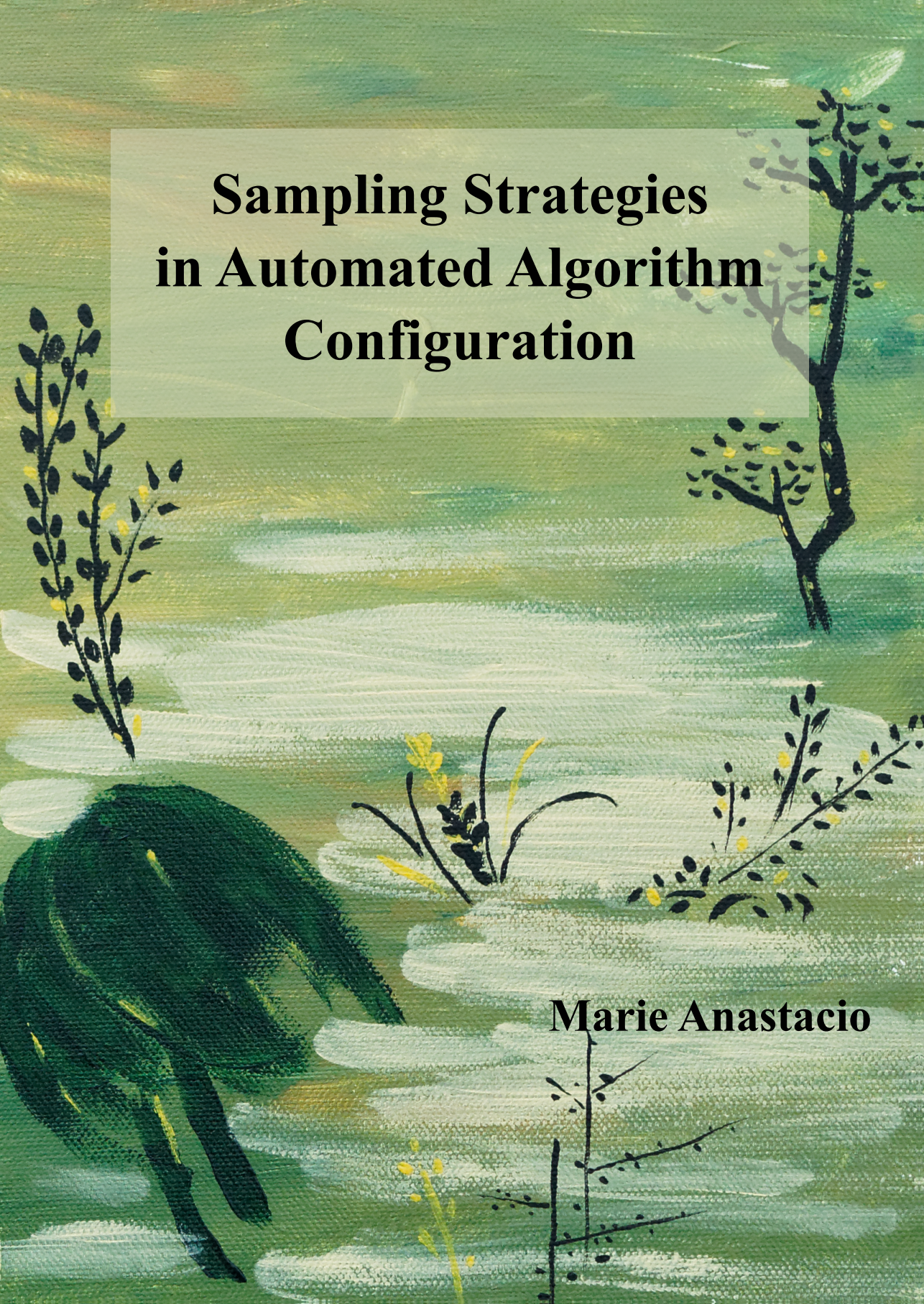
Sampling strategies in automated algorithm configuration

Anastacio, M.I.A.


Sampling Strategies
in Automated Algorithm
Configuration
Marie Anastacio

Sampling Strategies in Automated Algorithm Configuration

Proefschrift

ter verkrijging van
de graad van doctor aan de Universiteit Leiden,
op gezag van rector magnificus prof.dr. S. de Rijcke,
volgens besluit van het college voor promoties
te verdedigen op dinsdag 23 juni 2026
klokke 10:00 uur

door

Marie Inès Adélaïde Anastacio
geboren te Frankrijk
in 1989

Promotores:	Prof. Dr. H.H. Hoos	
	Prof. Dr. T.H.W. Bäck	
Promotiecomissie:		
	Prof. Dr. K.J. Batenburg	
	Prof. Dr. M.M. Bonsangue	
	Dr. C. Doerr	CNRS, Sorbonne Université (France)
	Prof. Dr. M. Lindauer	Leibniz University Hannover (Germany)
	Dr. E. Raponi	

Contents

Summary

When confronted with complex real-world problems, the first step is often to express them mathematically to make them accessible to a multitude of tools that can potentially be used for solving them. Among those mathematical representations comes one dreaded family: the non-deterministic polynomial-time hard (NP-hard) problems, commonly considered unsolvable in polynomial time. NP-hard problems have two important characteristics: creating the correct solution from scratch is typically intractable, but verifying if a solution is correct does not require as much effort. Thus came heuristic methods: through approximations and trial and error, they can often solve NP-hard problems, even large ones, within an acceptable timeframe.

Researchers gathered a large variety of methods to solve increasingly complex problems. Once the proper tool has been selected, which in itself is already the subject of an entire area of research, the next step is to decide how best to use it. Heuristic algorithms come with various parameters corresponding to switches and knobs one can play with until the fastest path to a solution is found. They allow, for example, to activate pre-processing steps, to select among several metrics, to decide the direction a search process should branch first, or to define the probability of restarting from a random position. All these can be tailored to the characteristics of the problem instances at hand, and setting them correctly can make the difference between solving a problem in minutes, hours, or weeks. To optimise the performance of such algorithms by setting the values of their parameters is an optimisation problem known as the automated algorithm configuration problem. Over the last twenty years, researchers have developed heuristic algorithms (referred to as configurators) to configure their heuristic algorithms (referred to as solvers). Optimisation methods based on various paradigms — such as evolutionary algorithms, racing, and Bayesian optimisation — have provided the bases for configuring the inner workings of algorithms from various fields of application. Those configurators, more specifically their strategies to sample

parameter values and problem instances, are the main topic of study in the present thesis. In particular, we are interested in their application to reduce the running time of solvers for NP-hard problems.

We start by assessing the state of the art, evaluating 7 different approaches on 20 datasets and configuration scenarios from the literature. We find configurators perform differently depending on the tasks, but also that recent ones do not always significantly outperform older approaches. This shows that there has been limited development in the state of the art in the last few years with running time optimisation. Indeed, many recent methods are tailored toward the specific problem of hyperparameter optimisation for machine learning models, which focus on solution quality. We also listed the characteristics of the configuration scenarios and found no straightforward link between them and the performance of the different configurators. Deciding which configurator would be best on which scenario is a difficult problem by itself.

Then, we focus our attention and the sampling distribution of new parameter values, around the default value. This value is typically given with solvers for NP-hard problems of interest in our work. We find that those values typically defined by the developer based on their own understanding of their algorithms nor on experiments on their own set of benchmarks, provide a strong prior in the sampling of new values. We show that using this prior, we can improve the performance of the state-of-the-art configurators based on Bayesian optimisation, SMAC.

After sampling parameter values, the configurator would sample the problem instances on which to test those parameter values. We developed several instance selection approaches based on the literature to decide which instances the solver should try to solve first, and to determine quickly if a set of parameter values leads to better performance. We evaluate them in several cases: comparing two algorithms, two parametrised version of one algorithm, and based on different amounts and types of prior information to represent several steps of the configuration process in our experiments. Then, we integrated them into SMAC and evaluated this new version of SMAC with instance selection. We reached better performances than ever reached before. However, it is difficult to decide which method should be used when, and these methods thus need more research before being applicable in practice.

Overall, in this thesis, we evaluated several methods to sample both new values of parameters and new instances on which to test them based on prior information either from the developer or from the data gathered so far. We showed that better leveraging this information allows the configurators to be more efficient.

Samenvatting

Bij complexe problemen uit de praktijk is de eerste stap vaak om ze wiskundig te formuleren, zodat ze toegankelijk worden voor een breed scala aan hulpmiddelen die mogelijk kunnen worden ingezet om ze op te lossen. Onder die wiskundige formuleringen bevindt zich een gevreesde groep: de niet-deterministische polynomiale-tijd-moeilijke (NP-moeilijke) problemen, die doorgaans als onoplosbaar in polynomiale tijd worden beschouwd. NP-moeilijke problemen hebben twee belangrijke kenmerken: efficient een juiste oplossing vanaf nul bedenken is doorgaans onmogelijk, maar controleren of een oplossing juist is, is doorgaans makkelijker. Zo ontstonden heuristische methoden: door middel van benaderingen en trial-and-error kunnen ze vaak NP-moeilijke problemen, zelfs grote, binnen een acceptabele tijd oplossen.

Onderzoekers hebben een breed scala aan methoden verzameld om steeds complexere problemen op te lossen. Zodra het juiste hulpmiddel is gekozen – wat op zich al een heel eigen onderzoeksgebied is – is de volgende stap te bepalen hoe dit het best kan worden ingezet. Heuristische algoritmen beschikken over diverse parameters die te vergelijken zijn met schakelaars en knoppen waarmee men kan spelen totdat de snelste weg naar een oplossing is gevonden. Ze maken het bijvoorbeeld mogelijk om voorbewerkingsstappen te activeren, te kiezen uit verschillende meetmethoden, te bepalen in welke richting een zoekproces zich eerst moet vertakken, of de kans te definiëren om vanaf een willekeurige positie opnieuw te starten. Dit alles kan worden afgestemd op de kenmerken van de betreffende probleeminstanties, en het correct instellen ervan kan het verschil maken tussen het oplossen van een probleem in minuten, uren of weken. Het optimaliseren van de prestaties van dergelijke algoritmen door de waarden van hun parameters in te stellen, is een optimalisatieprobleem dat bekend staat als het geautomatiseerde algoritmeconfiguratieprobleem. In de afgelopen twintig jaar hebben onderzoekers heuristische algoritmen (configurators genoemd) ontwikkeld om hun heuristische algoritmen (solvers genoemd) te configureren. Optimalisatiemetho-

den die zijn gebaseerd op verschillende paradigma's — zoals evolutionaire algoritmen, racing en Bayesiaanse optimalisatie — hebben de basis gelegd voor het configureren van de interne werking van algoritmen uit diverse toepassingsgebieden. Die configuratoren, en meer specifiek hun strategieën voor het kiezen van parameterwaarden en probleeminstanties, vormen het hoofdonderwerp van dit proefschrift. We zijn met name geïnteresseerd in de toepassing ervan om de looptijd van NP-moeilijke solvers te verkorten.

