bayesian optimization*, *iterated local search*, *genetic algorithms* and *direct search*) to the automated configuration of algorithms. Among these, by far the most common and popular methods for automated algorithm configuration tend to be those with powerful meta-heuristics, which are designed to ensure that sufficient exploration is performed of the parameter configuration space so as to avoid missing potentially high-quality regions of the landscapes. However, given that relatively little is known about algorithm con-

figuration landscapes (and given the relatively competitive behaviour observed by very simple methods like random sampling), it remains unclear whether or not the sophistication and strength of the exploration procedures used by these methods is justified.

We also discussed two methods for algorithm configuration that diverge from the classic approaches: *gradient-based optimization* (via finite difference approximations) and *local uni-modal sampling*. The design of each of these methods – the first, implicitly, and the second, explicitly – suggest that the authors of the methods assumed that algorithm configuration landscapes must not be as complex as is commonly assumed by the research community. The fact that these methods have been shown to perform relatively well, further motivates our questions regarding the simplicity of the structure of algorithm configuration landscapes.

Chapter 3

Methods: Analyzing a Landscape

All of the existing fitness landscape analysis methods reviewed in Chapter 2 were designed for deterministic landscapes, that is, landscapes for which a particular candidate solution results in a deterministic measure of fitness. However, in our application the algorithm performance objectives are non-deterministic. Even for deterministic algorithms, the performance objective must be a random variable that depends upon the particular choice of problem instances used to evaluate the algorithm; however, for some performance objectives (for example, running time) the results are stochastic even with fixed random seeds and sets of problem instances (due to interference by background processes and other types of noise in the execution environment).

For some fitness landscape analysis methods, such as the fitness distance correlation (FDC) coefficient [97], this is only mildly problematic. We can still approximate the FDC coefficient by taking the mean performance for each configuration and calculating the metric as usual. While the number of problem instances used to evaluate the configurations will have some impact on how rugged the landscape appears and thus on the estimate of the FDC, this is likely to be negligible compared to the particular choice of configurations used to calculate the FDC coefficient. Provided that the variance between independent runs of the target algorithm is bounded and not too large, and that we use a sufficiently large number of problem instances to evaluate each configuration, we can still obtain a reasonably accurate estimate for the expectation of the FDC coefficient over the distribution of problem

instances.

However, some of the simplest forms of fitness landscape analysis, for example, counting the number of local minima, can no longer be trivially performed via an exhaustive search of the landscape. As a result, all of the landscape analysis methods that we introduce in this chapter place particular emphasis on the variability in an algorithm's performance estimates. To this end, many of the landscape analysis methods require a confidence interval $[\underline{m}(c), \overline{m}(c)]$ for each $c \in C$ that captures the best known estimate for the performance of configuration c according to some performance metric m. Depending on the application scenario, we calculate these confidence intervals in different ways (see Chapters 4.1 and 6.1).

In this chapter,[1] we introduce the methods that we use in Chapters 4 and 6 to analyze AutoML loss landscapes and running time minimization landscapes, respectively. By making use of confidence intervals these methods can help us answer several basic questions about algorithm configuration landscapes, which, as motivated in Chapters 1 and 2, may yield insights about whether or not existing algorithm configuration procedures are unnecessarily inefficient.

To summarize, if algorithm configuration landscapes tend to be globally unimodal or even convex (for numerical hyper-parameters), this would have significant ramifications on the kinds of search procedures that could be used. This motivates our first research question: **RQ 1. Is the global structure of typical algorithm configuration landscapes relatively benign; in particular, are they (approximately) uni-modal or convex?**

It is already well-known that most algorithm configuration landscapes depend most strongly on a small number of parameters [22]; if these parameters tend to interact weakly or not at all, they could be configured independently. This would also have substantial ramifications for many existing Gaussian-process-based methods, for which a primary bottleneck is fitting a Gaussian-process model to high-dimensional landscapes [98]. This yields our second research question: **RQ 2. Do most parameters interact strongly; if not, to what extent do they interact and**

[1] This chapter is based on the methods introduced in joint work with Holger Hoos. An early version of some of these methods first appeared in Pushak and Hoos [148], where we received the 2018 PPSN best paper award. A single award is given out every two years, selected by an expert committee with guidance from a popular vote. An extended version of the methods has been accepted for publication, pending minor revisions [151].

where?

3.1 Parameter Response Slices

Ideally, landscape analysis would be performed using the entire configuration space of the target algorithm A to be configured. However, for many applications fully evaluating the set of configurations arising from the cross-product of a modest number of values for each parameter can be prohibitively expensive. Let A be an algorithm with n parameters $P = \{p_1, p_2, ..., p_n\}$. We define a *parameter response slice* for parameters $P_{slice} = \{p_a, p_b, ..., p_k\} \subseteq P$ to be obtained by fixing all other parameters $P \backslash P_{slice}$ to their respective values in some configuration $c \in C$ and measuring the performance of A as a function of the parameters in P_{slice}; formally,

$$r(v_a, v_b, ..., v_k) = m(c|_{p_a=v_a, p_b=v_b, ..., p_k=v_k}). \tag{3.1}$$

Intuitively, a one-dimensional parameter response slice corresponds to a single, axis-aligned slice through the configuration landscape of A. Technically, it can be seen as a conditional response, subject to all other parameters being held to fixed values.

3.2 Analysis of Global Landscape Shape

To address our first research question regarding the global shape of typical algorithm configuration landscapes, we designed two methods that initially assume that the landscapes are very simple (that is, uni-modal and convex) and then test to see whether or not there is sufficient evidence to reject these hypotheses. As with any type of statistical test, the outcome can vary depending on the particular sample of data used for the test; in our case, this corresponds to the grid of parameter values that are evaluated and used to form the neighbourhood relation graph $G = (C, E)$ (see Chapter 1.3). In particular, for continuous-valued parameters, it is always possible that our test will fail to detect a barrier that separates two or more modes because it falls between two adjacent parameter values. However, all of the methods described in Chapter 2 share similar weaknesses when used to study any continuous-valued domain.

In this section we also briefly review an existing related method of analysis, FDC, which we use to study the shape of the landscapes.

3.2.1 Test for Uni-Modality

The test for uni-modality attempts to construct a piece-wise affine landscape that is both uni-modal and contained within the confidence intervals. If no such landscape exists, it rejects uni-modality. To do this, we define an augmented graph $G' = (C', E')$, where $(c, m) \in C'$ if and only if $c \in C$ and $\underline{m}(c) \leq m \leq \overline{m}(c)$; and where a directional edge $e' = ((c_a, m_a), (c_b, m_b))$ is in E' if and only if $e = (c_a, c_b) \in E$ and $m_a \leq m_b$. The method begins by finding a vertex $(c^*, \overline{m}(c^*))$ such that $\overline{m}(c^*) \leq \overline{m}(c)$ for all $c \in C$. We define c_k to be *reachable from* c^*, if and only if there exists a *certifying path* $p = (c^*, m^*), (c_1, m_1), ..., (c_k, m_k)$ in G' – that is, c_k can be reached via a path p from c^* with a sequence of non-decreasing performance values that stay within the confidence intervals for each c on p (see Figure 3.1). Technically, a certifying path p is a chain of tuples (c_i, m_i) such that

$$\begin{aligned} & m_i \leq m_{i+1}, \\ & distance(c_i, c_{i+1}) = 1, \\ & m_i \in [\underline{m}(c_i), \overline{m}(c_i)] \text{ and} \\ & c_0 = c^*. \end{aligned} \tag{3.2}$$

Clearly, if each $c \in C$ is reachable from c^*, we cannot reject uni-modality for the landscape that induced G'. In Thereom 1, we show that if there exists some $c_0 \in C$ that is not reachable from c^* then no piece-wise affine, uni-modal landscape exists within the confidence intervals of C, hence we can safely reject uni-modality for the landscape that induced G'.

Theorem 1 (Correctness of Test for Uni-Modality)**.** *Let $G' = (V', E')$ be a neighbourhood relation graph defined for a landscape that contains a set of pre-evaluated configurations C, such that each configuration $c \in C$ has a corresponding confidence interval $[\underline{m}(c), \overline{m}(c)]$ for the performance of the algorithm. If $\overline{m}(c^*) \leq \overline{m}(c)$ for all $c \in C$, and there exists $c_0 \in C$ that is not reachable from c^*, then no uni-modal, piece-wise affine function exists that is contained within the confidence in-*

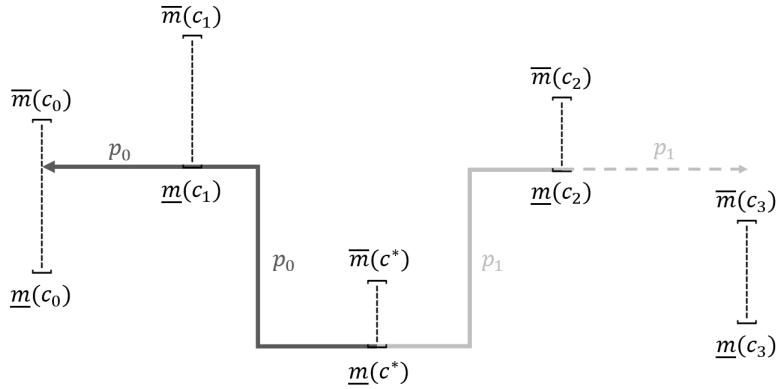

Figure 3.1: An example discretization of an algorithm configuration landscape with confidence intervals. The horizontal axis corresponds to a numerical parameter and the vertical axis corresponds to the performance of the configurations (lower is better). Dijsktra's algorithm would find certifying path p_0 from c^* to c_0 through c_1 that does not decrease in solution quality, thereby indicating that both c_0 and c_1 are reachable from c^*. However, no monotonically increasing path from c^* to c_3 exists because the lower bound of c_2 is above the upper bound of c_3. Therefore, c_3 is not reachable from c^*, and uni-modality can be rejected for the landscape.

tervals, and hence uni-modality can be rejected for the landscape.

A proof of Theorem 1 is in Appendix A.1.

We test if each configuration is reachable from c^* by running Dijkstra's algorithm [50] on the modified graph G' starting at the vertex $(c^*, \underline{m}(c^*))$. Clearly, if c_k is reachable from c^*, Dijkstra's algorithm will find a certifying path from $(c^*, \underline{m}(c^*))$ to (c_k, m_k). Furthermore, Dijsktra's algorithm finds the shortest path to each vertex and visits them in order. Hence, for each c, the first vertex (c, m) visited by Dijkstra's algorithm must contain the smallest value m out of all vertices (c, m') for which a path exists from $(c^*, \underline{m}(c^*))$. Trivially, if an edge $e' = ((c, m'), \cdot)$ is in E' and $m < m'$, then $e = ((c, m), \cdot)$ must also be in E'. We can therefore keep the time complexity of the test log-linear, by pruning all paths to vertices (c, m')

such that $m' > m$.

3.2.2 Counting the Number of Modes (Local Minima)

One commonly used feature to describe the ruggedness of a search landscape is the number and density of local optima (for example, see Chapter 5 of Hoos and Stützle [84]). A lower bound on the number of local minimum in a landscape can be obtained by repeatedly applying the test for uni-modality. Initially, the test for uni-modality is run as normal, keeping track of all reachable configurations. If there remain configurations that are unreachable, then the test is re-applied, this time starting with the configuration $(c', \overline{m}(c'))$ such that $\overline{m}(c') \leq \overline{m}(c)$ for all c that were unreachable in all previous applications of the test for uni-modality. This process then repeats until all configurations have been reached during at least one of the applications of the test for uni-modality. The number of times the test was applied corresponds to the lower bound for the number of local minima in the landscape.

3.2.3 Test for Convexity

Convexity does not apply to categorical parameters, so we fix these to their respective values in the optimal configuration prior to applying the test. Convexity is also not defined for discrete, numerical parameters. However, we can still include them in our test by checking if the piece-wise affine function that linearly interpolates between their discrete values is convex. The test converts the n numerical parameters of each configuration into a set of points with $n+1$ dimensions, where the last dimension is the upper bound of the performance metric confidence intervals.

The lower half of the convex hull of these upper bounds then corresponds to a convex, piece-wise linear function that has the largest possible performance value for all configurations, without exceeding any of the upper bounds. If this function is contained within all of the confidence intervals for the configurations, then there exists at least one convex function within the confidence intervals, and thus we cannot reject the hypothesis of the landscape being convex. We say that a configuration is *interior* to the convex hull if the lower bound of its confidence interval is contained within the convex hull. If any of the configurations are interior, then our

test rejects convexity.

Determining whether or not the hypothesis of convexity should be rejected therefore amounts to checking whether or not any of the lower bounds for a given configuration are contained within the convex hull of the upper bounds. Let $X \in \mathbb{R}^{m \times n+1}$ be a matrix containing all of the configurations of the landscape for which the first n columns correspond to the parameter values and the $n+1^{\text{th}}$ column represents the corresponding upper bounds. Let $x \in \mathbb{R}^{n+1}$ be an $n+1$-dimensional point that corresponds to a particular configuration and the corresponding lower bound.

A computationally efficient way to determine whether or not x is in the convex hull of X is to check if x can be represented as a convex combination of X – that is, if there exists $\alpha \in \mathbb{R}^m$ such that

$$\sum_{i=1}^{m} \alpha_i \cdot X_i = x, \tag{3.3}$$

where

$$\sum_{i=1}^{m} \alpha_i = 1 \text{ and } \alpha_i \geq 0 \text{ for all } \alpha_i. \tag{3.4}$$

This test can be easily implemented by encoding Equations 3.3 and 3.4 as the constraints of a linear program and then using a linear programming solver to verify whether or not any feasible solutions exist. We used the default configuration of the interior-point linear programming solver available in `scipy` [185].

One edge case that needs to be considered when applying this method is how lower bounds that are co-planar with the convex hull of the upper bounds are treated. Technically, if a single lower bound is co-planar with the convex hull of the upper bounds, then there exists only a single value that a convex function contained within the confidence intervals can take on for the corresponding configuration, and therefore convexity should not be ruled out. However, testing whether or not x is a convex combination of X does not allow us to easily distinguish between whether x is co-planar to the convex hull of X or strictly interior to it. This problem is further complicated by the numerical imprecision of floating point operations, which may introduce possible rounding errors.

Therefore, for any configuration and lower bound x that is found to be a convex

combination of X, we propose to perform a second test with x', where x' contains the same set of parameter values as x, but where its lower bound has been slightly decreased by a small absolute and relative tolerance. If x' can also be represented as a convex combination of X, then we say that the lower bound for the configuration is interior and we reject the hypothesis of convexity, otherwise we say that it is co-planar within machine precision and we do not reject the hypothesis of convexity. In our experiments, we used a relative and absolute tolerance of 10^{-5} and 10^{-8}, respectively.

3.2.4 Validation of Sensitivity

If the confidence intervals are all very large or similar (for example, the intersection of every confidence interval is non-empty), then our tests will be trivially unable to reject their null hypotheses due to a lack of sensitivity. The situation is further complicated in high dimensions by the connectedness of the neighbourhood relation graph (for example, a fully connected graph is uni-modal for any set of confidence intervals). We would not expect a landscape with completely random performance values to be either uni-modal or convex. We therefore randomly permute the confidence intervals for each landscape, thereby breaking the association between the original configurations and performance estimates, but maintaining the distribution of confidence intervals and the neighbourhood graph. If a test for convexity or uni-modality fails to reject their null hypotheses for a permuted landscape, then a similar result on the original landscape should be considered meaningless. In our experiments, we repeated this procedure three times for each landscape and considered the test *sensitive* if convexity or uni-modality was rejected for all three of the permuted landscapes.

3.2.5 Identifying "Interesting" Parameters

Parameters with almost flat responses (that is, robust ones, whose settings have little or no effect on the performance of the algorithm between their minimum and maximum allowed values) are of limited interest to our investigation. We therefore used a simple heuristic procedure to identify parameters with interesting (that is, non-flat or sensitive) responses, based on the sizes of and overlap between the

confidence intervals for each value of a parameter. Intuitively, this measure is related to how we validate the sensitivity of the statistical tests as described in Section 3.2.4, since if the response of a parameter is flat, then we should not expect the permuted response to be non-flat.

To be precise, we define a parameter's response slice to be *interesting*, if the size of the overlap between the two confidence intervals with the least amount of overlap is at most half of the average size of the confidence intervals. For scenarios where the performance of the algorithm is measured in running time, we perform this check on the log-transform of the running times, since in these scenarios, we are typically most interested in constant-factor changes in the running times.

3.2.6 Fitness Distance Correlation

Fitness-distance correlation (FDC) [97] measures the degree to which the "fitness" of an objective function is correlated with the distance from an optimal configuration. Technically, let $d(c)$ be the distance of configuration c from the nearest optimal solution c^*. Then, given the fitness-distance pairs $(m(c), d(c))$ for all candidate configurations $c \in C' \subseteq C$, FDC is defined as

$$\rho_{FDC}(m,d) := \frac{Cov(m,d)}{\sigma(m) \cdot \sigma(d)}, \tag{3.5}$$

where *Cov* and σ represent the covariance and standard deviations over all fitness-distance pairs, respectively.

High FDC corresponds to globally funnel-shaped landscapes that are easy to optimize and low FDC corresponds to random or deceptive landscapes. FDC has since been prominently applied to study many problems (see for example, Harrison et al. [79], Hoos and Stützle [84], Nunes et al. [134], Pimenta et al. [142], Pitzer and Affenzeller [143], Tanabe [173], Treimun-Costa et al. [179] or Yuan et al. [196]).

3.3 Analysis of Parameter Interactions

To address our second research question, regarding the importance of the interactions between the parameters of typical algorithm configuration landscapes, we introduce two new methods. The first assesses the significance of local estimates

for partial derivatives, and the second attempts to naïvely configure the parameters of the algorithms independently. We also use functional ANOVA [93].

3.3.1 Locally Significant Partial Derivatives

For each combination of parameters, we calculate finite difference approximations for the partial derivatives of the performance measure with respect to the parameters. We use the grid of parameter configurations to obtain multiple local estimates for each partial derivative. For scenarios with k estimates for the performance measure, we calculate k paired estimates for each local partial derivative. Then, for each one, we check to see if the partial derivative is significantly different from zero. Finally, we count the percentage of local partial derivatives that are different from zero at a 5% significance level.

Counter-intuitively, this method can even be generalized from continuous parameters for use with categorical parameters via a "discrete gradient" approximation. Let $m_{\{i_k\}}(c|_{p=v_l})$ be the estimate for the performance measure of a response slice evaluated at the l^{th} value v_l of parameter p on the k^{th} problem instance i_k. We calculate the l^{th} local partial derivative for the k^{th} problem instance as

$$\frac{\partial m_{\{i_k\}}}{\partial p} = \frac{m_{\{i_k\}}(c|_{p=v_l}) - m_{\{i_k\}}(c|_{p=v_{l-1}})}{v_l - v_{l-1}}. \tag{3.6}$$

However, since the distance $v_l - v_{l-1}$ is a constant for each value of k in $\frac{\partial m_{\{i_k\}}}{\partial p}$, it can be ignored without changing the outcome of the statistical test. Therefore, we can generalize our notion of local parameter significance and apply it to each pair of values for a categorical parameter by simply omitting the distance normalization term.

In some scenarios, only a single point estimate for the performance measurement of each configuration was available; however, confidence intervals for the performance could still be obtained by making additional assumptions. For example, from a single run of a latent structured SVM binary classifier (see Chapter 4.1) we can obtain a point estimate of the error rate obtained with a given configuration. We then assume the test errors are binomially distributed, which allows us to calculate a confidence interval for the estimate of the error rate.

In these cases, we applied a pessimistic version of the test that operates directly on the confidence intervals for two adjacent configurations. In particular, to calculate the upper bound of a derivative for two adjacent parameter values v_l and v_{l-1}, we take the difference between the upper bound of v_l and the lower bound of v_{l-1}, that is,

$$\frac{\partial \overline{m}}{\partial p} = \overline{m}(c|_{p=v_l}) - \underline{m}(c|_{p=v_{l-1}}). \tag{3.7}$$

Similarly, to calculate a lower bound, we use

$$\frac{\partial \underline{m}}{\partial p} = \underline{m}(c|_{p=v_l}) - \overline{m}(c|_{p=v_{l-1}}). \tag{3.8}$$

We then say that the derivative is significantly different from zero, if the confidence interval does not contain zero. For first-order derivatives, this is equivalent to checking if two neighbouring configurations have overlapping confidence intervals for their performance measurements.

3.3.2 Functional Analysis of Variance (fANOVA)

Functional analysis of variance has been prominently applied to study the importance of parameters and their interactions [93]. It decomposes the variance observed in the performance into components associated with each parameter individually and with their interactions. However, instead of using the implementation by Hutter et al. [93], which operates on a landscape approximated using a random forest, we re-implemented the analysis to compute the exact results for a grid of parameter configurations.

Since the complexity of fANOVA grows exponentially with the order of the interaction effects computed, previous work has been restricted to studying the importance of low-order interaction effects (for example 2^{nd}- or 3^{rd}-order effects) [93, 103]. However, our re-implementation came with a surprising bonus; rather than storing a list of tuples of n parameter values and their mapping to performance values, we stored the performance value of each configuration in an n-dimensional `numpy` array, which allowed us to compute the importance of the 11^{th} order parameter interactions for our largest scenario within a few minutes using simple matrix operations (not including the time required to evaluate the grid of parameter val-

ues). While purely due to software engineering, this is a substantial improvement, since calculating only low-order importance scores for landscapes with strong interactions could fail to reveal the important parameters due to marginalization: for example, the function defined by $f(x,y) = |x-y|$ for $x,y \in [-1,1]$ would be attributed with no importance to the first order effects and 100% importance to the interaction effect.

3.3.3 Configuring Parameters Independently

Locally significant partial derivatives and fANOVA both seek to quantify the degree to which the interactions of two or more parameters impact the response in in terms of performance. The first speaks to the percentage of the landscape for which the impact of interactions are statistically significant, and the second speaks to the magnitude of those interactions. However, a much more natural and practical question to ask is: can we configure each of the parameters of our algorithm independently?

In practice, this is quite likely how most algorithm designers choose the default values of each of the parameters of their algorithms, that is, through manual exploration of the response of varying each parameter independently, in sequence. After settling on a good value for a parameter, it would then typically be held fixed for the remainder of the manual configuration process.

We therefore propose to use the grids of the evaluated parameter configurations to emulate a simplistic configuration procedure, for which we can quantify how frequently it can be expected to produce high-quality results. In particular, the procedure assumes that the default configuration (that is, the initial incumbent) is the one in the grid that is estimated to obtain the worst performance. Then, for a random ordering of the parameters, the simplistic configuration procedure looks at the point-estimates for the performance of all of the configurations in the one-dimensional slice centered around the current incumbent and updates the incumbent to be the one with the best performance. This process is carried out for each of the parameters, until each of them has been configured once. At that point, we say that its final incumbent configuration is "tied with optimal" if the lower bound for its performance (where lower is considered better) is at least as good as the upper

bound for the performance of the configuration with the best point-estimate for the performance.

For a given grid of pre-evaluated parameter configurations, this procedure can be run for all possible permutations of the parameters (which determines the sequence in which these are configured), in order to obtain the probability that the simplistic configuration procedure will obtain a final incumbent that is tied with optimal.

As stated previously, this procedure is designed to emulate a common practice for configuring parameters by hand. However, this procedure evaluates the performance of an algorithm on a large set of problem instances, whereas most manual parameter configuration is likely done using only a small number of different problem instances, many of which may not even be representative of the true problem instances on which the algorithm must perform well in practice. As a result, we do not expect the results of this method to be representative of the quality that can be expected with manual configuration. Instead, we use it as a proxy to understand the extent to which a more advanced automated algorithm configuration procedure can assume that parameters are independent before we should expect it to start producing poor-quality results.

3.4 Chapter Summary

In this chapter, we formally defined *parameter response slices*, which relate the performance of an algorithm to one or more parameters when the remaining parameters are held fixed. In addition, we presented two new methods for the analysis of the global shape of a landscape, as well as two new methods for the analysis of the importance of interactions between the parameters of a landscape. In each case, we paid special attention to a critical property of algorithm configuration landscapes: the stochastic nature of measurements of an algorithm's performance. This feature of algorithm configuration landscapes is important, because it makes the direct application of many existing fitness landscape analysis methods inappropriate.

Our first two measures for global landscape shape test the landscapes for statistically significant deviations from two simple properties, uni-modality and convexity. If the test for uni-modality rejects its null hypothesis then we showed how

it can be adapted to find a lower bound on the number of local minima in a landscape. We also reviewed fitness distance correlation (FDC) [97], a related measure of global landscape shape that we apply throughout this thesis.

In some cases, our tests for statistical deviations from uni-modality and convexity may fail to reject their null hypotheses. When this happens, it may not be because the null hypotheses are correct, but instead because of a lack of sensitivity that can arise when the performance measurements are highly stochastic or when relatively little data could be collected. To safeguard against drawing incorrect conclusions from the interpretation of such results, we further proposed a method that validates the sensitivity of our tests using a technique similar to a permutation test. In a somewhat similar vein, we also introduced a heuristic for identifying parameters which yield "interesting" or non-flat responses around the best known configuration in a landscape.

In addition to reviewing the definition of fANOVA [93], an existing technique that can measure the importance of parameter interactions, we proposed two new methods with the same goal. The first method we proposed quantifies the fraction of local estimates for partial derivatives that are significantly different from zero. If this fraction is large for a given partial derivative, then we can conclude that the impact of the interactions between the parameters in the partial derivative are important for a large fraction of the landscape.

The second method we proposed for measuring the importance of parameter interactions is more direct; we run a simplistic configuration procedure to configure each parameter independently in a random sequence. If this procedure can typically find configurations that are statistically tied with optimal, regardless of the sequence in which they are configured, then we conclude that the effects of the interactions do not substantially increase the complexity of the configuration scenario. Note that this can occur even when the interaction effects are large and statistically significant according to the other two methods.

Chapter 4

Analysis I: AutoML Loss Landscapes

As stated in the introduction, the recent rise in popularity of automated machine learning (AutoML) has lead to a plethora of proposed techniques for the configuration of a machine learning method's hyper-parameters (for example, see Bergstra and Bengio [21], Bergstra et al. [22], Feurer et al. [62], Kandasamy et al. [98, 99], Li et al. [113], Olson and Moore [137], Snoek et al. [164], Springenberg et al. [166] or Falkner et al. [59]). However, despite this, very little is known about the structure of the AutoML loss landscapes that are searched by these techniques [142]. This makes analyzing the structure present in AutoML loss landscapes an important next step in the maturity of the field. Without a better understanding of this structure, it can be challenging for AutoML researchers to understand the relative performance of different approaches, or to judge whether or not a potential new method is likely to improve over the current state of the art. In particular, if these landscapes are found to be simpler than expected, then it may be possible to develop faster configuration procedures by exploiting this structure. To address this gap, in this chapter[1] we applied the methods for landscape analysis described in Chapter 3 to landscapes from a variety of machine learning scenarios.

[1] This Chapter is based on the experiments and results of joint work with Holger Hoos that has been accepted for publication, pending minor revisions [151].

Table 4.1: The machine learning scenarios for which AutoML loss landscape analysis was performed.

Model	Dataset	# HP		# Instances			# loss samples	
		Num	Cat	Train	Val	Test	Val	Test
FCNet	SL	6	3	32K	11K	11K	4	4
	PS	6	3	27K	9K	9K	4	4
	NP	6	3	7K	2K	2K	4	4
	PT	6	3	4K	2K	2K	4	4
XGBoost	CT	11	–	47K	5K	–	5	–
LogReg	MNIST	4	–	471K	52K	10K	1	1
LSSVM	UP	3	–	20K	–	20K	–	1
OLDA	Wiki	3	–	200K	25K	25K	–	1

4.1 Experimental Setup

Ideally, each model should be trained and evaluated on separate, randomly sampled training and validation sets (for example, using k-fold cross validation), as this allows for confidence intervals that capture three sources of variance: 1) independent training runs, and the instances included in the particular 2) training and 3) validation datasets. However, we found several existing scenarios with grids of pre-evaluated hyper-parameter configurations that provided sufficient data to calculate confidence intervals that captured subsets of these sources of variance. These eight machine learning scenarios are summarized in Table 4.1.

Klein and Hutter [103] pre-evaluated a grid of 62 208 joint hyper-parameter and neural architecture configurations for a feed-forward neural network (FCNet)[2] applied to four different UCI datasets [52]: slice localization (SL) [73], protein structure (PS) [153], naval propulsion (NP) [44] and Parkinsons telemonitoring (PT) [180]. They performed 4 training runs for each configuration using different random seeds, yielding a total of 4 validation and 4 test loss scores (for the final, 100-epoch training budget that we analyze here). To obtain the best-possible estimate for generalization error, we took the mean of all 8 loss samples and calculated 95% Student-t-based confidence intervals by assuming that all 8 loss samples were independently and identically distributed. When looking at the validation and test scores separately, 11.92% of the samples between all of the scenarios have normality rejected at a 5% significance level using a Shapiro-Wilk test. However, by

[2] Available at https://github.com/automl/nas_benchmarks.

naïvely combining the validation and test scores in this way, these samples are not in fact independently and identically distributed, and hence 32.89% of the sets of samples have normality rejected. This indicates that these confidence intervals are likely slightly over-confident, and thus some of the statistical tests that we perform may incorrectly reject their null hypothesis more frequently than expected for the given significance level.

Our second scenario comes from ACLib [94], where Xgboost [38] is applied to the covertype dataset [52]. We applied fANOVA [93] to some preliminary data and found that `eta` was by far the most important hyper-parameter. This hyper-parameter, `eta`, is sometimes also referred to as the shrinkage factor or learning rate, because it controls the weight assigned to each subsequently-fit tree in the ensemble, and hence the ability of the model to quickly over-fit to the training data. Given that it was the most important hyper-parameter, we evaluated a uniform grid of hyper-parameter values with 7 values for `eta` and 3 values for each of the remaining 10 hyper-parameters (see Appendix A.2.1 for the table of values).

ACLib provides 10 splits for performing 10-fold cross validation. To keep within our computational budget, we trained and evaluated the model on the first 5 splits. ACLib also specifies a 1 000 second running time cut-off for training. In the scripts provided by ACLib, this time includes reading the data from disk and splitting it. However, since this required about 40 seconds per run, and we performed just over 2 million runs of Xgboost, we reduced this cutoff to 960 seconds and used our own scripts that pre-loaded the instances and held them in memory. Nevertheless, collecting all of the data took 14.7 CPU years. We calculated 95% confidence intervals for the generalization loss of Xgboost using Student-t-based confidence intervals (normality was rejected for only 5.37% of the samples at a 5% significance level).

Our remaining three scenarios came from pre-computed grids that are available in HPOLib [55]. The first scenario [164], logistic regression on the MNIST digits dataset [109], contained one validation and one test loss, so we again calculated Student-t-based confidence intervals[3]. The hyper-parameter configuration grids for the latent structured SVM [127] applied to a DNA motif-finding dataset (UniProbe

[3]With only two samples per configuration, we were unable to test for deviations from normality.

or UP) and the online LDA model [83] applied to Wikipedia articles were evaluated by Snoek et al. [164]. However, each only had a single estimate of generalization loss available. Since the SVM scenario was binary classification, we assumed that the errors were binomially distributed[4] and calculated 95% confidence intervals using the Wald method. We were unable to make a similar assumption for the perplexity loss from online LDA. We therefore assumed all confidence intervals could be expressed as a percentage of the loss and performed a binary search to find the smallest value for the size of the interval for which the tests failed to reject their null hypotheses. If the confidence intervals are small, we can still conclude that the landscape is close to uni-modal or convex.

These grids of hyper-parameter values contained values that were chosen with domain knowledge that reflects the expected changes in the response to the hyper-parameters. For example, online LDA's hyper-parameters `tau` (a learning parameter that down weights initial iterations) and `s` (the size of the mini-batches) were both sampled such that each parameter value was four times larger than the previous one, whereas its `kappa` hyper-parameter (the learning rate's exponential decay rate) values were sampled with uniform spacing. Thus, to reflect this prior knowledge when calculating FDC, we defined the distance between any two adjacent numerical parameter values as 1, a constant value.

4.2 One-Dimensional Hyper-Parameter Response Slices

The results from our tests for uni-modality and convexity, as well as the median FDC for the 45 one-dimensional, numerical hyper-parameter response slices are summarized in Table 4.2. Confidence intervals of size $\pm$ 0.14% were large enough for all of the hyper-parameter response slices for the hyper-parameters of online LDA to appear both uni-modal and convex; since these are very small, we count them as being both uni-modal and convex, and we show them as such in the table.

Nearly all of the hyper-parameter response slices appear uni-modal (44 out of

[4]Technically, the errors are distributed according to the convolution of two binomial distributions – one for each class; however, without the error rates for each class it is impossible to obtain more precise confidence intervals. Using this method yields conservatively large confidence intervals [57], thus our test may have failed to detect slight deviations from uni-modality; however, our permutation tests still indicated that the results should be considered sensitive (see Tables 4.3 and 4.4), hence it is unlikely that we missed observing any large deviations from uni-modality.

Table 4.2: The percentage of the one-dimensional hyper-parameter response slices for which uni-modality (Uni-M) and convexity (Cvx) could not be rejected, and the median fitness distance correlation (FDC). All of the hyper-parameter response slices are centered around the global minima of the landscapes. Note that this table assumes the online LDA scenario has intervals of size 0.14%.

Type	# Slices	Uni-M	Cvx	FDC
All	45	97.8%	82.2%	0.93
Interesting	44	97.7%	81.8%	0.93

45) and most of them appear convex (37 out of 45). Furthermore, all but one of the 45 hyper-parameter response slices are considered to be interesting, according to our heuristic criterion, indicating that most AutoML hyper-parameters can be configured to improve the performance of their corresponding machine learning methods.

Xgboost's `subsample` hyper-parameter was the only one for which the response slice was determined to be neither uni-modal nor convex. Since this hyper-parameter was found to be relatively unimportant, according to fANOVA (its first order effect accounts for 2.5% of the variance in the loss), we originally only evaluated three different values for it. The middle hyper-parameter value had a loss substantially larger than both of the others, as well as a much wider confidence interval.

Curious, we investigated by increasing the number of hyper-parameter values from 3 to 21 for this individual hyper-parameter response slice (and we re-evaluated the original three values as well). We show the resulting hyper-parameter response slice in the top left pane of Figure 4.1. To our surprise, the large barrier in the response completely disappeared and our tests now failed to reject both uni-modality and convexity.

Looking more closely at the original data, we see that 4 out of the 5 evaluations on different cross-validation folds exceeded the 960 second running time cutoff; therefore, their error rates were recorded as 1. For the expanded response slice, the mean running time for `subsample` $= 0.50$ was 863 seconds, with a standard deviation of 42. Therefore, we surmise that background processes or other envi-

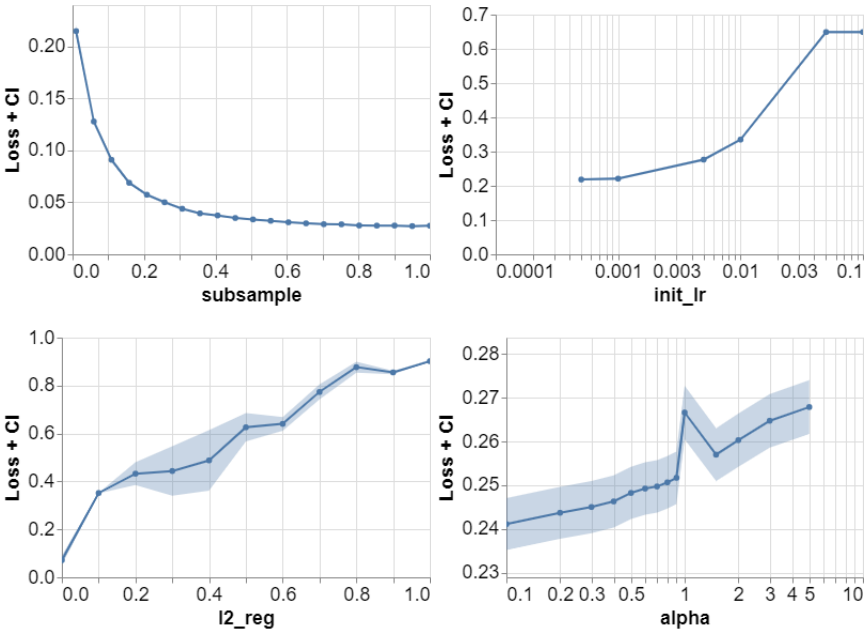

Figure 4.1: Four examples of one-dimensional hyper-parameter response slices. From top to bottom and left to right: Xgboost's `subsample` hyper-parameter, FCNet's `init_lr` hyper-parameter on the protein structure dataset, Logistic Regression's `l2_reg` hyper-parameter, and LSSVM's `alpha` hyper-parameter.

ronment noise likely caused several of the original runs to be censored. Given that this is, effectively, a spurious result, it would be reasonable to replace the original loss values with the new ones for the remaining analysis. However, in practice, many AutoML configurators are likely to encounter similar challenges from time to time. Therefore, we leave the data as is and continue to analyze the landscape with a spike in it.

Of the remaining seven hyper-parameters for which convexity was rejected, three were FCNet's `init_lr` hyper-parameter for three of the four datasets. For all four datasets, small values of the hyper-parameter provided the best loss, with a sometimes-abrupt transition to a sub-optimal, approximately plateau-shaped region. The response on the protein structure dataset (see the top right pane of Figure 4.1) is representative of the other responses; however, the response on the slice

localization dataset has confidence intervals that are too large to reject convexity.

The `epsilon` hyper-parameter for latent-structured SVMs also yielded a somewhat similar response slice; however, the difference between the good and bad values for the hyper-parameter were substantially smaller, there were only 4 values evaluated for the hyper-parameter, and the size of the confidence intervals were nearly as large as the transition, thus it is unclear whether or not this was merely due to random fluctuations in the performance measurements.

At the bottom left of Figure 4.1, we show the response slice for logistic regression's `l2_reg` hyper-parameter, for which convexity was also rejected. The mean loss increases nearly monotonically with the value of the hyper-parameter from 0.07 to 0.90; however, the response is more rugged than for most of the other hyper-parameters and appears to perhaps even be concave in shape. The only other hyper-parameter that shows some ruggedness in its one-dimensional response is latent-structured SVM's `alpha` hyper-parameter, which is shown in the bottom right pane of Figure 4.1. In this case, the response is very smooth apart from a single hyper-parameter value that yields an abnormally large loss. Visually, this hyper-parameter response slice at first appears bi-modal; however, careful inspection reveals that the confidence intervals are just large enough to admit the possibility of a uni-modal response. Nevertheless, in light of the spurious bi-modal response that we observed for Xgboost's `subsample` hyper-parameter, it is possible that the behaviour present in this response may have arisen in a similar fashion. However, without access to more detailed information or the original model and dataset, we were unable to verify this hypothesis.

Other than the seven exceptions discussed so far (and one more discussed in Chapter 4.3 and shown in the top left pane of Figure 4.2), the remaining 37 hyper-parameter response slices are surprisingly smooth, with many of them being qualitatively similar in smoothness to that seen at the top left of Figure 4.1. The final non-convex hyper-parameter response, for Xgboost's `min_child_weight`, also appears smooth, but has a slope that is slightly negatively correlated with the loss.

Table 4.3: Summary of the test results for uni-modality and convexity, and the median FDC applied to the two-dimensional hyper-parameter response slices centred around the global optima of the AutoML loss landscapes. Includes the mean percentage of the landscapes that is unreachable (UnR) from the global optima or interior (Int) to the convex hull of the upper bounds for those slices for which uni-modality or convexity was rejected, respectively; the mean percentage of the lower-bounds that are co-planar (Co-P) to the convex hull of the upper bounds; and the mean percentage of the landscapes for which the tests are sensitive (Sen) according to our permutation-based analysis.

Type	# Slices	Uni-M	UnR	Sen	Cvx	Int	Co-P	Sen	FDC
N×N	127	92.1%	22.3%	77.2%	58.3%	24.8%	22.2%	90.6%	0.71
N×C	72	100.0%	–	62.5%	–	–	–	–	0.77
C×C	12	100.0%	–	0.0%	–	–	–	–	0.72

4.3 Two-Dimensional Hyper-Parameter Response Slices

In Table 4.3 we show a summary of the results from our tests for uni-modality and convexity, as well as the median FDC values for the two-dimensional hyper-parameter response slices that are centered around the global optima of each landscape. The results are quite similar to those from the one-dimensional analysis (see Chapter 4.2); however, all of the numbers are slightly lower – especially those from the tests for convexity. In total, of the 208 hyper-parameter response slices we tested (this excludes those for online LDA, for which true confidence intervals could not be obtained), 95.19% appeared to be uni-modal (all but 10), and the median FDC is 0.73. However, the test for convexity rejected its null hypothesis more frequently, and only 58.87% of the response slices with two numeric hyper-parameters appear to be convex. In Table 4.3, we include the three response slices for online LDA as if it had confidence intervals of size $\pm$ 0.16%, which were small enough to fail to reject uni-modality, but not large enough to fail to reject convexity. Indeed, it appears that the landscape is most likely not convex, as it requires the intervals to be at least $\pm$ 8.70% before it fails to reject convexity.

The latent-structured SVM's `c` and `alpha` hyper-parameters yielded the most unusual two-dimensional response slice (see the top left pane of Figure 4.2). For many of the values of `alpha`, `c`'s response smoothly decreases monotonically

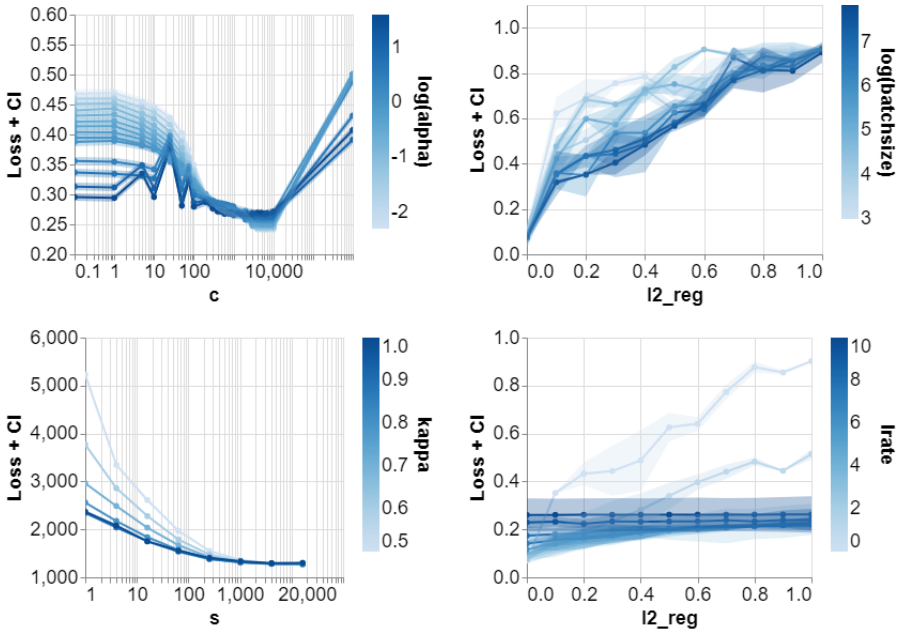

Figure 4.2: Four examples of two-dimensional hyper-parameter response slices. From top to bottom and left to right: latent-structured SVM's `c` and `alpha` hyper-parameters, logistic regression's `l2_reg` and `batchsize`, online LDA's `kappa` and `s`, and logistic regression's `l2_reg` and `lrate`.

prior to the optimal value near to $\texttt{c} = 10\,000$; however, there is a small "bump" around $\texttt{c} = 1\,000$ that causes the response to almost form a sub-optimal plateau. Most interestingly, for the four largest values of `alpha`, the smaller values of `c` yield a saw-tooth-style response instead of the smooth curves. Furthermore, each of the "teeth" in this part of the landscape align closely, which suggests that there exists a complex interaction between the two hyper-parameters. Despite this, only 2.5% of this response slice had locally significant partial derivatives, since with the exception of $\texttt{alpha} = 1$ and $\texttt{alpha} = 1.5$, all of the other neighbouring values of `alpha` yielded qualitatively similar hyper-parameter response slices in `c`. Note that this is the same `alpha` hyper-parameter that is pictured in the bottom right of Figure 4.1, hence the near-bi-modality in that one-dimensional response slice may in fact be due to a complex dependence of the loss upon the value of `alpha` rather

than a spurious result.

We observed another unusual, non-uni-modal hyper-parameter response slice for logistic regression's `l2_reg` and `batchsize` hyper-parameters (see the top right pane of Figure 4.2). Similar to the previously discussed case, we see more evidence that `l2_reg` has a highly rugged response; however, unlike the rugged portion of the response between the latent-structured SVM's `c` and `alpha` hyper-parameters, the jagged peaks of `l2_reg`'s response slices for varying values of `batchsize` do not align. Instead, the shape of the landscape is suggestive of a highly rugged and jagged mountainscape that slowly levels off towards the top. However, recall that for this scenario only one validation and one test loss value were available. Since these losses are likely correlated, it is possible that the landscape only appears to have numerous local minima due to our assumption that the two loss values were *i.i.d.* when we calculated the confidence intervals. If so, then the landscape may actually be uni-modal, albeit with much larger variance than most other cases.

Nevertheless, many hyper-parameter configurators never bother quantifying (accurately or at all) this source of variance in AutoML loss landscapes. Instead, they only perform a single run of the training algorithm on a single training and validation split (or they may take the mean over a few cross-validation folds). As a result, a robust hyper-parameter configurator must still be prepared to occasionally deal with some local ruggedness that can cause small, sub-optimal local minima to speckle the landscape.

FCNet's `dropout_1` and `dropout_2` hyper-parameters on the protein structure dataset yields a third non-uni-modal response slice. In this case, there is only a single point in the landscape that is unreachable from the global optima; however, if the confidence level of the test is just slightly increased from 95% to 96.52%, our test fails to reject its null hypothesis. It is also possible that this apparent sub-optimal local minima could be due to the discretization of the landscape, since its 3 by 3 grid may be too coarse to allow for diagonally-oriented basins to appear uni-modal. However, without access to the original model and dataset, we are unable to verify whether or not this is the case.

The remaining seven hyper-parameter response slices for which uni-modality was rejected all contain Xgboost's `subsample` hyper-parameter. In each case,

we continue to see that one or more of the configurations with `subsample` $= 0.5$ caused the algorithm to exceed the running time cutoff, thus once again introducing spike-like barriers into the landscape. We speculate that re-running each of these instances with larger running time cutoffs (or perhaps even with the same running time cutoff, but on a machine with a smaller background load) would yield uni-modal responses. However, given that hyper-parameter configurators would not have access to this kind of information while in use, we continue to analyze the landscape with these spikes present.

In the bottom right pane of Figure 4.2, we show the hyper-parameter response slice for online LDA's `kappa` and `s` hyper-parameters. Given the apparent smoothness in the response, it should come as no surprise that an interval size as small as $\pm$ 0.16% yields uni-modal responses for each of the three, two-dimensional hyper-parameter response slices. Out of the four response slices shown in Figure 4.2, both the smoothness and simplicity of this response is the most representative of all of the other hyper-parameter response slices (not shown).

The hyper-parameter response in the bottom right pane of Figure 4.2 also exhibits a very interesting property: if the hyper-parameters were to be configured individually a single time in sequence or independently in parallel, one would still find a configuration very near to optimal. This remains true even though 74.29% of the local second order partial derivatives are significant (when arbitrarily using an interval size of $\pm$ 0.16%). It was this observation that ultimately inspired the methodology described in Chapter 3.3.3, which seeks to answer the question: how often does this hold true in practice for other combinations of hyper-parameters? Clearly, it does not always hold (for example, see the bottom right pane of Figure 4.2); nevertheless, a surprising 80.77% of the hyper-parameter pairs and permutations of hyper-parameter configuration order yielded landscapes so benign that our simplistic configuration process was able to find a configuration tied with optimal. This shows that even though many of the hyper-parameters have statistically significant interactions within somewhat substantial fractions of their landscapes (see Table A.2), most of these interactions are relatively benign from the perspective of a hyper-parameter configurator.

This trend held true surprisingly often even for hyper-parameter response slices involving one or two categorical hyper-parameters. In fact, all of the response slices

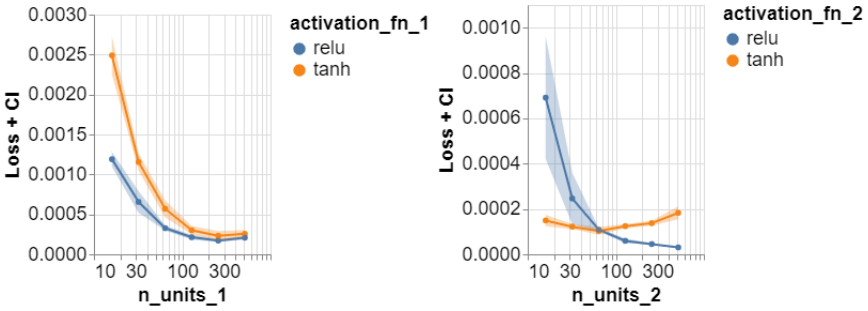

Figure 4.3: Two examples of two-dimensional hyper-parameter response slices that include numerical and categorical hyper-parameters. Left: FCNet's `n_units_1` and `activation_fn_1` on the slice localization dataset; right: FCNet's `n_units_2` and `activation_fn_2` on the naval propulsion dataset.

with at least one categorical hyper-parameter also turned out to be uni-modal. Unsurprisingly, in some cases, changing a categorical hyper-parameter shifts the loss up or down without substantially moving the optimum (for example, see the left pane of Figure 4.3); however, we would also expect to find cases where changing a categorical hyper-parameter would result in a completely different landscape with a new optimum for the other hyper-parameter. In fact, we did observe a small number of examples of the latter case (for example, see the right pane of Figure 4.3); however, in each case, the optimum of the numeric hyper-parameter for a sub-optimal categorical value yielded an equal or greater loss for the configuration with the same numeric value and the optimal categorical value. Therefore, the overall hyper-parameter response slices were still uni-modal. Nevertheless, since our study included a relatively small number of categorical hyper-parameters, we remain skeptical that this behaviour will generalize to all – or perhaps even most – other scenarios. We leave the study of this question as future work.

4.4 Higher-Dimensional AutoML Loss Landscapes

We show the results for uni-modality, convexity and FDC applied to the full AutoML loss landscapes in Table 4.4. Contrary to most of the lower-dimensional hyper-parameter response slices of the landscapes we studied, we see that all of the

complete AutoML loss landscapes have relatively low FDC, which suggests that they should be quite challenging to optimize. However, the test for uni-modality tells a different story: it rejects uni-modality for only two of the seven landscapes we were able to test. For the remaining scenario, online LDA, only 3.12% of the landscape is unreachable – even without using confidence intervals at all. Furthermore, no deep (if small) sub-optimal modes exist in this landscape, since an interval size of only $\pm 1.36\%$ is sufficient to fail to reject uni-modality, which limits the height of a barrier between two modes to being no greater than 2.76% of the loss of the best configuration corresponding to either mode (the derivation for this is in Appendix A.2.2).

Both scenarios for which uni-modality is rejected are for FCNet. In one case only three, and in the other only 41, out of 62 208 configurations are unreachable. Given that these numbers are so small, could this merely be due to random chance? To answer this question, we analyzed the distances between the unreachable configurations. If they were unreachable due to random chance, we might expect them to be spread out uniformly at random. For the protein structure instance set, the mean pairwise distance between unreachable configurations is 4.73. For comparison, we drew 1 000 sets of 41 random configurations from the landscape. The smallest mean pairwise distance out of the 1 000 samples was 9.61 and the mean was 10.36, hence the 41 unreachable configurations are clustered together.

In fact, many of them are adjacent, thus forming 18 very small, sub-optimal modes. We calculated the depth of these modes as the amount the smallest upper bound of each would need to increase to become reachable from the global optimum. Since there are a small number of configurations with extremely bad loss, we measure the size of this increase as a percentage of the range of losses spanned by the optimal configuration and the 95^{th} percentile of the losses. The median increase in loss required is 1.92% and the largest is only 6.30%; hence, these modes are rather shallow.

On the other hand, convexity is rejected for all of the complete AutoML loss landscapes that we studied. In particular, the FCNet landscapes appear to be quite far from convex, with 38.76% to 50.05% of the lower bounds being interior to the convex hull of the upper bounds.

One of the two scenarios that is closest to being convex is Xgboost on the

Table 4.4: The Uni-modality and convexity test results, and fitness-distance correlation for the full AutoML loss landscapes.

Model	Dataset	Interval	UnR	Sen	Int	Co-P	Sen	FDC
FCNet	SL	95%	0.00%	Yes	50.05%	0.00%	Yes	0.00
	PS	95%	0.03%	Yes	49.67%	0.00%	Yes	0.03
	NP	95%	0.00%	Yes	43.63%	0.03%	Yes	0.01
	PT	95%	$< 0.01\%$	Yes	38.76%	0.00%	Yes	0.01
LSSVM	UniProbe	95%	0.00%	Yes	18.86%	0.36%	Yes	0.43
LogReg	MNIST	95%	0.00%	Yes	28.25%	0.10%	Yes	-0.03
XGB	CT	95%	0.00%	Yes	9.65%	0.21%	Yes	0.33
OLDA	Wiki	$\pm$ 0.00%	3.12%	Yes	21.88%	61.46%	Yes	0.36
		$\pm$ 1.36%	0.00%	Yes	6.25%	0.00%	Yes	0.36
		$\pm$ 8.70%	0.00%	Yes	0.00%	0.00%	Yes	0.36

covertype dataset. For this scenario, it would have been computationally expensive (requiring approximately 1.5 months on our machines) to exactly calculate the percentage of the lower bounds that were interior; however, given that we only needed to find a single such configuration to reject convexity, it sufficed to randomly sample and evaluate 5% of the lower bounds to estimate the percentage that are interior. Using this method we calculated a 95% confidence interval for the percentage of points that are interior as [9.56%, 9.74%] – that is, just below 10% of the landscape would need to be altered for the entire landscape to appear convex.

The other landscape that is somewhat close to being convex is online LDA, for which the loss intervals need to be $\pm$ 8.70% in size to fail to reject convexity. However, even the much smaller interval size of $\pm$ 1.36% leaves only 6.25% of the lower bounds interior to the convex hull.

While these results from the test for convexity on the full landscapes are quite different than those on the lower-dimensional hyper-parameter response slices, they should come as no surprise. In fact, since every one of the scenarios had at least one two-dimensional hyper-parameter response slice that was non-convex, it follows that the full landscapes must also all be non-convex.

In Table 4.5, we show the mean percentage of each landscape that is dependent on the 1^{st}- through 6^{th}-order partial derivatives, that is, for each n^{th}-order interaction we calculated the percentage of significant partial derivatives and then report the means for each order n. For landscapes that had them, the higher order interaction results remained relatively similar to the 6^{th} order interaction results (see

Appendix A.2.3). Most of these scenarios behave roughly as we would expect: the first-order derivatives for most hyper-parameters affect relatively large portions of the landscapes. However, there are a few exceptions: for example, logistic regression's `n_epochs` hyper-parameter only affects 10.56% of the landscape. As the order of interactions increases, fewer tuples of hyper-parameters interact, and those that do affect smaller portions of the landscape. For example, 50% of the 2^{nd}-order interactions for logistic regression affect between $20.90\% - 26.61\%$ of the landscape, and the rest affect less than 7.07%; one of the 3^{rd}-order interactions affects 18.10% of the landscape, and the remaining four affect less than 5.21%.

However, the results are quite different for the FCNet landscapes. The higher-order parameter interactions are substantially more influential. For the four scenarios between 19.90% – 24.27% of the landscapes are dependent upon interactions between all 9 hyper-parameters. The results from fANOVA (see Table A.4) show additional support for the fact that the FCNet landscapes depend to a much larger degree on higher-order interactions than the other landscapes. In particular, fANOVA attributes between 17.01%–28.58% of the total variance in the FCNet landscapes to the 6^{th}-order interactions, compared to only 0.15%–4.50% for the 1^{st}-order effects. In comparison, for XGBoost, fANOVA attributes only 0.24% to the 6^{th}-order interactions and 58.94% to 1^{st}-order effects.

One possible explanation for this is that the FCNet scenarios are the only ones we study that fall into the over-parameterized regime [19], which may exhibit qualitatively different AutoML loss landscapes. In particular, because there may be a large number of models which all obtain (nearly) perfect training scores, the hyper-parameters may interact in complex ways to produce models that interpolate differently between the training instances.

The high degree of significant interactions between the FCNet hyper-parameters also provides a possible explanation for the sub-optimal modes that we observed in two of these landscapes. In particular, if the resolution of the grid of hyper-parameter values is too coarse, then a landscape that contains a diagonally-oriented basin with steep walls may appear to have multiple sub-optimal local minima because our neighbourhood relation graph does not allow for configurations to be connected even if they are diagonally adjacent. However, without the ability to collect additional performance measurements it is impossible to test whether or

not increasing the resolution of the grid of hyper-parameter values would alter the outcome of the test.

There is a qualitative difference between the fANOVA results and those regarding partial derivatives. For most scenarios, fANOVA reports that a small number of hyper-parameters are responsible for a large percentage of the variance in the objective function. The XGBoost landscape is a representative example: fANOVA reports that the 1^{st}-order effect of the hyper-parameter `eta` is responsible for 52.6% of the variance, and the rest are individually responsible for less than 2.54%. In comparison, `eta` has the third largest percentage of non-zero partial derivatives (67.65%) and all but one of the rest have between 17.06% – 75.16% (the one remaining has 3.01%). However, these results are not contradictory, since the statistical test can detect small but significant partial derivatives. Indeed, all that can be interpreted from fANOVA's result for `eta` is that at least one value of `eta` yields very different performance from the rest.

To determine where hyper-parameters are most important, we excluded from the analysis any partial derivative that contained a configuration in the worst $X\%$ of the landscape (see Tables A.5–A.8 for $X = 50\%$ and $X = 75\%$). The outcome of this analysis was mixed. For latent-structured SVMs, logistic regression and online LDA, the partial derivatives for most hyper-parameters were significant less often; for example, first-order partial derivatives of the hyper-parameters of latent-structured SVMs were found to be significant for 19.10% of the entire landscape on average, but only for 6.42% of the landscape when excluding the worst 50% of the configurations. However, we found the opposite to be true for FCnet and Xgboost; for example, the first-order partial derivatives for the FCNet landscapes were significant 9.75% more often on average when the worst 50% of the configurations were excluded from the analysis.

This result for FCNet is consistent with the sub-optimal, plateau-shaped regions that we observed in FCNet's `init_lr` hyper-parameter response, suggesting that this behaviour likely also occurs for more of the slices of AutoML loss landscapes that were excluded from our one-dimensional analysis. For Xgboost, it is possible that if there are regions of the configuration space for which the running time of the algorithm tends to be high, then entire portions of the landscape may have been artificially censored due to a running time cutoff. (We observed

Table 4.5: Hyper-parameter partial derivative significance result summary – part 1. Each column represents the mean percentage of the landscape with statistically significant partial derivatives for each partial derivative order.

Model	Dataset	Interval	1^{st}	2^{nd}	3^{rd}	4^{th}	5^{th}	6^{th}
FCNet	SL	95%	66.45%	45.25%	34.06%	28.66%	25.62%	23.59%
	PS	95%	68.16%	44.02%	31.68%	26.74%	24.58%	23.60%
	NP	95%	53.66%	37.29%	29.93%	26.68%	25.15%	24.33%
	PT	95%	70.96%	45.82%	31.62%	26.25%	24.08%	22.98%
LSSVM	UniProbe	95%	19.10%	4.43%	0.96%	–	–	–
LogReg	MNIST	95%	36.21%	15.21%	7.77%	4.24%	–	–
XGB	CT	95%	43.57%	13.35%	4.88%	3.30%	2.98%	2.91%
OLDA	Wiki	$\pm$ 1.36%	67.53%	17.07%	0.00%	–	–	–
		$\pm$ 5.10%	29.77%	0.95%	0.00%	–	–	–

one example of this bevahiour for `subample` $= 0.5$, which tended to have many censored runs reported as a loss of 1.) If this were to happen sufficiently often in clustered areas, then it could be the case that the worst configurations form regions of sub-optimal plateaus where the loss is 1.

We also observed one hyper-parameter with a particularly notable increase in the significance of its partial derivatives: logistic regression's `l2_reg` hyper-parameter. For example, when excluding the worst 75% of configurations, its 1^{st}-order partial derivative became significant 25.37% more often. This indicates that `l2_reg`'s response must be steepest nearest to its global optimum, possibly with a sub-optimal plateau embedded in its response – indeed, this is precisely what we observed in the one- and two-dimensional hyper-parameter response slices for it (see Figures 4.1 and 4.2).

Finally, we applied our simple test, wherein we configured each hyper-parameter independently a single time in a random sequence. Because Xgboost has nearly 40 million possible permutations for the order in which its hyper-parameters can be configured, we restricted ourselves to a small, 30 minute computation budget, which allowed for a random 537 100 permutations. For all of the other scenarios, we performed the analysis for all possible hyper-parameter permutations. The only scenario for which we found any permutations of the hyper-parameter configuration order that did not yield a result tied with optimal was online LDA, with an interval size of $\pm 0.00\%$. However, even in this case, the configuration procedure

found the optimal configuration in all but one of the six permutations.

This result is rather surprising, especially given that we have observed non-trivial percentages of the landscapes that are dependent upon hyper-parameter interactions, as it suggests that AutoML loss landscapes may be much simpler to optimize than previously assumed. One possible explanation for this result is our previous observation that in many (albeit not all) cases, the hyper-parameters tend to interact most strongly in regions of the configuration space that are far from optimal. An example can be seen in the bottom left pane of Figure 4.2, where the effect of `kappa` has no significant impact on the response in the three best values of `s`. Alternatively, as motivated in Chapter 1, the compensatory nature of some hyper-parameters may yield responses for which fixing a single value of a hyper-parameter does not harm the performance of the algorithm, since a different hyper-parameter can be modified to compensate for this, thereby still yielding an approximately-optimal configuration.

4.5 Chapter Summary

In this chapter, we applied the landscape analysis methods from Chapter 3 to empirical data from a diverse set of state-of-the-art machine learning models and algorithms. All but two of the eight AutoML loss landscapes we studied appear to be uni-modal at a 95% significance level, and those that significantly deviate from uni-modality do so only slightly. At first glance, this result may appear contradictory with the 2.2% of the one-dimensional and 4.8% of the two-dimensional hyper-parameter response slices for which uni-modality was rejected. However, the reason for this must lie in the hyper-parameter interactions, which allow a path from the sub-optimal modes in the lower-dimensional slices to circumnavigate their barriers. We were able to reject convexity for all of the landscapes we studied at a 95% significance level; however, we nevertheless observed that 82.2% of the hyper-parameters yield convex responses when considered independently.

Furthermore, we showed that an intuitively related metric, fitness distance correlation (FDC) [97], fails to accurately characterize the simplicity of the structure present in most of the full AutoML loss landscapes, which is consistent with previous results [142]. However, despite the clear evidence that most hyper-parameters

induce highly-structured and exploitable responses in the corresponding loss landscapes, we also observed a small number of exceptions to this rule. For example, small values of latent-structured SVM's `alpha` hyper-parameter yielded a very smooth, wavy response in `c`, whereas large values of `alpha` yielded a saw-tooth response (as seen in the top left pane of Figure 4.2). Somewhat similarly, logistic regression's `l2_reg` hyper-parameter yielded a relatively noisy and rugged response, with some sub-optimal plateaus (see the top right and bottom right panes of Figure 4.2). Finally, we also observed that certain values of hyper-parameters (for example, `subsample` in xgboost) are correlated with longer running times, which can cause the training algorithm to be censored, thus yielding spurious clusters of spikes and ridges, due to censored loss values.

We found that most landscapes have only a small number of hyper-parameters that interact strongly. However, the FCNet landscapes we studied were qualitatively very different and exhibited surprisingly large percentages of statistically significant high-order partial derivatives. We leave as future work the investigation of whether or not this behaviour could be attributed to the fact that the FCNet landscapes we studied may have been in the so-called "over-parameterized regime" [19].

We observed that many hyper-parameters tend to have flatter responses near high-quality solutions; however, we did observe some exceptions – the most notable of which was logistic regression (particularly its `l2_reg` hyper-parameter), for which the response contains some sub-optimal plateaus and a steep drop near the optimum (for example, see the top right and bottom right panes of Figure 4.2). Nevertheless, from this general trend we can surmise why random search can often find high-quality configurations, even though there is typically only a single region in the configuration space that contains high-quality solutions – that is, the region of high-quality solutions must usually not be too small.

Finally, we used our simplistic configuration procedure that naïvely configures each hyper-parameter independently, a single time and in a random order, to determine the extent to which interactions between hyper-parameters increase the complexity of the configuration problem. Surprisingly, we found that in all cases this procedure was able to find final configurations that were statistically tied with the optimal configuration, according to the respective 95% confidence

intervals. While this configuration procedure lacks several essential ingredients of state-of-the-art hyper-parameter configurators (for example, the ability to make use of multi-fidelity estimates for the loss of a hyper-parameter configuration), it nevertheless demonstrates that the effects of the hyper-parameter interactions are relatively benign, despite their statistically significant presence.

Also somewhat unexpectedly, we found that the landscapes induced by the categorical hyper-parameters for the FCNet benchmarks are either completely or very close to uni-modal. Indeed, we speculate that this behaviour will not generalize to the combinatorial landscapes induced by model and pre-processor selection in machine learning pipelines. We leave the study of this question as future work.

Chapter 5

Exploitation I: AutoML Loss Landscapes

In Chapter 4, we learned that most AutoML loss landscapes are more benign than may initially be expected. In particular, this is because they appear to often be unimodal (or very close to it) and their hyper-parameters tend not to interact in ways that cause challenges for a simplistic configuration procedure that configures each hyper-parameter independently, a single time and in a random order. Of course, the most interesting question raised by this analysis is how to best exploit these insights for the development of faster, more efficient and more effective AutoML methods.

In this chapter,[1] we propose and evaluate two prototype variations of a recent, state-of-the-art hyper-parameter configurator, BOHB [59]. BOHB combines Bayesian optimization with Hyperband [113], which exploits low-fidelity approximations of configurations' performances to quickly eliminate them. To perform Bayesian optimization, BOHB uses a tree-structured Parzen estimator (TPE) to model the high- and poor-quality regions of the AutoML loss landscape. While this has been shown to perform well in practice [22], it may be unnecessarily inefficient for two possible reasons.

- First, all Bayesian optimization methods are designed to trade off between

[1] This chapter is based on joint work with Holger Hoos that is currently under review [147].

both *exploitation* and *exploration*. However, if the landscapes of AutoML scenarios are typically very simple, there may be little need to spend time evaluating unexplored regions of the configuration space if they can be reasonably predicted to have poor performance.

- Second, the TPE model used by BOHB is a very flexible, non-parametric model. In many machine learning applications, this can be advantageous; however, when very little training data is available early in the configuration process, this may cause the procedure to produce models that have overfit and are not representative of the actual loss landscape. Using simpler models with stronger priors regarding the structure in the landscape may help overcome this, thereby expediting the configuration process as a whole.

For these reasons, using relatively simple surrogate models that incorporate our prior expectation regarding the structure of the landscapes (either that they are unimodal or that the hyper-parameter interactions are relatively benign) may help to improve the efficiency of the configuration procedure by cutting down on unnecessary configuration evaluations. However, as we will show in Chapter 5.3, both of our alternative surrogate models (a convex quadratic and a B-spline model) proved to be too simple, resulting in similar or worse performance compared to BOHB for most of the AutoML scenarios we studied. Nevertheless, our experiments provide additional insights into a) the structure of AutoML problem scenarios, b) the reasons why these particular methods did not perform as well as expected, and c) the conditions under which they can be expected to perform well.

5.1 Experimental AutoML Procedures

Both of our modifications to BOHB involved replacing the TPE model with simpler, more constrained surrogate models, which were chosen to reflect one of the structural properties we observed in AutoML loss landscapes (see Chapter 4). Similar to BOHB's TPE model, each of our surrogate models attempt to approximate the full AutoML loss landscapes. In the case of the convex quadratic surrogate model, this resulted in substantially increased overhead for fitting each of the models. Therefore, in order to avoid slowing down the configuration procedure unnecessarily, we fit the models asynchronously. However, this means that we are

required to suggest multiple configurations using the same surrogate model. As a result, we employed a similar, but adapted version of BOHB's method for suggesting configurations, which does not completely optimize the surrogate model.

5.1.1 Convex Quadratic Surrogate Models

Even though the results of our landscape analysis (See Chapter 4.4) revealed that AutoML loss landscapes are not convex, it may still be helpful to use a convex quadratic function as a surrogate model for the landscape, as it is perhaps the simplest n-dimensional model that is constrained to be uni-modal.

To fit convex quadratic models, we use the methods proposed by Rosen and Marcia [158]. One advantage of their methods compared to others is that the models are fitted using the L_1 norm, which is more robust than the L_2 norm in the presence of noise and outliers. They further impose the constraint that the fitted models must under-estimate all of the available training data. While it is unclear whether or not this will be beneficial for our application, we do not expect it to harm the performance of the method. Adding such a constraint will increase the residuals of an optimally-fitted model and thus will decrease the accuracy of the model's predictions. However, a model need not have high predictive accuracy in order to be an effective surrogate model; indeed, an ideal surrogate model only needs two properties: it should be easy to optimize and its global optimum should be in a similar location to the global optimum of the original landscape.

To fit the convex quadratic under-estimator models, Rosen and Marcia [158] proposed two methods, both of which formulate the problem as a two-step procedure. In the first step of each, the problem is simplified by imposing additional constraints on the quadratic model, thereby allowing the problem to be efficiently solved using linear programming. They then proposed to use the incumbent solution from the linear program to initialize the parameters of the non-linear program that is used to fit the general convex quadratic approximation model. This initial guess for the solution to the non-linear program can then be iteratively refined using an interior-point method.

In the first method, they force the Hessian to be a diagonal matrix, and in the second, they force it to be a diagonally-dominant matrix. Rosen and Marcia [158]

found that the second method yielded higher-quality final solutions, therefore, we chose to use it when possible. For a set of m data points in $\mathbb{R}^n$, the general convex quadratic model fit using this procedure uses $s = (n+1)\cdot(n+2)/2$ variables, and thus requires at least $m \geq s$ data points before models can be fit. However, the separable convex quadratic model obtained using a diagonal Hessian matrix only uses $s = 2\cdot n+1$ variables. Since we know from Chapter 4 that most AutoML hyper-parameters do not interact strongly in the vicinity of high-quality solutions, we therefore chose to fit a separable convex quadratic surrogate model whenever $2\cdot n+1 < m < (n+1)\cdot(n+2)/2$, and we fit a general convex quadratic surrogate model when $m \geq (n+1)\cdot(n+2)/2$.

The methods proposed by Rosen and Marcia [158] also require a parameter ε, which is used to constrain the minimum eigenvalue of the Hessian, thereby improving the conditioning of the surrogate-fitting optimization problems and ensuring that the optimum of the surrogate is unique. In our experiments, we set $\varepsilon = 10^{-3}$, since it is well-known that some hyper-parameters may have relatively little impact on the solution quality (see for example, Hutter et al. [93] or Chapter 4). Furthermore, we optimized the linear programs using the default linear-programming solver available in python's `SciPy` package [185]. To optimize the non-linear programs, we used `SciPy`'s sequential least squares quadratic programming solver (SLSQP).

5.1.2 B-Spline Basis Surrogate Models

Using a convex quadratic as a surrogate model has the advantage that it enforces our prior expectation that most AutoML loss landscapes are uni-modal. However, quadratic models are much less flexable than the original TPE model, and as shown in Chapter 4.4, AutoML loss landscapes are not typically convex. As an alternative, we focus on one of the other properties that we observed in Chapter 4, that is, that interactions between hyper-parameters in AutoML loss landscapes are not often relatively strong, particularly in the vicinity of the global optimum. Given this observation, fitting a linear model to a B-spline basis function [56] is a natural choice, as they are known to provide high-quality and stable approximations, and they force the fitted model to contain no hyper-parameter interactions.

Gaussian processes are a more commonly used model in AutoML methods that would likely provide similar-quality results to spline models. Gaussian processes are typically chosen, because they can provide a confidence interval around their predictions, which is necessary for most Bayesian optimization methods. However, in our application, where we wish to focus on exploitation rather than trading off between exploration and exploitation, we do not require this confidence interval. Instead, we prefer splines for their lower computational complexity.

Another advantage of using splines is that they support various methods for extrapolation. For our application, we chose to extrapolate using a constant function, as this will ensure that unpromising directions for exploration remain unexplored, whereas promising directions for exploration will continue to be explored.

To compute the spline basis function, we used the implementation available in `scikit-learn`'s latest development branch,[2] with the default settings of its parameters: five knots per feature with a third-degree basis. For hyper-parameters that are searched on a linear scale, we used a uniform spacing for the knots, whereas we spaced the knots geometrically for hyper-parameters that are searched on a logarithmic scale. Because we anticipated that the models would frequently be under-constrained, particularly early in the configuration process when relatively little configuration performance data is available, we used LASSO regression with five-fold cross-validation to choose the value of the regularization parameter.

5.1.3 Asynchronous Model Fitting and Selection

Since fitting these surrogate models typically requires substantially more time than fitting a TPE model, we fit the models asynchronously. In BOHB, new models are fitted when worker processes report new configuration results to the main process. In our methods, we initiate and check for completed model-fitting processes in this function as well (see Algorithm 1). We limit the number of models being fitted at any given time to one, to ensure that it does not overburden the machine and slow down the evaluation of the configurations.

Similarly, BOHB also evaluates each candidate configuration asynchronously. Each time a worker process for BOHB is done evaluating a configuration, the

[2] As of 2021-06-24.

Algorithm 1 The report function for our experimental AutoML configurators.

```
input
    m_I(c), the performance of a newly evaluated configuration
    f, the level of fidelity at which the configuration was evaluated
    R, a dictionary mapping fidelities to configuration evaluation results
    model, a dictionary mapping fidelities to previously-fit surrogate models (or None)
output
    None
procedure report
    # Save the configuration results and check for model-fitting results
    R[f][c] := m_I(c)
    Check if a model-fitting process has completed
    # Save the new model, if applicable
    if a model-fitting process reported model_new at fidelity level f', then
        if model[f'] is None or model_new better approximates R[f'] than model[f'], then
            model[f'] := model_new
    # Start a new model-fitting process, if applicable
    if no model-fitting processes are running, then
        for each fidelity f ∈ R in descending order, do
            if length(R[f]) ≥ 2 · |P| + 1, then
                Initiate a model-fitting process with R[f]
                break
    return
```

worker process reports its results to the main process. When the main process receives this data, it checks to see if there is sufficient data available to fit a model and if there are no models that are currently being fitted. If both of these conditions are met, the main process will initiate an asynchronous model-fitting process. Once the model-fitting process is complete, it reports its results back to the main process.