We beginnen met het in kaart brengen van de stand van zaken, waarbij we 7 verschillende benaderingen evalueren op 20 datasets en configuratiescenario's uit de literatuur. We constateren dat configuratoren verschillend presteren afhankelijk van de taken, maar ook dat recentere benaderingen niet altijd significant beter presteren dan oudere. Dit toont aan dat er de afgelopen jaren beperkte vooruitgang is geboekt in de stand van zaken op het gebied van looptijdoptimalisatie. Veel recente methoden zijn namelijk afgestemd op het specifieke probleem van hyperparameteroptimalisatie voor machine learning-modellen, waarbij de nadruk ligt op de kwaliteit van de oplossing. We hebben ook de kenmerken van de configuratiescenario's op een rijtje gezet en geen duidelijk verband gevonden tussen deze kenmerken en de prestaties van de verschillende configurators. Bepalen welke configurator het beste is voor welk scenario is op zich al een lastig probleem.

Vervolgens richten we onze aandacht op de steekproefverdeling van nieuwe parameterwaarden rond de standaardwaarde. Deze waarde wordt doorgaans opgegeven bij de NP-moeilijke solvers die in ons onderzoek van belang zijn. We constateren dat deze waarden, die doorgaans door de ontwikkelaar worden vastgesteld op basis van hun eigen inzicht in de algoritmen of op basis van experimenten met hun eigen reeks benchmarks, een sterke a-priori-verdeling opleveren bij het genereren van nieuwe waarden. We laten zien dat we met behulp van deze a-priori-informatie de prestaties kunnen verbeteren van de state-of-the-art configurators op basis van Bayesiaanse optimalisatie, SMAC.

Na het kiezen van parameterwaarden zou de configurator de probleeminstanties kiezen waarop die parameterwaarden getest moeten worden. We hebben verschillende benaderingen voor instantieselectie ontwikkeld op basis van de literatuur om te bepalen welke instanties de solver als eerste moet proberen op te lossen, en om snel vast te stellen of een set parameterwaarden tot betere prestaties leidt. We evalueren deze in verschillende gevallen: door twee algoritmen te vergelijken, twee geparametriseerde versies van één algoritme, en op basis van verschillende hoeveelheden en soorten a-priori-informatie om verschillende stappen van het configuratieproces in onze experi-

menten weer te geven. Vervolgens hebben we ze geïntegreerd in SMAC en deze nieuwe versie van SMAC met instantieselectie geëvalueerd. We bereikten betere prestaties dan nooit eerder waren bereikt. Het is echter moeilijk te bepalen welke methode wanneer moet worden gebruikt, en deze methoden vereisen daarom meer onderzoek voordat ze in de praktijk toepasbaar zijn.

In deze scriptie hebben we verschillende methoden geëvalueerd om zowel nieuwe parameterwaarden als nieuwe probleeminstanties te genereren op basis van voorafgaande informatie, afkomstig van de ontwikkelaar of uit de tot nu toe verzamelde gegevens. We hebben aangetoond dat configuratietools efficiënter kunnen werken als deze informatie beter wordt benut.

List of Figures

List of Tables

List of Algorithms

List of Symbols

A The algorithm to configure

$T_{\mathbf{cut}}$ Target algorithm cutoff time

Ω The set of valid configurations

ω A valid configuration from Ω

$\omega_{\mathbf{inc}}$ The current best known configuration

$\omega_{\mathbf{ch}}$ A challenger configuration to evaluate

$\mathcal{P}$ A parameter from the configuration space Ω

$\mathcal{D}$ The domain of possible values of a parameter

$\mathcal{I}$ A set of problem instances

I A single instance

$\mathbf{f}$ The feature values of an instance I

rt The running time of an algorithm applied with a specific configuration ω on a specific instance I

$\mathcal{M}$ Performance of the algorithm at hand, applied with a specific configuration ω on a set of instances $\mathcal{I}$ (typically $\mathcal{M} = rt$)

$\mathcal{C}$ A set of configurators

C A configurator

LIST OF SYMBOLS

$\mathcal{K}$ A set of configuration scenarios

K A configuration scenario

C_{thres} a confidence threshold to take a decision

1

Introduction

When faced with a new problem, a common approach is to transform it into a mathematical representation, which can then be solved using the laws of mathematics. We learn to do so from the early years of primary school and, as the years pass, we gather methods to solve increasingly complex mathematical problems. Among those more complex problems, nondeterministic polynomial time (NP) hard problems are a family commonly considered non-solvable in polynomial time. While creating the correct solution from scratch is typically intractable, verifying if a solution is correct does not require as much time, which leads researchers to develop heuristic methods. Through approximations, trials and errors, those methods are often able to solve NP-hard problems, even large ones, within an acceptable time frame. Not only are heuristic algorithms powerful, they are also configurable: they come with various parameters, corresponding to many switches and knobs one can play with until the fastest path to a solution is found, and they can be tailored to the specificity of the problem instance at hand.

However, to optimise the performance of such algorithms by setting the values of their parameters requires expert knowledge, a lot of time, or often both. Reproducing the exact same process as described in the previous paragraph, researchers took this new optimisation problem, represented it mathematically as the automated algorithm configuration (AAC) problem, and developed heuristic algorithms (referred to as configurators) to configure their heuristic algorithms (referred to as solvers). The configurator will search the space of possible values for the parameters of the solver

and evaluate their impact on its performance on the instances of interest, according to a specific performance measure (often the running time of the solver or the problem-specific quality of the solution it could find). Tackling the AAC problem requires searching a large space of configurations and performing time-consuming solver runs on instances. From local search methods to model-based approaches , state-of-the-art configurators are complex search algorithms tailored to mitigate those two difficulties.

In this thesis, we examine the elements of the AAC problem and the decisions made in past works regarding sampling new values or instances. We investigate sampling approaches – in theory, configurator-agnostic – and hypothesise that different approaches would allow to more efficiently search for high-performing configurations by focusing the search and reducing the time required to evaluate the configured solver.

1.1 Background

As mentioned before, one of the most challenging family of problems obtained by representing mathematically real-world problems are the NP-hard problems. Finding algorithms and developing software that can efficiently solve these is an active area of scientific research, due to their high complexity and wide range of applications (see *e.g.* Burch et al., 1994; Matai et al., 2010; Verbert et al., 2017). New solvers for NP-hard problems are regularly developed and compete with each other at yearly competitions, such as the SAT competition (see *e.g.* Heule et al., 2024), the SMT competition (see *e.g.* Weber et al., 2019) and the planning competition (see *e.g.* Taitler et al., 2024). Among these solvers, many can be tailored to a specific application domain through parameters that influence their inner workings and thus their performance. However, manually tuning those parameters is a lengthy and tedious process that relies heavily on expert knowledge. From the early 1990s onwards, researchers began to explore approaches to automatically configure search heuristics (Minton, 1993) and thus solvers based on such heuristics, which later gave rise to the field of AAC and the paradigm of programming by optimisation (Hoos, 2012b). AAC is particularly beneficial when confronted to a large set of similar problems on which the same configuration is expected to perform similarly well. In recent decades, many automated methods to configure – or tune – those solvers have been proposed (see *e.g.* Ansótegui et al., 2009; López-Ibáñez et al., 2016; Pushak and Hoos, 2020). Their great success encouraged researchers to apply them to a range of other algorithms from areas such as material science (see *e.g.* Packwood et al., 2017; Wahab et al., 2020), economics (see *e.g.* Balcan et al., 2018) and process mining(see *e.g.* Ramos-Gutiérrez et al., 2021). When applied to machine

learning (see *e.g.* Bergstra and Bengio, 2012; Li et al., 2018), this gave rise to the nowadays prominent research area of automated machine learning (autoML) (Hutter et al., 2019).

1.2 Outline and research questions

Chapter 2 introduces in more detail the problem at hand, explains the diverse approaches that have been used to tackle it in the literature, and describes the configuration scenarios that will be used in the remainder of the thesis.

RQ1 (Chapter 2) How can the configuration scenarios be best described and characterised?

To better understand the challenges of AAC, we examine widely studied configuration scenarios and attempt to characterise them with the goal of facilitating principled comparisons.

Chapter 3 describes the protocol we follow to compare the performance of configurators. It then compares the configurators introduced in the previous chapter and presents insights regarding their strengths and weaknesses.

RQ2 (Chapter 3) How do state-of-the-art configurators compare to each other on a variety of scenarios?

We run various state-of-the-art general-purpose algorithm configurators on the scenarios that we previously identified. We evaluate them in terms of aggregate performance and complementarity across diverse scenarios.

RQ3 (Chapter 3) How do scenario characteristics influence the performance of state-of-the-art configurators?

We relate the performance to the characteristics we defined in Chapter 2 and draw general conclusions about their strengths and weaknesses.

Chapter 4 explores the extent to which the default parameter values are used in current configurators and proposes an approach to make better use of them.

RQ4 (Chapter 4) How and to which extent do current configurators use the default values usually provided by algorithm developers?

Algorithms typically come with a default configuration. We investigate how frequently configurators use this configuration and their impact on performance.

RQ5 (Chapter 4) How can we make better use of known good parameter values – in our case, the default value – to guide the search strategy?

We evaluate a simple strategy to prune the search space around the default value. Based on its success, we propose a sampling strategy that focuses the search around the default values and evaluates its efficiency in a prominent configurator.

Chapter 5 proposes approaches to compare the performance of algorithms against each other while minimising time spent on non-informative problem instances. It is used as a first step towards the comparison of several configurations studied in the following chapter.

RQ6 (Chapter 5) How can we smartly select on which instances to run our evaluation to lower the time spent evaluating bad algorithms?

We evaluate several metrics to select a subset of instances for comparison between two algorithms. Our goal is to make a decision to discard less promising challengers faster.

Chapter 6 evaluates those approaches to compare the performance of several configurations of a single algorithm and integrates them inside a configurator.

RQ7 (Chapter 6) How can we smartly select on which instances to run our evaluation to lower the time spent evaluating bad configurations?

We apply the previously developed metrics to select a subset of instances for comparison between two configurations. To do so, we define two comparison situations that arise in a configurator and evaluate the selection metrics for both.

RQ8 (Chapter 6) How can instance selection boost the configurator performance or speed?

We integrate the most efficient metrics in a configurator to speed up the comparison of configurations and allow more configurations to be compared to the incumbent.

1.3 Publications based on material in this thesis

Parts of this work have given rise to peer-reviewed publications. We give below a short overview of the content of each publication.

> Anastacio, M., Luo, C., and Hoos, H. (2019). Exploitation of default parameter values in automated algorithm configuration. In *Workshop Data Science meets Optimisation, DSO, in conjunction with IJCAI*.

In this work, we studied the impact of the given default on the prominent configurators and introduced a naïve approach to focus the search on the default value without needing to make any changes in the configurators. To do so, we reduced the search space around the default value and showed that, for 15 out of 20 scenarios, the prominent configurator SMAC found better configuration on the reduced search space. This work is covered in Chapter 4.

> Anastacio, M. and Hoos, H. H. (2020a). Combining sequential model-based algorithm configuration with default-guided probabilistic sampling. In *GECCO 2020: Genetic and Evolutionary Computation Conference 2020, Companion Volume*, pages 301–302. ACM.

This extended abstract laid the foundations of a sampling method integrated inside of the configurator SMAC to follow truncated normal distributions centered around the default values of the parameters. This work is covered in Chapter 4.