Similar to BOHB, we fit separate CQA or spline models for each level of fidelity of the performance estimates. In some early prototypes, we experimented with using a weighted combination of some or all of the available performance estimates, regardless of their level of fidelity. However, we observed that because some measures of fidelity not only decrease the variability in the performance estimates, but also substantially change the performance values for a given configuration, doing so typically caused the model-fitting procedure to diverge, thereby producing unreliable surrogate models. Therefore, as soon as sufficient data to fit a model is available for a higher-fidelity performance estimate, the methods switch to fitting models using the higher level of fidelity instead.

In some rare cases the addition of new configuration performance data, or the switch from a large training set with low-fidelity performance estimates to a small training set with higher-fidelity performance estimates, can de-stabilize the model-fitting procedure, thereby causing it to diverge. To protect against this, we only accept a new surrogate model if the mean absolute error of a newly-fitted model is not worse than the current accepted model, if one exists.

5.1.4 Suggesting a Configuration

The simplest way to use a surrogate model to guide an optimization process is to find the globally optimal solution to the surrogate model and then to evaluate that solution using the original objective function. When the surrogate model is a convex quadratic model, this would be both easy and inexpensive to do. However, surrogate models that are fitted to previous performance data are typically only good approximations of the original landscape near to the previously-evaluated solutions. Therefore, it is common practice to employ, for example, a trust region, which limits how far away from the current incumbent the surrogate model should be trusted. The next configuration to evaluate is then the configuration predicted to have the optimal performance by the surrogate model that is contained within the trust region.

However, in our application, the surrogate models are fitted asynchronously, so it is also sometimes necessary for multiple new configurations to be suggested using the same surrogate model. Therefore, rather than using a trust region, we employ a somewhat similar trick to BOHB, whereby we randomly sample new configurations from a mixture of three Gaussian models, where each of the three Gaussian models are parameterized to be centered around the three best known configurations. To determine which configurations are best, we simply used the largest available fidelity budget evaluation for each of the configurations (see Algorithm 2).

We chose to use a mixture of three Gaussian models instead of a single Gaussian model, in case any of the low-fidelity budgets yield high-variance performance estimates that could cause the procedure to select the wrong configuration as the incumbent. We heuristically set the covariance of each of the Gaussian models to

Algorithm 2 The suggest function for our experimental AutoML configurators.

```
input
    R, A dictionary mapping fidelities to configuration evaluation results
    model, A dictionary mapping fidelities to previously-fit surrogate models (or None)
    C, the parameter configuration space
output
    c_next, a configuration to evaluate
procedure suggest
    # Find the largest-fidelity model, if any
    Initialize model_current := None
    for each fidelity f ∈ R in descending order, do
        if model[f] is not None, then
            model_current := model[f]
            break
    # Sample a new configuration using the model, if possible
    if model_current is None, then
        Pick c_next by sampling a random configuration from C
    else
        # Find the best configuration at any fidelity level
        Initialize an empty dictionary, performance := {}
        for each fidelity f ∈ R, in ascending order, do
            for each configuration c ∈ R[f], do
                performance[c] := R[f][c]
        # Sample from a mixture of 3 Gaussian models (see text for details)
        Sample 9 configurations from around the best 3 configurations in performance
        Pick c_next as the best of the 9 configurations according to model_current
    return c_next
```

be a diagonal matrix parameterized such that the standard deviation of each hyper-parameter is equal to 5% of the total range of values for the hyper-parameter. Finally, each time a configuration needs to be suggested we sample a total of nine candidate configurations from the mixture of Gaussian models and suggest the one predicted to have the best performance by the surrogate model. The number of Gaussians, their standard deviations, and the number of samples drawn were each chosen based on some preliminary experiments on hand-crafted benchmarks.

For spline surrogate models, this procedure has the added benefit that we do not need to be able to locate the optimal configuration according the spline model, which may be non-trivial to do, as the model is not guaranteed to be uni-modal.

Table 5.1: The five real-world and four hand-crafted scenarios used to compare the four different hyper-parameter configurators.

Model/Algorithm	Dataset/Task	Fidelity Type	Fidelity Range	# HP
Bayesian Neural Network	Boston Housing	MCMC Steps	[500, 10 000]	7
	Protein Structure	MCMC Steps	[500, 10 000]	7
Proximal Policy Optimization	Cart Pole Swing-Up	# Training Runs	[1, 9]	5
Xgboost	King-Rook *vs* King-Pawn	# Estimators	[2, 128]	6
Histogram Gradient Boosting	Covertype	Dataset Fraction	[1%, 100%]	7
Simulated Binary Classifier	Symmetric	Dataset Size	[500, 5 000]	1
	Asymmetric	Dataset Size	[500, 5 000]	1
	No Interactions	Dataset Size	[500, 5 000]	2
	Interactions	Dataset Size	[500, 5 000]	2

5.2 Experimental Setup

In preliminary experiments, we compared prototypes of our methods against several existing state-of-the-art baseline hyper-parameter configuration procedures, including BOHB [59], HyperBand [113] and ASHA [114]; however, we quickly realized that the method used to exploit low-fidelity approximations of configuration performance often had an equal or greater effect on the performance of the method than changing the search procedure. However, since our primary goal in this study is to answer the question: "Can AutoML loss landscape structure be exploited to improve configurator performance?", the fidelity budget allocation strategy is effectively an orthogonal design decision that acts as a confounding factor. Therefore, to isolate the impact of the search method from the fidelity budget allocation strategy, our preliminary analysis of the methods only compares them to BOHB [59] and Hyperband [113]. To further eliminate confounding factors, we used the implementation of Hyperband that is available with the original implementation of BOHB, and used the same values for all of the shared parameters of the configurators.

All but one of the AutoML scenarios in Chapter 4 contained pre-collected configuration performance data, and we did not have access to the original training algorithms and datasets. As a result, we evaluated each configurator on five new scenarios, spanning a range of different machine learning methods and datasets, as well as on four hand-crafted benchmark scenarios. We present a summary of the scenarios in Table 5.1. For each of the real-world scenarios, we performed 25

independent runs of the configurators, and we report the median test loss with 95% bootstrap percentile confidence intervals. To obtain a stable estimate for the test loss, we performed 5 independent training runs of the machine learning methods using each configuration and recorded their mean test losses. For the hand-crafted benchmark scenarios, which require substantially less running time to use, we performed 101 independent runs of the configurators and we report the exact loss underlying the landscape.

The first three scenarios we studied were also used in the original evaluation of BOHB [59]. The first two are for a Bayesian neural network trained using stochastic gradient Hamiltonian Monte-Carlo sampling [39] with scale adaption [166] on two UCI datsets [52], Boston housing and protein structure. We were unable to reproduce qualitatively similar results compared to those reported in Falkner et al. [59] for either of these two scenarios.[3] In each case, we observed that the gap between the performance for BOHB and Hyperband was much smaller than reported in their original study.

The third scenario we used from their study configures the hyper-parameters of proximal policy optimization [160] on the cart pole swing-up reinforcement learning task. In this case, even though the final performance gap between BOHB and Hyperband was also slightly smaller than originally reported, the results were still qualitatively similar.

For the next two scenarios, we configured the hyper-parameters of Xgboost [38] on the UCI [52] king-rook *vs* king-pawn [162] dataset and histogram gradient boosting (`scikit-learn`'s [141] implementation of a machine learning method inspired by LightGBM [100]) on the covertype dataset [28].

We also evaluated the four configurators on four artificial scenarios that simulate a binary classifier's error rate, thereby allowing us to control the precise shape of the AutoML loss landscape optimized by the configuration procedures. This control allowed us to test hypotheses about the relative behaviour of the methods under various circumstances, thereby providing answers to questions that arose

[3] From personal communication with the authors, we confirmed that we correctly set up the experiments and reproduced their method of analysis. To the best of our knowledge, the only differences in our execution environments were the particular machines used and, perhaps, the versions of some of the software packages.

from our results on the real-world benchmark scenarios.

The simulated classifier was constructed such that the variability in the measurements of the landscape mimic the behaviour of a binary classifier with a given error rate on a validation set of a given size. In particular, for each scenario, we defined functions that describe the landscape by mapping configurations to error rates, p, such that $0 < p < 1$. Then, given the error rate, p, of a particular configuration and a validation dataset size of n (the level of fidelity), we simulated an evaluation of a configuration by randomly drawing n trials from a binomial distribution with a success rate of p, and we recorded the loss as the number of successful trials divided by the total number of trials, n. We simulated short training times for the procedure by sleeping for one second per 1 000 instances in the dataset. This simulation ignores the variability in performance due to the random seed used when training a machine learning model; however, it still provides a cheap-to-evaluate and approximately realistic method for generating landscapes with exactly known structure that can be used to evaluate the configurators.

We motivate and define the precise landscapes that we used for the simulated classifier scenarios in Chapter 5.3, based on the results of the real-world scenarios.

5.3 Results

We show the anytime configuration results for three configuration budgets for the five real-world scenarios in Table 5.2. A similar analysis of other configuration budgets (not shown) yields qualitatively similar results. Overall, none of the methods consistently perform substantially better than any of the others; however, BOHB, which uses the TPE model, found the best or tied for the best on each scenario for all configuration budgets. Unfortunately, while both of the new methods we proposed can find high-quality configurations, neither of them appear to consistently work as well as BOHB. In fact, Hyperband, which only relies on random search, often performs similarly or only slightly worse than BOHB, and often performs similarly or slightly better than the methods using the CQA or spline models.

Why does a CQA surrogate model not work better? Given that we observed in Chapter 4 that most AutoML loss landscapes are uni-modal, it seems reasonable to

Table 5.2: Results from applying the hyper-parameter configurators to the real-world AutoML scenarios. At each configuration budget we show the median test loss over the independent configurator runs with 95% bootstrap percentile confidence intervals. The median losses are shown in boldface if they are not worse than the best loss for a given configuration budget, according to a one-sided Welch t-test with a 5% significance level. All losses are scaled by dividing by the loss of the single best known configuration for each scenario. Smaller is better.

	BNN-BH	**BNN-PS**	**PPO-CP**	**XGB-KR**	**HGB-CT**
Budget (10%)	1 580 Seconds	2 420 Seconds	4 667 Seconds	148 Seconds	888 Seconds
CQA	**1.91** [1.46, 2.42]	**2.14** [1.63, 4.30]	**10.49** [7.67, 13.74]	**1.72** [1.44, 2.28]	1.28 [1.18, 1.35]
Spline	**1.52** [1.40, 1.79]	**1.77** [1.63, 1.87]	13.74 [9.42, 13.74]	**1.72** [1.41, 2.41]	1.30 [1.21, 1.42]
Random (HB)	3.10 [1.60, 4.00]	**2.13** [1.77, 4.74]	**7.73** [6.21, 10.13]	**1.91** [1.50, 2.00]	**1.22** [1.13, 1.42]
TPE (BOHB)	**1.76** [1.45, 2.05]	**1.87** [1.56, 3.09]	**7.70** [4.57, 9.84]	**1.69** [1.47, 2.00]	**1.23** [1.14, 1.48]
Budget (50%)	7 899 Seconds	12 102 Seconds	23 337 Seconds	742 Seconds	4 442 Seconds
CQA	**1.44** [1.31, 1.67]	**1.97** [1.72, 2.60]	5.73 [3.06, 6.47]	1.91 [1.44, 2.12]	1.17 [1.12, 1.24]
Spline	**1.50** [1.42, 1.70]	1.85 [1.55, 2.24]	**5.31** [4.10, 8.43]	**1.56** [1.41, 1.94]	1.11 [1.10, 1.14]
Random (HB)	**1.39** [1.36, 1.57]	**1.50** [1.31, 1.83]	**4.97** [3.25, 6.86]	**1.78** [1.53, 1.97]	**1.08** [1.07, 1.10]
TPE (BOHB)	**1.38** [1.27, 1.68]	**1.24** [1.10, 1.40]	**3.59** [2.31, 5.37]	**1.37** [1.28, 1.53]	**1.09** [1.07, 1.14]
Budget (100%)	15 798 Seconds	24 204 Seconds	46 674 Seconds	1 484 Seconds	8 884 Seconds
CQA	**1.44** [1.31, 1.67]	**1.97** [1.62, 2.32]	**4.45** [2.92, 6.11]	1.72 [1.41, 2.00]	1.12 [1.12, 1.17]
Spline	**1.50** [1.43, 1.77]	1.85 [1.35, 2.24]	4.44 [3.53, 6.51]	**1.56** [1.41, 1.94]	1.09 [1.06, 1.13]
Random (HB)	**1.39** [1.36, 1.57]	**1.50** [1.31, 1.83]	**4.67** [2.69, 6.66]	1.72 [1.53, 1.97]	1.08 [1.06, 1.09]
TPE (BOHB)	**1.38** [1.27, 1.68]	**1.25** [1.10, 1.43]	**2.22** [1.76, 3.21]	**1.38** [1.31, 1.59]	**1.05** [1.03, 1.08]

assume that a uni-modal surrogate function should be able to improve the performance of BOHB. However, since the CQA surrogate model is a quadratic function, it implicitly assumes that the landscape is symmetric around the globally optimal configuration – which was frequently not the case in the one- and two-dimensional hyper-parameter response slices that we observed in Chapter 4. Therefore, this could be at least partly responsible for the poor performance that we observed with the CQA surrogate model.

To investigate this hypothesis, we compared the performance of the four configuration procedures on two artificial scenarios, where the machine learning algorithm's loss is simulated as described in Chapter 5.2. In both cases, the simulated classifier had a single real-valued hyper-parameter, x, that was searched over the range of values $[-1, 1]$. For the first scenario, we set the error rate to be

$$p_{\text{symmetric}}(x) = |x|^3 + 0.01 \tag{5.1}$$

and for the second scenario we set it to

$$p_{\text{asymmetric}}(x) = \begin{cases} |x|^3 + 0.01 & \text{if } x < 0 \\ \frac{1}{5} \cdot |x|^3 + 0.01 & \text{otherwise.} \end{cases} \tag{5.2}$$

In both cases, we clipped the error rate to be within [0, 1].

In the asymmetric scenario, it *should* be easier for the configuration procedures to find better-quality configurations, because 50% of the configurations obtain better error rates. However, we observed that when using a CQA surrogate model, the anytime configuration performance on the asymmetric scenario was actually worse than on the symmetric scenario (see Table 5.3). Furthermore, we see that on the symmetric scenario, using the CQA surrogate model yields significantly better performance than all of the other methods for most configuration budgets; however, on the asymmetric scenario it instead ties with all of the other methods for all but the 10% configuration budget, at which point it still has a small advantage.

This example clearly illustrates a rather large failure mode for using a CQA model as a surrogate function; even though the asymmetric artificial classifier scenario aligns very closely with the types of landscapes for which we had designed it to perform well, we see that it does not outperform any of the other methods at all.

Why does the spline surrogate model not work better? Similar to the CQA model, we had expected the spline surrogate model to yield better results, given that we observed in Chapter 4 that most hyper-parameters could be safely configured independently in a random sequence. However, note that this definition of simplicity among hyper-parameter interactions may not actually indicate that a *surrogate model* can assume that the hyper-parameters are independent of each other. Indeed, the function $f(x,y) = |x-y|$ is trivial to minimize when considering either x or y independently – for any fixed value of one, the other can be adjusted to yield an optimal solution. Despite this "simplicity", the spline surrogate model would be unable to accurately model the function due to the strong interaction between the hyper-parameters x and y.

To investigate, we again compared the performance of the four configuration procedures on two artificial classifier scenarios. Each scenario contained two

Table 5.3: Results from applying the hyper-parameter configurators to the artificial, hand-crafted AutoML scenarios. At each configuration budget (the cumulative number of training examples used to "train" candidate models) we show the median loss (the percentage of errors) over the independent configurator runs with 95% bootstrap percentile confidence intervals. The median losses are shown in boldface if they are within 0.01% of the best loss for a given configuration budget. Smaller is better.

	Symmetric	**Asymmetric**	**No Interactions**	**Interactions**
Budget (10%)	13k Examples	13k Examples	13k Examples	13k Examples
CQA	**1.01** [1.01, 1.02]	**1.04** [1.03, 1.09]	**3.56** [2.89, 4.88]	**3.08** [2.73, 3.70]
Spline	1.06 [1.03, 1.10]	1.08 [1.04, 1.13]	5.19 [3.93, 6.24]	3.50 [2.99, 4.19]
Random (HB)	1.11 [1.05, 1.20]	1.08 [1.05, 1.12]	5.26 [3.75, 6.39]	4.12 [3.40, 4.43]
TPE (BOHB)	1.12 [1.07, 1.25]	1.08 [1.04, 1.14]	4.32 [3.36, 5.48]	3.68 [3.03, 4.32]
Budget (50%)	67k Examples	67k Examples	67k Examples	67k Examples
CQA	**1.01** [1.00, 1.01]	**1.02** [1.01, 1.03]	2.12 [1.86, 2.29]	**1.27** [1.19, 1.37]
Spline	1.02 [1.01, 1.04]	**1.02** [1.01, 1.03]	**1.27** [1.20, 1.44]	1.60 [1.47, 1.74]
Random (HB)	1.04 [1.02, 1.05]	**1.02** [1.01, 1.03]	2.06 [1.84, 2.40]	1.91 [1.77, 2.25]
TPE (BOHB)	1.04 [1.03, 1.10]	**1.02** [1.01, 1.04]	2.40 [1.90, 2.83]	1.64 [1.38, 2.04]
Budget (100%)	135k Examples	135k Examples	135k Examples	135k Examples
CQA	**1.00** [1.00, 1.00]	**1.01** [1.00, 1.01]	1.32 [1.21, 1.47]	**1.15** [1.12, 1.18]
Spline	**1.01** [1.00, 1.02]	**1.01** [1.00, 1.02]	**1.11** [1.09, 1.13]	1.26 [1.19, 1.31]
Random (HB)	1.02 [1.01, 1.04]	**1.01** [1.01, 1.02]	1.65 [1.54, 1.85]	1.59 [1.48, 1.73]
TPE (BOHB)	1.03 [1.01, 1.04]	**1.01** [1.01, 1.02]	1.38 [1.23, 1.59]	1.27 [1.16, 1.32]

hyper-parameters, x and y in $[-1, 1]$. In one case there were no interactions and the error rate was defined as

$$p_{\text{no interactions}}(x, y) = \frac{1}{2} \cdot |x| + 0.01 \tag{5.3}$$

(y was ignored). The second case used the same objective function after applying a 45 degree rotation, that is, the error rate was set to

$$p_{\text{interactions}}(x, y) = \frac{1}{2 \cdot \sqrt{2}} \cdot |x - y| + 0.01. \tag{5.4}$$

Similar to the previous experiment, we clipped the error rate to the range [0, 1].

In this experiment, the scenario with hyper-parameter interactions *should* be easier for the configurators, because the basin of optimal solutions lies along the diagonal $x = -y$ (which has length $2 \cdot \sqrt{2}$ for $x, y \in [-1, 1]$) instead of along the

line $x = 0$ (which has length 2 over the same domain). However, similar to the case where we challenged the CQA model, we see that this simple rotation of the landscape causes the scenario to become substantially more challenging for the spline model. For both of the larger two configuration budgets, the spline model finds configurations with smaller error rates when there are no hyper-parameter interactions (see Table 5.3). Similarly, for all three configuration budgets, we see that the spline model finds the best solutions when there are no interactions, whereas when there are interactions, the CQA model performs best. (Note that both of the landscapes in these scenarios are perfectly symmetric along two axes around the global optima, hence we would expect CQA to do well in both cases – which it does.)

This example clearly illustrates an important failure mode for the spline model; even though it may often be sufficient to configure each hyper-parameter independently in a random sequence, it is not necessarily safe to build surrogate models that assume that they are independent.

Furthermore, similar to the parameters of the meta-heuristics studied by Yuan et al. [196] (see Chapter 2.1), we expect there to be some hyper-parameters with this type of interaction in AutoML scenarios. For example, consider the learning rate (`l_rate`) and the momentum decay rate (`mdecay`) hyper-parameters for the Bayesian neural network scenarios. The learning rate corresponds to the step size taken in each iteration of the optimization process, and the momentum decay rate corresponds to how quickly old gradient information is lost. Specifically, a large value for momentum corresponds to making heavy use of the previous gradients, whereas a small value corresponds to heavily trusting the most recently observed gradient.

In cases where gradient measurements are noisy, it is helpful to use large values of momentum to take an average over a larger number of previous gradient observations, thereby smoothing the path taken by the optimizer. This should be particularly important for large step sizes, since otherwise the optimizer will be taking large steps in nearly-random directions, effectively placing too much trust in each gradient observation. Indeed, we empirically observed precisely this relationship between these two hyper-parameters. For example, the best 5% of the 1 234 anytime configurations suggested by any of the configuration procedures have a

correlation coefficient of 0.83 for the `l_rate` and `mdecay` hyper-parameters on the protein structure dataset. These observations likely explain why the spline model did not produce better results on the real-world scenarios.

5.4 Chapter Summary

In this chapter, we introduced two experimental modifications to a state-of-the-art AutoML hyper-parameter configuration procedure, BOHB. In its original form, BOHB combines Bayesian optimization (using a TPE model) with Hyperband (which exploits low-fidelity approximations of configuration performance). Both of the modifications we made were inspired by the simplicity of the structure we observed in most AutoML loss landscapes in Chapter 4. Our hypothesis was that it should be more efficient to search landscapes that are relatively simple by using surrogate models that mimic aspects of that simplicity to guide the search process. However, in both cases, we observed that the models we chose were too simple to be effective on real-world problems.

First, we replaced the TPE model in BOHB with a convex quadratic approximation (CQA) surrogate model. We assumed that because the CQA model is uni-modal (like most AutoML loss landscapes), it should help guide the search process away from regions of the configuration space that we can reasonably assume contain poor configurations – regardless of whether or not configurations in that part of the search space have previously been evaluated.

However, we demonstrated that even landscapes that we originally assumed would be easy to optimize when using the CQA model turned out not to be. In particular, we constructed two hand-crafted benchmarks with known objective functions: one that was both uni-modal and symmetric around the global optimum, and a second that was still uni-modal, but not symmetric. On the symmetric landscape, the CQA model outperforms the other baselines; however, for the slightly altered asymmetric landscape, the CQA model does worse and instead ties with the other baselines. Since we do not expect most landscapes to be symmetric around the global optimum, this clearly explains why we did not observe competitive behaviour on the real-world benchmark problems.

Our second attempt replaced the TPE model in BOHB with a B-spline model.

Since the convex quadratic approximation model was not expressive enough to properly model the landscapes, we chose a more flexible class of models, in the hope that it could exploit a different kind of simplicity in AutoML loss landscapes. In particular, in Chapter 4, we showed that our simplistic configuration procedure that configures each hyper-parameter independently, a single time and in a random order, was able to find configurations tied with optimal for all of the scenarios we studied. This made spline models a natural choice, since they are known to provide high-quality approximations of functions with no interactions between the variables. Unlike CQA models, splines are not constrained to be uni-modal; however, by extrapolating with a constant, a spline model should still help avoid suggesting configurations that are likely to be bad in regions of the configuration space that have been unexplored.

Unfortunately, spline models turned out to not be helpful on the real-world benchmark problems either. In this case, we demonstrated on two additional hand-crafted benchmarks that while spline models do in fact perform better than the other baselines on objective functions that contain independent hyper-parameters, even a simple 45 degree rotation of the landscape ($p_{\text{no interactions}}(x,y) = \frac{1}{2} \cdot |x| + 0.01 \rightarrow p_{\text{interactions}}(x,y) = \frac{1}{2 \cdot \sqrt{2}} \cdot |x - y| + 0.01$) is enough to cause it to perform worse than the other baselines. Note that even though $p_{\text{interactions}}(x,y) = \frac{1}{2 \cdot \sqrt{2}} \cdot |x - y| + 0.01$ contains strong interactions between x and y, it would still be trivial to optimize by varying either x or y independently of the other. In practice, we expect some interactions to arise between hyper-parameters that cause landscapes with this type of structure. For example, we showed that the high-quality configurations for the Bayesian neural network scenarios have a correlation coefficient of 0.83 between the values of their momentum decay rate and learning rate hyper-parameters.

Finally, we also observed that even though the original version of BOHB performed best in all of the real-world scenarios we studied, Hyperband often performed the same or only slightly worse. The only difference between the two methods is that BOHB uses a TPE model to search the configuration space, whereas Hyperband uses random search. This suggests that while using a surrogate model to guide the search process can make it more efficient, the benefit of doing so appears to be relatively small. We suspect that this is quite likely because of Hyperband's ability to make use of cheap, low-fidelity approximations of the landscapes. If

the cost of evaluating a modest number of additional configurations is very small, then the performance penalty from using random search instead of something more intelligent will also be small. For this reason, the benefit that can be obtained by using something even more intelligent than BOHB's TPE model is likely also rather small.

Chapter 6

Analysis II: Running Time Minimization Landscapes

In Chapters 4 and 5 we gained insights into the structure of AutoML loss landscapes and how that structure can be exploited. However, in Chapter 5, we were unable to reach our ultimate goal to develop a new hyper-parameter configuration procedure that improves upon the state of the art. Therefore, we may wonder why should we continue to analyze the structure of landscapes for other, related algorithm configuration problems? In fact, there are several good reasons. As motivated in Chapter 1, landscape analysis is useful for other reasons; for example, to build intuition about existing configuration procedures to retrospectively analyze their relative performance.

Furthermore, even though we observed that BOHB was already performing reasonably well at learning and exploiting the structure present in AutoML loss landscapes, this may not be the case for similar algorithm configuration procedures that seek to minimize the running time of algorithms for solving $\mathcal{NP}$-hard and $\mathcal{NP}$-complete problems. In fact, for many $\mathcal{NP}$-hard problems it is unclear whether or not it is possible to obtain low-fidelity approximations of an algorithm's performance, whereas for AutoML scenarios the pervasiveness of low-fidelity approximations allows a large number of low-quality configurations to be quickly eliminated. Therefore, the fraction of the configuration budget spent evaluating poorly-performing configurations is typically larger in scenarios with $\mathcal{NP}$-hard

problem instances. As a result, running time minimization scenarios with $\mathcal{NP}$-hard problems is arguably an even more promising application area for landscape analysis than AutoML.

In this chapter,[1] we apply the methods for landscapes analysis described in Chapter 3 to the running time minimization landscapes of several prominent algorithms for three widely-studied $\mathcal{NP}$-hard problems: SAT, TSP and MIP. Our goal is to determine whether or not these landscapes are similar to AutoML loss landscapes – that is, are they globally uni-modal (or convex) and are their parameter interactions strong or weak?

6.1 Experimental Setup

Unlike in Chapter 4, we were unable to find any existing data sets relating grids of algorithm configurations to performance, so we collected our own data. The evaluation of algorithm configurations for solving $\mathcal{NP}$-hard problems is typically rather costly, so we focus our analysis primarily on one-dimensional parameter response slices of the landscapes, with some limited analysis on two-dimensional slices.

6.1.1 Parameter Response Slices

To focus our analysis on the most interesting portion of the landscapes, we studied parameter response slices centered around high-quality configurations. To do this, we first performed 25 independent runs of SMAC [89] for each scenario (configuring both numerical and categorical parameters), and subsequently evaluated the resulting configurations on the entire training set. We then used the best-performing configuration on the training set as the centre point for each of the one- and two-dimensional parameter response slices studied in Chapters 6.2 and 6.3.

Even evaluating every possible value for both of the one dimensional parameter response slices in our EAX scenario (described in more detail in Chapter 6.1) would take 6 CPU years, so we reduced the one-dimensional slices to only 15 pa-

[1] This chapter is based on the experiments and results from joint work with Holger Hoos. The results for the one-dimensional landscape analysis were published in Pushak and Hoos [148], where they received the 2018 PPSN best paper award. A single award is given out every two years, selected by an expert committee with guidance from a popular vote.

rameter values each. We further reduced the two-dimensional slices to 7 values for each parameter, resulting in up to 49 pairs of parameter values for each slice. If a parameter had an insufficient number of possible values, then we used all of them; otherwise, to obtain high resolution near the best known parameter setting, we increased the spacing between adjacent numeric values exponentially with increasing distance from the best known value.

We added an additional constraint to ensure that we obtained some coverage of the parameter response on either side of the best known parameter setting. In particular, we restricted at most 75% of the points to be on one side of the best known value. (Note that this constraint could not always be enforced, for example, if the best known value took the maximum or minimum allowed value.) Since there were still many possible locations for the points satisfying these constraints, we multiplied the grid of points by a randomly chosen weight to choose their exact location.

For any integer-valued parameter, we first determined a set of values as for real-valued parameters and then rounded each setting thus obtained to the nearest valid and previously unused value. Finally, we performed 10 independent runs per instance to obtain median performance values to ensure that the performance estimates for each configuration were robust.

This method produces grids of irregularly spaced parameter values that are mostly concentrated around the best known configuration. Some of the metrics we used (for example, FDC, see Chapter 3.2.6) require knowledge of the distance between two configurations. Therefore, we normalized the range of values for each numerical parameter to the range [0, 1] when calculating distances to ensure that they were all on the same scale. For categorical parameters, we treated the distance between each pair of possible values as 1.

6.1.2 Bootstrap Analysis and Confidence Intervals

To account for the variance between independent target algorithm runs and problem instances, we used a nested bootstrap procedure similar to the one by Mu and Hoos [130] with 1 001 outer and inner bootstrap samples. To be precise, for each configuration and problem instance, we created 1 001 (inner) bootstrap samples of

the 10 independent runs to obtain a distribution of median performance values for each problem instance. Next, we created 1 001 bootstrap samples of the instance set. For each occurrence of a problem instance in a sample of the instance set, we sampled a value from the corresponding distribution of median performance values. Finally, we calculated the (outer) median performance observed on the instance sets for each bootstrap sample of the per-instance medians. We used this distribution to calculate 95% bootstrap percentile confidence intervals for the performance observed for each configuration. Since each parameter response slice contained up to 49 unique configurations (each with a corresponding confidence interval), we used Bonferroni multiple testing correction to adjust their confidence intervals.

6.1.3 Algorithms and Problem Instances

We studied 10 different algorithm configuration scenarios, spanning three widely studied, $\mathcal{NP}$-hard problems (SAT, MIP and TSP), 6 prominent algorithms and 5 well-known instance sets. All of these scenarios involve the minimization of running time, measured in PAR10 – that is, mean running time with timed-out runs counted at 10 times the running time cutoff. In Table 6.1, we summarize the configuration budgets and running time cutoffs used for our scenarios. All instance sets are readily available online in ACLib scenarios that have been identified as interesting and challenging benchmarks for algorithm configurators [94]. For the SAT and MIP instance sets, we used the same budgets and running time cutoffs as specified in the corresponding ACLib scenarios. We increased the running time cutoff for the test set (and parameter slices) for the SAT and TSP scenarios, in order to better assess poorly performing configurations.

Table 6.2 provides an overview of the 6 algorithms we studied. We introduce a few new, state-of-the-art algorithms not found in existing ACLib scenarios.

For the SAT scenarios, we studied CaDiCaL [25], because it was one of the top-performing, configurable solvers in the application track of the 2017 SAT competition; lingeling [25], because it was the winner of the 2014 Configurable SAT Solver challenge on the circuit-fuzz and BMC08 instances; and cryptominisat [165], because it is a variant of the well-known and commonly used minisat algorithm. Ref-

Table 6.1: The instance sets we studied from ACLib scenarios and the configuration budgets and training/testing running time cutoffs (all measured in CPU Seconds) used for their scenarios.

Problem	Instance Set	Config Budget	Training Cutoff	Test Cutoff	# Instances
SAT	circuit-fuzz	172 800	300	600	586
	BMC08	172 800	300	600	718
MIP	CLS	172 800	10 000	10 000	50
	Regions200	172 800	10 000	10 000	1 000
TSP	TSP-RUE-1000-3000	86 400	86	3 600	250

Table 6.2: The 6 algorithms we studied for the analysis or running time minimization landscapes for $\mathcal{NP}$-hard problems.

Problem	Algorithm	Version	# Num	# Cat
SAT	CaDiCaL	sc17	40	22
	lingeling	azf	185[1]	137
	cryptominisat	4.1	22	36
MIP	CPLEX	12.6	22	52
TSP	EAX+restart	JDL	2	0
	LKH+restart	2.0.7	12	9

[1]We configured all 185 numerical parameters, but only studied slices for the 10 most important (see text for details).

erence implementations of lingeling and cryptominisat were directly obtained from ACLib 2.0, whereas that of CaDiCaL was taken from the 2017 SAT competition.

For the TSP scenarios, we chose two extensively studied [53, 131], state-of-the-art, inexact solvers: EAX [132] and LKH [82]. We used the same implementations as Mu *et al.* [131] and Dubois-Lacoste *et al.* [53], which were modified to use a restarting mechanism and terminate upon reaching optimal solution quality values (which are known from long runs of an exact solver). The TSP scenarios in ACLib configure for solution quality, so we chose these solvers to focus on running time minimization.

For the MIP scenarios, we studied the high-performance commercial solver IBM ILOG CPLEX [1], version 12.6 (featured in several ACLib scenarios), which terminates upon finding an optimal solution to a given MIP instance and completing a proof of optimality. We slightly modified the CPLEX scenarios from ACLib,

by treating CPLEX as a randomized algorithm. Earlier versions of CPLEX used a fixed random seed that was not exposed to the user; however, CPLEX is in fact a randomized solver, and treating it as such avoids potential problems arising from bias due to the use of a specific random seed.

For every algorithm except lingeling we were able to evaluate one-dimensional parameter response slices for all of their numerical parameters; however, since lingeling had so many parameters, we restricted our analysis to a subset of them. Falkner *et al.* [58] reported the 10 most important parameters according to functional ANOVA [93] (all of which were numerical) for lingeling on the circuit-fuzz instance set, so we only used these 10. We also slightly modified the ranges for a few parameters for LKH and CPLEX. Some of the numerical parameters use values 0 or -1 to encode special behaviour, for example, for the automatic setting of the parameter value or the deactivation of the mechanism controlled by the parameter. In cases where the documentation was unclear, we erred on the side of caution and removed a parameter value or treated the special value as a categorical parameter.

We also performed an analysis of a limited number of two-dimensional slices of the landscapes. In particular, we looked at data from one scenario for each of the three types of problems: EAX on the TSP RUE instances, CaDiCaL on the circuit-fuzz SAT instances and CPLEX on the Regions200 MIP instances. Since fully evaluating all possible two-dimensional slices would have been prohibitively expensive, we further limited our analysis to the cross product of only those parameters which were classified as interesting by our heuristic criterion (see Chapter 3.2.5).

In order to assess to what extent the categorical parameters interact with the numeric parameters (and each other) we first performed the same method of analysis to determine the subset of categorical parameters that were also interesting according to our criterion. We summarize the number of numeric and categorical parameters deemed interesting in each of the scenarios in Table 6.3. In total, this resulted in 17 two-dimensional parameter response slices between numerical parameters, 64 slices between one numerical and one categorical parameter and 120 slices between two categorical parameters.

For the CPLEX scenario, we further reduced the computational cost of collecting the data by decreasing the running time cutoff from 10 000 CPU seconds to

Table 6.3: The number of numeric and categorical parameters deemed interesting by our heuristic criterion (see Chapter 3.2.5). These are the parameters that we used for the analysis of the two-dimensional slices of the algorithm configuration landscapes.

Problem	Algorithm	Instance Set	# Num	# Cat
SAT	CaDiCaL	circuit-fuzz	5	0
MIP	CPLEX	CLS	4	16
TSP	EAX	TSP-RUE-1000-3000	2	0

Table 6.4: PAR10 values on the test sets for the default configuration versus the configuration with the best training PAR10. All times are in CPU seconds.

Problem	Algorithm	Instance Set	Default PAR10	Configured PAR10	Speedup
SAT	CaDiCaL	circuit-fuzz	468.71	252.34	1.86
		BMC08	638.57	637.93	1.00
	lingeling	circuit-fuzz	382.23	279.29	1.37
		BMC08	692.80	691.51	1.00
	cryptominisat	circuit-fuzz	444.68	276.83	1.61
		BMC08	938.61	970.07	0.97
MIP	CPLEX	CLS	40.39	3.39	11.91
		Regions200	106.77	6.40	16.68
TSP	EAX	TSP-RUE-1000-3000	65.99	56.84	1.16
	LKH	TSP-RUE-1000-3000	428.60	228.62	1.87

1 200 CPU seconds, since we observed that a relatively small number of the configurations in the original analysis required a disproportionately large amount of the computational budget to be fully evaluated. However, even with these modifications it still took 58.9 CPU years to collect all of the data for the two-dimensional parameter response slices.

In Table 6.4, we show the results from configuring our 10 scenarios, using 25 runs of SMAC [89] per scenario. These results are consistent with the literature. We note that in some cases, configuration did not result in significant performance improvements over the default parameter settings of a given algorithm; this is unproblematic, since our goal in performing automated configuration was not to obtain improved performance, but rather to ensure we used high-performance configurations as reference points for the parameter response slices that formed the basis for our algorithm configuration landscape analysis.

Table 6.5: Summary of the shape analysis results for the one-dimensional parameter response slices.

Parameters	# Slices	Uni-M	Cvx	FDC
All	193	99.5%	99.5%	0.20
Interesting	18	94.4%	94.4%	0.72

We ran all of our experiments on Ada, a cluster of 20 nodes, equipped with 32 2.10 GHz Intel Xeon E5-2683 v4 CPUs with 40 960 KB cache and 96 GB RAM each, running openSUSE Leap 42.1 (x86_64). To minimize detrimental cache effects and memory contention, in all experiments, we used a single core per CPU and limited RAM use to 3 GB. In total, we used 102.4 CPU years for automated configuration and collection of parameter response slice data.