> Anastacio, M. and Hoos, H. H. (2020b). Model-based algorithm configuration with default-guided probabilistic sampling. In *Proceedings of Parallel Problem Solving from Nature - PPSN XVI, Part I*, volume 12269 of *Lecture Notes in Computer Science*, pages 95–110. Springer.

In this paper, we tested the sampling method presented previously and compared the obtained configurator SMACPS to the two prominent configurators SMAC and irace, reaching better configurations than both of them on more than half of the 16 studied scenarios. This work is covered in Chapter 4.

> Matricon, T., Anastacio, M., Fijalkow, N., Simon, L., and Hoos, H. H. (2021). Statistical comparison of algorithm performance through instance selection. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming, CP*, volume 210 of *LIPIcs*, pages 43:1–43:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

In this paper, we sped up the time required to compare the performance of pairs of solvers for NP-hard problems by automatically selecting on which problem instances they should be evaluated and applying a statistical test to decide when enough evidence has been gathered. This work is covered in Chapter 5.

Anastacio, M., Matricon, T., and Hoos, H. H. (2022). Instance selection for configuration performance comparison. In *Meta-knowledge transfer workshop, in conjunction with ECML-PKDD*.

Building on the previous paper, we studied the impact of instance selection methods to compare configurations of a single algorithm instead of comparing algorithms. We showed that similarly, the time required to get enough evidence to decide which is better than the other can be significantly lower than when running them on all problem instances. This work is covered in Chapter 6.

Anastacio, M. (2021). Greybox algorithm configuration. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI*, pages 4875–4876.

This extended abstract was prepared when planning the content of this thesis. While the final content deviates slightly from the one presented then, this extended abstract describes at a high-level the content covered in Chapters 4 to 6.

1.4 Other work published by the author

During the thesis project, the author also contributed to other publications that are only loosely related to this dissertation. Their content is not part of this thesis, but, for completeness, these are outlined in the following.

Fokkinga, D., Latour, A. L. D., Anastacio, M., Nijssen, S., and Hoos, H. (2019). Programming a stochastic constraint optimisation algorithm, by optimisation. In *Workshop Data Science meets Optimisation, DSO, in conjunction with IJCAI*.

Latour, A. L. D., Babaki, B., Fokkinga, D., Anastacio, M., Hoos, H. H., and Nijssen, S. (2020). Stochastic constraint optimisation with applications in network analysis (extended abstract). In *International Workshop on Model Counting, MCW, in conjunction with SAT*.

Latour, A. L., Babaki, B., Fokkinga, D., Anastacio, M., Hoos, H. H., and Nijssen, S. (2022a). Exact stochastic constraint optimisation with applications in network analysis. *Artificial Intelligence*, 304:103650.

Latour, A. L. D., Babaki, B., Fokkinga, D., Anastacio, M., Hoos, H. H., and Nijssen, S. (2022b). Stochastic constraint optimisation with applications in network

analysis (extended abstract). In *Workshop on Counting and Sampling 2022, in conjunction with FLoC 2022 and SAT 2022*.

These publications are part of a line of research in which automated algorithm configuration was applied to a solver for stochastic constraint optimisation problems. In the framework of his master project, D. Fokkinga used SMAC to optimise the parameters of the approach developed by A.L.D. Latour (then a PhD student).

Pulatov, D., Anastacio, M., Kotthoff, L., and Hoos, H. H. (2022). Opening the black box: Automated software analysis for algorithm selection. In *International Conference on Automated Machine Learning, AutoML*, volume 188 of *Proceedings of Machine Learning Research*, pages 6/1–18. Proceedings of Machine Learning Research PMLR.

Purucker, L. O., Schneider, L., Anastacio, M., Beel, J., Bischl, B., and Hoos, H. H. (2023). Q(D)O-ES: population-based quality (diversity) optimisation for post hoc ensemble selection in automl. In *International Conference on Automated Machine Learning, AutoML*, volume 224 of *Proceedings of Machine Learning Research*, pages 10/1–34. Proceedings of Machine Learning Research PMLR.

These publications arose from the projects of other PhD students, whom the author assisted in conceptualising the methods, implementing them and analysing the obtained results. The first one explored the usage of algorithm source code features in the context of automated algorithm selection, and the second proposed a quality diversity optimisation ensemble selection method to ensemble machine learning models after their hyperparameters have been optimised.

Rogers, J., Anastacio, M., Bernard, J., Chakhchoukh, M., Faust, R., Kerren, A., Koch, S., Kotthoff, L., Turkay, C., and Wall, E. (2024). Visualization and automation in data science: Exploring the paradox of humans-in-the-loop. In *Workshop on Visualization in Data Science VDS, in conjunction with IEEE Visualization and Visual Analytics Conference VIS*.

This publication resulted from a Dagstuhl seminar in which the author actively participated and calls for bringing back humans in the loop in automated data science. It won a best paper award at the Workshop on Visualization in Data Science.

Kalkreuth, R., de França, F. O., Dierkes, J., Anastacio, M., Jankovic, A., Vasicek, Z., and Hoos, H. (2025). Tinyversegp: Towards a modular cross-domain

benchmarking framework for genetic programming. In *GECCO 2025: Genetic and Evolutionary Computation Conference, Companion Volume*. ACM.

This work presents a benchmarking tool for genetic programming. It includes a pipeline implemented by the author for optimising the hyperparameters of genetic programming methods.

Gerlach, B., Anastacio, M., and Hoos, H. H. (2025). On the efficiency of training robust decision trees. In *Poster at the Symposium on AI Verification SAIV, adjunct to the International Conference on Computer-Aided Verification CAV*.

Moeini, E., Vox, C., Anastacio, M., Skaf, W., Barachi, M., and Hoos, H. H. (2026). Neural architecture and hyperparameter selection through meta-learning on time series. In *Proceedings of the AAAI Conference on Artificial Intelligence*. AAAI Press.

These publications were the result of master and bachelor projects of students co-supervised by the author. Gerlach *et al.* studies the robustness of decision trees and ensembles thereof. Moeini *et al.* applied meta-learning to jointly predict the architecture and hyperparameters one should use on a new time series dataset.

2

Automated Algorithm Configuration

In this chapter, we define the automated algorithm configuration (AAC) problem, its applications and challenges, particularly when optimising the running time of solvers for NP-hard problems. We list existing state-of-the-art configurators and delve into their inner workings to gain a deeper understanding of how they address these challenges during their search procedure. We then introduce the set of configuration scenarios and benchmarks that have been used in the literature and will be used throughout the thesis. For each of them, we attempt to define characteristics that might impact the performance of configurators. An empirical evaluation of configurator performance is presented in Chapter 3.

2.1 Algorithm configuration problem

Many state-of-the-art algorithms, especially solvers for NP-hard problems, come with parameters that enable users to fine-tune the inner workings of the solver to the specific problem instances they are trying to solve. This tuning has traditionally been conducted manually or through simple search procedures, such as random search, which remains the approach in many fields despite the availability of tools to automate parameter tuning. The question of finding which parameter values should be used for an algorithm to perform well on a set of problem instances is known as the *automated algorithm configuration problem*.

2.1.1 Problem definition

The AAC problem can be defined as follows (see *e.g.* Hoos, 2012a).

Definition 2.1. Given:

- a target algorithm A with k parameters $p_1, p_2, ..., p_k$, a domain $\mathcal{D}_j$ of possible values and a default value $D_j \in \mathcal{D}_j$ for each parameter p_j;
- a configuration space Ω, containing all valid combinations of parameter values of A;
- a set of problem instances $\mathcal{I}$;
- a performance metric $\mathcal{M}$ that measures the performance on $\mathcal{I}$ of target algorithm A configured according to $\omega \in \Omega$;

find $\omega^* \in \Omega$ that optimises the performance of A on instance set $\mathcal{I}$, according to metric $\mathcal{M}$.

Note that typically the running time of the target algorithm is limited by an upper bound defined by means of a cutoff time T_{cut}. This limit has an impact on any attempt to model the running time, as a saturation phenomenon occurs at the upper bound. Instead of following the underlying distribution of running time, data gathered with such a cutoff time will be set to T_{cut} for any algorithm that needs more time. Some configuration approaches will implement methods to mitigate this impact, such as SMAC proposed by Hutter et al. (2011b) (described in Section 2.2.2).

2.1.2 Challenges

The AAC problem comprises three main elements, each coming with its own challenges.

Target algorithm

The target algorithm is typically treated as a black-box algorithm with complex inner workings. Although some methods tried to open the black box (see *e.g.* Adriaensen and Nowé, 2016; Pulatov et al., 2022), these are still preliminary and, to the best of our knowledge, there exists no configurator that integrates such methods natively. In this work, we are particularly interested in solvers for NP-hard problems. Such algorithms present two significant challenges. First, their typical running time varies widely depending on the specific problem instance they solve, especially for solvers of NP-hard problems and the longer evaluations can be very costly. Thus, the configuration process can benefit from mechanisms to reduce the number of evaluations. Second, they do not typically come with a notion of anytime performance, preventing us from using methods typically used for machine learning algorithms, such as successive halving Jamieson and Talwalkar (2016) or learning curve-based performance prediction Domhan et al. (2015).

Configuration space

Each parameter comes with a domain of possible values. This domain can be of different types: categorical, ordinal, or numerical. Categorical parameters have an unordered finite set of possible values and are often used to select between several heuristic components or mechanisms. Numerical parameters are real- or integer-valued and are often used to calibrate heuristic mechanisms or components. Parameters can also conditionally depend on each other, such that one is active only when another takes a specific value. For example, consider a Boolean parameter that activates a mechanism, which is itself adjusted using a numerical parameter. While in some scenarios $\Omega = \mathcal{D}_1 \times \ldots \times \mathcal{D}_k$, these conditional relationships induce complex shapes in the resulting configuration spaces, as some parts are accessible only under the condition that other parameters are set to a specific value. Despite those challenges, Pushak and Hoos (2018) demonstrated that the performance landscapes (*i.e.* how the performance of the target algorithm varies depending on the variations in its configuration) tend rather benign, which can facilitate accurate prediction of the performance of the algorithm. Chapter 4 will delve more into the configuration space.

Instance set

The instance set comprises a set of instances that are considered representative of the given problem. Specifically, the instances in this set are expected to be drawn from

the same underlying distribution, such that the configuration obtained based on a part of this set performs similarly well on the whole set. Indeed, the AAC problem aims to find a well-performing configuration for a specific type of problem instances, rather than a configuration that performs well on any instance. The set can contain instances that are more or less difficult, and thus take more or less time to solve. This affects the configurator in many ways. Indeed, long running times mean that fewer evaluations can be performed within a given time budget. To avoid the configuration process from stalling on an unsolvable or extremely costly instance, each algorithm run is stopped after a cutoff time T_{cut}. However, if too many instances cannot be solved by the solver within this time, there is very little to be learnt from the runs. We look into this in more detail in Chapters 5 and 6.

2.2 Configuration approaches

In recent years, much work has been done on AAC, resulting in several general-purpose automatic algorithm *configurators*, based on different approaches. Each requires and includes three key mechanisms: a method to generate a challenger configuration from Ω, a method to estimate the performance of a configuration on the instances from $\mathcal{I}$, and a method to decide which configuration(s) to keep for the next iteration of the optimisation process.

2.2.1 Scope

Most configurators have limitations, strengths and weaknesses that are based on their underlying search algorithm and the configuration scenarios for which they were designed. In this work, we are interested in configuring algorithms that support a large number of parameters of any type (numerical, Boolean, or categorical) with possible conditional dependencies between them. We thus do not consider configurators that do not handle categorical parameters, such as REVAC (Nannen and Eiben, 2007). While we acknowledge that for some algorithms it is relevant to optimise along several objectives, we focus solely on single-objective configuration for the running time of NP-hard problems. This restriction, which corresponds to many practical applications of algorithm configuration, allows us to use well-studied configuration scenarios from the library AClib (Hutter et al., 2014a). Moreover, the analysis of our results is easier to interpret, without the added complexity of a Pareto front analysis. For many such scenarios, we have no access to anytime performance or any way to approximate

a result with a lower budget, which excludes configurators relying on this kind of information, such as BOHB (Falkner et al., 2018) and Optuna (Akiba et al., 2019). We also limited our use of proprietary configurators that are not readily available for use, such as GGA (Ansótegui et al., 2009).

For our performance comparisons (see Chapter 3), we thus consider the following configurators: paramILS (Hutter et al., 2009), GGA++ (Ansótegui et al., 2015), irace (Birattari et al., 2010), SMAC (Hutter et al., 2011b) and GPS (Pushak and Hoos, 2020). While there might be more available, we believe that this selection covers prominent fundamental methods (evolutionary computation, local search, Bayesian optimisation and racing). In the following, we delve deeper into the inner workings of these configurators.

2.2.2 Details on prominent AAC procedures

As mentioned earlier, the AAC problem presents numerous challenges, and each configurator approaches them in a different way. To handle the extensive running time of the target algorithm, configurators can use an empirical performance model (*e.g.* SMAC and GGA++) that allows them to estimate the running time based on features characterising the instances, similarly to the work done in algorithm selection (AS) (Xu et al., 2008; Lindauer et al., 2015). Capping mechanisms reduce the number of algorithm runs by stopping the evaluation of poorly performing configurations (*e.g.* paramILS and irace). To handle the vast search space, numerous search algorithms are available in the literature. Each configurator is based on well-known search strategies that have already demonstrated success in other fields, such as genetic algorithms, estimation of distribution, or golden search. Each configurator considered in our work is described in more detail in the following.

paramILS

paramILS (Hutter et al., 2007, 2009) is the earliest method that we consider. It is an iterated local search (ILS) algorithm (Lourenço et al., 2003), designed to optimise the parameters of heuristic algorithms. Following ILS, it starts from a point in the search space and, through small perturbations, explores its neighbourhood. It restarts this process several times to collect a set of good configurations and returns the best one when the configuration budget is exhausted. Additionally, it extends ILS with an adaptive capping mechanism that limits the time spent on less promising sets of parameters.

The high-level procedure of paramILS is described in Algorithm 2.1. ParamILS is initialised with random configurations (line 1) from which it keeps the one determined to perform best according to $\mathcal{M}$. It perturbs this configuration with small changes to one parameter at a time and retains the first found improvement (line 2). Then, until the assigned budget is exhausted, it applies s perturbations to the configuration (line 5) and improves it by taking the first neighbouring configuration that improves upon it (line 6). If the best found configuration so far, ω_{inc}, is outperformed by the newly found configuration, its value is updated (line 7). paramILS restarts the local search at random with a probability of $p_{restart}$ set to 1% by default (line 5).

Hutter *et al.* propose two approaches to decide, line 7, which of the two configurations performs best. This leads to two variants of the paramILS procedure. The first variant, basicILS, compares directly the performance of two configurations on a fixed subset of instances. The second variant, focusedILS, gradually increases the number of runs performed with the configuration that was evaluated the fewest times, until one configuration dominates the other (ω_1 dominates ω_2 if it performs at least as well on all instances evaluated with ω_2). At each iteration of the core loop (lines 3 to 9), focusedILS increases the number of instances on which ω_{inc} is evaluated. Moreover, to avoid spending excessive time evaluating poorly performing configurations, an adaptive capping mechanism is employed. If, during a target algorithm run, the perturbed configuration is deemed to perform significantly worse than the current best, the run

Algorithm 2.1 ParamILS

Input Ω: configuration space, r: number of initial configurations, $p_{restart}$: probability of restarting, s: number of perturbations, *Budget*: the maximum time allowed to find a configuration.
Output ω_{inc}: the best found configuration.

1: Sample r random configurations in Ω, set ω to the best-performing.
2: Perform iterative first improvement on neighbourhood of ω (the configurations that differ by one parameter) and set ω_{inc} to the result
3: **while** *Budget* not exhausted **do**
4: $\omega \leftarrow \omega_{\text{inc}}$
5: Apply s random perturbations (change one parameter value) to ω
6: Perform iterative first improvement on neighbourhood of ω (the configurations that differ by one parameter)
7: If ω is better that the best known configuration ω_{inc}, update ω_{inc}
8: With probability $p_{restart}$, reset ω to a random configuration
9: **end while**
10: **Return** ω_{inc}

is terminated. According to the evaluation presented by (Hutter et al., 2009), this reduced up to 10-folds the required configuration time.

GGA++

GGA (Ansótegui et al., 2009) is a genetic algorithm that considers a population in which each individual is a configuration and the performance of the target algorithm is used as the fitness function. Following the work of Lis and Eiben (1997), the individuals are assigned a "gender" that separates them into two groups: the competitive and the non-competitive. Among the competitive group, only the $c\%$ best individuals can produce offsprings, based on a tournament evaluating their fitness on a subset of instances from the training set. Among the non-competitive group, a portion of randomly selected individuals will produce offsprings (their number is tailored to keep the size of the population stable). This approach allows to minimise the number of direct evaluations of configurations and to keep diversity in the population thanks to the non-competitive individuals. Following standard practice in the field, Ansótegui *et al.* apply crossovers and mutations to the offspring produced at each generation, and kill individuals when they reach the age limit (set to 3 by default).

Later, Ansótegui et al. (2015) proposed GGA++. They added a random forest surrogate model to predict the performance of the non-competitive individuals and

Algorithm 2.2 GGA++

Input Ω: configuration space, *Age*: maximum age of the population, c: portion of the individuals kept through competition, g: portion of the offspring genetically engineered, r: portion of individuals replaced by random at each generation, *Budget*: the maximum number of generations allowed to find a configuration.
Output ω_{inc}: the best found configuration.

1: pop $\leftarrow$ SampleUniform(Ω) ▷ Initialise the population.
2: Assign each individual an age (from 1 to *Age*)
3: separate into $\text{pop}_{competition}$ and $\text{pop}_{attraction}$
4: **while** *Budget* not exhausted **do**
5: $\quad \text{pop}_{\text{best}} \leftarrow$ best $c\%$ of $\text{pop}_{competition}$
6: $\quad \text{pop}_{\text{new}} \leftarrow$ offsprings of pop_{best} and $\text{pop}_{attraction}$ with probability g of genetic engineering
7: $\quad$ Mutate offspring.
8: $\quad$ Replace portion r of the non-competitive population with random individuals
9: $\quad$ Evaluate pop_{best} and update the incumbent ω_{inc}
10: **end while**
11: **Return** ω_{inc}

used it as a measure of attractiveness to select which of them would mate. This surrogate model also provides a heuristic to predict the performance of possible offsprings and decide which one should be produced. The model is specifically tuned to be more precise in the top-performing areas of the search space, and is used at several steps of the algorithm (Algorithm 2.2). Compared to GGA, competitive individuals are still selected based on a tournament (line 4), but the non-competitive individuals are selected with a probability based on their predicted performance instead of uniformly at random. Moreover, a portion g (set to 1 in the original publication) of the offsprings are built using genetic engineering, meaning that the surrogate model indicates which offspring should be produced by the mating process (line 5). This engineering process, coupled to the selection of parents based on the same surrogate model, would lower the diversity of the population, which is key in evolutionary computing for balancing exploration and exploitation. Thus, part of the population is replaced by random individuals at each generation (line 9).

irace

The configurator irace (Birattari et al., 2010) is based on racing methods, drawing from prior work by Maron and Moore (1997). The idea, as seen in Algorithm 2.3, is to perform a race among a set of configurations, *i.e.*, to run them incrementally on more and more instances, dropping configurations as soon as they have been found to perform statistically worse than others. Then, irace generates new configurations using an estimation of distribution mechanism (see *e.g.* Hauschild and Pelikan, 2011) that

Algorithm 2.3 irace (Based on Algorithm 1 from Birattari et al. (2010))

Input $I = [I_1, I_2, \ldots] \in \mathcal{I}$, Ω: the parameter space, $\mathcal{M}(\omega, I) \in \mathbb{R}$: performance measure, $Budget$: the maximum number of iterations allowed to find a configuration.
Output Ω_{elite}: a set of best found configurations.

1: $\Omega_1 \leftarrow$ configurations sampled uniformly at random from Ω
2: $\Omega_{\text{elite}} \leftarrow$ elites of the race among Ω_1
3: $j \leftarrow 1$
4: **while** $j < Budget$ **do**
5: $\quad j \leftarrow j + 1$
6: $\quad \Omega_{\text{new}} \leftarrow$ configurations sampled from Ω around Ω_{elite}
7: $\quad \Omega_j \leftarrow \Omega_{\text{new}} \cup \Omega_{\text{elite}}$
8: $\quad \Omega_{\text{elite}} \leftarrow$ elites of the race among Ω_j
9: **end while**
10: **Return** Ω_{elite}

builds a probabilistic model to capture the distribution of promising configurations based on the best configurations seen in the previous race, relying on the assumption that good configurations are likely near each other within the configuration space.

The races (lines 2 and 8) proceed as follows: each configuration is evaluated on a fixed number of instances T^{first}; then, after each T^{each} evaluations on a new instance, a Friedman two-way analysis of variance by ranks (with a significance level of 0.05 by default) is applied to discard less performing configurations. By default, $T^{first} = 5$ and $T^{each} = 1$. The irace procedure does not perform multiple hypothesis testing correction, which leads to a higher probability to mistakenly discard configurations.

To sample new configurations (line 6), parent configurations are first sampled with a higher probability for higher-ranked configurations. Then, children configurations are sampled nearby following a truncated normal distribution centred around their parent configuration and with a standard deviation σ_d^j for parameter d at iteration j. To sample configurations increasingly closer to known elites, at each iteration this standard deviation is updated according to $\sigma_d^j = \sigma_d^{j-1} \cdot \left(\frac{1}{N_j^{new}}\right)^{1/N^{param}}$, where N_j^{new} is the number of new configurations sampled in iteration j and N^{param} is the number of parameters.

Later, inspired by paramILS previously described, it was extended with a capping mechanism (López-Ibáñez et al., 2016; Cáceres et al., 2017) to stop evaluating configurations as soon as they are clearly worse performing than the elites.