6.2 One-Dimensional Parameter Response Slices

In total, we collected and analyzed 193 one-dimensional parameter response slices as described in Chapter 6.1, the results for which are summarized in Table 6.5. Overall, similar to in Chapter 4, the parameter response slices appear to be more benign than one might expect. Our tests for uni-modality and convexity failed to reject their null hypotheses for all but one of the parameter response slices. That is, 99.48% of the slices appear to be both uni-modal and convex.

Somewhat surprisingly, our heuristic method outlined in Chapter 3.2.5 identified only 18 of the slices as interesting (compared to 44 out of the 45 studied in Chapter 4). In Figure 6.1, we show four parameter response slices that are representative of the qualities we observed in this set of 18 (the remaining 14 are available in Appendix A.3.1). To our surprise, neither lingeling nor cryptominisat had any interesting parameter response slices. The parameter that fANOVA rated to be the most important for lingeling [58] shows a slight dip at the smallest parameter value in the slice for the circuit-fuzz instance set. To investigate further, we evaluated an additional 15 parameter values, but still found it to be un-interesting according to our criterion.

The only parameter we found that had a non-unimodal and non-convex response was LKH's `BACKBONE_TRIALS` parameter (see Figure 6.1), which spec-

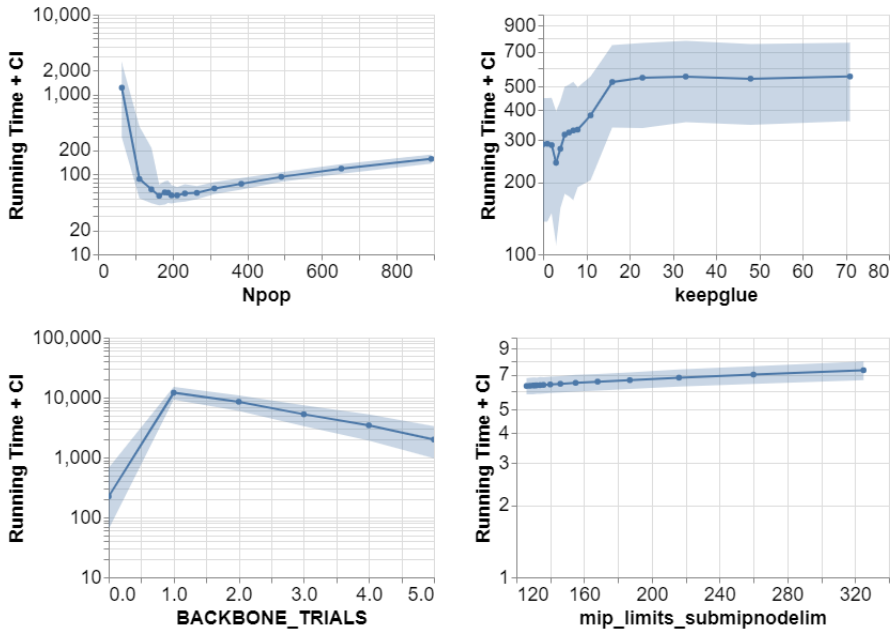

Figure 6.1: Four examples of one-dimensional parameter response slices. From left to right and top to bottom: EAX's `Npop` on the TSP-RUE-1000-3000 instance set, CaDiCaL's `keepglue` on the circuit-fuzz instance set, LKH's `BACKBONE_TRIALS` on the TSP-RUE-1000-3000 instance set and CPLEX's `mip_limits_submipnodelim` on the Regions200 instance set.

ifies the number of backbone trials in each run. Apart from `BACKBONE_TRIALS`=0, even this response slice appears to be both convex and uni-modal. To the best of our knowledge, a value of 0 does not have special meaning, apart from the obvious semantic difference of *some* versus *no* backbone trials, which may alone account for this difference, since it likely corresponds to a (poorly performing) heuristic component of the algorithm that is turned on or off.

Some of the parameter responses (for example, `keepglue` in Figure 6.1), appear to be flat for poorly performing parameter values, and hence non-convex overall. However, our tests were unable to reject convexity, despite this visual evidence, because of the relatively wide bootstrap confidence intervals. Nevertheless, we believe that these flat regions are artifacts of how PAR10 scores treat censored

runs and hence a sufficiently large running time cutoff may continue to yield convex responses.

In the three SAT scenarios involving the BMC08 instance set, we found only one parameter response slice considered interesting according to our criterion: CaDiCaL's `restartmargin`. This is consistent with the fact that SMAC was unable to achieve significant performance improvements for these scenarios. We further note that the default value for `restartmargin` is very near to the best known value. Hence, it appears that better configurations may not exist rather than that they are hard to find due to highly irregular or rugged landscapes.

These results paint a very different picture of the difficulty of the landscapes than our analysis using FDC would suggest. The median FDC value for the 193 parameter response slices is 0.20 – in fact 75% of the response slices have FDC values less than 0.35. These FDC values suggest that most of these parameters should be challenging to configure. However, if we look only at the 18 parameters with interesting response slices, then we see the median increases to 0.72 and 75% of these slices have FDC values less than 0.88. These results indicate that many of the low FDC scores are in fact due to parameters with mostly-flat responses. Given this result, we conclude that FDC is a less useful metric for characterizing the difficulty of algorithm configuration landscapes for running time minimization.

For a more detailed discussion of the FDC analysis as well as an analysis of the algorithm configuration landscapes that arise when optimizing algorithm performance on individual instances (instead of on sets of instances as done here), see Appendices A.3.2 and A.3.3.

6.3 Two-Dimensional Parameter Response Slices

Overall, none of the 201 two-dimensional parameter response slices appeared to be multi-modal according to our test (see Chapter 3.2.1). However, only 19.40% of these tests were considered sensitive by our validation procedure (see Chapter 3.2.4), so it is possible that some of these parameter response slices could appear multi-modal if analyzed with a larger computational budget. Of the 17 parameter response slices with two numerical parameters, our test for convexity only rejected its null hypothesis for one of them – that is, 16 of the 17 parameter response slices

appear to be convex. Since the test for convexity imposes a stronger requirement on its null hypothesis, all of these tests were deemed sensitive by our validation procedure. The only response slice for which convexity was rejected contained the population size, `Npop`, and number of children, `Nch`, used by EAX (see the top left pane of Figure 6.2). However, even in this case only 6.12% of the landscape was interior to the convex hull of the upper bounds, which indicates that only a small portion of the landscape would need to be modified to make it indistinguishable from a convex function.

Of the two-dimensional slices with two numerical parameters, EAX's two parameters also had the lowest FDC, at 0.11. As discussed previously, low FDC values are typically believed to represent landscapes that are challenging to optimize. However, from inspection of the top left pane of Figure 6.2, it can be seen that the FDC is low in this landscape because configurations with smaller-than-optimal values for `Npop` yield worse performance much more quickly than configurations with larger-than-optimal values. In comparison, most of the other numerical parameter response slices had relatively large FDC values, as seen in Table 6.6.

From Table 6.6 we can also see that the FDC for the parameter response slices with at least one categorical parameter tend to be quite a bit lower than those with only numerical parameters; however, this observation should come as no surprise, since each categorical value for a particular parameter is considered equidistant from each other. This result also suggests that categorical parameters typically contain less exploitable structure than their numerical counterparts.

In the summary of the results for the parameter interactions (see Table 6.7), we again see that the parameter response slice from EAX is an outlier compared to the rest of the slices with two numerical parameters. Apart from EAX, all but three of the parameter response slices have very few significant interactions among their numerical parameters. The rest of the numerical parameters interact in no more than 11.11% of their landscapes. However, EAX has interactions among its two parameters that span nearly the entire landscape, that is, 97.22% of it. In Figure 6.2, we can see that the interactions are compensatory in nature; we surmise that this is because both the number of children, `Nch`, and the population size, `Npop`, influence the degree of exploration performed within each iteration of EAX.

The percentage of the two-dimensional parameter response slices that have

Table 6.6: The uni-modality and convexity test results and the fitness distance correlations for the two-dimensional parameter response slices obtained for the parameters in Table 6.3. Includes the mean percentage of the landscapes that were unreachable (UnR) from the global optima and interior (Int) to the convex hull of the upper-bounds for those slices for which uni-modality or convexity could be rejected, respectively; the mean percentage of the lower-bounds that were co-planar (Co-P) to the convex hull of the upper bounds; and the mean percentage of the landscapes for which the tests appeared sensitive (Sen) according to our permutation-based analysis.

Type	# Slices	Algorithm	UnR	Sen	Int	Co-P	Pow	FDC
N×N	1	EAX	0.00	100.00	6.12	0.00	100.00	-0.21
	10	CaDiCaL	0.00	0.00	0.00	0.00	40.00	0.70
	6	CPLEX	0.00	0.00	0.00	0.00	0.00	0.60
N×C	64	CPLEX	0.00	32.81	–	–	–	0.16
C×C	120	CPLEX	0.00	14.17	–	–	–	0.31

Table 6.7: The mean percentage of locally significant parameter interactions based on the analysis of the partial derivatives (Sig ∂^2), the 2nd order fANOVA scores (fANOVA), and the mean probability of obtaining a configuration that is tied with optimal if each parameter is configured independently, a single time and in a random order (Tied $\overline{\text{w}}$ Opt). All results are for the two-dimensional parameter response slices obtained for the parameters in Table 6.3.

Type	# Slices	Algorithm	Sig ∂^2	fANOVA	Tied $\overline{\text{w}}$ Opt
N×N	1	EAX	97.22%	42.61%	50.00%
	10	CaDiCaL	2.78%	7.36%	100.00%
	6	CPLEX	0.00%	16.39%	100.00%
N×C	64	CPLEX	4.98%	0.68%	100.00%
C×C	120	CPLEX	31.82%	3.28%	97.92%

locally significant interactions tends to be larger between categorical parameters than numerical parameters. Nevertheless, the median numbers for these are still rather small (see Table 6.7), indicating that even the interactions among categorical parameters appear to be relatively infrequent.

The fANOVA results (also summarized in Table 6.7) are consistent with those from the locally significant partial derivatives; however, they provide a different lens to the analysis. Typically, fANOVA scores are calculated for the full landscapes; however, ours are computed using only two-dimensional parameter response slices rather than complete landscapes, hence they need to be interpreted in a slightly different way. Effectively, the 2nd order fANOVA score presented here measures the percentage of the variance in the two-dimensional slice of the landscape that is explained by the interactions of the two parameters compared to each of the parameters independently. That is, the sum of all of the fANOVA scores for all of the 190 CPLEX parameter response slices will sum to 19 000%, instead of 100%. With this in mind, we again see the EAX landscape has much more dependence upon the interaction of its parameters, since only 57.39% of its response can be explained by the two parameters independently. Similarly, we see that the interactions among many of the categorical parameters are surprisingly small.

Finally, to put our hypotheses regarding the relative simplicity of the interactions of most parameters to the test, we used our simplistic configuration procedure from Chapter 3.3.3 to determine to what extent the parameters can be configured independently. We show these results in the final column of Table 6.7.

Similar to in Chapter 4, we observe the surprising result that the configuration procedure found configurations tied with optimal in nearly all cases, regardless of the order in which the parameters were configured. EAX's two parameters were, once again, the only exception among the parameter response slices with only numerical parameters. However, in many cases this is no doubt due to the large variability between the performance of a given configuration on different problem instances. For example, in the top right of Figure 6.2, we can see that the order in which the parameters are configured would yield different median running times; however, these differences are insignificant when taking into account the size of the confidence intervals. Nevertheless, the bottom two response slices in Figure 6.2 – for both of which it is quite clear that the order of configuration would be unimpor-

tant – are representative of a surprising fraction of the two-dimensional parameter response slices. In fact, the bottom right pane of Figure 6.2 is representative of all six of CPLEX's parameter response slices with two numerical parameters.

Every parameter response slice had at least one order for the configuration of the parameters that yielded a configuration tied with optimal; however, there were five cases among CPLEX's parameter responses with two categorical parameters for which no more than one of the two orders yielded a configuration that was tied with optimal. This seems quite surprising; we would expect that in many cases a change to a categorical parameter could yield a completely different response in the remaining parameters. However, nearly all of CPLEX's parameter response slices with one numerical parameter and one categorical parameter were qualitatively similar to the right pane of Figure 6.3, wherein the change to the categorical parameter `mip_strategy_subalgorithm` yielded no observable effect on the response in the numerical `mip_limits_submipnodelim` parameter, other than to translate it up or down. One of only four exceptions to this trend is shown in the left pane of Figure 6.3, for which qualitatively different responses to the values of `simplex_refactor` can be seen for different values of `mip_strategy_subalgorithm`. The `simplex_refactor` parameter determines the number of iterations that must pass before the MIP instance's basis matrix is refactored, whereas `mip_strategy_subalgorithm` determines which optimization algorithm is used to solve sub-problems in a MIP instance. These results suggest that the optimal simplex refactoring frequency must be different for different optimization algorithms, which may make more or less progress within each iteration.

6.4 Chapter Summary

In this chapter, we applied all of the methods of analysis from Chapter 3 to running time minimization landscapes from several prominent $\mathcal{NP}$-hard and $\mathcal{NP}$-complete AClib [94] scenarios spanning SAT, TSP and MIP. Due to prohibitively expensive data collection times, we could not perform an analysis of the full algorithm configuration landscapes. Instead, we focused primarily on one-dimensional parameter response slices with some limited analysis of the effects of parameter interactions on two-dimensional parameter response slices.

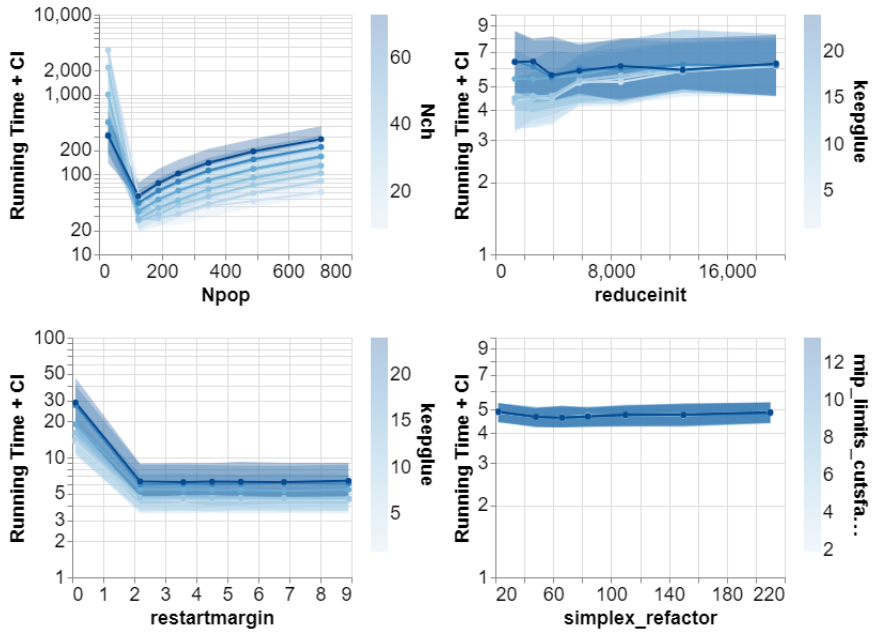

Figure 6.2: Four examples of two-dimensional parameter response slices that include two numerical parameters. From top to bottom and left to right: EAX's `Npop` and `Nch` on the TSP-RUE-1000-3000 instance set; CaDiCaL's `keepglue` and `reduceinit` on the circuit fuzz instance set; CaDiCaL's `keepglue` and `restartmargin` on the circuit fuzz instance set; and, CPLEX's `mip_limits_cutsfactor` and `simplex_refactor` on the capacitated lot sizing (CLS) instance set.

Overall, the running time minimization landscapes we observed have several characteristics similar to those of AutoML loss landscapes (see Chapter 4). For example, we showed that FDC can often be misleading, since a landscape that is completely flat, or one that is uni-modal but with very different penalties for being too large or too small, yield low FDC despite being very easy to optimize. Both of these cases appear to occur relatively frequently among algorithm configuration landscapes.

Similar to in Chapter 4, we observed that most of the individual parameter response slices appear to yield uni-modal and – surprisingly – often even convex responses. The shapes of the landscapes that we observed in our limited analysis of

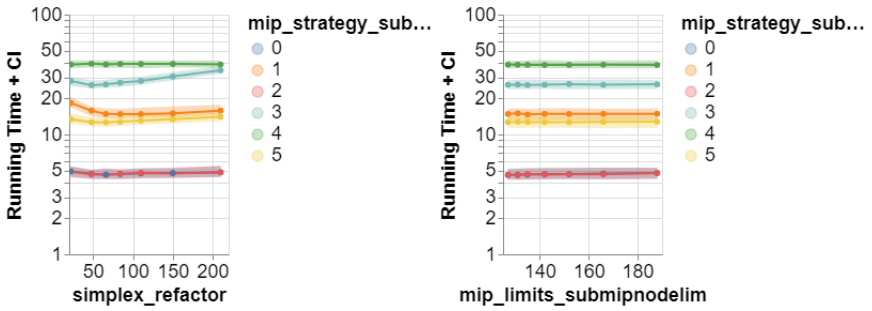

Figure 6.3: Two examples of two-dimensional parameter response slices that include both one numerical and one categorical parameter. Left: CPLEX's `mip_strategy_subalgorithm` and `simplex_refactor` parameters on the capacitated lot sizing (CLS) instance set; right: CPLEX's `mip_strategy_subalgorithm` and `mip_limits_submipnodelim` parameters on the capacitated lot sizing (CLS) instance set.

two-dimensional parameter response slices were qualitatively similar to those for one-dimensional slices; however, many applications of the tests for uni-modality lacked sensitivity when applied to the two-dimensional parameter response slices. As a result, collecting more data on larger problem instance sets may be necessary to detect deviations from such structure, if they are present. To save on computation costs, this could be done in an adaptive way, where additional runs are only performed in high-variance regions of the landscapes. Nevertheless, the confidence intervals of the parameter response slices still provide upper bounds on the size of any barriers that can exist in the landscapes. In most applications, algorithm configuration procedures do not evaluate configurations as extensively as we do in our analysis. Therefore, it is unlikely that such deviations from uni-modality would be detectable by an algorithm configurator.

Overall, the parameter interactions were surprisingly benign for all but one outlying case: EAX's population size, `Npop`, and number of children, `Nch`. These two parameters yielded a basin of high-quality configurations that is slightly diagonally oriented relative to the axes of the parameters. It also contains a relatively steep penalty when either parameter value is set too small. Otherwise the numerical parameters had significant interactions that covered less than 3% of their response

slices. In fact, even the response slices with two categorical parameters showed significant interactions among less than 32% of their responses.

Furthermore, configuring each parameter independently, a single time and in a random sequence, produced configurations that are tied with optimal for all but six of the 201 two-dimensional parameter response slices we studied. While in many cases, this may be due to the large size of the confidence intervals, it nevertheless suggests that even relatively simple methods may obtain satisfactory performance for many similar scenarios.

Compared to AutoML loss landscapes, the biggest difference we observed in these landscapes is that the variance in the performance estimates is orders of magnitude larger. In particular, in this chapter we performed 10 independent runs of each algorithm configuration on each problem instance. We then recorded the performance of a configuration as the median over per-instance median running times. Each scenario had between 50 to 1 000 instances. In contrast, for the AutoML scenarios we typically studied means, a less robust statistic, over one to five independent runs of the machine learning method. Despite this, we still observed that the size of the confidence intervals were almost always substantially larger in the running time minimization scenarios relative to the AutoML scenarios when looking at the one- and two-dimensional (hyper-)parameter response slices.

However, the difference in the variability of the performance in the two applications makes sense. In the AutoML scenarios, each problem instance corresponds to a random training and validation split of a given data set. For most data sets, this will likely result in problem instances that are relatively similar. In contrast, for the running time minimization scenarios the diversity in the problem instances is likely much larger. For example, consider the bounded model checking SAT instance sets, for which each problem instance corresponds to unrolling different (possibly unrelated) hardware circuits to different depths, thereby creating a much more heterogeneous collection of problem instances. Furthermore, running time is an inherently noisy metric, since even repeated calls to a deterministic algorithm for solving the same problem instance will most often result in different running times due to concurrently-running background processes on the machine. Conversely, some of the performance metrics used in the AutoML loss landscapes such as accuracy are bounded to the interval [0, 1], which further restricts the maximum size

of the confidence intervals.

Nevertheless, despite this performance variability difference, we still see overall that algorithm configuration landscapes for running time minimization appear to be relatively benign. This stands in contrast with the most popular algorithm configuration procedures, which typically employ powerful diversification mechanisms. Indeed, our results suggest that the main source of difficulty for the configuration of these scenarios is not that they are highly multi-modal, but rather that the high variance in the performance measurements makes them appear highly rugged to configuration procedures that do not properly account for their stochasticity.

Chapter 7

Exploitation II: Running Time Minimization Landscapes

As motivated in the introduction, automated algorithm configuration procedures such as SMAC, GGA++ and `irace` can often find parameter configurations that substantially improve the performance of state-of-the-art algorithms for difficult problems – for example, a three-fold speedup in the running time required by EAX, a genetic algorithm, to find optimal solutions to a set of widely studied TSP instances. However, it is usually recommended to provide these methods with running time budgets of one or two days of wall-clock time as well as dozens of CPU cores. Most general-purpose algorithm configuration methods are based on powerful meta-heuristics that are designed for challenging and complex search landscapes; however, in Chapter 6 we showed that many scenarios for running time minimization appear to have algorithm configuration landscapes with a relatively simple structure. That is, most parameters typically yield responses in algorithm performance that are uni-modal. Furthermore, many of the parameters do not interact strongly, and even when they do, they frequently do not substantially increase the complexity of the configuration scenarios.

In light of these results, we introduce in this chapter[1] the golden parameter

[1]This chapter is based on joint work with Holger Hoos, which was published at GECCO 2020 [149], where it received the ECOM track best paper award. One award is given out each year at GECCO to each of nine tracks, by a vote among the attendees. An early prototype of GPS was

search (GPS) configuration procedure, which is designed to exploit this structure. In contrast to our prototype methods in Chapter 5, which made assumptions that were too strong to exploit similar structure in AutoML loss landscapes, we show that GPS is competitive with state-of-the-art baselines. In particular, it often finds similar or better parameter configurations using a fraction of the computing time budget across a broad range of scenarios spanning TSP, SAT and MIP. We believe there are four reasons behind this success when compared with our previous attempts in Chapter 5.

First, in Chapter 5, we introduced two separate configurators, each of which were designed to exploit a single structural property (uni-modality and benign parameter interactions) that we observed in AutoML loss landscapes. In GPS, we exploit both of these simultaneously. We exploit the simplicity of the parameter interactions by configuring each parameter of the target algorithm semi-independently in parallel using a coordinate-descent-based [192] approach. To exploit the uni-modality of the parameter responses, the search procedure for each parameter employs a variant of the golden section search algorithm [102], which obtains optimal worst-case performance for one-dimensional uni-modal functions.

We believe that the second reason GPS performs well is also due to the golden section search algorithm. At each iteration, golden section search picks a new configuration to evaluate that guarantees that it eliminates the largest possible fraction of the search space in the worst case. In many line search applications, it is instead frequently recommended to use quadratic or cubic interpolation to pick the next candidate to evaluate – for example, the well-known Brent's method [34], which combines golden section search with interpolation. While it is possible that interpolation could improve the performance of GPS, based on our observations in Chapter 5, we do not expect this to be the case. In particular, given that in Chapter 6 we observed that running time minimization landscapes tend to be substantially noisier than the AutoML loss landscapes, we suspect that most attempts to interpolate would only result in worse performance due to over-fitting to random noise.

Similarly, we believe that the use of a coordinate-descent-based [192] algo-

submitted as part of a course project for CPSC 521: Parallel Algorithms and Architectures, taught by Mark Greenstreet.

rithm is responsible for the third reason behind GPS's strong performance. In Chapter 5, when we used a spline model to exploit the weak parameter interactions, the method failed to perform well in scenarios where there were strong interactions, even though these scenarios were considered easy by our simplistic parameter configuration procedure. In particular, we refer to the diagonally-oriented basin structure (for example, $f(x,y) = |x-y|$), that appears to be relatively common among parameters that interact strongly in various algorithm configuration scenarios. For example, see a) our discussion of the momentum and learning rate hyper-parameters in Chapter 5.3; b) EAX's population size and number of children parameters in Chapter 6.3; c) the population size and the number of fitness evaluations of differential evolution [140]; or d) the acceleration and constriction factors of particle swarm optimization [196].

And finally, fourth, to the best of our knowledge, there are no simple, universal methods for obtaining low-fidelity approximations of an algorithm configuration's performance for $\mathcal{NP}$-hard and $\mathcal{NP}$-complete problems. Therefore, reducing the number of configurations that need to be evaluated by a modest percentage will yield a substantially larger speedup to the configuration procedure than for AutoML scenarios.

7.1 Golden Parameter Search (GPS)

GPS conducts an efficient search process on each parameter semi-independently. It begins with a bracket B_p for each parameter $p \in P$, where a bracket corresponds to a set of parameter values believed to contain the optimum parameter value. Each bracket is evaluated in parallel and shrunk around the optimum value. A bracket may be expanded if there is evidence that it no longer contains the optimum (due to, for example, parameter interactions). For numerical parameters, this search procedure is based on the golden section search algorithm [102], which has the optimal worst-case bound for one dimensional, uni-modal functions [145]. To save on computational resources, GPS uses a racing procedure based on a permutation test. Once a better value for a parameter is found, the search procedure for each of the other parameters is updated to use this value. Since parameter interactions may cause old target algorithm runs to become stale, these old runs are slowly forgotten

(down-weighted), and then eventually re-run. When GPS terminates, the incumbents for each parameter value are returned as the final incumbent configuration.

Many algorithms contain conditional parameters, whose values are only used when their "parent" parameters are set to certain values. GPS configures the values of all such child parameters in parallel by appropriately modifying their ancestors' parameter values to enable the child parameters when they are being evaluated. This avoids incorrectly discarding parent parameter values because their children were sub-optimally configured.

To avoid wasting time evaluating poor configurations on a large number of instances, GPS uses an intensification mechanism that determines how many runs are performed for each value, and slowly introduces more instances into the set $I' \subseteq I$ used to evaluate parameter values. A crucial component of many algorithm configuration procedures are capping mechanisms [35, 87], which limit the running time cutoffs for each target algorithm run. GPS employs a novel capping mechanism that avoids being overly aggressive.

The primary bottleneck in algorithm configuration is running many configurations on sets of problem instances; therefore, GPS uses a master-worker process design, whereby the master process loops through each parameter $p \in P$ to check for newly completed target algorithm runs, updates the bracket B_p and incumbent value $c^*[p]$, and then queues new runs. We show high-level pseudo-code for the master process in Algorithm 3. The worker processes repeatedly query a queue in a database to obtain new runs to perform. GPS queues an exponentially increasing number of instances on which each parameter will be evaluated each time new instances are queued for a given parameter. For some highly-parameterized algorithms, only a small fraction of parameters affect the performance of the algorithm (for example, see Falkner et al. [58] or Chapter 6). GPS therefore uses a multi-armed bandit procedure to prioritize parameters believed to be important.

For some scenarios, GPS may be unable to keep the target algorithm run queue sufficiently populated. Hence, GPS dynamically adjusts an instance increment parameter, *instIncr*, such that the intensification and queuing mechanisms operate on batches of *instIncr* instances (target algorithm runs).

Algorithm 3 The main algorithm for Golden Parameter Search.

input
    A, the algorithm to be configured
    I, the training instance set
    m, the metric with respect to which A is configured
    P, the set of configurable parameters of A
    C, the configuration space (ranges and constraints)
    c_0, the default configuration
    numInitInst, an integer in $[1, \infty)$ (default: 1)
    decayRate, a real number in [0,1] (default: 0.2)
    α, a real number in $(0, 1)$ (default: 0.05)
    instIncr, a positive Fibonacci number (default: 1).
output
    c^*, the best configuration found so far
procedure GPS
    # Initialization
    Initialize the incumbent $c^* := c_0$
    for each parameter $p \in P$, **do**
        Initialize a bracket B_p
        Initialize I_p with *numInitInst* random instances
        Queue a run for the default value $c_0[p]$
    Initialize empty arrays R and Q
    # Main Procedure
    while budget not exhausted, **do**
        # See Chapter 7.1.9 for the bandit queue
        Sample a parameter $p \in P$ using the bandit queue
        # R and Q help make use of all workers (see Chapter 7.1.10)
        Append values to R and Q
        if sufficient time has elapsed, **then**
            Update *instIncr*, if needed
            Re-initialize R and Q as empty arrays
        # Compare performances (see Chapter 7.1.1)
        for each pair $v_1, v_2 \in B_p$, **do**
            Perform permutation test with significance level α
        # Incumbent update (see Chapter 7.1.6)
        if there exists v such that $m(c^*|_{p=v}) \prec_\alpha m(c^*)$, **then**
            Update the incumbent $c^*[p] := v$
        # Bracket Update (see Chapter 7.1.2)
        if there is sufficient evidence for improvement, **then**
            Expand/shrink the bracket, B_p
            # Intensification (see Chapter 7.1.8)
            Add *instIncr* random instances to I_p
        else if each $v \in B_p$ has been run on each $i \in I_p$, **then**
            Add *instIncr* random instances to I_p
        Append values to R and Q
        # Offload tasks to the workers (see Chapter 7.1.8)
        Queue new target algorithm runs as needed
    return c^*

7.1.1 Test for Significance (Permutation Test)

Algorithm performance measures are typically noisy, randomized objective functions [88]. Therefore, a core component of GPS is a permutation test, which is used to determine, in a distribution-free way, whether or not there is sufficient evidence at a given significance level α to conclude that one parameter value is worse than another. Let $m(c^*|_{p=v_1}) \prec_\alpha m(c^*|_{p=v_2})$ denote a significant difference, let $m(c^*|_{p=v_1}) \approx_\alpha m(c^*|_{p=v_2})$ denote a statistical tie, and let $m(c^*|_{p=v_1}) \preceq_\alpha m(c^*|_{p=v_2})$ denote that $m(c^*|_{p=v_1}) \prec_\alpha m(c^*|_{p=v_2})$ or $m(c^*|_{p=v_1}) \approx_\alpha m(c^*|_{p=v_2})$. Note that $m(c^*|_{p=v})$ is an estimate of the performance on a subset of I. If the intersection of completed runs (technically: run equivalents – see Chapter 7.1.4) for $m(c^*|_{p=v_1})$ and $m(c^*|_{p=v_2})$ is less than *numInitInst*, we skip the test and record $m(c^*|_{p=v_1}) \approx_\alpha m(c^*|_{p=v_2})$.

7.1.2 Expanding/Shrinking the Bracket

For a parameter $p \in P$, the bracket B_p is simply a set of values that are believed to contain or bound the optimal value for p. Different rules for how to expand and shrink the bracket are used for real-valued, integer-valued and categorical parameters.

Real-Valued Parameters For real-valued parameters, the bracket updating procedure is inspired by the *golden section search algorithm* [102], which provides optimal worst-case performance for one-dimensional, uni-modal functions (an assumption which we assume is true for most algorithms' parameters – see Chapter 6). The bracket contains two end points v_a and v_b (believed to bound the optimal value) and two interior points v_c and v_d, such that $v_a < v_c < v_d < v_b$ and $\frac{v_b - v_a}{v_d - v_a} = \frac{v_b - v_a}{v_b - v_c} = \frac{\sqrt{5}+1}{2} \equiv \phi$ (the golden ratio), which implies $\frac{v_d - v_a}{v_c - v_a} = \frac{v_b - v_c}{v_b - v_d} = \phi$ as well. Both an expand and shrink operation corresponds to replacing one parameter value in the set $\{v_a, v_b, v_c, v_d\}$ with a new value. For example, we expand the bracket in the direction of v_a if we observe $m(c^*|_{p=v_a}) \prec_\alpha m(c^*|_{p=v_c}) \preceq_\alpha m(c^*|_{p=v_d}) \preceq_\alpha m(c^*|_{p=v_b})$, where an expand operation corresponds to updating $v_d := v_c$ and $v_c := v_a$ followed by $v_a := v_c \cdot (\phi + 1) - v_d \cdot \phi$. The value v_b remains unchanged. Similarly, we shrink the bracket around v_c if we observe $m(c^*|_{p=v_a}) \succeq_\alpha m(c^*|_{p=v_c}) \prec_\alpha m(c^*|_{p=v_d}) \preceq_\alpha$

$m(c^*|_{p=v_b})$, where a shrink operation corresponds to updating $v_b := v_d$ and $v_d := v_c$ followed by $v_c := v_b - \frac{v_b - v_a}{\phi}$. Note that GPS does not require that parameter values stay within the maximum and minimum values specified in the parameter configuration space file. While this may lead GPS to evaluate configurations that cause the algorithm to crash, it also allows GPS to recover from situations where the user chose a poor maximum or minimum value.

Integer-Valued Parameters We use the same procedure as for real-valued parameters; however, parameter values are rounded to the nearest integer value not already contained in the bracket. Technically, this is similar to the *Fibonacci search algorithm* [102] with a custom bracket initialization strategy (see Chapter 7.1.3).

Categorical Parameters The bracket B_p may contain any subset of the parameter values available for p. When a bracket is shrunk, a value is simply removed from the set. We perform this update as soon as we have seen sufficient evidence that one value $v_{bad} \in B_p$ performs worse than the current incumbent (that is, $m(c^*) \prec_\alpha m(c^*|_{p=v_{bad}})$). Parameter values may be re-added to the bracket if we lose confidence in the target algorithm runs upon which we based this decision due to parameter interactions (see Chapter 7.1.4).

7.1.3 Bracket Initialization

Real-Valued Parameters Existing algorithm configurators [7, 35, 87, 89] require that a default, maximum and minimum value are provided. Our bracket initialization procedure guarantees that at least one of v_a, v_b, v_c and v_d are the default value, while making the bracket as large as possible, without exceeding the maximum and minimum range. We further require that the golden ratio properties between v_a, v_b, v_c and v_d are satisfied (see Chapter 7.1.2).

Integer-Valued Parameters These brackets are initialized using the same procedure as for real-valued parameters, with rounding.

Categorical Parameters All parameter values are initially added to the bracket, that is $B_p = V_p$.

7.1.4 Parameter Interactions (Decaying Memory)

Each time one of the concurrent search processes updates its incumbent parameter value, the algorithm now contains stale performance measurements for the other parameters. Instead of completely trusting or discarding this information, GPS uses a heuristic to slowly forget stale information. For a target algorithm run performed on instance i with configuration c_i, we compute a weight, $w_i \in [0,1]$, which encodes the amount of trust we have in the corresponding performance measurement. Let c^* be the current incumbent configuration and let c_i be the configuration that was the incumbent when the target algorithm run on i was performed. Then,

$$w_i = decayRate^{\Delta(c_i,c^*)}, \tag{7.1}$$

where $\Delta(c_i,c^*)$ denotes the difference between the previous incumbent c_i and the current incumbent c^*, and $decayRate \in [0,1]$ is a parameter that controls how quickly information is forgotten. Specifically,

$$\Delta(c_i,c^*) = \sqrt{\sum_{p \in P_i \cap P^*} \delta(c_i[p],c^*[p])^2}, \tag{7.2}$$

where P_i and P^* denote the set of active parameters (see Chapter 7.1.5) in c_i and c^*, respectively and where $\delta(c_i[p],c^*[p])$ measures the difference in parameter values $c_i[p]$ and $c^*[p]$. For numerical parameters,

$$\delta(c_i[p],c^*[p]) = \frac{|c_i[p] - c^*[p]|}{|v_{\max} - v_{\min}|}, \tag{7.3}$$

where $v_{\max}$ and $v_{\min}$ correspond to the maximum and minimum values for parameter p, respectively. For categorical parameters,

$$\delta(c_i[p],c^*[p]) = \begin{cases} 0 & \text{if } c_i[p] = c^*[p], \\ 1 & \text{otherwise.} \end{cases} \tag{7.4}$$

When performing permutation tests, we multiply the weights w_i and w'_i to combine them for the target algorithm runs that approximately measure $m_{\{i\}}(c^*|_{p=v})$ and $m_{\{i\}}(c^*|_{p=v'})$; this intuitively corresponds to adding the uncertainties associated with the respective performance estimates.

Throughout the following, we refer to the number of *run equivalents* performed for a particular parameter value, which is simply the sum of the weights for the target algorithm runs performed for that parameter value. Intuitively, this approximates the total number of reliable target algorithm runs for that parameter value.

7.1.5 Evaluating Conditional Parameters

Some parameters, known as conditional (or child) parameters, are only active if their parent parameter is set to a certain value or set of values. With GPS, we only allow parameters that have a single parent value that activates them (this only affected one of the algorithms we studied, see Chapter 7.2). In practice, if there are multiple parent values that activate a child, this can be encoded by creating multiple copies of the child parameter, one for each parent value. To configure a conditional parameter p_c, with parent (or ancestor) p_a, the value of p_a must be set to a specific value v_{on}. If $c^*[p_a] \neq v_{\text{on}}$, then GPS will temporarily set $c^*[p_a]$ to v_{on} when evaluating values of p_c, that is, instead of evaluating $m(c^*|_{p_c=v_c})$ it will evaluate $m(c^*|_{p_c=v_c,\ p_a=v_{\text{on}}})$. If p_a also has one or more ancestors, then each of these are checked and set such that p_a is active as well. An improvement to $c^*[p_c]$ will not immediately improve $m(c^*)$ when $c^*[p_a] \neq v_{\text{on}}$, because the parameter value will still be ignored. However, it will decrease the confidence in the old target algorithm runs for $m(c^*|_{p_a=v_{\text{on}}})$ (see Chapter 7.1.4), which may eventually lead to setting $c^*[p_a] := v_{\text{on}}$.

7.1.6 Updating the Incumbent

GPS uses a conservative incumbent updating procedure, whereby an incumbent parameter value is only updated if it is believed it *improves* the performance of the incumbent configuration. This helps avoid over-fitting, losing confidence in old target algorithm runs (see Chapter 7.1.4) and breaking things (on unseen instances) that do not need fixing (on seen instances). GPS picks the incumbent for each

parameter using a 7-step process. Steps 1–2 make sure that each candidate has been run on a sufficient instance set and 3 checks to see if any challengers are better than the incumbent. Steps 4–7 are a series of tie-breakers. The process begins with all of the values in the bracket as candidates, and in each, a filter is used to reject some of the candidates. Unless otherwise specified, if no candidate remains at the end of a round, the previous incumbent is returned, and if only one candidate remains, it becomes the new incumbent. Specifically, the steps are as follows:

1. Admit candidates v with $\geq$ *numInitInst* run equivalents.
2. Admit candidates v that have been run on a (non-strict) super-set of the instances upon which the last incumbent was run when it was chosen as the incumbent.
3. Admit candidates v with statistically sufficient evidence of improved performance compared to the previous incumbent, according to a permutation test, that is, $m(c^*|_{p=v}) \prec_\alpha m(c^*)$.
4. Admit candidates v that are not worse than any of the other candidates according to the permutation test. If every candidate is eliminated (for example, a triangle, $m(c^*|_{p=v_a}) \prec_\alpha m(c^*|_{p=v_b}) \prec_\alpha m(c^*|_{p=v_c}) \prec_\alpha m(c^*|_{p=v_a})$, can occur when each configuration is evaluated on different subsets of instances), then skip this filter.
5. Admit candidates v with the best performance $m(c^*|_{p=v})$.
6. Admit candidates v with the most run equivalents.
7. Select one of the remaining candidates uniformly at random.

7.1.7 *"Adaptive"* Adaptive Capping

An important component of many algorithm configurators, such as SMAC [89], ParamILS [87] and `irace` [35], is a so-called "adaptive capping mechanism" that selects running time cutoffs for individual target algorithm runs. This mechanism is designed to avoid running a very poor configuration for a very long time on a

single instance. All three previously mentioned configurators use a simple heuristic for their adaptive capping mechanism, which is based on the performance of (one of) the best known configuration(s) found so far. This mechanism can be easily modified for use in GPS; for example, if $m(\cdot) = \text{PAR10}(\cdot)$, then the adaptive cap is calculated as

$$\begin{aligned} AC = {} & \text{PAR10}_{(I^* \cap I_{\text{new}}) \cup \{i'\}}(c^*) \cdot (|I^* \cap I_{\text{new}}| + 1) \cdot BM \\ & - \text{PAR10}_{I^* \cap I_{\text{new}}}(c^*|_{p=v_{\text{new}}}) \cdot |I^* \cap I_{\text{new}}|, \end{aligned} \tag{7.5}$$

where $|I^*|$ and $|I_{\text{new}}|$ denote the number of instances upon which c^* and $c^*|_{p=v_{\text{new}}}$ have each been evaluated so far, respectively; BM (the bound multiplier) is a constant (typically set to 2); and where $\text{PAR10}(\cdot)$ denotes mean running time with censored runs replaced by 10 times their running time cutoff.

However, this heuristic is overly aggressive, which causes high-quality challengers to be rejected before the permutation test used by GPS (see Chapter 7.1.6). In fact, it often takes only a single unlucky target algorithm run for a challenger to be prematurely rejected with this cap. Since GPS eliminates entire regions of the configuration space after observing a poorly performing parameter value, such a mistake can lead to a very large performance penalty for GPS's final incumbent (preliminary experiments indicated that slow-down factors of up to 10 were not uncommon). We therefore chose to modify the capping mechanism to make it even more adaptive, by using a bound multiplier that depends on $|I^* \cap I_{\text{new}}|$.

We estimated safe values for $BM(x)$ by performing a large number of repeated simulations using two identical exponential distributions to simulate the running times of two hypothetical algorithm configurations. We performed this analysis for varying values of x, the number of independent runs of each simulated algorithm configuration. We then calculated the ratio of the means for each pair of running times for each value of x, to obtain a distribution of bound multipliers that would not incorrectly lead GPS to reject one candidate as worse than the other. Using these distributions for each x, we then calculated the 99.95% percentile of the bound multipliers to obtain a bound multiplier that should work approximately 99.95% of the time (under the given assumptions). Finally, we plotted the data, performed various log transforms, and then used the Levenberg-Mardquart algorithm

to fit the function

$$f(x) = \exp(7.21 \cdot x^{-0.63}), \tag{7.6}$$

which provided a reasonably tight visual fit for the data, although we observed that it slightly under-estimated new data that we generated for larger values of x. Since this analysis makes several assumptions which may not hold for real-world scenarios, we make the rule even more conservative by using

$$BM(x) = \max(\exp(7.21 \cdot x^{-0.63}), 2), \tag{7.7}$$

which ensures that it is never more aggressive than the original heuristic.

To incorporate the effects of the decaying memory (see Chapter 7.1.4), everywhere in Equation 7.5 we use the number of run equivalents instead of the size of the instance sets (where we multiply the weights from $m(c^*)$ and $m(c^*|_{p=v_{\text{new}}})$ to obtain a combined level of trust, as for permutation tests).

Finally, as soon as a run with a parameter value has exhausted an adaptive cap for a single instance, we stop evaluating that value on new instances. Furthermore, for the purposes of the permutation test (see Chapter 7.1.1), it is considered worse than all other parameter values in its bracket which have not been capped.

7.1.8 Intensification & Queuing Runs

One of the most important components of algorithm configuration procedures are intensification mechanisms [88, 89], which control the number of target algorithm runs performed to evaluate each candidate configuration. Poorly performing configurations can typically be eliminated using only a small number of runs, whereas high-quality configurations require many more. It is also advantageous to use only a small number of benchmark instances when comparing configurations at early stages of the process (when the goal is to quickly distinguish between good and very bad regions of the configuration space), and to increase this number slowly as the search progresses. This is especially important in a parallel setting, where a large number of target algorithm runs could otherwise be queued all at once for an extremely bad configuration.

GPS addresses this with a two-part intensification mechanism: first, for each

parameter p, GPS starts with a very small set of *numInitInst* random instances, I_p. Each time B_p is updated, or when each $v \in B_p$ has been evaluated on each $i \in I_p$, the mechanism adds one randomly chosen instance (without replacement) to I_p. If there are no more new instances to add to I_p, then GPS starts adding instances it has already used with new random seeds.

The second component of the intensification mechanism is captured by how many target algorithm runs GPS queues to be performed for each candidate. GPS only queues target algorithm runs in powers of 2; that is, for parameter value $v \in B_p$, we find the largest power q for which $c^*|_{p=v}$ has been run on at least 2^q instances, and then we queue all of the first 2^{q+1} instances in I_p that have not yet been queued. If an old target algorithm run has become stale ($w \leq 0.05$), then we also re-queue that run. GPS does not queue any target algorithm runs for parameter values for which there is sufficient information to reject those values as worse than the current incumbent (unless it detects that the response for a given parameter is not uni-modal, in which case it assumes that it does not yet have enough data to correctly determine the ordering of the values and continues to queue runs for all of them). Sets of target algorithm runs are pushed into the queue in a random order.

7.1.9 Bandit Queue

Often, only a small fraction of parameters for highly-parameterized algorithms affect the algorithm's performance (for example, see Chapter 6 or Falkner et al. [58]). To avoid spending a large amount of time evaluating parameters that are unimportant, GPS uses a multi-armed-bandit-style mechanism to determine the order in which parameters are considered for queuing runs. GPS approximates the relative importance of each parameter p by counting the number of times k_p that p has been updated in c^*. Since most parameters are unlikely to be updated more than a small handful of times, we need a mechanism that increases the probability of choosing a parameter very quickly (for example, exponentially). We therefore start with a set $P' = P$, and when we are picking a new parameter to evaluate, we sample parameter p_{next} with probability

$$\frac{\text{Fib}(k_{p_{\text{next}}})}{\sum_{p \in P'} \text{Fib}(k_{p_{\text{next}}})}, \tag{7.8}$$

where Fib(k) is the k^{th} Fibonacci number. If we sample a parameter that has not been updated, then we remove it from P' and pick a new sample. Once a parameter has been picked that has been updated, or once P' is empty, we reinitialize $P' = P$.

7.1.10 Instance Increment

When algorithm runs can be performed very quickly (for example ≤ 10 CPU seconds per run) or when there are lots of parallel resources available, then GPS may be unable to keep the queue sufficiently populated to keep all of the workers busy. To compensate, GPS dynamically updates the value for an instance increment, *instIncr*. This is a multiplier used to make the operations in the intensification mechanism (see Chapter 7.1.8) operate on batches of I instances.

To dynamically update *instIncr*, GPS periodically records the number of worker processes, r, that are currently performing target algorithm runs and the number of tasks, q, that are waiting in the queue. Let R and Q denote the sequences of these recently recorded values. If median$(Q) < \frac{\max(R)}{2}$ or median$(Q) \leq 4$, then *instIncr* is increased. If median$(Q) \geq 2 \cdot \max(R)$, then *instIncr* is decreased. Otherwise, *instIncr* is left unchanged. To allow *instIncr* to quickly respond to changes, the value is always set to a positive Fibonacci number. We use max(R) to approximate the number of available workers, since GPS can operate with a dynamically changing amount of parallel resources.

7.1.11 The Worker Process

The worker processes query a database for new target algorithm runs (that is, a parameter p, a value v and an instance i). For each run, the worker calculates an adaptive cap (see Chapter 7.1.7), creates a temporary entry in the database that is set to expire in twice that run's cutoff time, and then begins the target algorithm run to obtain $m_{\{i\}}(c^*|_{p=v})$. Once the run is complete, the worker pushes the result to the database and removes the temporary entry. The temporary entry is a back-up designed to prevent GPS from stalling in the event of an unexpected crash occurring for a worker process. Once the temporary entry expires, if the result has not been pushed to the database, the master process will re-queue the run.

7.2 Experimental Setup

We compared GPS to the latest versions of three other state-of-the-art general-purpose algorithm configurators: SMAC 3.0[2] [89], GGA++ [7] and `irace` 3.3 [35]. Due to time constraints, we did not compare GPS to earlier versions of these configurators, nor to ParamILS, since the authors of ParamILS and SMAC recommend to use SMAC, unless there is sufficient budget available to use both.[3] GPS, `irace` and GGA++ all support parallel execution; for each run of these configurators we used 8 CPU cores. Unfortunately, to the best of our knowledge, the parallel version of SMAC [91] was never made publicly available; however, the quality of the configurations found between independent runs of SMAC can vary substantially, and it is typically recommended to exploit this via multiple independent parallel runs of SMAC [91, 172]. The so-called *standard protocol* is to evaluate all of the final incumbents on the entire training set and return the one with the best performance [172]. We applied SMAC in precisely this way by performing 8 independent runs of SMAC per scenario.

However, one may wonder whether the quality of the configurations obtained by GPS also varies substantially, and if the standard protocol might also benefit GPS. To answer this question, we performed 5 independent runs of each parallel algorithm configurator (and 40 independent runs of SMAC) for each scenario. From this single set of results, we then performed an analysis that simulates two experiments: One *small parallel budget* experiment, where each parallel configurator was run only once and SMAC was run 8 times in parallel; and a second *large parallel budget* experiment, where each parallel configurator was run 3 times independently in parallel (using a total of 24 cores each) and SMAC was run 24 times in parallel. We then used bootstrap sub-sampling to simulate 1 001 independent trials of each type of experiment to obtain median speedups and 95% bootstrap percentile confidence intervals.

We evaluated the performance of the configuration procedures on six ACLib [94] scenarios, which we summarize in Table 7.1. Four of these scenarios (LKH [82] and EAX [132] on TSP RUE 1000-3000 instances, CaDiCaL [25] on circuit fuzz

[2] We chose to use SMAC 3 after checking with the authors and in light of the fact that all recent publications regarding SMAC also use SMAC 3.

[3] Personal communication with the authors.

Table 7.1: The ACLib benchmark scenarios studied and their numbers of numerical and categorical parameters.

Problem	Algorithm	Instance Set	# Num	# Cat
TSP	LKH	TSP RUE 1000-3000	12	11
	EAX	TSP RUE 1000-3000	2	0
SAT	CaDiCaL	Circuit Fuzz	40	22
	probSAT	7SAT90	4	5
MIP	CPLEX	Regions200	22	63
		RCW2	22	63

SAT instances and CPLEX [1] on Regions200 MIP instances) were chosen, because they were among the scenarios that we identified as having "interesting" parameter responses in the landscape analysis (see Chapter 6) that inspired GPS. However, since we found that the EAX scenario had only a very minor possible speedup available over the default configuration, we modified the default parameter values to make this scenario more interesting. In particular, we changed the default for the population size from 100 to 418 and the number of children from 30 to 36. In this landscape work we also found that one of the parameters for LKH, the number of backbone trials, had a non-unimodal response; we hypothesize that this was because the value 0 had a special meaning (that a particular heuristic should be disabled). Between the scenarios, there were 10 other parameters (1 other for LKH, 9 for CPLEX) where the algorithm's documentation indicated that a similar special meaning was encoded in a particular value of a numerical parameter. GPS makes strong assumptions about the structure of algorithm configuration landscapes, therefore, when using GPS, parameters should not be encoded in this way. We therefore modified these two scenarios to introduce additional categorical parent parameters that control whether or not the special value for the numerical child parameter is used, or if the child parameter is otherwise used as normal. Finally, since GPS is also only able to handle conditional parameters that are active when their parents take on a single value (rather than a set or range of values), we further modified two of the parameters of CPLEX by creating multiple copies of the child parameter (this causes the number of parameters for LKH and CPLEX in Table 7.1 to differ from the respective AClib scenarios [94]).

To compare the performance of each configurator for different overall config-

uration budgets, we performed the previously described analysis for the anytime incumbent configurations after 30 minutes, 1 hour, 3 hours, 6 hours, 12 hours and 24 hours. We note that for most of these scenarios, ACLib recommends to use 48 hours for each configurator run; however, we shortened this to 24 hours to keep the total CPU time required for our experiments within our available budget. Even with this reduced budget, for all but the final scenario (CPLEX on RCW2), at least one of the configurators was still able to find a configuration with a significant speedup over the default with high probability. However, for CPLEX on RCW2, only SMAC and GPS returned configurations better than the default, and most of these were found within approximately 3 minutes, that is, when the configurators had not yet evaluated the configurations on enough instances to have any kind of statistical confidence of an improvement. Since this scenario was more challenging than the others, we turned it into a stretch test for GPS by increasing the total degree of parallelism from 8 cores per run to 32 cores per run for the parallel configurators (and increased the total SMAC runs from 40 to 160). In the following, we report the results for this more challenging setting.

All experiments were run on a cluster of 20 nodes, each equipped with 32 2.10 GHz Intel Xeon E5-2683 v4 CPUs with 40 960 KB cache and 96 GB RAM each, running openSUSE Leap 42.1 (x86_64). We allotted 3 GB of RAM to each core used by an algorithm configurator, and further restricted each target algorithm run to a maximum of 3 GB of RAM. We measured the performance of each target algorithm using penalized average running time 10 (PAR10), where censored runs (runs unable to complete within their running time cutoff or within the 3 GB RAM limit) are replaced with 10 times their running time cutoff. When evaluating the configurations on the test set, we ran each configuration on each instance in random order, to ensure that background environmental noise effects (for example, cache effects or dynamic CPU clock speed changes, which are designed to avoid overheating) were independently and identically distributed.

7.3 Results

We encountered varying degrees of difficulty with each of the configurators we used as baselines. For SMAC, 2.5% of the runs we performed terminated early

after raising an uncaught exception. We treated these runs as if the final incumbent found remained unchanged for all time budgets after the runs crashed. For all of GGA++'s runs on the CPLEX scenarios, we provided 52 hour job allocations on our cluster (which should provide ample slack for the 24-hour configuration budget). However, GGA++ was unable to complete any of these runs within the 52 hours. Since GGA++ does not output any information about its anytime configurations until the very end of its runs, we were unable evaluate its performance on these scenarios. We suspect GGA++ did not respect its configuration budgets on these scenarios because the running time cutoffs for CPLEX are 10 000 seconds, which is much larger than all the other scenarios and it does not use adaptive capping or intensification mechanisms.

When we gave `irace` a wall-clock budget of 24 hours we quickly observed that it used only a small fraction of the available budget before terminating. Initially, `irace` estimates the performance of the default configuration and uses this to convert the remaining time budget into a target algorithm run budget, a number which it consistently under-estimated. Since we still had most of `irace`'s computational budget remaining, we repeated all of these runs with a configuration budget of 48 hours of wall-clock time and report the anytime results for these instead. Nevertheless, 80% of `irace`'s runs terminated after using between 25-75% of the available budget (and the rest used less – as little as 3%); however, we had exhausted the computational resources we had available to evaluate `irace`, therefore, we treat these runs similarly to SMAC's crashed runs.

In Table 7.2 we show selected results using the large parallel budget analysis method as described in Chapter 7.2. In particular, we compare the 24 hour configuration budget results (which is on the small end of a typical algorithm configuration budget) with the 6 hour budget results (a time for which the best configurator for each scenario has typically found a high-quality solution). While the absolute and relative performance of each configurator varies a little between configuration budgets, the results presented here nevertheless reflect the main trends for each scenario. The remainder of the results are available as supplementary material in Appendix A.4. We mark the median speedups in boldface if they are not statistically worse than the best speedup within each configuration budget according to a permutation test with a 5% significance level.

Table 7.2: Large parallel budget analysis speedups (medians and 95% confidence intervals). Median speedups not worse than the best speedup for each time budget according to a permutation test with a 5% significance level are shown in boldface.

	TSP		SAT		MIP	
	LKH	EAX	CaDiCaL	probSAT	CPLEX	
	TSP RUE 1000-3000		Circuit Fuzz	7SAT90	Regions200	RCW2
Configuration budget (excluding validation) = 6.0 wall-clock hours						
GPS	**1.20** [0.95, 1.20]	**3.05** [2.82, 3.21]	**1.41** [1.12, 1.41]	**3.76** [1.93, 4.73]	**1.16** [1.00, 1.31]	1.01 [1.00, 1.05]
SMAC3.0	1.00 [1.00, 1.00]	**2.62** [2.53, 3.55]	1.25 [1.09, 1.36]	**4.90** [4.28, 13.41]	0.96 [0.77, 1.18]	**1.26** [1.22, 1.27]
irace3.3	0.94 [0.92, 1.11]	2.48 [1.85, 3.38]	0.87 [0.83, 0.95]	**4.90** [4.90, 13.06]	0.01 [0.01, 1.00]	1.00 [1.00, 1.00]
GGA++	0.66 [0.58, 1.05]	0.61 [0.61, 0.61]	0.92 [0.89, 1.02]	**7.14** [5.38, 8.58]	–	–
Configuration budget (excluding validation) = 24.0 wall-clock hours						
GPS	**1.21** [1.18, 1.28]	**3.22** [2.36, 3.46]	**1.44** [1.16, 1.55]	3.03 [1.93, 5.52]	0.68 [0.01, 1.12]	**1.41** [1.09, 1.41]
SMAC3.0	1.00 [1.00, 1.00]	2.73 [2.62, 3.08]	**1.36** [1.16, 1.60]	**5.76** [4.28, 13.41]	**1.18** [0.77, 1.23]	1.26 [1.22, 1.27]
irace3.3	1.03 [0.79, 1.14]	2.72 [2.33, 3.00]	0.90 [0.84, 1.01]	**5.86** [4.97, 12.92]	0.01 [0.01, 1.00]	1.00 [1.00, 1.00]
GGA++	1.02 [0.77, 1.11]	0.61 [0.61, 0.61]	1.02 [0.84, 1.02]	**8.87** [6.73, 8.89]	–	–

When we compare the small parallel budget analysis results (available in the supplementary material) to the large parallel budget analysis results, we can see that all procedures tend to perform better using the large budget analysis and that GPS performs best (relative to the other configurators) in the large budget results. We therefore refer the reader to the supplementary material for the complete results, and discuss the 6 and 24 hour large budget analysis results in the remainder of this chapter.

Overall, we can see that GPS consistently obtained the greatest median speedup for three of the five scenarios with 8 cores per configurator run: LKH and EAX on TSP RUE 1000-3000 instances, and CaDiCaL on circuit fuzz instances, with SMAC finding similar-performing configurations for EAX at 6 hours and for CaDiCaL at 24 hours.

In some cases, the configurators returned incumbents with speedups less than one. Most often, this is early on in the configuration process when the configurators have not evaluated the configurations on enough problem instances to have a good approximation of their final performance. However, it can also happen for larger budgets. For example, for the CPLEX on Regions200 scenario GPS initially obtains the best speedup, but then suffers from some performance degradation, such that SMAC ends up winning for the 24 hour configuration budget. In cases where GPS's performance degrades with larger budgets, it is possible that

GPS updated two parameter values in short succession. When this happens, GPS's new incumbent may initially not have been evaluated on any problem instances due to the decaying memory heuristic (see Chapter 7.1.4). If the two parameters have a strong compensatory interaction, then the double update may cause GPS to over-shoot the basin of high-quality configurations, thus possibly leading to an incumbent with substantially worse performance.

Nevertheless, the only 8 core scenario for which GPS did not perform competitively with all (in fact any) of the other configurators was probSAT on the 7SAT90 instance set. GPS did find configurations with speedups greater than 3 for both time budgets; however, the other procedures obtained even greater speedups. Despite this, GPS still managed to statistically tie for the 3 hour configuration budget. However, this tie is primarily because poorly-performing configurations are unable to complete a relatively large fraction of their runs within the running time cutoff, which substantially increases their variance. For example, 10.44% of the runs of the default configuration are censored.

For CPLEX on the RCW2 instance set (the stretch test, for which we provided the parallel configurators with 32 cores per run instead of 8), we see that SMAC initially finds the greatest speedup, but that GPS's median speedup eventually surpasses SMAC. Surprisingly, we actually see that SMAC obtains exactly the same speedups for all configuration budgets (see Table A.10). In fact, only 17.5% of the runs of SMAC updated the incumbent after the first 3 minutes; however, 100% of the runs updated the incumbent within the first 3 minutes. This indicates that SMAC may be getting lucky by finding better-than-default configurations prior to having any statistical confidence of their improvement. Therefore, these improvements are likely due to the standard protocol when applied to mostly-random configurations.

Overall, GPS appears to be the top-performing configuration procedure, followed by SMAC. However, one advantage shared by GPS, GGA++ and `irace` is that they are all able to make use of parallel resources within individual configurator runs, whereas SMAC must be run multiple times independently in parallel. In Figure 7.1, we show the anytime speedups obtained by the four configurators such that the x-axis includes both the configuration budget and the budget required to validate the anytime incumbent configurations with the standard protocol (see also

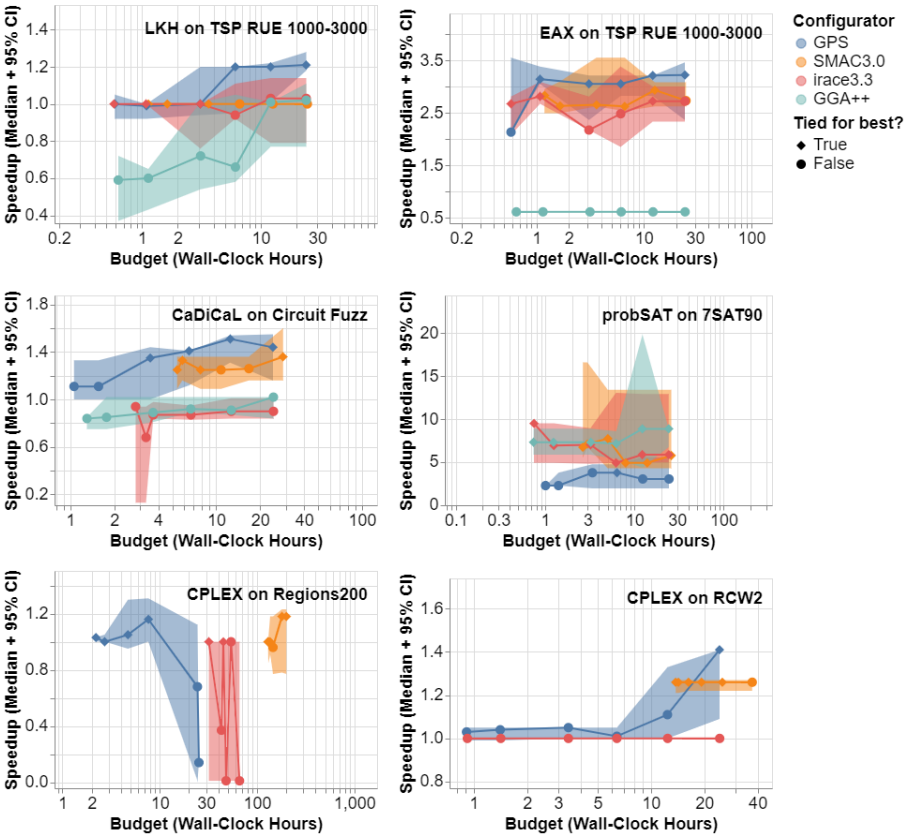

Figure 7.1: The anytime performance results comparing the four configurators. The wall-clock budgets include both the configuration and validation budgets, where the validation budgets are assumed to be perfectly parallelizable. Configurations obtained for a given configuration budget (excluding validation time) are considered tied for best if the difference between the median speedups is not significant according to a permutation test with a 5% significance level.

Table A.12, which includes these combined configuration budgets). The time to validate all of SMAC's configurations is sometimes very large. The most extreme example is for the six hour configuration budget for CPLEX on Regions200, where GPS required 1.64 hours for validation and SMAC required 139.79 hours. While there are also some less extreme examples – for example, the 0.08 *vs* 0.63 hours required to validate the 24 hour configuration budget runs on LKH – SMAC always required more validation time overall. This becomes even more pronounced for scenarios with large and difficult instance sets. An alternative is to perform fewer independent runs of SMAC; this gives it less total parallel resources, but a fair chance during validation. However, for challenging scenarios, this typically yields worse performance for SMAC.

7.4 Chapter Summary

In this chapter, we introduced a powerful automated algorithm configurator, GPS, designed to operate in parallel and exploit the insights gained throughout each chapter of this thesis. In particular, in Chapter 6 we observed that algorithm configuration landscapes for running time minimization are often uni-modal and typically contain relatively weak parameter interactions. However unlike in Chapter 5, where we attempted to exploit similar structure by using simple classes of surrogate models (which proved to be too simple), we instead combined coordinate descent with golden section search [102], which obtains optimal worst-case performance on one-dimensional uni-modal functions without attempting to model the landscape.

We believe that our choice of coordinate descent worked well, because it is very efficient in situations where parameters do not interact strongly. Furthermore, in cases where the structure of the interactions is similar to the function $f(x,y) = |x-y|$, coordinate descent can solve the problem quite efficiently by updating only a single one of the parameters. This likely helped GPS perform well, given that it is a type of structure that appears to be rather common among those parameters that do interact.

We also attribute GPS's success to two other key attributes: the permutation test used to quickly reject poorly performing configurations (see Chapter 7.1.1)

and our "adaptive" adaptive capping procedure (see Chapter 7.1.7), which does the same without causing GPS to eliminate candidates without reasonable statistical (or heuristic) confidence that it should do so. In particular, in some preliminary experiments with an early version of GPS that used a fixed bound multiplier, we observed that GPS frequently eliminated good configurations, which caused it to get stuck in very poor regions of the configuration space.

Despite strong assumptions made by GPS about algorithm configuration landscapes – that is, that parameter responses are uni-modal and that most parameters do not strongly interact – GPS found the best configurations in 5 out of 6 scenarios, often at a fraction of the time budget required by other methods. For example, for CaDiCaL on the circuit fuzz SAT instance set, GPS achieved a median speedup of 1.41 after a 6-wall-clock-hour configuration budget and 0.51 wall-clock hours for validation, compared to SMAC, which obtained a speedup of 1.25 after a 6-wall-clock-hour configuration budget and 4.79 wall-clock hours for validation. (Note that SMAC actually obtained a slightly better speedup of 1.31 within a 1-hour configuration budget; this phenomenon might be caused by SMAC's incumbent updating procedure, which does not make use of statistical tests for performance improvements.)

In two cases, GPS exhibited undesirable behaviour, that is, premature stagnation for probSAT on the 7SAT90 instances, and returning substantially worse configurations after more configuration time for CPLEX on Regions200. We suspect that parameter interactions may have caused this behaviour, since GPS can occasionally return incumbents which have never been evaluated on any instances (see the discussion of the CPLEX on Regions200 scenario in Chapter 7.3). However, two possible ways to safeguard against such effects include increasing the decay rate and using the standard protocol, which validates a set of final incumbent configurations (see Chapter 2.3.3).

Nevertheless, despite the overall strong performance of GPS on running time minimization algorithm configuration scenarios, we do not include an analysis of its performance on AutoML scenarios. Due to GPS's iterative nature and its strong reliance on a permutation test to avoid prematurely rejecting poor configurations, it is unclear how to effectively make use of low-fidelity estimates of configuration performance. Since our preliminary experiments showed that exploiting this infor-

mation is required to obtain state-of-the-art performance for AutoML, we do not expect GPS to be competitive on AutoML scenarios. We performed some preliminary experiments that supported this hypothesis; however, we did not fully evaluate GPS on the AutoML scenarios in Chapter 5 as it did not seem an effective use of our limited computational budget.

Chapter 8

Conclusion

In this thesis, we provided the first statistically-principled methods for analyzing the structure of landscapes of optimization problems with stochastic objectives (see Chapter 3). We applied these methods to study the landscapes arising in two very important and prominent application areas of automated algorithm configuration: automated machine learning (AutoML – see Chapter 4), and the configuration of algorithms for solving $\mathcal{NP}$-hard and $\mathcal{NP}$-complete problems (see Chapter 6). To exploit the insights we learned, we developed and analyzed three new automated algorithm configuration procedures; two were designed specifically for AutoML scenarios (see Chapter 5) and one for running time minimization (see Chapter 7).

8.1 Algorithm Configuration Landscape Analysis

Very little previous work has been done to analyze the landscapes of AutoML scenarios and running time minimization scenarios. However, a third application area of automated algorithm configuration, the optimization of meta-heuristic performance for solving optimization problems, had a small amount landscape analysis done prior to the work for this thesis. The methods used to perform these prior analyses lacked statistical sophistication; however, their results suggested that this class of algorithm configuration landscapes may be relatively simple. Most notably, their results provided some evidence that these landscapes may often be unimodal. If this structure is pervasive among algorithm configuration landscapes, it

raises questions about why nearly all existing automated algorithm configuration procedures rely on powerful meta-heuristics to explore the full extent of the landscapes of algorithm configuration scenarios. Indeed, in some early work in this space, Pedersen [140] even showed that a simple configuration procedure that assumes uni-modality of the landscapes can perform well for differential evolution and particle swarm optimization, when configuring up to 9 numerical parameters to improve the final incumbent's solution quality.

We showed that nearly all of the AutoML loss landscapes and running time minimziation landscapes we studied share the structural simplicity suggested by these early studies on meta-heuristic configuration landscapes. Most notably, we showed that all 18 of the landscapes we studied between the two application areas appear to be very close to uni-modal. Our statistical tests only detected deviations from uni-modality in three of these landscapes. Two of these cases were for the loss landscapes of FCNet [103] (when applied to two different data sets); however, in each case, those deviations from uni-modality were very small. In particular, the sub-optimal modes in the landscape accounted for up to 0.03% of the landscape, indicating that even a weak perturbation strategy should be sufficient for escaping such local minima. The remaining case was for LKH when solving TSP RUE instances, for which a single value ($\texttt{BACKBONE\_TRIALS} = 0$) of a parameter yielded a sub-optimal local minimum.

We also performed an analysis of the parameters in these landscapes. Most notably, we showed that a simplistic configuration procedure that naïvely assumes each parameter can be configured independently, a single time and in a random order, can obtain surprisingly high-quality results. In particular, this procedure found configurations that were statistically tied with optimal in every AutoML scenario we studied as well as for 97% of the 201 two-dimensional parameter response slices we studied for the running time minimization scenarios. This result suggests that even though there can be strong interactions between some parameters, these interactions still tend to be rather simple. Intuitively, we surmise that many of these interactions that are both "simple" and strong, can be explained by the compensatory interaction that we would expect to observe between the time and temperature used to bake a cake. That is, if one is slightly larger than optimal the other can compensate by decreasing slightly (see Figure 1.1).

Apart from these general trends, we observed a small handful of examples of more complex behaviour. For example, when Xgboost's `subsample` parameter is set to 0.5, the algorithm frequently exceeded its running time cutoff of 960 seconds. This lead to the presence of numerous "spikes" and "ridges" in an otherwise relatively smooth landscape. In fact, their presence was not enough to form sub-optimal modes, as they could be circumnavigated. Nevertheless, automated algorithm configuration procedures must still be equipped to occasionally encounter such artifacts, as they could otherwise cause the configurators to incorrectly assume that certain directions in the landscape are unlikely to yield improvement. For a more thorough discussion of this and some other oddities we observed, see Chapter 4.

8.2 Algorithm Configuration Landscape Exploitation

Most algorithm configuration landscapes appear to be relatively simple. This result suggests that their structure should be exploitable. To this end, we introduced three new automated algorithm configuration procedures.

The first two configuration procedures were targeted specifically for AutoML scenarios (see Chapter 5). In particular, we modified an existing state-of-the-art hyper-parameter configuration procedure, BOHB [59], which combines Bayesian optimization with Hyperband [113] – that is, a procedure for exploiting the low-fidelity approximations uniquely ubiquitous among AutoML scenarios (see Chapter 2.3.2). Our two variants of BOHB swapped out the Bayesian optimization component in favour of simple surrogate models that could be used to guide the search process: a convex quadratic approximation model (CQA) and a spline model. The choice of the CQA model was motivated by our observation that AutoML loss landscapes are often uni-modal. We chose the spline model because the interactions of the hyper-parameters tended to be rather benign. In each case, we showed that the models can perform well under ideal circumstances – that is, when the landscape is symmetric around the global minimum or that there are no parameter interactions, respectively. However, we also showed that even small deviations from these conditions caused the new methods to perform poorly, thereby revealing why they were not competitive on the realistic application scenarios.

Furthermore, while traditional Bayesian optimization does improve over the random search normally employed by Hyperband, we observed that because Hyperband is able to eliminate poorly performing configurations at a very low cost, the performance gains that can be realized by a more efficient landscape search procedure are likely to be relatively small. We therefore concentrated our efforts on algorithm configuration procedures for minimizing the running time of algorithms for solving $\mathcal{NP}$-hard and $\mathcal{NP}$-complete problems instead, where cheap, low-fidelity approximations are typically not readily available.

To this end, we introduced a powerful automated algorithm configurator, GPS, designed to operate in parallel and exploit our insights on running time minimization landscapes. GPS combines essential ingredients of state-of-the-art algorithm configuration procedures [35, 87, 89] with a simple one-dimensional bracketing procedure [102] applied to each parameter semi-independently in parallel. We compared GPS to three state-of-the-art baselines, SMAC [89], irace [35] and GGA++ [7] on six scenarios spanning TSP, SAT and MIP. Despite strong assumptions made by GPS about algorithm configuration landscapes – that is, that parameter responses are uni-modal and that most parameters do not strongly interact – GPS found the best configurations in five out of the six scenarios, often at a fraction of the time budget required by other methods.

However, in two cases, GPS exhibited undesirable behaviour – that is, premature stagnation and returning substantially worse configurations after more configuration time. We suspect that parameter interactions and/or sub-optimal plateaus may be responsible for this behaviour; however, future work needs to be done to determine how frequently this occurs in practice, and whether or not our hypotheses are correct.

We believe that there are several essential ingredients underlying GPS's success. First, it exploits the frequent uni-modality of the parameter responses via golden section search [102]. Second, it exploits the relative simplicity of parameters interactions, by configuring each parameter semi-independently in parallel. Note that this strategy is effective if strong interactions do not exist, but also when parameters have strong but still simple, compensatory relationships. And third, it uses a novel adaptive capping mechanism (a procedure for picking a running time cutoff for each target algorithm run), that is less likely to incorrectly reject

high-quality configurations before there is sufficient evidence to do so. This final component is particularly important for the running time minimization landscapes of algorithms for solving $\mathcal{NP}$-hard and $\mathcal{NP}$-complete problems, since they exhibit high stochasticity in algorithm performance relative to the AutoML loss landscapes that we studied in Chapters 4 and 5.

8.3 Promising Directions for Future Research

As with most lines of research, digging deeper resolves many questions but often raises even more. We therefore see many promising directions for future improvements to the work we have done, as well as numerous promising ideas for future work.

8.3.1 Landscape Analysis for Randomized Objectives

A common theme among the new landscape analysis techniques we introduced in Chapter 6 was their ability to handle the noisy and randomized performance measurements that arise in the configuration of algorithms. To the best of our knowledge, all other existing fitness landscape analysis techniques – even those that have been applied specifically to algorithm configuration landscapes [42, 79–81, 134, 142, 157, 179, 196] – incorrectly assume the performance of a given configuration is deterministic.

In particular, we believe that no existing measure of ruggedness (for example, see Harrison et al. [80], Stadler and Schnabl [167], Vassilev et al. [184] or Weinberger [187]) is adequate for the analysis of algorithm configuration landscapes. Ruggedness generally relates the tendency of neighbouring candidate solutions to yield similar solution quality. However, for algorithm configuration scenarios, repeated calls to the target algorithm typically yields different solution qualities. This is often true even when the algorithm is used to solve the exact same set of problem instances. Existing measures of ruggedness may still be useful if they are applied to some population statistic (for example, the mean or median) of the algorithm's performance on a suitably large population of problem instances with a suitably large number of independent runs of the algorithm on each problem instance. However, we firmly believe that new measures of ruggedness must be defined that treat and

quantify each dimension of variability separately: the variability between independent runs of the algorithm on different problem instances, and the variability between nearby candidate configurations.