SMAC

SMAC (Hutter et al., 2011b; Lindauer et al., 2022) is based on a sequential model-based optimisation approach (also known as Bayesian optimisation). Bayesian optimisation allows for the replacement of costly evaluation of the real performance with a surrogate model that estimates the performance of configurations cheaply (see *e.g.* Mockus, 1989). As shown in Algorithm 2.4, SMAC constructs a random forest surrogate model to predict the performance of configurations on given instances (line 5). It samples random configurations (lines 6 and 8), and some of them are optimised using local search on the expected improvement obtained from the predictions of the model (line 7). The local search procedure uses a notion of expected improvement to produce promising configurations. This expected improvement is defined as the expected difference between the performance of the best known ω_{inc} and the performance of the sampled configuration. To ensure diversity in the evaluated configurations, this local search procedure is applied only to half of the configurations. The configurations found

through local search are then sorted from highest to lowest expected improvement, and interleaved with those random configurations (line 9). During the intensification phase (line 10), the challenger configurations thus found are compared to the incumbent, first by running them on the same instances as those on which ω_{inc} has already been run, and then by adding more instances randomly to slowly increase the set of instances on which the runs are collected and avoid overfitting to a subset of instances from the training set. To avoid spending too much time on non-promising configurations, as soon as the challenger configuration ω_{ch} has used up as much time budget as ω_{inc} needs for all instances it has been evaluated on, the evaluation of ω_{ch} is stopped.

GPS

More recently, GPS (Pushak and Hoos, 2020) introduced a new, highly parallelisable search approach, described in Algorithm 2.5 (we omit the steps related to the queues and workers management and focus on the configuration aspect of the procedure). GPS relies on two fundamental assumptions. The first assumption is that the interactions between parameters are limited. This allows each parameter to be optimised independently and the incumbent configuration ω_{inc} to be update parameter per parameter (line 10). The second assumption, as per the findings of Pushak and Hoos (2018), is that the response landscape of the solvers is unimodal. This assumption led

Algorithm 2.4 SMAC

Input Ω: configuration space, ω_d: the default configuration, *Budget*: the maximum time allowed to find a configuration.
Output ω_{inc}: the best found configuration.

1: $\mathcal{R}$: target algorithm runs performed, M: performance model.
2: $\omega_{\text{inc}} \leftarrow \omega_d$
3: $\mathcal{R} \leftarrow run\,(\omega_d)$
4: **while** *Budget* not exhausted **do**
5: $M \leftarrow$ update model M from $\mathcal{R}$
6: $\Omega_{\text{prom}} \leftarrow$ sample configurations from Ω uniformly
7: $\Omega_{\text{prom}} \leftarrow$ refine Ω_{prom} using local search on the expected improvement derived from M
8: $\Omega_{\text{rand}} \leftarrow$ sample configurations from Ω uniformly
9: $\Omega_{\text{new}} \leftarrow$ interleave Ω_{rand} and Ω_{prom}
10: $\omega_{\text{inc}} \leftarrow$ compare configurations from Ω_{new} to ω_{inc} until a better one is found or a time limit is reached
11: **end while**
12: **Return** ω_{inc}

to the use of the golden search algorithm from by Kiefer (1953), which uses the golden ratio to select interior points within an interval and discards subintervals where the extremum cannot lie.

For each parameter p, GPS maintains a bracket of possible values instead of keeping the entire domain $\mathcal{D}_p$. It uses the golden search algorithm to adjust this bracket, such that it is expected to still contain the optimum value (line 13). Since golden search only works for strictly unimodal functions, a property that is likely to be violated for some AAC scenarios, GPS includes a mechanism to expand back the bracket thus reduced. To accept new values for the parameters, it uses a permutation test with significance value $\alpha = 0.05$ (lines $9 - 10$). It evaluates configurations on parallel cores, prioritising the most promising parameters using a bandit approach (line 8). It gradually increases the number of instances on which configurations are evaluated (lines 14 and 16), while using an adaptive capping mechanism to prevent excessive time spent on poorly performing configurations. GPS is the most recent algorithm included in our comparison. It combines mechanisms from earlier works, including adaptive capping from paramILS, racing from irace and the intensification approach from SMAC.

Algorithm 2.5 GPS

Require: numInitInst: Initial number of instances, instIncr: Instance increment.

1: Initialise incumbent ω_{inc} with default
2: **for** each parameter p **do**
3: Initialise bracket B_p
4: Initialise $\mathcal{I}_p$ with numInitInst random instances
5: Queue a run for default value $\omega_{\text{inc}}[p]$
6: **end for**
7: **while** *Budget* not exhausted **do**
8: Sample parameter p using bandit queue
9: **if** $\exists v$ such that $m(\omega_{\text{inc}}|_{p=v}) \prec_\alpha m(\omega_{\text{inc}})$ **then** ▷ based on a permutation test
10: $\omega_{\text{inc}}[p] \leftarrow v$ ▷ Update incumbent
11: **end if**
12: **if** sufficient evidence for improvement **then**
13: Adjust bracket B_p
14: Add instIncr random instances to $\mathcal{I}_p$
15: **else if** each $v \in B_p$ has been run on each $I \in \mathcal{I}_p$ **then**
16: Add instIncr random instances to $\mathcal{I}_p$
17: **end if**
18: Queue new target algorithm runs for B_p
19: **end while**
20: **return** ω_{inc}

Table 2.1: List of scenario characteristics

Category	Name	Domain	Description
Algorithm	Deterministic	Boolean	Is the algorithm deterministic or does it need to run with several seeds
Parameters	Number	$\mathbb{N}+$	Number of parameters
Parameters	Categoricals	$\mathbb{N}+$	Number of categoricals parameters
Parameters	Integers	$\mathbb{N}+$	Number of integers parameters
Parameters	Continuous	$\mathbb{N}+$	Number of continuous parameters
Parameters	Conditionals	$\mathbb{N}+$	Number of Conditionals parameters
Parameters	Defaults	$\mathbb{N}+$	Number of default configurations
Parameters	Forbidden	$\mathbb{N}+$	Number of forbidden configurations
Instances	Origin	String	Origin (generated, real-world)
Instances	Training size	$\mathbb{N}+$	Number of training instances
Instances	Testing size	$\mathbb{N}+$	Number of testing instances
Instances	Features	$\mathbb{N}+$	Number of features per instance
Instances	Clusters	$\mathbb{N}+$	As a measure of dataset homogeneity
Scenario	Cutoff time	$\mathbb{N}+$	Maximum running time before interrupting the algorithm run
Scenario	Timeouts	$[0, 100]$	Percent of training instances that timeout with the default configuration
Scenario	Budget	$\mathbb{N}+$	Time given to the configurator

2.3 Configuration scenarios

In this section, our goal will be to answer RQ1 – *How can the configuration scenarios be best described and characterised?*. To do so, we list scenarios from the literature and define characteristics to compare them, as well as the performance of configurators on these scenarios. Based on the definition introduced in Section 2.1.1, a configuration scenario would need to describe the target algorithm A with possibly a cutoff time T_{cut}, a configuration space Ω with possibly a default configuration, a set of problem instances $\mathcal{I}$, and a metric to optimise $\mathcal{M}$. The considered list of characteristics is listed in Table 2.1. We attempt to be representative of the problems considered in this thesis, but do not claim to be exhaustive.

2.3.1 Benchmark instance sets

The configuration scenarios we considered throughout this thesis are based on ten sets of randomly generated and real-world instances across four NP-hard problems:

Table 2.2: Benchmark instance sets characteristics.

	Name	Origin	*Training size*	*Testing size*	*Features*	*Clusters*
SAT	CF	generated	298	301	113	14
	LABS	generated	350	350	119	6
	UNSAT	generated	299	249	113	12
Planning	Satellite	generated	2000	2000	305	7
	Zenotravel	generated	2000	2000	305	5
MIP	CLS	generated	50	50	148	3
	COR-LAT	real-world	1000	1000	148	19
	RCW2	real-world	495	495	148	6
	REG200	generated	999	999	148	2
TSP	rue-1000-3000	generated	50	250	64	9

Boolean satisfiability (SAT), automated planning (AI planning), mixed integer programming (MIP) and traveling salesperson problem (TSP). They are all part of the library AClib introduced by Hutter et al. (2014a) and have been widely used in the papers introducing new AAC methods and related mechanisms. We used the training and testing sets as provided by AClib. A more detailed description of each of those datasets is provided below, and their characteristics are listed in Table 2.2. The number of clusters is computed based on the instance features using the mean-shift (Comaniciu and Meer, 2002) implementation of `scikitlearn`. Each cluster is thus a group of instances which are close to each other in the feature space according to their Euclidean distance.

Boolean satisfiability

The SAT problem is a classical mathematical problem that has been proven to be NP-complete in the 1970s (Cook, 1971). Given a Boolean formula, the goal is to either find an assignment of truth values to its variables that satisfies the formula or prove that it is unsatisfiable. Such Boolean formulas can describe many real-world problems, and thus the SAT problem appears in a wide range of applications – such as model checking (see *e.g.* Biere et al., 1999), software and hardware verification (see *e.g.* Burch et al., 1994) or automated theorem proving (see *e.g.* Brown, 2013).

We use three SAT benchmarks that originate from the configurable SAT solver challenge (Hutter et al., 2017) and have been widely used in the subsequent AAC literature (see *e.g.* Lindauer and Hutter, 2018; Pushak and Hoos, 2020): a set of instances generated by a CNF fuzzing tool (CF) (Brummayer et al., 2010), a set of low auto-correlation binary sequence problems converted into SAT (LABS) (Mugrauer and Balin, 2013) and a set of 5-SAT problems generated uniformly at random from which only unsatisfiable instances have been kept (UNSAT).

Automated planning

AI planning aims at generating a sequence of actions to reach a goal. It has many applications to real-world problems, such as cyber security (see *e.g.* Boddy et al., 2005) or maintenance scheduling (*e.g.* Verbert et al., 2017). While AI planning scenarios have rarely been used in work introducing new AAC procedures, they were in analysis papers such as the work of Fawcett and Hoos (2016) and the prominence of AI planning led us to include those benchmarks

Our two automated planning benchmarks originate from the third International Planning Competition (Long and Fox, 2003) and were included in the analysis of Fawcett and Hoos (2016); a set about the control and observation scheduling of satellites (Satellite) and a set of route planning problems (Zenotravel) (Penberthy and Weld, 1994).

Mixed integer programming

The mixed integer programming problem is a well-studied constraint programming problem, in which we try to minimise or maximise an objective function while satisfying constraints on a given set of variables. Many problems can be formulated as instances of MIP, which leads to a wide range of applications.

Three of our four MIP benchmarks originate from a study on MIP solver configuration (Hutter et al., 2010a) and were included in follow-up work (see *e.g.* Ansótegui et al., 2015; Pushak and Hoos, 2020): a set of capacitated lot-sizing benchmark (CLS) (Atamtürk and Muñoz, 2004), a set of MIP problem instances generated with the combinatorial auction test suite (Leyton-Brown et al., 2000) and a set of real-life data for wildlife corridors for grizzly bears in the Northern Rockies (Gomes et al., 2008). The fourth benchmark stems from work on combining AAC and AS (Xu et al., 2011): a set of MIP-encoded habitat preservation data for the endangered red-cockaded woodpecker (Ahmadizadeh et al., 2010).

Table 2.3: Characteristics of the target algorithms.