Similarly, other existing methods for performing fitness landscape analysis would benefit greatly from improved handling of this additional source of variance. For example, Treimun-Costa et al. [179] generated local optima networks using BasicILS [88]. They assumed that when BasicILS's local search procedure stopped, it stopped at a local optimum. However, that local search procedure relies only on a fixed number of independent runs of the target algorithm, and thus those "local optima" may have merely been artifacts introduced by the variability in performance measurements.

One potentially simple way to improve local optima networks for algorithm configuration landscapes would be to adapt a statistical test to quantify the probability for which each vertex is in fact a local optimum compared to a random artifact. This additional information would be extremely valuable: not only would it help to quantify the likelihood that the full local optima network accurately depicts the global structure of the landscape, but it would also contrast that view with a representation of the structure that a naïve algorithm configuration procedure like BasicILS believes is present in the landscape.

8.3.2 Low Fidelity Approximations

Perhaps the biggest limitation with our analysis of AutoML loss landscapes is that we did not include any discussion of low-fidelity landscapes. That is, a *low-fidelity algorithm configuration landscape* is the same as a typical algorithm configuration landscape where the performance measurement for the algorithm has been replaced by a low-fidelity approximation. For example, in Chapter 5, we obtained low-fidelity approximations by decreasing the number of MCMC steps while training a Bayesian neural network or by decreasing the number of estimators used by XGboost.

The main reason that we did not include an analysis of low-fidelity landscapes in Chapter 4 is that most of the landscape data that was available for us to use simply did not contain the information necessary to perform this analysis. However,

a better understanding of the extent to which low-fidelity algorithm configuration landscapes resemble their full-fidelity counterparts would be immensely useful.

Clearly, the success of successive halving [95], Hyperband [113], ASHA [114] and BOHB [59] indicates that there must be at least *some* similarity between low-fideltiy landscapes and full-fidelity landscapes. However, while we observed that the full-fidelity landscapes typically have relatively simple structure (for example, uni-modality and simple parameter interactions – see Chapter 4), this may not necessarily be true for their low-fidelity counterparts. For example, low-fidelity landscapes may have similar global structure to their full-fidelity counterparts; however, they may also have substantially increased ruggedness, thereby introducing numerous sub-optimal local minima. If true, this could serve as an additional explanation for the relatively poor performance of the methods we introduced in Chapter 5 for exploiting the structure of AutoML loss landscapes.

Furthermore, this finding would have a significant impact on the way in which we believe the structure of AutoML loss landscapes could be exploited in future work. For example, a simple adaptation of GPS (see Chapter 7) to support multiple levels of fidelity would not be successful. Indeed, if the low-fidelity landscapes prove to be highly rugged with numerous local minima then any method designed to search these landscapes would need to make use of strong diversification mechanisms.

A second, but perhaps equally useful question to answer is the extent to which different *types* of low-fidelity approximations yield landscapes that remain similar for their full-fidelity counterparts. For example, it may be the case that decreasing the number of training iterations [95] compared to decreasing the size of the training set [104] may yield low-fidelity landscapes that are either more or less similar to their full-fidelity counterparts. In fact, the reality may be more nuanced; for example, one type of low-fidelity approximation may tend to increase the ruggedness of a landscape without substantially affecting the position of high-quality solutions. A different type of approximation could instead cause the global shape of the landscape to change, thereby moving the high-quality configurations from one region of the configuration space to another, while leaving the ruggedness of the landscape relatively unchanged. Insights such as these could help to further improve methods which seek to make use of multiple types of low-fidelity landscapes, or

at least provide guidance regarding which types of low-fidelity approximations are best suited for different types of configurators.

Another research direction that has the potential to be highly impactful is the exploitation of low-fidelity approximations in applications other than those arising in AutoML. However, the availability and applicability of a given method of obtaining low-fidelity approximations of solution quality may vary substantially between different algorithm configuration scenarios. Audet et al. [14] suggested that optimization algorithms could be stopped early to obtain a surrogate function. Clearly, this is an easy way that low-fidelity approximations could be obtained for anytime optimization algorithms. However, it is also possible to imagine other methods as well. For example, if configuring a sorting algorithm, it may be sufficient to randomly sample a subset of the items being sorted. Similarly, for problems like the TSP, it may be effective to randomly subsample cities from a given problem instance in order to obtain one that is similar, but less costly to solve. Alternatively, easier versions of bounded model checking SAT instances can be obtained by simply unrolling the hardware circuits to smaller depths [150]. However, to the best of our knowledge, all of these options remain almost entirely unexplored for improving the performance of automated algorithm configuration outside of AutoML.

While not motivated by our observations on low-fidelity approximations, we spent some time working on a closely related problem. In particular, we searched for features of a given problem instance that could be varied to control the difficulty of individual problem instances, for example, the size of the problem instance. We then hoped that by using methods related to those for empirical running time scaling analysis (for example, see Mu and Hoos [130], Pushak and Hoos [150] or Pushak et al. [152]) we could predict which configuration would perform well on larger problem instances. More specifically, we hoped to identify which parameters of an algorithm need to be *modified* to obtain configurations that scale well with problem difficulty.

However, our results were mixed, with no clear indication that we had obtained any method that could work robustly on a diverse set of application scenarios. In hind sight, and in light of the success of existing methods in AutoML for exploiting low-fidelity approximations, perhaps a more successful approach would be to

search for features of a problem instance that can be varied such that the problem difficulty can be scaled *without* needing to modify the parameters of an algorithm in order to obtain optimal performance.

8.3.3 Other Objectives and Multi-objective Algorithm Configuration

In some applications, the running time of an algorithm may not be a huge concern (provided that it stays within some reasonable limit). Similarly, there many not necessarily be any meaningful notion of solution quality. For example, when the target algorithm sorts a set of numbers, the resulting output is either sorted or it is not. In these contexts, a very natural metric which may be useful to optimize is the amount of energy that is consumed by running the algorithm, as this often directly relates to the cost of running the algorithm. Energy consumption minimization is particularly important in the context of mobile application development [188] and data center operation [20].

One of the key challenges in minimizing the consumption of energy is in measuring the energy that is used by the algorithm in isolation of other concurrently running processes. A direct approach is to use a smart power meter that can measure and report the energy consumed by, for example, a laptop [177]; however, obtaining accurate readings with such an approach requires minimizing all other concurrent processes running on the machine. As a result, one of the most common practices is to use simple features to construct a model that can predict the energy usage of an algorithm [60, 194, 197]. For example, Zhang et al. [197] and Fan et al. [60] propose to simply predict energy as a function of CPU usage. However, this method can clearly produce misleading results when an algorithm is paralellized – particularly if it is parallelized across resources other than CPUs.

One potentially natural method for overcoming this limitation is to look at two different measures of algorithm resource usage: the running time of the algorithm *and* its average degree of parallelism when solving a particular problem instance. Clearly, these metrics could be combined to produce a more accurate model of energy consumption. Alternatively, they could both be used as two competing objectives in a multi-objective algorithm configuration scenario, thereby allowing the user to obtain a Pareto front of solutions that can trade off between running

time and energy consumption.

In fact, multi-objective optimization is a very natural choice for many algorithm configuration scenarios which can either include energy consumption as one of the objectives (for example, see Balaprakash et al. [16], Gschwandtner et al. [74] or Tiwari et al. [177]) or those that focus only on other kinds of objectives (for example, see Blot et al. [29] or Tari et al. [174]). However, to the best of our knowledge, no existing multi-objective automated algorithm configuration procedure makes use of insights from landscape analysis techniques. Indeed, the adaptation and application of some of the recent advances in multi-objective fitness landscape analysis techniques [48, 70, 85] remains a promising area for future research.

8.3.4 Additional Algorithm Configuration Landscape Analysis

In addition to improving algorithm configuration landscape analysis techniques, there is still plenty of room for additional analysis using the methods we have introduced in this thesis.

For example, our analysis contained only a limited number of scenarios with categorical parameters. Overall, we observed that the effects of these categorical parameters were relatively benign, in that they did not cause the landscapes to become multi-modal. However, we speculate that this behaviour will not generalize to, for example, the combinatorial landscapes induced by model and pre-processor selection in machine learning pipelines. In particular, while a machine learning method that uses model m_1 might perform best with feature encoding mechanism e_1, it is not clear that e_1 will be optimal for a qualitatively different machine learning method that relies on model m_2. Therefore, if the choice for each pre-processor and model are encoded as categorical hyper-parameters (as is often done, see for example, Feurer et al. [63]) it is not clear that the neighbourhood relation we used in this thesis will induce a uni-modal landscape.

In Chapter 4, we also observed that the FCNet landscapes were qualitatively different from the rest that we studied. In particular, they exhibited surprisingly large percentages of statistically significant high-order partial derivatives. Future work should be done to investigate whether or not this behaviour should be attributed to the fact that the FCNet landscape we studied may have been in the over-

parameterized regime [19]. Given that we also saw relatively strong interactions between two of the parameters of the Bayesian neural network that we configured in Chapter 5, this seems to be at least a reasonable hypothesis.

Our new landscape analysis methods could also be applied to other similar algorithm configuration problems. For example, another very common objective for algorithm configuration is the solution quality returned by optimization algorithms (for example, see Balaprakash et al. [15] or López-Ibáñez et al. [117]). Augmenting the existing analysis of these landscapes with our new statistically principled techniques has the potential to bring similar insights and improvements to these applications as well. Similarly, it would be useful to apply our analysis to more scenarios from AutoML and for running time minimization, to determine whether or not the observations we have made can be expected to be reproduced reliably among many different applications, or whether or not there is some degree of variability among the landscapes, which would necessitate quantifying the frequency with which exceptions occur or determining how they can be predicted without performing such costly forms of analysis.

8.3.5 Extensions to GPS

Future work could combine multiple independent parallel runs of GPS with a racing procedure that evaluates the anytime incumbents of each GPS run as these runs are performed. We believe that this could not only improve the performance of GPS, but also remove the need for post-configuration validation. Such a procedure could also be combined with a mechanism to detect stagnation and initiate random restarts, if needed.

Given that we observed that most algorithm configuration landscapes appear to be uni-modal, it may seem counter-intuitive for us to be recommending random restarts, a diversification mechanism. However, one obvious failure mode for GPS occurs in landscapes that contain a high degree of neutrality. In particular, we expect this to occur for any scenario in which a large fraction of the landscape contains poorly-performing configurations, thereby leading to censoring of the true performance values. If GPS starts its search in any location for which two or more parameters have bad default values, it is likely that GPS will fail to ever update

its incumbent configuration. This is because GPS will see that by modifying each parameter independently, no improvements to the performance can be obtained.

The detrimental effects of such stagnation in the presence of highly censored landscapes may also be mitigated by modifying GPS to initialize its search procedure with random search. Random search been shown to be highly effective under tight configuration budget constraints [59], which makes it a common initial search strategy for many configuration procedures (for example, see Ansótegui et al. [7], Cáceres et al. [35], Falkner et al. [59] or Hutter et al. [89]). Initialization with random search would also allow GPS to start off at a configuration that is potentially much better than the default, thereby eliminating GPS's dependence on the quality of the default configuration.

Similarly, modifying GPS to make use of more general pattern search [178] or mesh adaptive direct search [12] methods instead of coordinate descent may help to improve the performance of GPS, both on landscapes with high degrees of neutrality arising from censoring, and potentially also on landscapes with high correlation among the high-quality values of two or more parameters. However, one of the possible downsides of such adaptations is that GPS would no longer be able to make use of golden section search to minimize the number of objective evaluations that need to be performed at each iteration, which may outweigh the benefits of searching along non-axis-aligned directions.

It may also be advantageous to improve GPS's current bandit-queue procedure, which helps GPS spend more time evaluating important parameters (see Chapter 7.1.9). For example, consider the scenario where an important parameter is initialized to a locally-optimal value, but for which compensatory parameter interactions between it and another parameter mean that it will eventually need to be updated to obtain optimal performance. In such cases, the first parameter may initially appear unimportant, because it does not initially need to be updated. As other parameters are updated, GPS's bandit queue identifies them as important, thereby decreasing the probability of selecting the first parameter exponentially. This could lead to the first parameter never being evaluated during the later stages of the configuration procedure. To help protect against this problem, a small but constant fraction of the budget, for example 1%, could be reserved for evaluating parameters that are believed to be unimportant. Alternatively, better methods for

identifying which parameters are most important (for example, see Chapter 2.2.11) could be adapted for use within GPS.

As stated in the introduction of Chapter 7, it may also be beneficial to include interpolation in GPS's bracketing procedure to help pick the next configuration to evaluate, for example, using Brent's method [34]. However, as indicated previously, we remain skeptical that this would be substantially beneficial, particularly in the case of automated algorithm configuration for the minimization of running time for algorithms that solve $\mathcal{NP}$-hard and $\mathcal{NP}$-complete problems, due to the large variability in performance between independent runs of the algorithms. However, it may be more suitable for other types of objectives in other application scenarios, such as AutoML. Therefore, we believe that it is still a potential avenue for improvement.

However, before GPS can be made a competitive option in AutoML scenarios, it must be adapted to make use of low-fidelity approximations of configuration performance. Future work should also be done to investigate the performance of GPS on other algorithm configuration problems for solution quality minimization. Furthermore, while we designed GPS with the goal that it would scale well to large degrees of parallelism, all of the applications on which it has been applied so far contained only a moderate degree of parallelism. It would be useful to perform an extended analysis on configuration scenarios involving substantially increased parallel computing resources.

Finally, it may be advantageous to extend GPS to allow for both soft and hard bounds on numerical parameter values, since exceeding some bounds may lead GPS to finding incumbents that produce incorrect output.

8.3.6 Explainability and Automated Algorithm Configuration

Recently, explainable artificial intelligence, and in particular, explainable machine learning, has gained significant amounts of attention (for example, see Molnar [128] and references therein). To the best of our knowledge, explainability has not yet been adopted among the algorithm configuration community; however, we believe that they are naturally complementary.

For example, the most obvious combination of the two uses methods for auto-

mated machine learning to generate powerful (and potentially complex) machine learning pipelines, and then uses explainability techniques to make the decisions or predictions made by those pipelines more transparent to end users. Methods for explaining the decisions made by complex machine learning pipelines, and in fact, any powerful artificial intelligence tool, are increasingly important, as they allow non-expert users to build trust in the complex, black-box AI technology that they use. Explainability techniques also have a key role to play in any applications where AI is used to make decisions that impact the lives of real people. In these cases, there are an increasing number of regulations providing those impacted with rights to explanations regarding why the particular AI system made a given decision regarding, for example, their application for a loan or permit.

While some of these situations may seem somewhat far removed from the topics of this thesis, we also see many areas in which explainability can be used in the context of algorithm configuration. For example, there exist numerous different methods for calculating the importance of features for a given machine learning model. This can be done either globally (for example, Breiman [33] or Frye et al. [68]) or locally (for example, Ribeiro et al. [155] or Štrumbelj and Kononenko [169]). Clearly, any of these methods could be adapted for use to study algorithm configuration scenarios in order to determine the importance of each of the parameters, either globally or locally, thereby providing alternatives to the methods discussed in Chapter 2.2.11. Similarly, techniques developed to help visualize the dependence of a machine learning model's predictions upon individual features (for example, see Apley and Zhu [8] or Friedman [67]) could be adapted for the visualization of algorithm configuration landscapes (for some recent and early work in this space, see Moosbauer et al. [129]).

The most straightforward way for any these methods to be applied to algorithm configuration problems is through the use of a machine learning surrogate model trained to predict the performance of the algorithm in question as a function of its parameters. However, it may also be possible to adapt some of these methods to bypass such a step, since many machine learning explainability techniques treat the machine learning model as a black box function.

8.4 Outlook

The landscape analysis techniques that we have developed and employed throughout this thesis allowed us to obtain deep insights into the common patterns and structure present in algorithm configuration landscapes. However, we believe that we have only scratched the surface of what can be achieved through the study of algorithm configuration landscapes.

Due to the substantial practical importance of techniques for automated algorithm configuration, we believe that algorithm configuration landscape analysis will attract significant attention in the years to come. In particular, future work will be done that provides further support for (or perhaps against) the hypotheses we have made regarding the common structure of algorithm configuration landscapes. Future work will also address other types of insights; for example, the typical ruggedness of various kinds of algorithm configuration landscapes.

The algorithm configuration landscape analysis methods we have introduced can help researchers to see which directions in the design space of algorithm configuration procedures are likely candidates for improvements. Designing configuration procedures that exploit specific types of landscape structure will necessitate the creation of a variety of algorithm configurators with complementary strengths and weaknesses. However, in their current form, our landscape analysis methods remain prohibitively expensive to be useful as a means of *selecting* a given algorithm configuration procedure for solving a particular algorithm configuration scenario. Therefore, the logical next step from this line of research is to develop features of algorithm configuration landscapes that can be cheaply computed, thereby enabling an algorithm selection procedure [2, 18, 101, 119] to decide which automated algorithm configurator is most likely to produce high-quality results on a given algorithm configuration scenario.

Nevertheless, we hope that our research – and the additional algorithm configuration landscape research it inspires – will lift the dense fog blanketing algorithm configuration landscapes, thereby inspiring the next generation of automated algorithm configuration procedures.

Bibliography

[1] IBM Corp. IBM ILOG CPLEX Optimizer. https://www.ibm.com/analytics/cplex-optimizer, 2021. Last accessed on October 10th, 2021. → pages 108, 137

[2] T. Abell, Y. Malitsky, and K. Tierney. *Fitness landscape based features for exploiting black-box optimization problem structure*. IT University of Copenhagen, 2012. → pages 5, 30, 31, 160

[3] M. Anastacio and H. H. Hoos. Combining sequential model-based algorithm configuration with default-guided probabilistic sampling. In *Proceedings of the Twenty-Second International Genetic and Evolutionary Computation Conference Companion (GECCO 2020 Companion)*, 2020. → page 42

[4] M. Anastacio and H. H. Hoos. Model-based algorithm configuration with default-guided probabilistic sampling. In *Proceedings of the Sixteenth International Conference on Parallel Problem Solving from Nature (PPSN 2020)*, 2020. → page 42

[5] C. Angermueller, T. Pärnamaa, L. Parts, and O. Stegle. Deep learning for computational biology. *Molecular Systems Biology*, 12(7), 2016. → page 22

[6] C. Ansótegui, M. Sellmann, and K. Tierney. A gender-based genetic algorithm for the automatic configuration of algorithms. In *Proceedings of the Fifteenth International Conference on Principles and Practice of Constraint Programming (CP 2009)*, pages 142–157, 2009. → pages 3, 44

[7] C. Ansótegui, Y. Malitsky, H. Samulowitz, M. Sellmann, and K. Tierney. Model-based genetic algorithms for algorithm configuration. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)*, pages 733–739, 2015. → pages 44, 128, 136, 149, 157

[8] D. W. Apley and J. Zhu. Visualizing the effects of predictor variables in black box supervised learning models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 82(4):1059–1086, 2020. → page 159

[9] H. Asi and J. C. Duchi. The importance of better models in stochastic optimization. *Proceedings of the National Academy of Sciences*, 116(46): 22924–22930, 2019. → page 22

[10] C. Audet and J. E. Dennis Jr. Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on Optimization*, 17(1):188–217, 2006. → page 40

[11] C. Audet and W. Hare. *Derivative-Free and Blackbox Optimization*. Springer International Publishing, 2017. → page 4

[12] C. Audet and D. Orban. Finding optimal algorithmic parameters using derivative-free optimization. *SIAM Journal on Optimization*, 17(3): 642–664, 2006. → pages 3, 46, 157

[13] C. Audet, J. Denni, D. Moore, A. Booker, and P. Frank. A surrogate-model-based method for constrained optimization. In *Proceedings of the Eighth Symposium on Multidisciplinary Analysis and Optimization*, 2000. → page 3

[14] C. Audet, K.-C. Dang, and D. Orban. Optimization of algorithms with OPAL. *Mathematical Programming Computation*, 6(3):233–254, 2014. → pages 4, 42, 46, 153

[15] P. Balaprakash, M. Birattari, and T. Stützle. Improvement strategies for the F-Race algorithm: Sampling design and iterative refinement. In *Proceedings of the Fourth International Workshop on Hybrid Metaheuristics (HM 2007)*, pages 108–122. Springer International Publishing, 2007. → pages 3, 40, 46, 156

[16] P. Balaprakash, A. Tiwari, and S. M. Wild. Multi objective optimization of HPC kernels for performance, power, and energy. In *Proceedings of the Fourth International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS 2013)*, pages 239–260. Springer International Publishing, 2013. → pages 4, 155

[17] T. Bartz-Beielstein, C. W. Lasarczyk, and M. Preuss. Sequential parameter optimization. In *Proceedings of the 2005 IEEE Congress on Evolutionary*

Computation (CEC 2005), volume 1, pages 773–780. IEEE, 2005. → page 42

[18] N. Belkhir, J. Dréo, P. Savéant, and M. Schoenauer. Feature based algorithm configuration: A case study with differential evolution. In *Proceedings of the Fourteenth International Conference on Parallel Problem Solving from Nature (PPSN 2016)*, pages 156–166, 2016. → pages 5, 19, 160

[19] M. Belkin, D. Hsu, S. Ma, and S. Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019. → pages 24, 25, 80, 84, 156

[20] A. Beloglazov, R. Buyya, Y. C. Lee, and A. Zomaya. A taxonomy and survey of energy-efficient data centers and cloud computing systems. In *Advances in Computers*, volume 82, pages 47–111. 2011. → page 154

[21] J. S. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(10):281–305, 2012. → pages 3, 23, 39, 66

[22] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Proceedings of the Twenty-fifth Conference on Advances in Neural Information Processing Systems (NeurIPS 2011)*, pages 2546–2554, 2011. → pages 22, 23, 43, 53, 66, 86

[23] A. Biedenkapp, M. Lindauer, K. Eggensperger, F. Hutter, C. Fawcett, and H. H. Hoos. Efficient parameter importance analysis via ablation with surrogates. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI 2017)*, 2017. → pages 6, 37

[24] A. Biedenkapp, J. Marben, M. Lindauer, and F. Hutter. CAVE: Configuration assessment, visualization and evaluation. In *Proceedings of the Twelfth International Conference on Learning and Intelligent Optimization (LION 2018)*, 2018. → page 37

[25] A. Biere. CaDiCaL, Lingeling, Plingeling, Treengeling and YalSAT entering the SAT Competition 2017. In *Proceedings of the 2017 SAT Competition: Solver and Benchmark Descriptions*, pages 14–15, 2017. → pages 107, 136

[26] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A racing algorithm for configuring metaheuristics. In *Proceedings of the Fourth Genetic and Evolutionary Computation Conference (GECCO 2002)*, pages 11–18, 2002. → pages 3, 37, 39, 40, 45, 46

[27] M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle. F-Race and iterated F-Race: An overview. In *Experimental Methods for the Analysis of Optimization Algorithms*, pages 311–336. Springer International Publishing, 2010. → page 40

[28] J. A. Blackard and D. J. Dean. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and Electronics in Agriculture*, 24(3): 131–151, 1999. → page 95

[29] A. Blot, H. H. Hoos, L. Jourdan, M.-É. Kessaci-Marmion, and H. Trautmann. MO-ParamILS: A multi-objective automatic algorithm configuration framework. In *Proceedings of the Tenth International Conference on Learning and Intelligent Optimization (LION 2016)*, pages 32–47. Springer International Publishing, 2016. → pages 44, 155

[30] Y. Borenstein and R. Poli. Information landscapes. In *Proceedings of the Seventh International Conference on Genetic and Evolutionary Computation (GECCO 2005)*, pages 1515–1522, 2005. → pages 29, 30

[31] L. Bottou and Y. LeCun. Large scale online learning. *Proceedings of the Eighteenth Conference on Advances in Neural Information Processing Systems (NeurIPS 2004)*, 16:217–224, 2004. → page 47

[32] S. Boyd, S. P. Boyd, and L. Vandenberghe. *Convex Optimization*. Cambridge university press, 2004. → page 4

[33] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. → pages 22, 159

[34] R. P. Brent. *Algorithms for minimization without derivatives*. Courier Corporation, 2013. → pages 123, 158

[35] L. P. Cáceres, M. López-Ibáñez, H. H. Hoos, and T. Stützle. An experimental study of adaptive capping in `irace`. In *Proceedings of the Eleventh International Conference on Learning and Intelligent Optimization (LION 2017)*, pages 235–250. Springer International Publishing, 2017. → pages 40, 45, 47, 125, 128, 131, 136, 149, 157

[36] A. Cauchy et al. Méthode générale pour la résolution des systemes d'équations simultanées. *Comptes Rendus des Séances de l'Académie des Sciences de Paris*, 25(1847):536–538, 1847. → page 47

[37] J. Cavazos and M. F. O'Boyle. Automatic tuning of inlining heuristics. In *Proceedings of the Eighteenth ACM/IEEE Conference on Supercomputing (SC 2005)*, pages 14–14. IEEE, 2005. → page 4

[38] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the Twenty-Second ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD 2016)*, pages 785–794, 2016. → pages 68, 95

[39] T. Chen, E. Fox, and C. Guestrin. Stochastic gradient hamiltonian monte carlo. In *Proceedings of the Thirty-First International Conference on Machine Learning (ICML 2014)*, pages 1683–1691. PMLR, 2014. → page 95

[40] C. W. Cleghorn. Particle swarm optimization: Understanding order-2 stability guarantees. In *Proceedings of the International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*, pages 535–549. Springer International Publishing, 2019. → page 17

[41] C. W. Cleghorn and A. P. Engelbrecht. Particle swarm stability: A theoretical extension using the non-stagnate distribution assumption. *Swarm Intelligence*, 12(1):1–22, 2018. → page 17

[42] C. W. Cleghorn and G. Ochoa. Understanding parameter spaces using local optima networks: A case study on particle swarm optimization. In *Proceedings of the Twenty-Third International Genetic and Evolutionary Computation Conference Companion (GECCO 2021 Companion)*, pages 1657–1664, 2021. → pages 17, 150

[43] M. Conforti, G. Cornuéjols, and G. Zambelli. *Integer programming*, volume 271. Springer International Publishing, 2014. → page 4

[44] A. Coraddu, L. Oneto, A. Ghio, S. Savio, D. Anguita, and M. Figari. Machine learning approaches for improving condition-based maintenance of naval propulsion plants. *Proceedings of the Institution of Mechanical Engineers, Part M: Journal of Engineering for the Maritime Environment*, 230(1):136–153, 2016. → page 67

[45] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20 (3):273–297, 1995. → page 22

[46] T. L. Dean and M. S. Boddy. An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI 1988)*, volume 88, pages 49–54, 1988. → page 42

[47] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002. → page 45

[48] B. Derbel and S. Verel. Fitness landscape analysis to understand and predict algorithm performance for single-and multi-objective optimization. In *Proceedings of the Twenty-Second International Genetic and Evolutionary Computation Conference Companion (GECCO 2020 Companion)*, pages 993–1042, 2020. → page 155

[49] C. Di Francescomarino, M. Dumas, M. Federici, C. Ghidini, F. M. Maggi, W. Rizzi, and L. Simonetto. Genetic algorithms for hyperparameter optimization in predictive business process monitoring. *Information Systems*, 74:67–83, 2018. → page 45

[50] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959. → page 56

[51] S. Droste, T. Jansen, and I. Wegener. Perhaps not a free lunch but at least a free appetizer. In *Proceedings of the First International Genetic and Evolutionary Computation Conference (GECCO 1999)*, pages 833–839, 1999. → page 5

[52] D. Dua and C. Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml. → pages 67, 68, 95

[53] J. Dubois-Lacoste, H. H. Hoos, and T. Stützle. On the empirical scaling behaviour of state-of-the-art local search algorithms for the Euclidean TSP. In *Proceedings of the Seventeenth International Genetic and Evolutionary Computation Conference (GECCO 2015)*, pages 377–384, 2015. → page 108

[54] R. C. Eberhart, Y. Shi, and J. Kennedy. *Swarm Intelligence*. Elsevier, 2001. → page 16

[55] K. Eggensperger, M. Feurer, F. Hutter, J. S. Bergstra, J. Snoek, H. H. Hoos, and K. Leyton-Brown. Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In *Proceedings of the NeurIPS workshop on Bayesian Optimization in Theory and Practice*, 2013. → pages 23, 68

[56] P. H. Eilers and B. D. Marx. Flexible smoothing with B-splines and penalties. *Statistical Science*, 11(2):89–121, 1996. → page 89

[57] B. Emil and G. Tamar. Some statistical aspects of binary measuring systems. *Measurement*, 46(6):1922–1927, 2013. → page 69

[58] S. Falkner, M. Lindauer, and F. Hutter. SpySMAC: Automated configuration and performance analysis of SAT solvers. In *Proceedings of the Eighteenth International Conference on Theory and Applications of Satisfiability Testing (SAT 2015)*, pages 215–222, 2015. → pages 42, 109, 111, 125, 134

[59] S. Falkner, A. Klein, and F. Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the Thirty-Fifth International Conference on Machine Learning (ICML 2018)*, pages 1437–1446, 2018. → pages 23, 41, 43, 48, 66, 86, 94, 95, 148, 152, 157

[60] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *Proceedings of the Thirty-Fourth International Symposium on Computer Architecture (SIGARCH 2007)*, volume 35, pages 13–23, 2007. → page 154

[61] C. Fawcett and H. H. Hoos. Analysing differences between algorithm configurations through ablation. *Journal of Heuristics*, 22(4):431–458, 2016. → pages 6, 37

[62] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In *Proceedings of the Twenty-Ninth Conference on Advances in Neural Information Processing Systems (NeurIPS 2015)*, pages 2962–2970. 2015. → pages 3, 42, 43, 48, 66

[63] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter. Auto-sklearn: Efficient and robust automated machine learning. In *Automated Machine Learning*, pages 113–134. Springer International Publishing, 2019. → page 155

[64] M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter. Auto-sklearn 2.0. *arXiv:2007.04074 [cs.LG]*, 2020. → pages 42, 48

[65] F.-A. Fortin, F.-M. De Rainville, M.-A. G. Gardner, M. Parizeau, and C. Gagné. DEAP: Evolutionary algorithms made easy. *The Journal of Machine Learning Research*, 13(1):2171–2175, 2012. → page 45

[66] J. H. Friedman. Multivariate adaptive regression splines. *The Annals of Statistics*, pages 1–67, 1991. → page 38

[67] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, pages 1189–1232, 2001. → page 159

[68] C. Frye, D. de Mijolla, L. Cowton, M. Stanley, and I. Feige. Shapley-based explainability on the data manifold. *arXiv e-prints*, pages arXiv–2006, 2020. → page 159

[69] U. Garciarena, R. Santana, and A. Mendiburu. Analysis of the complexity of the automatic pipeline generation problem. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2018)*, pages 1–8. IEEE, 2018. → page 23

[70] D. Garrett and D. Dasgupta. Multiobjective landscape analysis and the generalized assignment problem. In *Proceedings of the Second International Conference on Learning and Intelligent Optimization (LION 2007)*, pages 110–124. Springer International Publishing, 2007. → page 155

[71] S. I. Gass. *Linear programming: methods and applications*. Courier Corporation, 2003. → page 4

[72] D. E. Goldberg and J. H. Holland. Genetic algorithms and machine learning. *Machine Learning*, 3(2):95–99, 1988. → page 20

[73] F. Graf, H.-P. Kriegel, M. Schubert, S. Pölsterl, and A. Cavallaro. 2D image registration in CT images using radial image descriptors. In *Proceedings of the Fourteenth International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI 2011)*, pages 607–614, 2011. → page 67

[74] P. Gschwandtner, J. J. Durillo, and T. Fahringer. Multi-objective auto-tuning with insieme: Optimization and trade-off analysis for time, energy and resource usage. In *Proceedings of the Twentieth International Conference on Parallel Processing (Euro-Par 2014)*, pages 87–98. Springer International Publishing, 2014. → pages 4, 155

[75] G. T. Hall, P. S. Oliveto, and D. Sudholt. On the impact of the cutoff time on the performance of algorithm configurators. In *Proceedings of the Twenty-First Genetic and Evolutionary Computation Conference (GECCO 2019)*, pages 907–915, 2019. → page 44

[76] G. T. Hall, P. S. Oliveto, and D. Sudholt. Analysis of the performance of algorithm configurators for search heuristics with global mutation operators. In *Proceedings of the Twenty-Second Genetic and Evolutionary Computation Conference (GECCO 2020)*, pages 823–831, 2020. → page 44

[77] G. T. Hall, P. S. Oliveto, and D. Sudholt. Fast perturbative algorithm configurators. In *Proceedings of the Sixteenth International Conference on Parallel Problem Solving from Nature (PPSN 2020)*, pages 19–32. Springer International Publishing, 2020. → page 44

[78] N. Hansen. The CMA evolution strategy: A comparing review. *Towards a New Evolutionary Computation*, pages 75–102, 2006. → page 40

[79] K. R. Harrison, B. M. Ombuki-Berman, and A. P. Engelbrecht. The parameter configuration landscape: A case study on particle swarm optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2019)*, pages 808–814, 2019. → pages 17, 28, 29, 30, 60, 150

[80] K. R. Harrison, B. M. Ombuki-Berman, and A. P. Engelbrecht. Visualizing and characterizing the parameter configuration landscape of differential evolution using physical landform classification. In *Proceedings of the IEEE Symposium Series on Computational Intelligence (SSCI 2020)*, pages 2437–2444, 2020. → pages 18, 19, 33, 34, 35, 150

[81] K. R. Harrison, B. M. Ombuki-Berman, and A. P. Engelbrecht. Visualizing and characterizing the parameter configuration landscape of particle swarm optimization using physical landform classification. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2021)*, pages 2299–2306. IEEE, 2021. → pages 17, 33, 34, 150

[82] K. Helsgaun. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126: 106–130, 2000. → pages 108, 136

[83] M. Hoffman, F. R. Bach, and D. M. Blei. Online learning for latent dirichlet allocation. In *Proceedings of the Twenty-fourth Conference on Advances in Neural Information Processing Systems (NeurIPS 2010)*, pages 856–864, 2010. → page 69

[84] H. H. Hoos and T. Stützle. *Stochastic Local Search: Foundations & Applications*. Morgan Kaufmann Publishers Inc., 2005. ISBN 1558608729. → pages 4, 25, 27, 31, 32, 35, 57, 60

[85] Y. Huang, W. Li, F. Tian, and X. Meng. A fitness landscape ruggedness multiobjective differential evolution algorithm with a reinforcement learning strategy. *Applied Soft Computing*, 96:106693, 2020. → page 155

[86] F. Hutter, H. H. Hoos, and T. Stützle. Automatic algorithm configuration based on local search. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI 2007)*, volume 7, pages 1152–1157, 2007. → page 43

[87] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle. ParamILS: An automatic algorithm configuration framework. *Journal of Artificitial Intelligence Research*, 36:267–306, 2009. → pages 3, 32, 43, 45, 47, 125, 128, 131, 149

[88] F. Hutter, H. H. Hoos, K. Leyton-Brown, and K. Murphy. Time-bounded sequential parameter optimization. In *Proceedings of the Fourth International Conference on Learning and Intelligent Optimization (LION 2010)*, pages 281–298. Springer International Publishing, 2010. → pages 42, 45, 127, 133, 151

[89] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the Fifth Learning and Intelligent Optimization Conference (LION 2011)*, volume 6683 of *LNCS*, pages 507–523, 2011. → pages 3, 42, 48, 105, 110, 128, 131, 133, 136, 149, 157

[90] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Bayesian optimization with censored response data. In *Procedings of 2011 NeurIPS workshop on Bayesian Optimization, Experimental Design, and Bandits*, 2011. → page 42

[91] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Parallel algorithm configuration. In *Proceedings of Sixth International Conference on Learning and Intelligent Optimization (LION 2012)*, pages 55–70, 2012. → pages 43, 136

[92] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Identifying key algorithm parameters and instance features using forward selection. In *Proceedings of the Seventh International Conference on Learning and Intelligent*

Optimization (LION 2013), Lecture Notes in Computer Science, pages 364–381. Springer International Publishing, 2013. → page 38

[93] F. Hutter, H. H. Hoos, and K. Leyton-Brown. An efficient approach for assessing hyperparameter importance. In *Proceedings of the Thirty-First International Conference on Machine Learning (ICML 2014)*, pages 754–762, June 2014. → pages 38, 61, 62, 65, 68, 89, 109

[94] F. Hutter, M. López-Ibáñez, C. Fawcett, M. Lindauer, H. H. Hoos, K. Leyton-Brown, and T. Stützle. AClib: A benchmark library for algorithm configuration. In *Proceedings of the Fourteenth International Conference on Learning and Intelligent Optimization (LION 2014)*, pages 36–40, 2014. → pages 68, 107, 117, 136, 137

[95] K. Jamieson and A. Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *Artificial Intelligence and Statistics*, pages 240–248. PMLR, 2016. → pages 3, 41, 47, 152

[96] J. Jasiewicz and T. F. Stepinski. Geomorphons – a pattern recognition approach to classification and mapping of landforms. *Geomorphology*, 182:147–156, 2013. → page 33

[97] T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA 1995)*, volume 95, pages 184–192, 1995. → pages 23, 27, 52, 60, 65, 83

[98] K. Kandasamy, J. Schneider, and B. Póczos. High dimensional bayesian optimisation and bandits via additive models. In *Proceedings of the Thirty-Second International Conference on Machine Learning (ICML 2015)*, pages 295–304, 2015. → pages 53, 66