Name	*Deterministic*	*Total*	*Categoricals*	*Integers*	*Continuous*	*Conditionals*	*Defaults*	*Forbidden*
Clasp	False	70	33	30	7	60	1	2
Lingeling	False	322	137	185	0	0	1	0
SpToRiss	False	222	170	36	16	190	1	21
Kissat	False	133	61	72	0	0	1	0
LPG	False	67	48	5	14	25	1	12
CPLEX	True	73	50	16	7	4	1	0
LKH	False	23	11	12	0	3	1	0
EAX	False	2	0	2	0	0	1	0

Traveling salesperson problem

The TSP is a well-studied optimisation problem. Given a set of points and the distance between them, one aims to find the shortest tour that visit each of the points at most once and returns to the starting point. Other than its application to package delivery, it is also widely used in industry for circuit engraving, for example (Matai et al., 2010).

Our benchmark contains two sets of generated random uniform Euclidean instances originally introduced with irace Ansótegui et al. (2009) and subsequently used in the GPS paper (Pushak and Hoos, 2020).

2.3.2 Target algorithms

In our experiments, we used prominent solvers for SAT, AI planning, MIP and TSP (see Table 2.3). In the following, we explain those choices.

Boolean satisfiability

Our SAT solvers were selected based on their performance in the Configurable SAT Solver Challenge (CSSC) 2014 (Hutter et al., 2017) : Lingeling (Biere, 2014) ranked first on the *industrial SAT+UNSAT* track and second on the *crafted SAT+UNSAT* track, Clasp (Gebser et al., 2012) first on the *crafted SAT+UNSAT* and *Random SAT+UNSAT* tracks and SparrowToRiss (SpToRiss) (Balint and Manthey, 2014) second on the *Random SAT* track. Because there have been advances in SAT solving

since then, we also added Kissat (Balyo et al., 2020), which was the winner of the SAT 2020 competition, is highly configurable, and is similar to CadiCal (Biere et al., 2020), which is known to benefit from configuration (Pushak and Hoos, 2020).

Automated planning

For automated planning, we selected LPG (Gerevini and Serina, 2002; Gerevini et al., 2003, 2008, 2011), as it has been successfully configured previously (Vallati et al., 2013; Fawcett and Hoos, 2016) and is also available through AClib. While there has been further development in AI planning, LPG has been very impactful in the field, as shown by the ICAPS 2019 Influential Paper Award received by Gerevini and Serina (2002).

Mixed integer programming

For MIP, we chose IBM's CPLEX solver, as it is widely used in practice and has shown great potential for performance improvement through AAC (Hutter et al., 2010a). We use version 12.6, since it is the version used by Hutter et al. (2010a) and thus included in AClib. CPLEX is widely used in the literature since it provides a state-of-the-art MIP solver free of charge for academic use.

Traveling salesperson problem

For TSP, we use LKH (Helsgaun, 2000) – an implementation of the Lin-Kernigan Heuristic – and EAX (Nagata and Kobayashi, 2013) – a Genetic Algorithm –, two prominent TSP solvers that have been configured in the study introducing GPS (Pushak and Hoos, 2020). They are considered to be state-of-the-art heuristic solvers (see *e.g.* Heins et al., 2024).

2.3.3 Scenarios

For all scenarios from AClib, we followed the setup defined there, including default configurations. As a configuration objective, we used minimisation of PAR10 (the average running time of the target algorithm, with timed-out runs counted as ten times the cutoff time). Moreover, for the MIP scenarios, we duplicated the scenarios to run them with a lower cutoff time, following the study introducing irace (Cáceres et al., 2017), in order to evaluate more closely the impact of the cutoff time on the performance of the configurators. The details are described in Table 2.4.

Table 2.4: Characteristics of the scenarios.

Benchmark	Algorithm	Cutoff [s]	Timeouts [*count (%)*]	Budget [s]
CF	Clasp	300	8 (2.7)	172800
	Lingeling		12 (4.0)	
	SpToRiss		16 (5.4)	
	Kissat		8 (2.7)	
LABS	Clasp		81 (23)	172800
	Lingeling		85 (24)	
	SpToRiss		89 (25)	
	Kissat		79 (23)	
UNSAT	Clasp		0 (0.0)	172800
	Lingeling		0 (0.0)	
	SpToRiss		0 (0.0)	
	Kissat		0 (0.0)	
Satellite	LPG	300	0 (0.0)	172800
Zenotravel			0 (0.0)	
CLS	CPLEX	10000	0 (0.0)	172800
COR-LAT			1 (0.1)	
RCW2			0 (0.0)	
REG200			0 (0.0)	
CLS	CPLEX	300	0 (0.0)	172800
COR-LAT			3 (0.3)	
RCW2			15 (3.0)	
REG200			0 (0.0)	
rue-1000-3000	LKH	86	12 (24)	86400
	EAX		7 (14)	

Table 2.5: Statistics on scenario characteristics

Category	Name	Domain	Min	Max	Median
Parameters	Number	$\mathbb{N}+$	2	322	71.5
Parameters	Categoricals	$\mathbb{N}+$	0	170	49
Parameters	Integers	$\mathbb{N}+$	2	185	23
Parameters	Continuous	$\mathbb{N}+$	0	16	3.5
Parameters	Conditionals	$\mathbb{N}+$	0	190	3.5
Parameters	Defaults	$\mathbb{N}+$	1	1	1
Parameters	Forbidden	$\mathbb{N}+$	0	21	0
Instances	Training size	$\mathbb{N}+$	50	2000	422
Instances	Testing size	$\mathbb{N}+$	50	2000	422
Instances	Features	$\mathbb{N}+$	64	305	148
Instances	Clusters	$\mathbb{N}+$	2	21	6.5
Scenario	Cutoff time	$\mathbb{N}+$	86	10000	300
Scenario	Timeouts	$[0, 100]$	0	1.5	25
Scenario	Budget	$\mathbb{N}+$	86400	172800	172800

Analysis

An overview of the salient characteristics of the AAC scenarios is given in Table 2.5.

Our set of scenarios shows a wide variety of search spaces with various numbers of parameters in each category. We note that, while some configurators can handle several default configurations as a starting point, all of our scenarios have only one default configuration, which is the case for most solvers, but could be extended by using configurations found to be well-performing in the literature. Additionally, the scenarios we consider contain only one deterministic algorithm. While this is a small number, it is also representative of NP-hard solvers, which often rely on stochasticity to more effectively search for solutions.

The number of instances in the datasets varies. We note that all scenarios come with instance features, which are exploited by some but not all configurators. This give an advantage to configurators that are using those features, *e.g.* to learn a surrogate model. We note that most of our datasets are artificially generated, albeit in such a way that they capture important characteristics of real-world applications. Also, almost all our benchmarks have an equal number of instances in the training and testing sets. This allows the testing set to contain a large variety of instances and have a testing performance that would better represent the distribution of instances, while having enough variety in the training set to avoid overfitting.

Finally, the scenarios show little diversity in their cutoff times and overall wall-clock time budget. However, this is a feature of AClib, and our decision to follow AClib's approach limits us in this aspect.

2.4 Conclusion

In this chapter, we introduced the Automated algorithm configuration problem. We highlighted three main challenges of this problem: the expensive evaluations of the target algorithm, the complexity of the configuration space, and the variability. Then, we described standard methods for solving it. We described a large and varied set of scenarios and datasets to be used in the remainder of this thesis and characterised them by means of a set of features, in an attempt to highlight the ways in which they relate to or differ from each other. The following chapter will provide an in-depth evaluation of the AAC methods introduced here.

3

Critical Assessment of the State of the Art in Automated Algorithm Configuration

As seen in the previous chapter, the automated algorithm configuration (AAC) literature contains many different scenarios. Each time a new configurator is introduced, it is evaluated on its own set of old and new scenarios, sometimes completely disjoint from each others, which greatly complicates comparisons between their performance based on their respective publications. Moreover, different publications tend to follow a different setup of experiments and run the algorithms on different machine execution environments which, for running time optimisation tasks, makes the comparison between results from several papers nearly impossible. In this chapter, we evaluate a diverse set of state-of-the-art configurators on the scenarios described in the previous chapter and present the first such evaluation for running time optimisation AAC scenarios. We present the obtained data and analyse them in light of the characteristics described in the previous chapter. To facilitate further research, we make all performance data openly available.

3.1 Background

As described in Chapter 2, there have been many attempts to build configurators, each focusing on specific challenges ofthe AAC problem and providing more insights on it. However, due to the variation in focus of the community, the continuous development of new solvers and the evolution of hardware, they were often tested on different datasets and benchmarks. An attempt at proposing a unified library of configuration scenarios has been made by Hutter *et al.* with AClib (Hutter et al., 2014a) but due to the extensive computation time typically required by configurators, researchers will typically limit the scope of their experiments to a subset of those scenarios. To the best of our knowledge, there has been no attempt in extensively comparing the current prominent configurators on a large set of scenarios nor to draw clear conclusions about their strength and weaknesses. This is the aim of the present chapter. To answer RQ2 – *How do state-of-the-art configurators compare to each others on a various set of scenarios?* – we run state of the art configurators on a large set of scenarios and compare their expected performance at the end of the given configuration budget. Then, we use the scenario features defined in Chapter 2 to answer to RQ3 – *How do scenario characteristics influence the performance of state-of-the-art configurators?* – and attempt to draw general conclusions about the strength and weaknesses of each configurator.

3.2 Evaluating the state of the art

Existing configurators, introduced in the previous chapter, have been developped and tested on different kind of target algorithms and with different applications in mind. Many configurators come with specific sets of features that reflect the requirements of the intended application. When performing an evaluation such as the one conducted here, it is hard to account for those differences, and for practical purposes, we reduced the considered options to those shared among most configurators and not requiring extensive pre-processing by end users (such as organising instances by families, deciding in which order they should be included in the search, or coding complex wrappers for the target algorithm). Moreover, we are aware that there is potential in tuning the hyper-parameters of the configurators, but this would involve a large computational cost. Thus, we are applying all configurators out of the box, following the developper's recommendations on their use. In the remainder of this section, we introduce the choices we made regarding the evaluation protocol and how we handled the level of

parallelisation of the configurators as well as their limitations in term of search space.

3.2.1 Evaluation protocol

To decide how we should evaluate a configurator, we have to go back to what we aim at measuring. As we configure, the aim of the configurator is to produce a configuration that will best generalise to other instances from the same underlying distribution. Since the configurators are not deterministic, the standard protocol is to run the configuration process several times on a training set and keep the configuration that performs best on that set (see the recommendations of Eggensperger et al., 2019). Because in many cases only a subset of instances is seen during the configuration process, it is common practice to validate on the training set rather than on a separate validation set (Eggensperger et al., 2019). To evaluate the ability to generalise, we can then look at the performance on a test set. In a case were the goal would be to improve the algorithm performance or analyse its best performing configurations, one would then compare the performance of the default configuration, or an expert-chosen set of parameter values, and the configured algorithm (*e.g.* Fokkinga et al., 2019).

When evaluating the configurator, we want to know how likely it is that the target algorithm performs better once configured than using the default configuration. Thus, we are interested in the distribution of possible configurations that this procedure would output. To do so, we would need to perform the above described procedure several times. To simulate this, we configure the target algorithm N times to create a pool of best found configurations, sample $n < N$ configurations that represent the set of configurator runs performed by a practitioner, and keep the best performing configuration on the training set out of those n configurations.