[99] K. Kandasamy, G. Dasarathy, J. Schneider, and B. Póczos. Multi-fidelity bayesian optimisation with continuous approximations. In *Proceedings of the Thirty-Fourth International Conference on Machine Learning (ICML 2017)*, pages 1799–1808, 2017. → pages 41, 43, 66

[100] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. LightGBM: A highly efficient gradient boosting decision tree. *Proceedings of the Thirty-First Conference on Advances in Neural Information Processing Systems (NeurIPS 2017)*, 30:3146–3154, 2017. → page 95

[101] P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann. Automated algorithm selection: Survey and perspectives. *Evolutionary Computation*, 27(1):3–45, 2019. → pages 5, 160

[102] J. Kiefer. Sequential minimax search for a maximum. *Proceedings of the American Mathematical Society*, 4(3):502–506, 1953. → pages 123, 124, 127, 128, 143, 149

[103] A. Klein and F. Hutter. Tabular benchmarks for joint architecture and hyperparameter optimization. *arXiv preprint arXiv:1905.04970*, 2019. → pages 38, 62, 67, 147

[104] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter. Fast Bayesian optimization of machine learning hyperparameters on large datasets. In *Artificial Intelligence and Statistics*, pages 528–536. PMLR, 2017. → pages 41, 152

[105] M. J. Kochenderfer and T. A. Wheeler. *Algorithms for Optimization*. Mit Press, 2019. → page 22

[106] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown. Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. *Journal of Machine Learning Research*, 18(25): 1–5, 2017. URL http://jmlr.org/papers/v18/16-261.html. → page 48

[107] A. Krizhevsky. Learning multiple layers of features from tiny images. 2009. → page 24

[108] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the Twenty-Sixth Conference on Neural Information Processing Systems (NeurIPS 2012)*, pages 1097–1105, 2012. → page 22

[109] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324, 1998. → pages 24, 68

[110] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553): 436–444, 2015. → page 22

[111] S. Lessmann, R. Stahlbock, and S. F. Crone. Optimizing hyperparameters of support vector machines by genetic algorithms. In *Proceedings of the International Conference on Artificial Intelligence (IC-AI 2005)*, pages 74–82, 2005. → page 45

[112] K. Leyton-Brown, E. Nudelman, and Y. Shoham. Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming (CP 2002)*, pages 556–572, 2002. → page 38

[113] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1): 6765–6816, 2017. → pages 3, 23, 41, 43, 66, 86, 94, 148, 152

[114] L. Li, K. Jamieson, A. Rostamizadeh, E. Gonina, J. Ben-tzur, M. Hardt, B. Recht, and A. Talwalkar. A system for massively parallel hyperparameter tuning. In *Proceedings of Machine Learning and Systems (MLSys 2020)*, volume 2, pages 230–246. 2020. → pages 41, 94, 152

[115] Y. Li, J. Jiang, J. Gao, Y. Shao, C. Zhang, and B. Cui. Efficient automatic CASH via rising bandits. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2020)*, number 04, pages 4763–4771, 2020. → page 48

[116] M. Lindauer and F. Hutter. Warmstarting of model-based algorithm configuration. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI 2018)*, pages 1355–1362, 2018. → page 42

[117] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, T. Stützle, and M. Birattari. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016. → pages 40, 156

[118] M. Lunacek and D. Whitley. The dispersion metric and the CMA evolution strategy. In *Proceedings of the Eighth International Conference on Genetic and Evolutionary Computation (GECCO 2006)*, pages 477–484, 2006. → page 28

[119] K. M. Malan. Landscape-aware constraint handling applied to differential evolution. In *Proceedings of the Seventh International Conference on Theory and Practice of Natural Computing (TPNC 2018)*, pages 176–187, 2018. → pages 5, 160

[120] K. M. Malan. A survey of advances in landscape analysis for optimisation. *Algorithms*, 14(2):40, 2021. → pages 4, 25

[121] K. M. Malan and A. P. Engelbrecht. A survey of techniques for characterising fitness landscapes and some possible ways forward. *Information Sciences*, 241:148–163, 2013. → pages 4, 25

[122] K. M. Malan and A. P. Engelbrecht. Characterising the searchability of continuous optimisation problems for PSO. *Swarm Intelligence*, 8(4): 275–302, 2014. → pages 29, 30

[123] A. Mametjanov, P. Balaprakash, C. Choudary, P. D. Hovland, S. M. Wild, and G. Sabin. Autotuning FPGA design parameters for performance and power. In *Proceedings of the Twenty-Third IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM 2015)*, pages 84–91, 2015. → page 4

[124] O. Maron and A. W. Moore. Hoeffding races: Accelerating model selection search for classification and function approximation. In *Proceedings of the Sixth International Conference on Advances in Neural Information Processing Systems (NeurIPS 1993)*, pages 59–66, 1993. → page 3

[125] D. C. Mattfeld, C. Bierwirth, and H. Kopfer. A search space analysis of the job shop scheduling problem. *The Annals of Operations Research*, 86: 441–453, 1999. → page 30

[126] O. J. Mengshoel. Understanding the role of noise in stochastic local search: Analysis and experiments. *Artificial Intelligence*, 172(8-9):955–990, 2008. → page 16

[127] K. Miller, M. P. Kumar, B. Packer, D. Goodman, and D. Koller. Max-margin min-entropy models. In *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics (AIStats 2012)*, pages 779–787, 2012. → page 68

[128] C. Molnar. *Interpretable machine learning*. Lulu. com, 2020. → page 158

[129] J. Moosbauer, J. Herbinger, G. Casalicchio, M. Lindauer, and B. Bischl. Towards explaining hyperparameter optimization via partial dependence plots. In *Proceedings of the Eighth International Conference on Machine Learning (ICML) Workshop on Automated Machine Learning (AutoML)*, 2021. → page 159

[130] Z. Mu and H. H. Hoos. Empirical scaling analyser: An automated system for empirical analysis of performance scaling. In *Proceedings of the Seventeenth International Genetic and Evolutionary Computation Conference (GECCO 2015)*, pages 771–772, 2015. → pages 106, 153

[131] Z. Mu, H. H. Hoos, and T. Stützle. The impact of automated algorithm configuration on the scaling behaviour of state-of-the-art inexact TSP solvers. In *Proceedings of the Tenth International Conference on Learning and Intelligent Optimization (LION 2016)*, volume 10079 of *LNCS*, pages 157–172, 2016. → page 108

[132] Y. Nagata and S. Kobayashi. A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. *Institute for Operations Research and the Management Sciences Journal on Computing*, 25(2):346–363, 2013. ISSN 1526-5528. → pages 108, 136

[133] Y. Nesterov. Gradient methods for minimizing composite functions. *Mathematical Programming*, 140(1):125–161, 2013. → page 47

[134] M. Nunes, P. M. Fraga, and G. L. Pappa. Fitness landscape analysis of graph neural network architecture search spaces. In *Proceedings of the Twenty-Third International Genetic and Evolutionary Computation Conference (GECCO 2021)*, pages 876–884, 2021. → pages 24, 28, 29, 36, 60, 150

[135] G. Ochoa, M. Tomassini, S. Vérel, and C. Darabos. A study of NK landscapes' basins and local optima networks. In *Proceedings of the Tenth International Conference on Genetic and Evolutionary Computation (GECCO 2008)*, pages 555–562, 2008. → page 32

[136] G. Ochoa, S. Verel, and M. Tomassini. First-improvement *vs* best-improvement local optima networks of NK landscapes. In *Proceedings of the Eleventh International Conference on Parallel Problem Solving from Nature (PPSN 2010)*, pages 104–113, 2010. → page 32

[137] R. S. Olson and J. H. Moore. TPOT: A tree-based pipeline optimization tool for automating machine learning. In *Proceedings of the Workshop on Automatic Machine Learning*, pages 66–74. PMLR, 2016. → pages 46, 66

[138] R. S. Olson, N. Bartley, R. J. Urbanowicz, and J. H. Moore. Evaluation of a tree-based pipeline optimization tool for automating data science. In *Proceedings of the Eighteenth International Genetic and Evolutionary Computation Conference (GECCO 2016)*, pages 485–492, 2016. → page 45

[139] R. S. Olson, R. J. Urbanowicz, P. C. Andrews, N. A. Lavender, L. C. Kidd, and J. H. Moore. Automating biomedical data science through tree-based pipeline optimization. In *Proceedings of the Nineteenth European*

Conference on the Applications of Evolutionary Computation (EvoApplications 2016), pages 123–137, 2016. → page 45

[140] M. E. H. Pedersen. *Tuning & simplifying heuristical optimization*. PhD thesis, University of Southampton, 2010. → pages 11, 16, 19, 47, 124, 147

[141] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research (JMLR)*, 12:2825–2830, 2011. → pages 23, 47, 95

[142] C. G. Pimenta, A. G. de Sá, G. Ochoa, and G. L. Pappa. Fitness landscape analysis of automated machine learning search spaces. In *Proceedings of the Twentieth European Conference on Evolutionary Computation in Combinatorial Optimization (EVOCOP 2020 – Part of EvoStar)*, pages 114–130. Springer International Publishing, 2020. → pages 23, 27, 28, 36, 60, 66, 83, 150

[143] E. Pitzer and M. Affenzeller. A comprehensive survey on fitness landscape analysis. *Recent Advances in Intelligent Engineering Systems*, pages 161–191, 2012. → pages 4, 25, 27, 28, 60

[144] M. J. Powell. The BOBYQA algorithm for bound constrained optimization without derivatives. *Cambridge NA Report NA2009/06, University of Cambridge, Cambridge*, pages 26–46, 2009. → page 40

[145] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. Golden section search in one dimension. *Numerical Recipes in C: The Art of Scientific Computing*, 1992. → page 124

[146] P. Probst, A.-L. Boulesteix, and B. Bischl. Tunability: Importance of hyperparameters of machine learning algorithms. *The Journal of Machine Learning Research*, 20(1):1934–1965, 2019. → pages 38, 39

[147] Y. Pushak and H. H. Hoos. Experimental procedures for exploiting structure in AutoML loss landscapes. Under review, 5 pages. → pages vii, 86

[148] Y. Pushak and H. H. Hoos. Algorithm configuration landscapes: More benign than expected? In *Proceedings of the Fifteenth International Conference on Parallel Problem Solving from Nature (PPSN 2018)*, pages 271–283, 2018. → pages vi, vii, 16, 53, 105

[149] Y. Pushak and H. H. Hoos. Golden parameter search: Exploiting structure to quickly configure parameters in parallel. In *Proceedings of the Twenty-Second International Genetic and Evolutionary Computation Conference (GECCO 2020)*, pages 245–253, 2020. → pages vii, 122

[150] Y. Pushak and H. H. Hoos. Advanced statistical analysis of empirical performance scaling. In *Proceedings of the Twenty-Second International Genetic and Evolutionary Computation Conference (GECCO 2020)*, pages 236–244, 2020. → page 153

[151] Y. Pushak and H. H. Hoos. AutoML loss landscapes. Under review, 26 pages. → pages vi, vii, 53, 66

[152] Y. Pushak, Z. Mu, and H. H. Hoos. Empirical scaling analyzer: An automated system for empirical analysis of performance scaling. *AI Communications*, pages 1–19, 2020. → page 153

[153] P. S. Rana. Physicochemical properties of protein tertiary structure data set. *UCI Machine Learning Repository*, 2013. → page 67

[154] C. M. Reidys and P. F. Stadler. Neutrality in fitness landscapes. *Applied Mathematics and Computation*, 117(2-3):321–350, 2001. → page 36

[155] M. T. Ribeiro, S. Singh, and C. Guestrin. "Why should I trust you?" Explaining the predictions of any classifier. In *Proceedings of the Twenty-Second ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD 2016)*, pages 1135–1144, 2016. → page 159

[156] H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, pages 400–407, 1951. → page 47

[157] N. M. Rodrigues, S. Silva, and L. Vanneschi. A study of fitness landscapes for neuroevolution. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2020)*, pages 1–8. IEEE, 2020. → pages 12, 23, 24, 35, 150

[158] J. B. Rosen and R. F. Marcia. Convex quadratic approximation. *Computational Optimization and Applications*, 28(2):173–184, 2004. → pages 88, 89

[159] N. L. Roux, M. Schmidt, and F. Bach. A stochastic gradient method with an exponential convergence rate for finite training sets. In *Proceedings of the Twenty-Fifth International Conference on Advances in Neural*

Information Processing Systems (NeurIPS 2012), volume 2, pages 2663–2671, 2012. → page 47

[160] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. → page 95

[161] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948. → page 35

[162] A. D. Shapiro. *Structured induction in expert systems*. Addison-Wesley Longman Publishing Co., Inc., 1987. → page 95

[163] G. Sierksma and Y. Zwols. *Linear and integer optimization: Theory and practice*. CRC Press, 2015. → page 27

[164] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Proceedings of the Twenty-Sixth Conference on Advances in Neural Information Processing Systems (NeurIPS 2012)*, pages 2951–2959, 2012. → pages 43, 66, 68, 69

[165] M. Soos. CryptoMiniSat v4. In *Proceedings of the 2014 SAT Competition: Solver and Benchmark Descriptions*, page 23, 2014. → page 107

[166] J. T. Springenberg, A. Klein, S. Falkner, and F. Hutter. Bayesian optimization with robust Bayesian neural networks. *The Thirtieth Conference on Advances in Neural Information Processing Systems (NeurIPS 2016)*, 29:4134–4142, 2016. → pages 66, 95

[167] P. F. Stadler and W. Schnabl. The landscape of the traveling salesman problem. *Physics Letters A*, 161(4):337–344, 1992. → pages 35, 150

[168] R. Storn and K. Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997. → page 18

[169] E. Štrumbelj and I. Kononenko. Explaining prediction models and individual predictions with feature contributions. *Knowledge and Information Systems*, 41(3):647–665, 2014. → page 159

[170] T. Stützle and H. H. Hoos. $\mathcal{MAX}$–$\mathcal{MIN}$ ant system. *Future Generation Computer Systems*, 16(8):889–914, 2000. → page 21

[171] J. Styles and H. H. Hoos. Ordered racing protocols for automatically configuring algorithms for scaling performance. In *Proceedings of the Fifteenth Conference on Genetic and Evolutionary Computation (GECCO 2013)*, pages 551–558. ACM, 2013. → page 40

[172] J. Styles, H. H. Hoos, and M. Müller. Automatically configuring algorithms for scaling performance. In *Proceedings of the Sixth International Conference on Learning and Intelligent Optimization (LION 2012)*, pages 205–219, 2012. → pages 43, 136

[173] R. Tanabe. Analyzing adaptive parameter landscapes in parameter adaptation methods for differential evolution. In *Proceedings of the Twenty-Second International Genetic and Evolutionary Computation Conference (GECCO 2020)*, pages 645–653, 2020. → pages 16, 28, 29, 60

[174] S. Tari, N. Szczepanski, L. Mousin, J. Jacques, M.-E. Kessaci, and L. Jourdan. Multi-objective automatic algorithm configuration for the classification problem of imbalanced data. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2020)*, pages 1–8. IEEE, 2020. → page 155

[175] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the Nineteenth ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD 2013)*, pages 847–855, 2013. → page 48

[176] A. Tiwari and J. K. Hollingsworth. Online adaptive code generation and tuning. In *Proceedings of the Twenty-Fifth IEEE International Symposium on Parallel & Distributed Processing (IPDPS 2011)*, pages 879–892, 2011. → page 4

[177] A. Tiwari, M. A. Laurenzano, L. Carrington, and A. Snavely. Auto-tuning for energy usage in scientific applications. In *Proceedings of the Seventeenth International Conference on Parallel Processing (Euro-Par 2011)*, pages 178–187, 2011. → pages 4, 154, 155

[178] V. Torczon. On the convergence of pattern search algorithms. *SIAM Journal on Optimization*, 7(1):1–25, 1997. → page 157

[179] G. Treimun-Costa, E. Montero, G. Ochoa, and N. Rojas-Morales. Modelling parameter configuration spaces with local optima networks. In *Proceedings of the Twenty-Second International Genetic and Evolutionary*

Computation Conference (GECCO 2020), pages 751–759, 2020. → pages 20, 28, 32, 60, 150, 151

[180] A. Tsanas, M. A. Little, P. E. McSharry, and L. O. Ramig. Accurate telemonitoring of Parkinson's disease progression by noninvasive speech tests. *IEEE Transactions on Biomedical Engineering*, 57(4):884–893, 2010. → page 67

[181] W. A. van Aardt, A. S. Bosman, and K. M. Malan. Characterising neutrality in neural network error landscapes. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2017)*, pages 1374–1381. IEEE, 2017. → page 36

[182] K. Van Den Doel and U. Ascher. The chaotic nature of faster gradient descent methods. *Journal of Scientific Computing*, 51(3):560–581, 2012. → page 4

[183] L. Vanneschi, Y. Pirola, G. Mauri, M. Tomassini, P. Collard, and S. Verel. A study of the neutrality of boolean function landscapes in genetic programming. *Theoretical Computer Science*, 425:34–57, 2012. → page 36

[184] V. K. Vassilev, T. C. Fogarty, and J. F. Miller. Information characteristics and the structure of landscapes. *Evolutionary Computation*, 8(1):31–60, 2000. → pages 35, 150

[185] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17: 261–272, 2020. → pages 58, 89

[186] J.-P. Watson. An introduction to fitness landscape analysis and cost models for local search. In *Handbook of Metaheuristics*, pages 599–623. Springer International Publishing, 2010. → pages 4, 25, 30

[187] E. Weinberger. Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics*, 63(5):325–336, 1990. → pages 34, 150

[188] F. Willnecker, A. Brunnert, and H. Krcmar. Model-based energy consumption prediction for mobile applications. In *Proceedings of the Twenty-Eighth International Conference on Informatics for Environmental Protection (EnviroInfo 2014)*, pages 747–752, 2014. → page 154

[189] D. H. Wolpert and W. G. Macready. No free lunch theorems for search. Technical report, SFI-TR-95-02-010, Santa Fe Institute, 1995. → page 5

[190] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1): 67–82, 1997. → page 5

[191] S. Wright. The roles of mutation, inbreeding, crossbreeding, and selection in evolution. *Proceedings of the Eleventh International Congress of Genetics*, 8, 1932. → pages 4, 25

[192] S. J. Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015. → page 123

[193] A. Yakovlev, H. F. Moghadam, A. Moharrer, J. Cai, N. Chavoshi, V. Varadarajan, S. R. Agrawal, S. Idicula, T. Karnagel, S. Jinturkar, and N. Agarwal. Oracle AutoML: A fast and predictive AutoML pipeline. *Proceedings of the Forty-Sixth International Conference on Very Large Data Bases (VLDB 2020)*, 13(12):3166–3180, 2020. → page 47

[194] H. Yang. *Energy prediction for I/O intensive workflow applications*. PhD thesis, University of British Columbia, 2014. → page 154

[195] Z. Yuan, T. Stützle, and M. Birattari. MADS/F-Race: Mesh adaptive direct search meets F-race. In *Proceedings of the Twenty-Third International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE 2010)*, pages 41–50, 2010. → pages 16, 21, 40, 46

[196] Z. Yuan, M. A. M. De Oca, M. Birattari, and T. Stützle. Continuous optimization algorithms for tuning real and integer parameters of swarm intelligence algorithms. *Swarm Intelligence*, 6(1):49–75, 2012. → pages 12, 16, 17, 21, 28, 40, 60, 100, 124, 150

[197] X. Zhang, J. Lu, and X. Qin. BFEPM: Best fit energy prediction modeling based on CPU utilization. In *Proceedings of the Eighth IEEE International Conference on Networking, Architecture and Storage (NAS 2013)*, pages 41–49, 2013. → page 154

Appendix A

Supporting Materials

A.1 Supporting Materials for Chapter 3

Theorem 1 (Correctness of Test for Uni-Modality)**.** *Let $G' = (V', E')$ be a neighbourhood relation graph defined for a landscape that contains a set of pre-evaluated configurations C, such that each configuration $c \in C$ has a corresponding confidence interval $[\underline{m}(c), \overline{m}(c)]$ for the performance of the algorithm. If $\overline{m}(c^*) \leq \overline{m}(c)$ for all $c \in C$, and there exists $c_0 \in C$ that is not reachable from c^*, then no uni-modal, piece-wise affine function exists that is contained within the confidence intervals, and hence uni-modality can be rejected for the landscape.*

Proof. Let c^* be a global minimum of C, that is,

$$\overline{m}(c^*) \leq \overline{m}(c) \text{ for all } c \in C, \tag{A.1}$$

and let $c_0 \in C$ be a configuration that is not reachable from c^*. Because this c_0 exists, we cannot construct a piece-wise affine function that is contained within the confidence intervals of C that also contains c^* as a global minima. It remains to be shown that this is a sufficient condition to conclude that no uni-modal, piece-wise affine function exists that is contained within the confidence intervals of C.

Assumption. For eventual contradiction, assume that a piece-wise affine uni-modal function does exists within the confidence intervals of C. This function must have

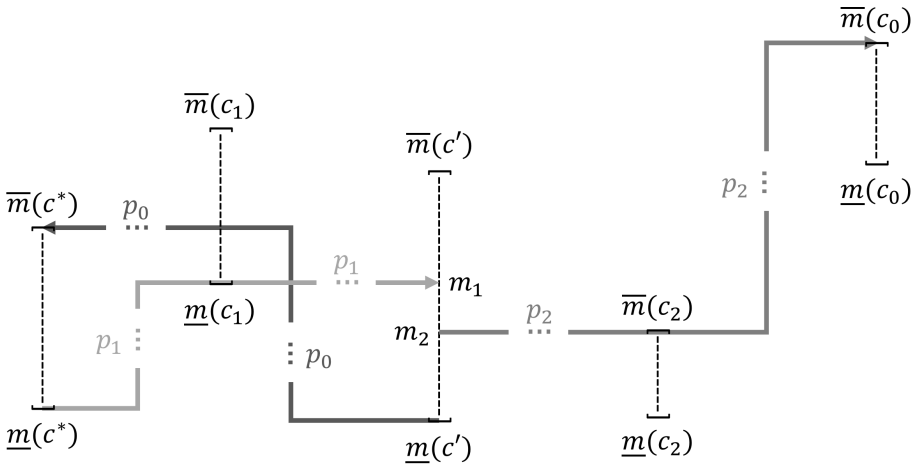

Figure A.1: Illustration of the items used in the proof of correctness for the test for statistically significant deviations from uni-modality.

at least one global minimum $c' \in C$ from which all other points $c \in C$ must be reachable. We will show (1) that there must exist a path p_1 from c^* to c' and a path p_2 from c' to c_0 (see Figure A.1); however, since they cannot be concatenated (otherwise c_0 would be reachable from c^*), we will show (2) that c^* must not have been a global minimum of C, which is a contradiction.

Step 1. First, we show that there must exist a path p_1 from $(c^*, \underline{m}(c^*))$ to (c', m_1). Since c^* is reachable from c', there must be a path p_0 from $(c', \underline{m}(c'))$ to $(c^*, \overline{m}(c^*))$. Furthermore, based on the definition of reachability, the height of path p_0 must never exceed $\overline{m}(c^*)$, therefore $\underline{m}(c) \leq \overline{m}(c^*)$ for all c on p_0. Combining this with Equation A.1, we obtain

$$\underline{m}(c) \leq \overline{m}(c^*) \leq \overline{m}(c) \text{ for all } c \text{ on } p_0. \tag{A.2}$$

Hence, we can clearly construct at least one path from c^* to c' with height less than or equal to $\overline{m}(c^*)$ for all c on p_0 that is the reverse of p_0. Let p_1 be the path from $(c^*, \underline{m}(c^*))$ to (c', m_1) with minimal height at all configurations c on p_1. The final height, m_1, of p_1 must be constrained by the lower bound of the configuration, c_1,

that has the smallest lower bound in p_1, that is, $m_1 = \underline{m}(c_1) \leq \underline{m}(c)$ for all c on p_1. Combining this with Equation A.2, we have

$$\underline{m}(c_1) = m_1 \leq \overline{m}(c^*). \tag{A.3}$$

Step 2. Let p_2 be the certifying path from (c', m_2) to $(c_0, \overline{m}(c_0))$ with the maximum possible height for all configurations c on p_2. The beginning height, m_2, of p_2 must be constrained by the upper bound of the configuration, c_2, that has the smallest upper bound in p_2, that is,

$$m_2 = \overline{m}(c_2) \leq \overline{m}(c) \text{ for all } c \text{ on } p_2. \tag{A.4}$$

Since c_0 is not reachable from c^*, we must not be able to concatenate the paths p_1 (from c^* to c') and p_2 (from c' to c_0). This can only happen if the height, m_1, at the end of p_1 is above the height, m_2, at the beginning of p_2, that is,

$$m_2 < m_1. \tag{A.5}$$

Contradiction. Finally, combining Equations A.3, A.4 and A.5, we obtain

$$\overline{m}(c_2) = m_2 < m_1 \leq \overline{m}(c^*), \tag{A.6}$$

that is,

$$\overline{m}(c_2) < \overline{m}(c^*). \tag{A.7}$$

However, this contradicts our precondition that c^* is a global minimum of C (see Equation A.1). Therefore, our original assumption must be false: no uni-modal, piece-wise affine function exists within the confidence intervals of C. ■

A.2 Supporting Materials for Chapter 4

In this section, we include the hyper-parameter grid used to evaluate Xgboost (see Appendix A.2.1), we derive the maximum barrier height for a local minimum in

online LDA's landscape (see Appendix A.2.2), and we include several tables with additional results from the analysis of the hyper-parameter interactions in the AutoML loss landscapes (see Appendix A.2.3).

A.2.1 Xgboost AutoML Loss Landscape Hyper-parameter Grid

The grid of hyper-parameters configurations we used to evaluate Xgboost on the covertype dataset in Chapter 4 can be obtained by taking the cross product of the lists of hyper-parameter values in Table A.1

Table A.1: XGBoost hyper-parameter grid.

Hyper-parameter	Grid of Values
`eta`	[0, 0.15, ..., 0.9]
`gamma`	[0, 5, 10]
`max_dept`	[2, 6, 10]
`min_child_weight`	[1, 10.5, 20]
`max_delta_step`	[0, 5, 10]
`subsample`	[0.01, 0.505, 1]
`colsample_bytree`	[0.5, 0.75, 1.0]
`colsample_bylevel`	[0.5, 0.75, 1.0]
`lambda`	[1, 5.5, 10]
`alpha`	[0, 5, 10]
`num_round`	[50, 150, 250]

A.2.2 Maximum Barrier Height for online LDA

Let c_a be a configuration that is a sub-optimal local minimum with a corresponding barrier c_b, blocking it from the optimal configuration c^*. That is, c_b is a configuration with the smallest loss through which any path from c_a to the c^* must pass. Technically, a barrier c_b for a local-minimum c_a is defined to be any configuration such that

$$m(c_b) = \min_{p_{c_a \to c^*}} \max_{c \text{ on } p} m(c), \tag{A.8}$$

where $p_{c_a \to c^*}$ denotes a path from c_a to c^*.

Let the height of a barrier c_b for local minimum c_a be defined as

$$H(c_b, c_a) := m(c_b) - m(c_a). \tag{A.9}$$

Let s define the size of the intervals for a configuration c, that is,

$$\begin{aligned} \overline{m}(c) &:= m(c) \cdot (1+s) \quad \text{and} \\ \underline{m}(c) &:= m(c) \cdot (1-s). \end{aligned} \tag{A.10}$$

If a test for uni-modality fails to reject its null hypothesis for a given interval size s, then we can conclude that any barriers within the landscape must be small enough that they are contained within the intervals (otherwise uni-modality would be rejected). Therefore, the value s can be used to calculate an upper-bound on the height of any barriers in the landscape.

Theorem 2 (Maximum Barrier Height)**.** *Any interval size, $s < 1$, for which the landscape cannot have uni-modality rejected, bounds the height, $H(c_b, c_a)$, of any barrier, c_b, for any local minima, c_a, such that*

$$H(c_b, c_a) \leq m(c_a) \cdot \left(\frac{1+s}{1-s} - 1 \right). \tag{A.11}$$

Proof. By definition, since interval size s yields a landscape for which uni-modality cannot be rejected, we know that c_a must be reachable from c^*, and hence

$$\underline{m}(c_b) \leq \overline{m}(c_a) \iff 0 \leq \overline{m}(c_a) - \underline{m}(c_b). \tag{A.12}$$

Trivially,

$$\begin{aligned} H(c_b, c_a) &= m(c_b) - m(c_a) \\ &\leq m(c_b) - m(c_a) + (\overline{m}(c_a) - \underline{m}(c_b)) \\ &\leq (m(c_b) - \underline{m}(c_b)) + (\overline{m}(c_a) - m(c_a)). \end{aligned} \tag{A.13}$$

From the definition in Equation A.10, we know that

$$\begin{aligned} m(c_b) - \underline{m}(c_b) &= s \cdot m(c_b) \quad \text{and} \\ \overline{m}(c_a) - m(c_a) &= s \cdot m(c_a), \end{aligned} \tag{A.14}$$

which we can substitute into Equation A.13 to obtain

$$m(c_b) - m(c_a) \leq s \cdot m(c_b) + s \cdot m(c_a). \tag{A.15}$$

By re-arranging and solving for $m(c_b)$ we obtain

$$m(c_b) \leq m(c_a) \cdot \frac{1+s}{1-s}, \tag{A.16}$$

which we can plug back into the definition in Equation A.9 to obtain the desired result. ■

Using Theorem 2, and the fact that a confidence interval of size $s = 0.0136$ yields a landscape for online lDA for which uni-modality cannot be rejected, we know that the maximum height of any barrier for any sub-optimal local minimum in the landscape can be no more than $\left(\frac{1+0.0136}{1-0.0136} - 1\right) \approx 0.0276$ times the height of the local minima.

A.2.3 Extended Hyper-Parameter Interaction Results

In Table A.2 we show the results of our analysis on the two-dimensional slices of the landscapes. Throughout the remainder of the tables, we denote by "–" a result does not exist because there are not enough hyper-parameters to compute the indicated quantity. In the case of the higher-order interaction tables wherein we excluded the worst 50% or 75% of the landscapes, we denote by "nan" the scenarios for which the exclusion of the indicated fraction of the configurations resulted in us being unable to calculate any of the indicated partial derivatives because all of them contained at least one such censored configuration. In Table A.3 we show the remaining partial derivative significance summary for the scenarios with more than seven hyper-parameters. In Table A.4 we show the sum of the variance explained by each order n of hyper-parameter interactions for each landscape. In Tables A.5, A.6, A.7 and A.8 we show the same analysis used to determine the fraction of the landscapes with locally significant partial derivatives of various orders; however, in the first and second two sets of tables we drop the worst 50% and 75% of the configurations from the analysis, respectively.

Table A.2: The mean percentage of locally significant hyper-parameter interactions based on an analysis of the partial derivatives (Sig ∂^2), the 2nd order fANOVA scores (fANOVA), and the mean probability of obtaining a configuration that is tied with optimal if each hyper-parameter is configured once sequentially in a random order (Tied $\overline{w}$ Opt). All results are for the two-dimensional hyper-parameter response slices centered around the global optima of the AutoML loss landscapes.

Type	# Slices	Model	Dataset	Interval	Sig ∂^2	fANOVA	Tied $\overline{w}$ Opt
N×N	15	FCNet	NP	95%	52.80%	15.05%	76.67%
			PS	95%	54.56%	4.26%	76.67%
			PT	95%	50.56%	10.20%	83.33%
			SL	95%	61.53%	11.36%	86.67%
	3	LSSVM	UP	95%	1.78%	5.38%	83.33%
	6	LogReg	MNIST	95%	20.97%	10.17%	91.67%
	55	XGB	CT	95%	20.91%	9.35%	84.55%
	3	OLDA	Wiki	± 8.70%	0.00%	16.61%	100.00%
				± 0.16%	75.81%	16.61%	50.00%
				± 0.00%	100.00%	16.61%	50.00%
N×C	18	FCNet	NP	95%	63.89%	18.95%	72.22%
			PS	95%	60.93%	4.14%	80.56%
			PT	95%	43.89%	15.15%	72.22%
			SL	95%	63.70%	15.26%	86.11%
C×C	3	FCNet	NP	95%	0.00%	17.24%	66.67%
			PS	95%	33.33%	3.38%	100.00%
			PT	95%	66.67%	28.60%	50.00%
			SL	95%	33.33%	2.81%	83.33%

Table A.3: The hyper-parameter partial derivative significance result summary – part 2.

Model	Dataset	Interval	7^{th}	8^{th}	9^{th}	10^{th}	11^{th}
FCNet	SL	95%	22.05	21.08	19.80	–	–
	PS	95%	23.23	23.36	23.80	–	–
	NP	95%	23.92	23.70	24.27	–	–
	PT	95%	22.24	22.05	20.20	–	–
XGB	CT	95%	2.92	2.97	3.05	3.14	3.04

A.3 Supporting Materials for Chapter 6

In this section, we show figures of the remaining one-dimensional parameter response slices not shown in Chapter 6 (see Appendix A.3.1), we include a discussion of the results of applying our tests for uni-modality and convexity to the one-dimensional parameter response slices on the individual problem instances (see

Table A.4: The hyper-parameter fANOVA importance result summary.

Model	Dataset	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	11th
FCNet	SL	0.44	2.80	9.75	20.46	27.13	23.03	12.20	3.70	0.49	–	–
	PS	4.50	8.88	14.22	15.50	16.13	17.01	14.68	7.63	1.46	–	–
	NP	0.15	0.48	2.58	9.05	20.21	28.58	24.69	11.85	2.41	–	–
	PT	0.27	0.82	3.36	10.53	21.58	28.27	22.82	10.34	2.01	–	–
XGB	CT	58.94	12.96	14.47	9.24	3.70	0.24	0.15	0.14	0.10	0.11	0.02
LogReg	MNIST	76.25	21.89	1.76	0.11	–	–	–	–	–	–	–
LSSVM	UP	87.39	11.59	1.01	–	–	–	–	–	–	–	–
OLDA	Wiki	70.16	29.50	0.34	–	–	–	–	–	–	–	–

Table A.5: The hyper-parameter partial derivative significance excluding the worst 50% of the configurations – part 1

Model	Dataset	Interval	1st	2nd	3rd	4th	5th	6th
FCNet	SL	95%	74.72	49.51	36.26	30.36	27.40	25.63
	PS	95%	81.17	50.58	33.46	26.70	23.98	23.05
	NP	95%	64.97	43.78	33.42	28.40	26.01	24.56
	PT	95%	77.29	50.04	32.37	25.60	22.90	21.84
XGB	CT	95%	54.98	13.41	4.34	3.44	3.36	3.36
LogReg	MNIST	95%	32.70	11.18	4.79	3.01	–	–
LSSVM	UniProbe	95%	6.42	0.00	0.00	–	–	–
OLDA	Wiki	± 1.36%	41.92	0.42	0.00	–	–	–
		± 5.10%	1.23	0.00	0.00	–	–	–

Appendix A.3.2), and we discuss a more in-depth analysis of the results for the FDC of the one-dimensional parameter response slices that uses bootstrap confidence intervals (see Appendix A.3.3).

A.3.1 Remaining Interesting Parameter Response Slices

We plot the 14 additional interesting one dimensional parameter response slices for the running time minimization algorithm configuration scenarios in Figures A.2 and A.3.

A.3.2 Uni-Modality and Convexity on Individual Instances

Since algorithm configuration landscapes are in fact comprised of the mean response to parameters over a set of instances, we ran our tests for convexity and uni-modality on the one-dimensional parameter response slices for each individual problem instance in the running time minimization scenarios to determine to what

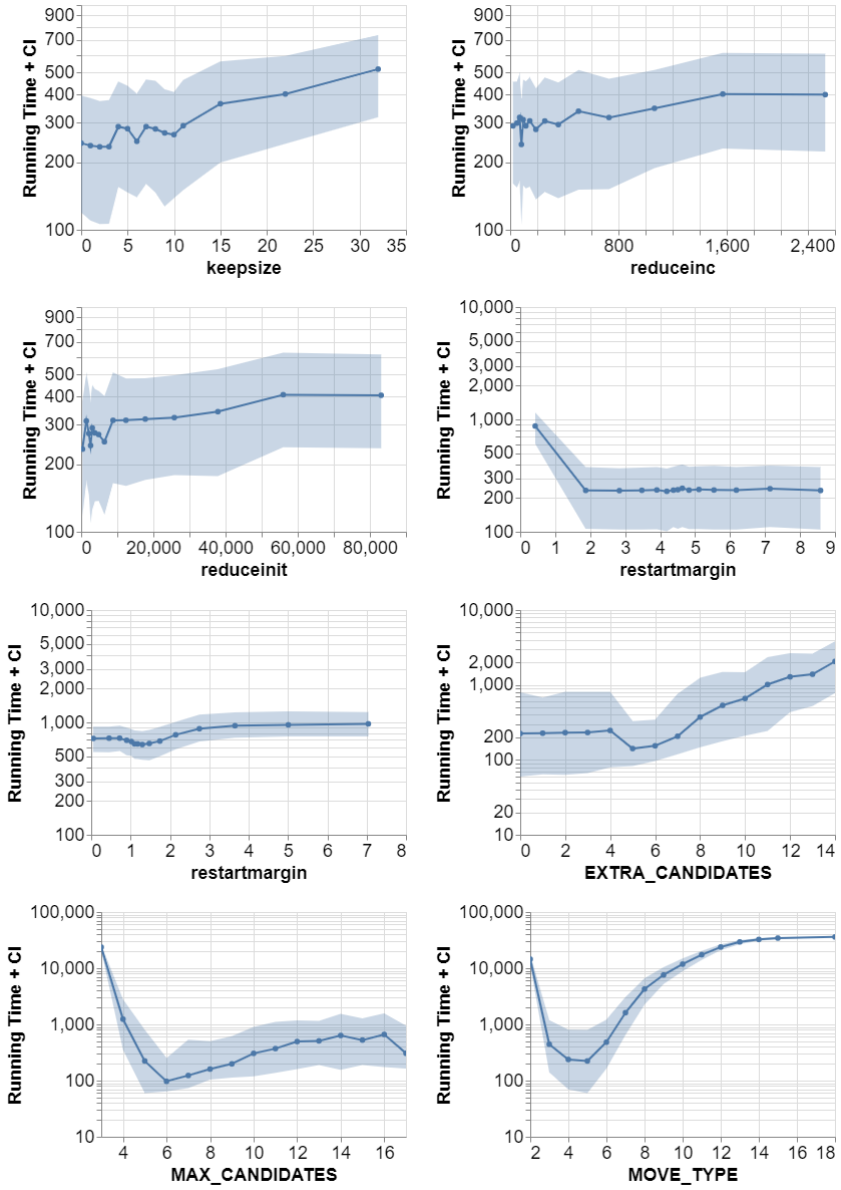

Figure A.2: From left to right and top to bottom: CaDiCaL's `keepsize`, `reduceinc`, `reduceinit` and `restartmargin` on the circuit-fuzz instance set and `restartmargin` on the BMC08 instance set; and, LKH's `EXTRA_CANDIDATES`, `MAX_CANDIDATES` and `MOVE_TYPE` on the TSP-RUE-1000-3000 instance set.

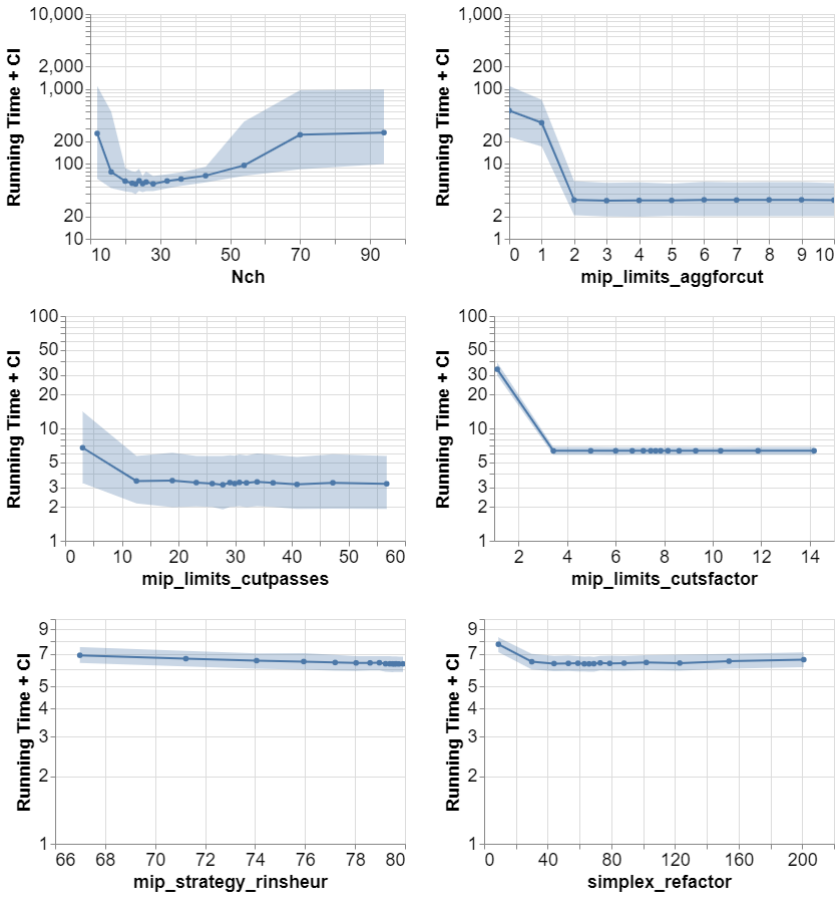

Figure A.3: From left to right and top to bottom: EAX's `Nch` on the TSP-RUE-1000-3000 instances; and CPLEX's `mip_limits_aggforcut` and `mip_limits_cutpasses` on the capacitated lot sizing (CLS) instances and `mip_limits_cutsfactor`, `mip_strategy_rinsheur` and `simplex_refactor` on the Regions200 instances.

Table A.6: Hyper-parameter partial derivative significance excluding the worst 50% of the configurations – part 2

Model	Dataset	Interval	7^{th}	8^{th}	9^{th}	10^{th}	11^{th}
FCNet	SL	95%	24.43	25.36	26.74	–	–
	PS	95%	23.11	24.60	22.00	–	–
	NP	95%	24.07	23.53	33.33	–	–
	PT	95%	21.98	23.34	21.05	–	–
XGB	CT	95%	3.41	3.47	3.51	3.40	3.70

Table A.7: Hyper-parameter partial derivative significance excluding the worst 75% of the configurations – part 1.

Model	Dataset	Interval	1^{st}	2^{nd}	3^{rd}	4^{th}	5^{th}	6^{th}
FCNet	SL	95%	75.30	50.46	35.61	29.34	26.52	23.74
	PS	95%	81.34	51.28	32.41	26.60	24.65	nan
	NP	95%	63.91	43.22	32.76	27.36	nan	nan
	PT	95%	72.67	46.60	30.42	24.86	nan	nan
XGB	CT	95%	64.86	15.53	3.95	3.35	3.28	3.26
LogReg	MNIST	95%	38.25	14.34	6.44	3.82	–	–
LSSVM	UniProbe	95%	6.11	0.00	0.00	–	–	–
OLDA	Wiki	$\pm$ 1.36%	11.70	0.00	0.00	–	–	–
		$\pm$ 5.10%	0.00	0.00	0.00	–	–	–

extent our results hold for individual problem instances. While less useful than our analysis on of the landscapes on the entire instance sets, this analysis nevertheless provides additional insights into algorithm configuration landscapes for running time minimization scenarios.

We consider the parameters independently by looking at statistics of their responses on each instance. For example, on the left pane of Figure A.4 we plot a cumulative distribution function (CDF) showing on the y-axis the percentage of parameters that had convex responses on a percentage of instances less than or equal to the value specified on the x-axis. Surprisingly, there is a large percentage of parameters with convex responses slices for most instances. However, nearly half of the parameters with interesting response slices on the entire instance set tend to have much fewer convex parameter responses on the individual instances. Our procedures (outlined in Chapters 3.2.1 and 3.2.3), sometimes assume uni-modality or convexity when there is insufficient data to perform a test because too many of the parameter values yielded running times above the running time cutoff. On average,

Table A.8: Hyper-parameter partial derivative significance excluding the worst 75% of the configurations – part 2.

Model	Dataset	Interval	7^{th}	8^{th}	9^{th}	10^{th}	11^{th}
FCNet	SL	95%	nan	nan	nan	–	–
	PS		nan	nan	nan	–	–
	NP		nan	nan	nan	–	–
	PT		nan	nan	nan	–	–
XGB	CT		nan	nan	nan	nan	nan

over all parameters, this happened for only 6.2% of the instances we considered (the parameter for which this happened the most had 16.9% of the instances with too few un-censored data points). Hence, even if all of these cases were instead assumed to be non-unimodal or non-convex, our overall results would not be substantially different.

Furthermore, looking at the CDF of the average numbers of modes for each instance parameter response slice on the right pane of Figure A.4, we see that just under 50% of the interesting parameters have an average of more than one mode for their individual problem instance responses. On the other hand, most of the parameters have an average of only one mode per instance, which is consistent with the fraction of parameters with primarily convex instance response slices.

Overall, there are a surprisingly large number of parameter response slices that are both uni-modal and convex on most or all of their individual problem instances. In Table A.9, we show a summary of these results. For comparison, we also include the corresponding results for the one-dimensional parameter responses on entire instance sets. Note that for the aggregate instance set parameter responses, we show the percentage of uni-modal and convex parameter response slices observed on different instance sets, whereas for the individual instances, we first computed the percentage of instances with uni-modal and convex responses for each parameter, and then report the average percentages over the set of all parameters.

Our analysis of the fitness distance correlation coefficient (FDC) for the parameter response slices supports our hypothesis that parameter responses on individual instances are more rugged than the aggregate responses on entire instance sets. In particular, we found that 80% of the parameters have an average instance response slice FDC less than 0.25, compared to 0.41 for the instance set responses. How-

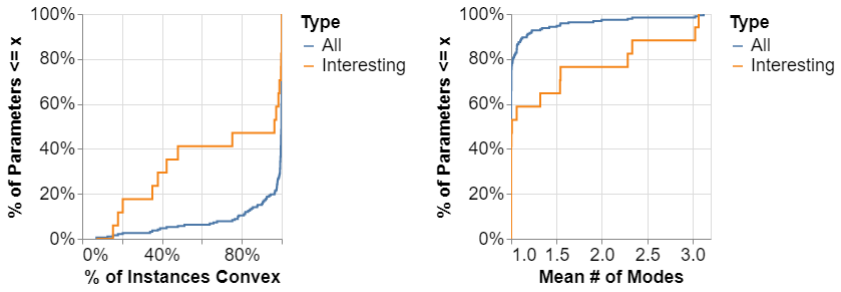

Figure A.4: Cumulative distribution functions (CDFs) that summarize our findings for the parameter response slices on individual instances. Left: for each parameter we computed the percentage of instances on which it had a convex response, and then we plot the CDF of these percentages; right: the CDF of the average number of modes observed in the responses for a parameter on each instance.

Table A.9: Left: the percentages of uni-modal and convex parameter response slices on entire instance sets; right: we computed the percentage of instances with convex or uni-modal responses for each parameter, and then show the average percentages over the parameters – that is, we show the percentage of convex and uni-modal responses for an "average" parameter on individual problem instances.

	Instance Sets		**Single Instances**	
	% Uni-M	**% Cvx**	**% Uni-M**	**% Cvx**
All Parameters	99.5	99.5	95.3	92.6
Interesting Parameters	94.4	94.4	76.1	66.1

ever, through manual inspection of the instance parameter slices, we found that some responses obtained low FDC scores simply because they are relatively flat (hence deviations in parameter value have low correlation with deviations in algorithm performance). Still, the high average numbers of modes observed for some of the parameters indicate that these responses are truly rugged.

To check that these were not spurious results, we performed exact replicates for three scenarios that were near the Pareto front of the largest average number of modes and the smallest average FDC: CaDiCaL's `posize` and `elimint` on circuit-fuzz instances, and CPLEX's `mip_limits_cutpasses` on CLS instances. Then, for each parameter, we chose three instances near to their respective

Pareto fronts. Through manual inspection of the original and replicate parameter responses in Figures A.5 and A.6, it is clear that in all cases the replicates were qualitatively similar to the original ones.

These results show that the parameter response slights on individual instances are somewhat more rugged than their counterparts on the full instance sets. This observation suggests that future work should be done to search for alternative forms of low-fidelity estimates of parameter configuration performance for running time minimization scenarios (as suggested in Chapter 8.3.2). If an alternative means of estimating configuration performance can be found for many scenarios, it may produce landscapes that are less rugged than these. Since less rugged landscapes are easier to optimize, such efforts may result a more effective means of reducing the cost of automated algorithm configuration for these scenarios than racing procedures alone can provide.

A.3.3 FDC Analysis with Bootstrap Confidence Intervals

In the left of Figure A.7 we plot the CDF of the FDC for the one-dimensional parameter response slices on the entire instance sets. Surprisingly, the variance in the FDC across the bootstrap samples of the parameter slices is very high and the median FDC is relatively low for most parameters, for example, 80% of the parameter responses have their median FDC less than 0.41. This large variance in the FDC indicates that the variation among the parameter responses on individual instances is quite high, meaning that the aggregate response on the instance set, and hence the FDC, depends strongly on the instances in a particular bootstrap sample. However, we can also see that the FDC tends to be higher for parameters with interesting, non-flat responses. As a result, we believe that the relatively low FDC values are primarily due to relatively small fluctuations occurring in large, flat regions of the parameter responses (which tend to occur for most parameters).

Despite the fact that many of the individual instance parameter response slices appear to be uni-modal and convex, we see in the right of Figure A.7 that the average FDC for many of the parameters is quite low, for example, 80% of the parameter response slices have an average instance FDC less than 0.25. while some of the instance parameter response slices have negative FDCs, this does not

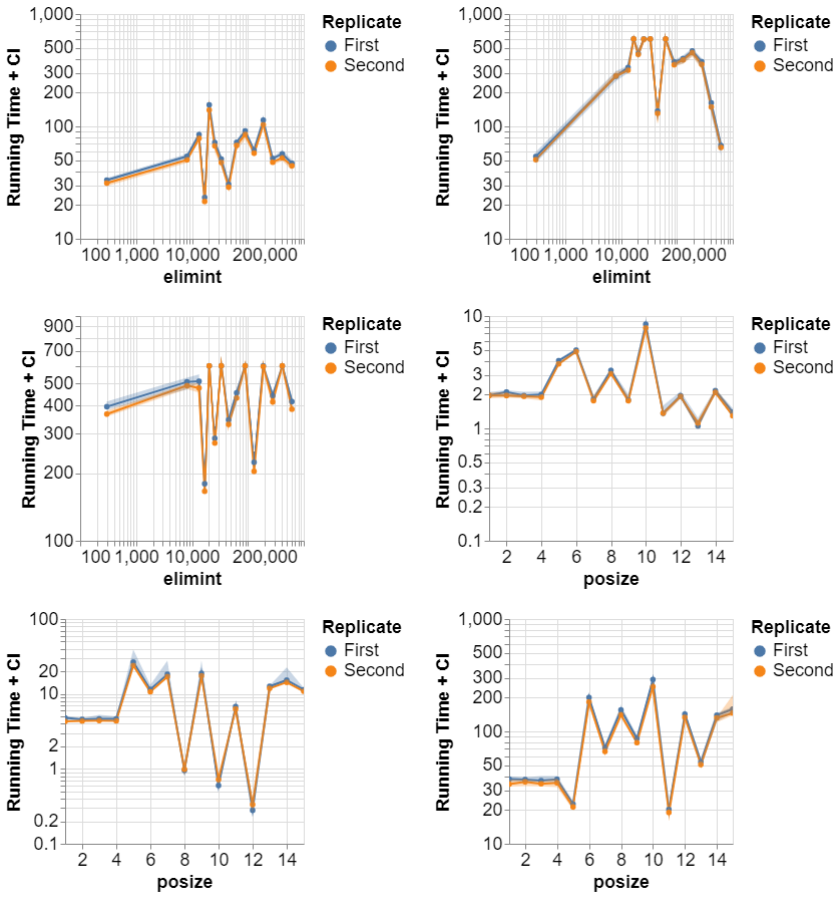

Figure A.5: Replicates of rugged parameter response slices for CaDiCaL's `elimint` and `posize` on three circuit-fuzz instances each. From top to bottom and left to right: `elimint` on fuzz_100_634.cnf, fuzz_100_30719.cnf and fuzz_100_9873.cnf; and `posize` on fuzz_100_5082.cnf, fuzz_100_29685.cnf and fuzz_100_16079.cnf.

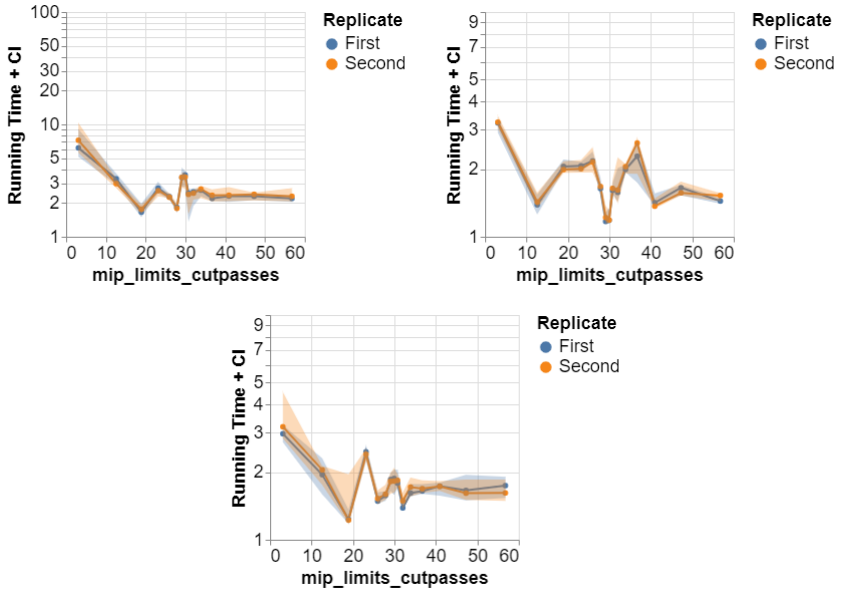

Figure A.6: Replicates of rugged parameter response slices for CPLEX's `mip_limits_cutpasses` on three capacitated lot sizing instances. From top to bottom and left to right: the instances named cls.T90.C3.F200.S3.mps, cls.T90.C3.F1000.S5.mps and cls.T90.C5.F200.S5.mps.

necessarily indicate that they are highly rugged (consider $\frac{1}{x} + \log(x)$, which has very high values close to the right of the optimum near 2 and relatively low values far to the left of 2). However, given that the bootstrap confidence intervals for the FDCs are very small compared to those for the instance set parameter responses, we can infer that the ruggedness we observe on individual instances is more often truly reflective of the nature of the parameters' responses, rather than being simply due variation between independent runs of the algorithms. We can again see that the parameters with interesting responses (on the instance sets), tend to have individual instance responses with a higher FDC, again indicating that some of the low FDC scores we are seeing are due to relatively flat regions of the parameter responses.

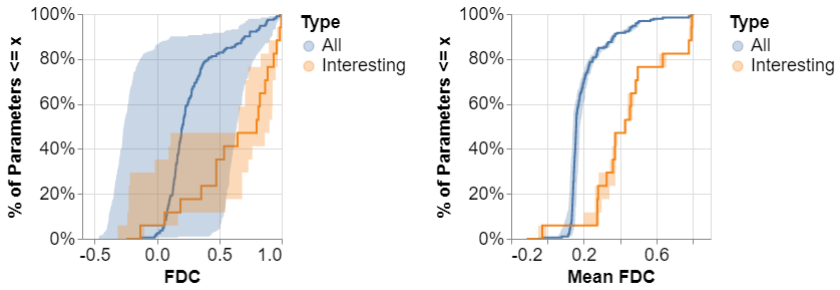

Figure A.7: Left: the CDF of the median and 95% confidence interval of the FDC coefficients of the parameter response slices on the entire instance sets; right: for each parameter we took the average FDC over each of the individual instance parameter response slices and we show the resulting CDF of the average median and 95% confidence intervals. The confidence intervals are based on bootstrap sampling.

A.4 Supporting Materials for Chapter 7

In Table A.10 we show the results from performing the large parallel budget analysis (see Chapter 7.2) for all of the anytime configurations that we evaluated. In Table A.11 we show the same results, but using the small parallel budget analysis (see Chapter 7.2). In both of these tables, we mark in boldface the configurations with speedups that are not statistically worse than the best speedup within each cell, according to a permutation test with a 5% significance level.

While each configuration procedure had the same total configuration budget within in cell, this does mean that they all required the same amount of time to validate the configurations. Table A.12 shows the total wall clock time (in hours) for configuration and validation for our large parallel budget analysis. These numbers assume that the validation times can be perfectly parallelized using the same number of processors as used during configuration. We also show the same results for the small parallel budget analysis in Table A.13. However, we note that in this case, since SMAC is the only configurator that requires validation of a set of configurations, all the other methods required exactly the configuration budget times. Rather than highlighting the configurators which had the smallest validation times, in these tables we mark in boldface the corresponding entries to the tables above. That is, to compare the configuration + validation budget of only the configurators

Table A.10: The large parallel budget analysis speedups (with medians and 95% confidence intervals). Median speedups not worse than the best speedup for each time budget according to a permutation test with a 5% signifance level are shown in boldface.

	TSP		SAT		MIP	
	LKH	EAX	CaDiCaL	probSAT	CPLEX	
	TSP RUE 1000-3000		Circuit Fuzz	7SAT90	Regions200	RCW2
Configuration budget (excluding validation) = 0.5 wall clock hours						
GPS	**1.00** [0.92, 1.05]	2.13 [2.13, 3.55]	1.11 [1.00, 1.33]	2.26 [2.26, 2.26]	**1.03** [1.00, 1.03]	1.03 [0.99, 1.05]
SMAC3.0	**1.00** [1.00, 1.00]	**2.83** [2.49, 2.95]	**1.25** [1.09, 1.36]	**6.72** [5.57, 16.60]	**1.00** [0.95, 1.22]	**1.26** [1.21, 1.27]
irace3.3	**1.00** [1.00, 1.00]	**2.67** [2.08, 2.81]	0.94 [0.13, 0.94]	**9.50** [4.91, 9.50]	**1.00** [0.01, 1.00]	1.00 [1.00, 1.00]
GGA++	0.59 [0.37, 0.72]	0.61 [0.61, 0.61]	0.84 [0.75, 0.85]	**7.30** [5.86, 8.89]	–	–
Configuration budget (excluding validation) = 1.0 wall clock hours						
GPS	**0.99** [0.92, 1.00]	**3.14** [3.05, 3.38]	1.11 [1.00, 1.33]	2.26 [2.26, 3.76]	**1.00** [1.00, 1.05]	1.04 [0.99, 1.05]
SMAC3.0	**1.00** [1.00, 1.00]	**2.63** [2.49, 3.10]	**1.33** [1.16, 1.36]	**7.10** [6.72, 16.60]	**1.00** [0.85, 1.02]	**1.26** [1.21, 1.27]
irace3.3	**1.00** [1.00, 1.00]	**2.81** [2.67, 3.09]	0.68 [0.13, 0.94]	**6.92** [4.91, 9.50]	0.37 [0.01, 1.00]	1.00 [1.00, 1.00]
GGA++	0.60 [0.43, 0.65]	0.61 [0.61, 0.61]	0.85 [0.75, 1.02]	**7.30** [5.86, 8.89]	–	–
Configuration budget (excluding validation) = 3.0 wall clock hours						
GPS	**1.00** [0.94, 1.20]	**3.05** [2.36, 3.21]	**1.35** [1.00, 1.44]	3.76 [1.93, 4.73]	**1.05** [0.95, 1.30]	1.05 [1.00, 1.05]
SMAC3.0	**1.00** [1.00, 1.00]	**2.65** [2.62, 3.55]	**1.25** [1.09, 1.36]	**7.73** [4.28, 13.41]	**1.00** [0.91, 1.18]	**1.26** [1.22, 1.27]
irace3.3	**1.00** [0.76, 1.00]	2.17 [2.17, 2.81]	0.87 [0.84, 0.98]	**7.01** [4.81, 8.95]	**1.00** [0.01, 1.00]	1.00 [1.00, 1.00]
GGA++	0.72 [0.54, 1.01]	0.61 [0.61, 0.61]	0.89 [0.81, 1.02]	**7.30** [5.86, 8.89]	–	–
Configuration budget (excluding validation) = 6.0 wall clock hours						
GPS	**1.20** [0.95, 1.20]	**3.05** [2.82, 3.21]	**1.41** [1.12, 1.41]	**3.76** [1.93, 4.73]	**1.16** [1.00, 1.31]	1.01 [1.00, 1.05]
SMAC3.0	1.00 [1.00, 1.00]	**2.62** [2.53, 3.55]	1.25 [1.09, 1.36]	**4.90** [4.28, 13.41]	0.96 [0.77, 1.18]	**1.26** [1.22, 1.27]
irace3.3	0.94 [0.92, 1.11]	2.48 [1.85, 3.38]	0.87 [0.83, 0.95]	**4.90** [4.90, 13.06]	0.01 [0.01, 1.00]	1.00 [1.00, 1.00]
GGA++	0.66 [0.58, 1.05]	0.61 [0.61, 0.61]	0.92 [0.89, 1.02]	**7.14** [5.38, 8.58]	–	–
Configuration budget (excluding validation) = 12.0 wall clock hours						
GPS	**1.20** [1.01, 1.22]	**3.21** [2.82, 3.21]	**1.51** [1.31, 1.54]	3.03 [1.93, 5.52]	0.14 [0.14, 0.18]	1.11 [1.00, 1.33]
SMAC3.0	1.00 [1.00, 1.00]	**2.93** [2.62, 3.08]	1.26 [1.16, 1.36]	**4.90** [4.28, 13.41]	**1.18** [0.78, 1.23]	**1.26** [1.22, 1.27]
irace3.3	1.03 [0.79, 1.14]	**2.72** [2.33, 3.00]	0.90 [0.84, 1.01]	**5.86** [4.97, 12.92]	1.00 [0.01, 1.00]	1.00 [1.00, 1.00]
GGA++	1.01 [0.77, 1.01]	0.61 [0.61, 0.61]	0.91 [0.87, 1.02]	**8.87** [4.35, 19.84]	–	–
Configuration budget (excluding validation) = 24.0 wall clock hours						
GPS	**1.21** [1.18, 1.28]	**3.22** [2.36, 3.46]	**1.44** [1.16, 1.55]	3.03 [1.93, 5.52]	0.68 [0.01, 1.12]	**1.41** [1.09, 1.41]
SMAC3.0	1.00 [1.00, 1.00]	2.73 [2.62, 3.08]	**1.36** [1.16, 1.60]	**5.76** [4.28, 13.41]	**1.18** [0.77, 1.23]	1.26 [1.22, 1.27]
irace3.3	1.03 [0.79, 1.14]	2.72 [2.33, 3.00]	0.90 [0.84, 1.01]	**5.86** [4.97, 12.92]	0.01 [0.01, 1.00]	1.00 [1.00, 1.00]
GGA++	1.02 [0.77, 1.11]	0.61 [0.61, 0.61]	1.02 [0.84, 1.02]	**8.87** [6.73, 8.89]	–	–

which found the best configurations, one only needs to look at the budgets that are marked in boldface.

Table A.11: The small parallel budget analysis speedups (with medians and 95% confidence intervals). Median speedups not worse than the best speedup for each time budget according to a permutation test with a 5% signifance level are shown in boldface.

	TSP		SAT		MIP	
	LKH	EAX	CaDiCaL	probSAT	CPLEX	
	TSP RUE 1000-3000		Circuit Fuzz	7SAT90	Regions200	RCW2
Configuration budget (excluding validation) = 0.5 wall clock hours						
GPS	**0.95** [0.92, 1.05]	2.13 [2.13, 3.55]	1.01 [1.00, 1.33]	2.26 [1.00, 2.26]	**1.00** [0.88, 1.03]	1.00 [0.99, 1.05]
SMAC3.0	**1.00** [1.00, 1.00]	**2.83** [2.46, 3.36]	**1.19** [1.01, 1.36]	**8.24** [5.36, 16.60]	0.96 [0.24, 1.22]	**1.22** [1.00, 1.27]
irace3.3	**1.00** [1.00, 1.00]	**2.59** [1.85, 2.81]	0.13 [0.13, 0.94]	**6.78** [4.67, 9.50]	0.01 [0.00, 1.00]	1.00 [1.00, 1.00]
GGA++	0.46 [0.37, 0.72]	0.61 [0.61, 0.61]	0.84 [0.52, 0.85]	**6.61** [3.49, 8.89]	–	–
Configuration budget (excluding validation) = 1.0 wall clock hours						
GPS	**0.99** [0.92, 1.00]	**3.14** [2.27, 3.38]	1.01 [1.00, 1.33]	2.26 [1.32, 3.76]	**1.00** [0.88, 1.05]	1.00 [0.99, 1.05]
SMAC3.0	**1.00** [1.00, 1.00]	**2.68** [2.40, 3.36]	**1.19** [1.01, 1.36]	**7.73** [5.37, 16.60]	0.96 [0.30, 1.02]	**1.22** [1.00, 1.27]
irace3.3	**1.00** [1.00, 1.00]	**2.76** [2.19, 3.09]	0.13 [0.13, 0.94]	**6.78** [4.67, 9.50]	0.01 [0.00, 1.00]	1.00 [1.00, 1.00]
GGA++	0.50 [0.43, 0.65]	0.61 [0.61, 0.61]	0.76 [0.52, 1.02]	**6.61** [3.49, 8.89]	–	–
Configuration budget (excluding validation) = 3.0 wall clock hours						
GPS	**1.00** [0.94, 1.20]	**3.05** [2.36, 3.21]	1.12 [1.00, 1.44]	2.26 [1.32, 4.73]	**1.01** [0.90, 1.30]	1.00 [0.98, 1.05]
SMAC3.0	**1.00** [1.00, 1.00]	**2.68** [2.62, 3.55]	**1.19** [1.01, 1.36]	**8.25** [4.28, 13.41]	**0.98** [0.46, 1.18]	**1.22** [1.00, 1.27]
irace3.3	**1.00** [0.76, 1.05]	2.57 [2.17, 2.81]	0.85 [0.71, 0.98]	**7.01** [4.81, 9.03]	0.01 [0.00, 1.00]	1.00 [1.00, 1.00]
GGA++	0.58 [0.46, 1.01]	0.61 [0.61, 0.61]	0.85 [0.78, 1.02]	**6.61** [3.49, 8.89]	–	–
Configuration budget (excluding validation) = 6.0 wall clock hours						
GPS	**1.01** [0.95, 1.20]	**3.03** [2.36, 3.21]	**1.31** [1.08, 1.41]	3.03 [1.32, 4.73]	0.65 [0.20, 1.21]	1.00 [0.99, 1.05]
SMAC3.0	**1.00** [1.00, 1.00]	**2.75** [2.53, 3.55]	1.19 [1.01, 1.36]	**5.47** [4.28, 13.42]	**1.18** [0.62, 1.55]	**1.22** [1.00, 1.27]
irace3.3	**0.94** [0.86, 1.11]	2.33 [1.85, 3.38]	0.85 [0.80, 0.95]	**5.59** [4.90, 13.06]	0.15 [0.06, 1.00]	1.00 [0.00, 1.00]
GGA++	0.66 [0.58, 1.05]	0.61 [0.61, 0.61]	0.90 [0.82, 1.02]	**5.70** [5.36, 8.58]	–	–
Configuration budget (excluding validation) = 12.0 wall clock hours						
GPS	**1.12** [0.89, 1.22]	**3.03** [2.36, 3.21]	**1.36** [1.23, 1.54]	3.03 [1.51, 5.52]	0.14 [0.01, 0.18]	1.09 [0.96, 1.33]
SMAC3.0	**1.00** [1.00, 1.00]	**2.89** [2.62, 3.28]	1.19 [1.01, 1.36]	**6.82** [4.28, 13.42]	0.83 [0.21, 1.23]	**1.22** [1.00, 1.27]
irace3.3	0.91 [0.79, 1.14]	**2.52** [2.33, 3.00]	0.87 [0.84, 1.01]	**5.86** [4.97, 12.92]	0.01 [0.00, 1.00]	1.00 [0.00, 1.00]
GGA++	0.82 [0.61, 1.01]	0.61 [0.61, 0.61]	0.88 [0.81, 1.02]	**7.08** [4.35, 19.84]	–	–
Configuration budget (excluding validation) = 24.0 wall clock hours						
GPS	**1.20** [1.18, 1.28]	**2.96** [2.36, 3.46]	**1.44** [1.16, 1.55]	3.03 [1.51, 5.52]	0.14 [0.11, 1.08]	1.17 [1.05, 1.41]
SMAC3.0	1.00 [1.00, 1.00]	**2.75** [2.62, 3.12]	1.23 [1.00, 1.60]	**7.73** [4.28, 13.42]	**0.96** [0.44, 1.01]	**1.22** [1.00, 1.27]
irace3.3	0.91 [0.79, 1.14]	**2.52** [2.33, 3.00]	0.87 [0.84, 1.01]	**5.86** [4.97, 12.92]	0.15 [0.06, 1.00]	1.00 [0.00, 1.00]
GGA++	0.86 [0.70, 1.11]	0.61 [0.61, 0.61]	0.89 [0.84, 1.02]	**6.98** [5.70, 8.89]	–	–

Table A.12: Median configuration budgets, including validation time for the large parallel budget analysis. The configuration budgets that correspond to median speedups not worse than the best speedup for each time budget according to a permutation test with a 5% signifance level are shown in boldface.

	TSP		**SAT**		**MIP**	
	LKH	**EAX**	**CaDiCaL**	**probSAT**	**CPLEX**	
	TSP RUE 1000-3000		**Circuit Fuzz**	**7SAT90**	**Regions200**	**RCW2**
Configuration budget (excluding validation) = 0.5 wall clock hours						
GPS	**0.59**	0.58	1.06	1.01	**2.21**	0.91
SMAC3.0	**1.13**	**1.19**	**5.40**	**2.64**	**129.63**	**13.78**
`irace`3.3	**0.58**	**0.58**	2.80	**0.75**	**31.94**	0.92
GGA++	0.63	0.65	1.30	**0.74**	–	–
Configuration budget (excluding validation) = 1.0 wall clock hours						
GPS	**1.09**	**1.08**	1.56	1.41	**2.71**	1.41
SMAC3.0	**1.63**	**1.66**	**5.86**	**2.97**	**130.57**	**14.28**
`irace`3.3	**1.08**	**1.08**	3.30	**1.25**	43.03	1.42
GGA++	1.13	1.15	1.77	**1.24**	–	–
Configuration budget (excluding validation) = 3.0 wall clock hours						
GPS	**3.09**	**3.08**	**3.53**	3.38	**4.71**	3.42
SMAC3.0	**3.63**	**3.63**	**7.80**	**5.03**	**136.23**	**16.28**
`irace`3.3	**3.09**	3.08	3.70	**3.22**	**45.03**	3.42
GGA++	3.11	3.15	3.69	**3.24**	–	–
Configuration budget (excluding validation) = 6.0 wall clock hours						
GPS	**6.09**	**6.08**	**6.51**	**6.38**	**7.64**	6.42
SMAC3.0	6.63	**6.62**	10.78	**7.95**	145.79	**19.28**
`irace`3.3	6.09	6.08	6.70	**6.22**	48.03	6.42
GGA++	6.11	6.15	6.68	**6.24**	–	–
Configuration budget (excluding validation) = 12.0 wall clock hours						
GPS	**12.08**	**12.08**	**12.46**	12.38	25.27	12.39
SMAC3.0	12.63	**12.62**	16.75	**13.88**	**180.50**	**25.28**
`irace`3.3	12.10	**12.08**	12.67	**12.22**	54.03	12.42
GGA++	12.10	12.15	12.68	**12.27**	–	–
Configuration budget (excluding validation) = 24.0 wall clock hours						
GPS	**24.08**	**24.08**	**24.47**	24.38	24.38	**24.36**
SMAC3.0	24.63	24.62	**28.68**	**25.86**	**199.97**	37.27
`irace`3.3	24.10	24.08	24.68	**24.22**	66.03	24.42
GGA++	24.10	24.15	24.65	**24.23**	–	–

Table A.13: Median configuration budget, including validation time for the small parallel budget analysis. The configuration budgets that correspond to median speedups not worse than the best speedup for each time budget according to a permutation test with a 5% signifance level are shown in boldface.

	TSP		**SAT**		**MIP**	
	LKH	**EAX**	**CaDiCaL**	**probSAT**	**CPLEX**	
	TSP RUE 1000-3000		**Circuit Fuzz**	**7SAT90**	**Regions200**	**RCW2**
Configuration budget (excluding validation) = 0.5 wall clock hours						
GPS	**0.50**	0.50	0.50	0.50	**0.50**	0.50
SMAC3.0	**1.13**	**1.19**	**5.41**	**2.61**	120.33	**13.79**
`irace`3.3	**0.50**	**0.50**	0.50	**0.50**	0.50	0.50
GGA++	0.50	0.50	0.50	**0.50**	–	–
Configuration budget (excluding validation) = 1.0 wall clock hours						
GPS	**1.00**	**1.00**	1.00	1.00	**1.00**	1.00
SMAC3.0	**1.63**	**1.66**	**5.85**	**2.92**	125.51	**14.29**
`irace`3.3	**1.00**	**1.00**	1.00	**1.00**	1.00	1.00
GGA++	1.00	1.00	1.00	**1.00**	–	–
Configuration budget (excluding validation) = 3.0 wall clock hours						
GPS	**3.00**	**3.00**	3.00	3.00	**3.00**	3.00
SMAC3.0	**3.63**	**3.63**	**7.79**	**4.98**	**126.26**	**16.27**
`irace`3.3	**3.00**	3.00	3.00	**3.00**	3.00	3.00
GGA++	3.00	3.00	3.00	**3.00**	–	–
Configuration budget (excluding validation) = 6.0 wall clock hours						
GPS	**6.00**	**6.00**	**6.00**	6.00	**6.00**	6.00
SMAC3.0	**6.63**	**6.62**	10.76	**7.91**	145.31	**19.28**
`irace`3.3	**6.00**	6.00	6.00	**6.00**	6.00	6.00
GGA++	6.00	6.00	6.00	**6.00**	–	–
Configuration budget (excluding validation) = 12.0 wall clock hours						
GPS	**12.00**	**12.00**	**12.00**	12.00	12.00	12.00
SMAC3.0	**12.63**	**12.62**	16.74	**13.85**	173.50	**25.29**
`irace`3.3	12.00	**12.00**	12.00	**12.00**	12.00	12.00
GGA++	12.00	12.00	12.00	**12.00**	–	–
Configuration budget (excluding validation) = 24.0 wall clock hours						
GPS	**24.00**	**24.00**	**24.00**	24.00	24.00	24.00
SMAC3.0	24.63	**24.62**	28.68	**25.81**	198.42	**37.28**
`irace`3.3	24.00	**24.00**	24.00	**24.00**	24.00	24.00
GGA++	24.00	24.00	24.00	**24.00**	–	–