Definition 3.1. We call *standard protocol* the following protocol:

1. run $N \in \mathbb{N}$ times the configurator to create a set of N configurations
2. evaluate the performance of those configurations on the training set
3. uniformly sample $n < N$ configurations
4. keep the configuration with the best performance among the n samples
5. go back to step 3 until you gather the desired number of configurations

This protocol allows us to gather a distribution of *expected best configurations.* Rather than running steps 3 to 5 of the standard protocol, we can calculate the probability of each configuration to be chosen.

Theorem 3.1. Given a list of N configurations ω_i with $i \in [0, N-1]$ in ascending order based on their performance on the training instances, *i.e.*

$$\forall j \in [1, N-1], \mathcal{M}(\omega_{j-1}, \mathcal{I}_{\text{train}}) < \mathcal{M}(\omega_j, \mathcal{I}_{\text{train}}),$$

with $\mathcal{I}_{\text{train}}$ the training instances set and $\mathcal{M}(\omega_i, \mathcal{I}_{\text{train}})$ the performance of the configuration ω_i on the set of instances $\mathcal{I}_{\text{train}}$ that we try to minimise.

The probability for ω_i to be sampled through the standard protocol can be expressed as follows:

$$P(\omega_i) = \frac{n}{N-i} \times \frac{\binom{N-i}{n}}{\binom{N}{n}}$$

Proof. The total number of sets $\mathcal{S}$ of n configurations is $\binom{N}{n}$.
There are i configurations performing better than ω_i, thus the number of sets $\mathcal{S}$ such that $\forall \omega \in \mathcal{S}, \mathcal{M}(\omega_i, \mathcal{I}_{\text{train}}) \leq \mathcal{M}(\omega, \mathcal{I}_{\text{train}})$ is $\binom{N-i}{n}$. Thus, the probability that $\forall \omega \in \mathcal{S}, \mathcal{M}(\omega_i, \mathcal{I}_{\text{train}}) \leq \mathcal{M}(\omega, \mathcal{I}_{\text{train}})$ is $\frac{\binom{N-i}{n}}{\binom{N}{n}}$. Finally, the probability that $\omega_i \in \mathcal{S}$ is $\frac{n}{N-i}$.
All together, the probability of ω_i being the best performing sample in $\mathcal{S}$ is

$$\frac{n}{N-i} \times \frac{\binom{N-i}{n}}{\binom{N}{n}}.$$

□

We note that, if ω_0 is sampled it will always be outputed, its probability in the final distribution is thus $\frac{n}{N}$. On the other opposite, the $n-1$ last configurations will never be outputed as there will always be one better configuration chosen.

In the following, when comparing configurators, we will always consider the distribution of performance values for the configurations obtained by means of the standard protocol.

3.2.2 Configurators specificities

Despite our intention to consider all configurators on equal ground, there are difference in the way they handle the scenarios, in the variety of scenarios they can consider and on the way they use the resources given to them. Those specificities and their expected impact are listed below.

Parallelisation

Among the considered configurators (listed in Section 2.2.2), GGA++ and GPS are the only ones which need parallelisation to work as intended. The typical case for both would be to launch 8 workers according to discussion with the developpers and the paper which introduced GPS (Pushak and Hoos, 2020). On the other hand, while SMAC and irace support parallelisation, it is not core to their usage and they would typically not be used in that fashion. To allow us to compare parallel algorithms to purely sequential algorithms, we follow an approach similar to the one used by Pushak and Hoos in the paper introducing GPS.

For each configurator and each scenario, we run the sequential configurators 24 times and apply the standard protocol with a sample size of 8, while we run the parallel configurators 8 times with 8 workers and apply bootstrap sampling to obtain a larger sample size. These two approaches allow us to obtain the same number of samples for each configurator while avoiding the large computational incurred by running the parallel configurators as many times as the sequential ones. Moreover, Eggensperger et al. (2019) recommends using parallel runs of sequential configurators. By comparing 8 parallel runs to one run on 8 cores, we simulate a situation where a practitioner has access to one machine with 8 cores.

Search spaces

Configurators have limitations regarding the search space they are able to handle. paramILS, for example, can only handle discrete parameter space, which means that we made a discrete version of the search spaces. To generate this discrete search space, we take 10 evenly spaced values and add the default value to this set if it is not part of the chosen values. Since EAX has only 2 hyperparameters, which are both integers, we did not need to modify its search space.

An other limitation in search space is brought in by the way it is described. GGA++ and GPS search spaces are presented in a tree-like way, which limits the ability to handle conditionals. In particular, if one parameter depends on more than one other parameters, the user needs to duplicate it and create dummies that the configurator will not consider as linked. Since this requires a non-negligible amount of work, those were run only on solvers which do not have this particular feature. In particular, Clasp and SpToRiss have many conditionals and have been excluded. LPG had three such parameters, which made the required changes easy to implement, so we kept it.

Configuration budget

While we consider the budget given to a configurator in terms of wall-clock time, it is not possible to give it as-is to all configurators. In particular, irace and GGA++ were not developped with a concept of wall-clock time as a budget. By design, the budget of irace is expressed as a number of iterations. For compatibility, it includes a mechanism that allows to estimate the running time of the target algorithm and through it the number of iteration which can be done within the given configuration budget. This mechanism sometimes leads to irace running overtime or to it refusing to run if the given budget is found to be too short. GGA++, on the other hand, needs to be configured in terms of size of population and number of generation for the genetic algorithm. OPTANO, the freely available implementation of GGA++ we are using, provides a tool which allows to estimate how long the configuration run will take based on a few of its parameters. Based on the 48 hours budget of most of our scenarios (see Table 2.4) and our 8 workers, this tool led us to use a population of 100 configurations over 100 generations.

For both configurators, we stop after the given wall-clock time and take the last incumbent found.

3.2.3 Normalised score

For some of our analysis, we want to be able to aggregate or put next to each others performance data collected over several scenarios. Because there is a large variation in the running time of the respective solvers, we normalise the performance of the solvers using the default value and the best final performance as reference points.

For any given $\omega \in \Omega$, the normalised performance $\mathcal{M}'$ is thus defined as:

$$\mathcal{M}'_K(\omega) = \frac{\mathcal{M}(\omega_{\text{def}}, \mathcal{I}) - \mathcal{M}(\omega, \mathcal{I})}{\mathcal{M}(\omega_{\text{def}}, \mathcal{I}) - \mathcal{M}(\omega^*_K, \mathcal{I})}$$

where ω^*_K denotes the best known configuration for a scenario K. Since we do not know the best configuration overall, we evaluate all obtained configurations after the full configuration time and take the best performing configuration on the test set as an estimate for the lower bound.

3.2.4 Implementation details

Eggensperger et al. (2019) listed avoidable pitfalls and best practices in algorithm configuration. To avoid inconsistencies in the evaluation of the target algorithm running

time, we follow AClib in using a standard wrapper based on the `runsolver` software to evaluate the running time of each run in a similar way. Moreover, we make sure to move all instance files into the RAM of the compute node on which the target algorithm is run to avoid the speed of the file system to impact the time required to read the instances from external memory (such as a hard drive). To decrease latency, the code and executables are all stored on a `BeeGFS` file system. We perform all experiments on a computing cluster running Rocky Linux 9.3. Each node is equipped with 2 AMD EPYC 7543 32-core CPUs with 256 MB L3 cache and has 1TB of memory.

3.3 Evaluation results

This section shows the results of the experiments described in Section 3.2.1. All numbers are based on the expected performance described there. We first show the results and then draw high-level conclusions about the evaluated configurators. All our results are available on `ada.liacs.nl/ac-comparison`.

3.3.1 Overall performance

Table 3.1 summarises the results of our experiments comparing configurators performance. It shows the median of the expected performance following the protocol described in Section 3.2.1 with the configurations found at the end of the configuration budget (as specified in Table 2.4). The lowest median values are underlined. To evaluate the statistical significance of the difference between the configurator reaching the lowest median and the others, we applied a Mann-Whitney U-test ($\alpha = 0.05$). We tested both the significance of the difference in performance on the raw distribution of performances from and on the distribution of best configurations obtained through our full protocol. Since our protocol resamples the distribution with a bias towards the lower tail, it enlarges the differences between them and the test indicated significant differences in all cases. The result of the test applied to the raw distributions is shown in the Table 3.1 by putting in bold values for which the underlying raw distribution is statistically tied with the best one.

As could be expected, our random baseline ROAR can not find a better configuration than the default one in most scenarios. It improves on it in 9 of our 20 scenarios, meaning that in more than half of our scenarios, running random configurations for two days failed to achieve improvements over the default configurations. This is consistent with the premise that configuration scenarios are difficult and reinforces the

need for more sophisticated search algorithms. We note, however, that for SpToRiss on CF ROAR outperforms more sophisticated approaches.

Looking at the difference between SMAC2 and SMAC3 shows a clear advantage for the former. On all of our mixed integer programming (MIP) and traveling salesperson problem (TSP) scenarios, as well as on more than half of our Boolean satisfiability (SAT) scenarios, SMAC2 performs better than SMAC3. While SMAC3 is a Python reimplementation of SMAC2, some changes that might affect its performance have been introduced. In discussions with the authors, we discovered that the procedure for introducing random configuration among the challenger configurations tested against the incumbent was changed, because the one used in SMAC2 was too exploratory for hyperparameter optimisation of machine learning models. Instead of alternating between a random one and one based on the surrogate model, like the earlier version, they use a random configuration with a set probability. Compared to other configurators, SMAC2 shows very strong performance.

Despite being model-free and configuring on a discrete space, paramILS is among the best-performing configurators. One could expect that the best performing configurations might be excluded from the search space in many cases due to discretisation, but it seems that the reduction in search space size often compensates for this. It would be interesting to evaluate if running the other configurators on this same discrete space would improve their performance.

The performance of irace varies widely across target algorithms. Notably, it is among the best on all SpToRiss scenarios but performs very poorly on all Lingeling scenarios. While it has been argued in the past that the high cutoff time of the CPLEX scenarios hinders the performance of irace (Cáceres et al., 2017), we do not observe a large drawback. We note, however, that the CPUs on which our experiments are running have a significantly higher performance. This leads to most configurations to complete runs within the cutoff time. Thus, larger cutoff time does not make much difference in the number of instances finishing in time. The results reported in Chapter 4, which ran on a different machine, point in that same direction.

GGA++ (Ansótegui et al., 2015) performs unevenly. The output on both TSP scenarios, which have a low configuration time, is especially problematic since it is worse than the default; On the other hand, it achieves the best performance on both LPG scenarios and among the best on several others (*e.g.* Kissat on CF, Lingeling on UNSAT).

GPS is the only configurator that never ranks first. On many scenarios, it struggles to do better than the default. We note that the fact GPS runs a database to keep track

Table 3.1: Median expected performance of configurators on continuous search space, the best values are underlined and the values statistically tied with the best are boldfaced

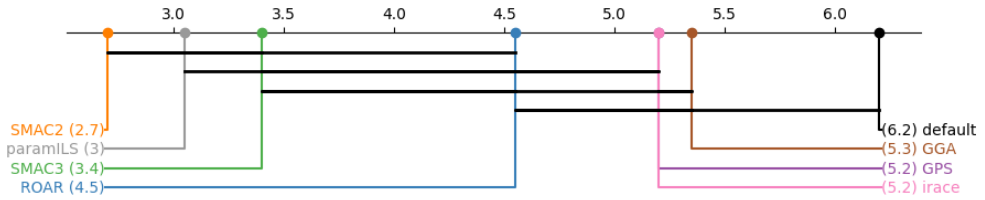
scenario	default	ROAR	SMAC2	SMAC3	irace	GGA++	GPS	paramILS
Clasp								
CF	103.29	103.29	102.73	**102.01**	**112.72**	X	X	110.81
LABS	707.25	707.25	**743.58**	699.40	**734.13**	X	X	**685.97**
UNSAT	0.42	0.19	0.18	0.18	0.18	X	X	**0.18**
Kissat								
CF	110.91	**110.91**	92.80	100.83	**140.48**	**90.57**	**100.85**	**89.83**
LABS	667.63	667.63	**691.16**	**676.01**	725.13	708.86	667.72	**666.69**
UNSAT	0.62	0.48	0.19	0.28	0.23	0.31	0.58	**0.19**
Lingeling								
CF	215.53	215.53	**165.12**	184.03	406.30	206.67	**224.52**	171.19
LABS	796.01	**796.01**	**780.05**	**795.42**	863.15	813.84	787.54	796.29
UNSAT	1.22	1.22	0.66	0.79	1.65	0.58	0.79	**0.55**
SpToRiss								
CF	326.40	**145.82**	**162.55**	**182.24**	151.37	X	X	183.35
LABS	811.09	747.13	751.49	752.49	744.46	X	X	**737.89**
UNSAT	151.42	0.97	0.84	0.88	**0.79**	X	X	0.83
CPLEX								
CLS	1.71	1.89	**1.24**	1.28	2.15	1.54	1.88	**1.07**
COR-LAT	10.95	10.47	**2.84**	9.46	5.06	18.96	10.47	10.65
RCW2	38.71	38.71	**25.94**	33.70	44.95	38.71	35.85	30.60
REG200	6.32	3.31	1.93	2.68	2.40	5.48	3.94	**1.73**
LPG								
Satellite	8.03	1.93	2.35	1.91	2.45	**1.90**	8.02	2.49
Zenotravel	12.56	**1.24**	1.06	**1.13**	0.90	**0.82**	12.52	**1.29**
EAX								
rue-1000-3000	120.82	**81.53**	**75.27**	**84.59**	**103.17**	**763.49**	**75.59**	**99.33**
LKH								
rue-1000-3000	229.22	**229.22**	**139.08**	**180.65**	227.97	586.82	**194.72**	**213.91**

of the performed runs might have affected our runs as a result of slow communication compared to the read and write speed in RAM for other configurators.

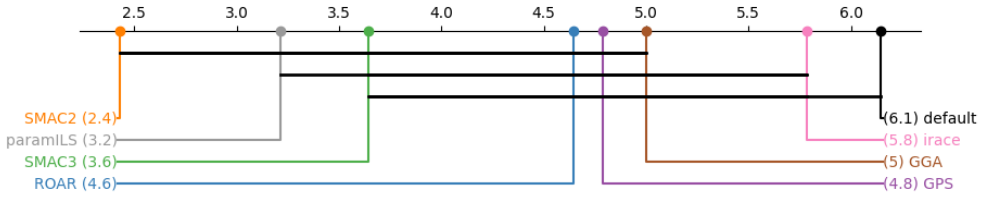
3.3.2 Comparative analysis

While the previous section explored the results in details, we now take a step back and look at the bigger picture. To do so, we first look at the widely used critical difference plot to determine if one of the configurators performs significantly better in terms of ranking over all of our datasets. Then, because aggregate performance over a broad range of scenarios is not always the desired goal, we evaluate the contribution each configurator would make to a configurator portfolio if we were to build one.

Since two of our configurators (GGA++ and GPS) could not run on all of the scenarios, we show our analysis separately for those compatible scenarios, named *tree-shaped scenarios* in the following, since they apply to target algorithms with a



(a) on all 20 configuration scenarios



(b) on the 12 tree-shaped scenarios

Figure 3.1: Critical difference diagram of the average score ranks of the configurators. Configurators linked with a black line were not found statistically different.

parameter space that can be represented as a tree.

Note that in this section, we base our evaluation on the average running time without penalty on timeouts rather than the penalised average running time. The factor applied could impact the soundness of the results of the statistical tests.

Critical difference diagram

In the machine learning community, a common approach for evaluating results across multiple datasets is the critical difference plot. Following the work of Demšar (2006), we first applied a non-parametric Friedman test with a significance threshold of 0.05 on our expected performances to verify that they come from different distributions. The test rejected the null hypothesis that the performance of the configurators is similar. We then applied a post hoc Nemenyi test with a significance threshold of 0.05 that compares each pair of configurators. The null hypothesis is that the two configurators have the same performances. We visualise the result in Figure 3.1, where the horizontal axis shows the average ranking of the configurator and pairs on which the Nemenyi test could not reject the null hypothesis are linked with a black line.

When looking at all scenarios in Figure 3.1a, we see that while all approaches achieve improvements over the default configurations, the differences are not statistically significant for ROAR, irace, GPS, and GGA++. This goes against the expecta-

tion that configurators consistently achieve improvements over the default configurations.

As mentioned in Section 3.3.1, the performance of paramILS is particularly strong considering that it is model-free and limited to searching a discrete space of parameter configurations. However, while it ranks first more often than SMAC2, looking at the average ranking over all scenarios gives SMAC2 an edge, albeit not a statistically significant one. ROAR, our random baseline with racing, performs fairly well and ends up tied with the best configurators: SMAC2, paramILS and SMAC3.

GGA++, irace and GPS perform similarly, with a slightly lower rank for GGA++ than for the other two. When looking at Figure 3.1b, we see that for tree-shaped scenarios, they are close to the performance of ROAR and join the statistically tied group of best-performing configurators. Interestingly, despite never placing first, GPS still achieves a better mean rank than GGA++ and irace.

Each configurator is expected to have strengths and weaknesses depending on the type of scenarios considered. Since we chose the scenarios in such a way that they cover a wide range of domains and characteristics, the fact that none of the configurators largely outperforms all the others in a statistically significant way aligns with our expectations.

Contribution to a portfolio

When evaluating the state of the art of NP-hard problems solvers, the idea of studying the contribution each solver makes to the state of the art, their *marginal contribution*, was first discussed by Xu et al. (2012) in the context of SAT solvers. To quantify the contribution of a solver, they compared the performance of portfolio techniques, such as automated selection or parallel portfolios, built with this solver to ones built without this solver. However, this approach suffers from several drawbacks, as pointed out by Fréchette et al. (2016). In particular, if two solvers solve the same set of instances, they would both have the same marginal contribution as a solver that solves no instance at all. Indeed, their marginal contribution compared to one another could be null.Removing both solvers from the portfolio would, however, degrade the performance of the portfolio. Thus, Fréchette et al. (2016) compute the contribution of the configurators with the Shapley value. For our application, the scores they used need to be slightly modified. In their case, they wanted to account more for the ability of a solver to solve an instance and thus computed a score that allowed them to give precedence to this objective. In our case, we are interested in the performance improvement achieved through automated configuration of a given target algorithm

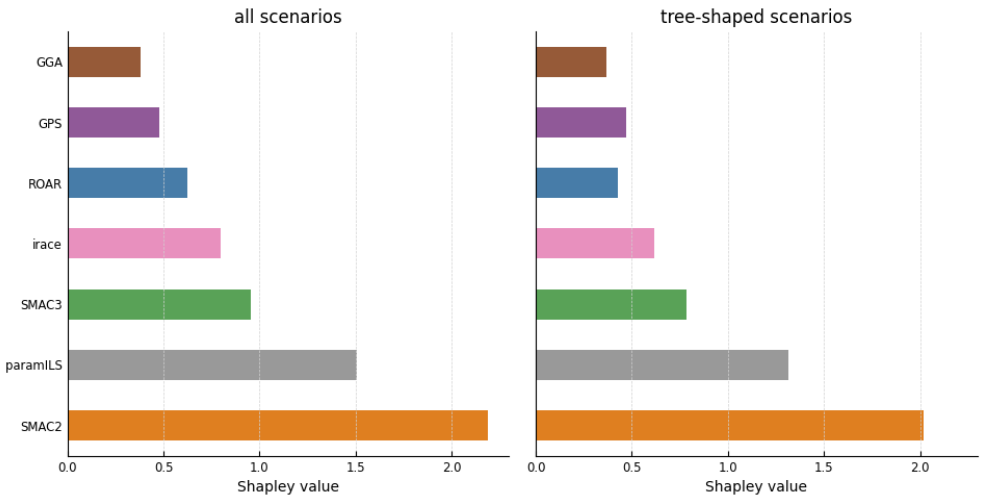


Figure 3.2: Shapley values of the configurators in an oracle portfolio

compared to using it with default parameter values. Thus, we use the normalised performance described in Section 3.2.3 and subtract it from 1 to obtain an objective function to be maximised.

Considering a set of configurators $\mathcal{C}$. The Shapley value $\phi(C)$ of a configurator $C \in \mathcal{C}$ is computed as follows:

$$\phi(C) = \frac{1}{|\mathcal{C}|} \cdot \sum_{G \subseteq \mathcal{C} \setminus \{C\}} \frac{v(G \cup \{C\}) - v(G)}{\binom{|\mathcal{C}-1|}{|G|}}$$

, where v returns the score of the best configurator for a given configuration scenario among the given set.For our set of scenarios $\mathcal{K}$,

$$v(G) = \sum_{K \in \mathcal{K}} \max_{C \in G} score_K(C) \text{ with } score_K(C) = 1 - \mathcal{Q}'_K(C)$$

, where $\mathcal{Q}'_K(C)$ denotes the median of the performance values of configurator C, normalised as per Section 3.2.3.

The obtained contributions are shown in Figure 3.2.

Consistent with the results shown in Section 3.3.2, SMAC2 achieves the highest contribution, followed by paramILS and SMAC3. However, despite its low overall ranking, irace makes a higher contribution than GGA++ and GPS, indicating that it complements the top three configurators. When only considering the tree-shaped

scenarios, GGA++ and GPS reach a higher contribution, but still fall short of irace.

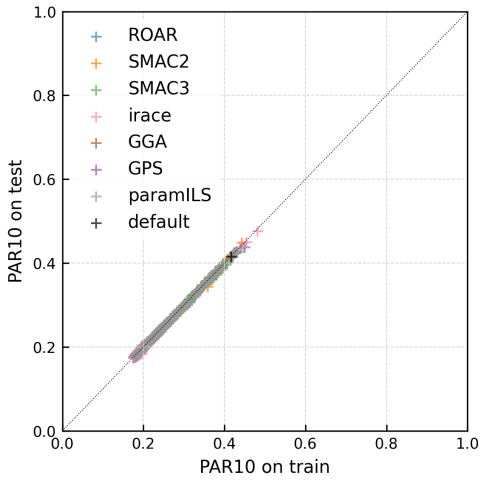
3.3.3 Overtuning analysis

A known problem of AAC is the difficulty to find configurations that generalise well to previously unseen instances, as shown by Eggensperger et al. (2019). To evaluate how much the configurators we studied are impacted by this difficulty, we traced back all configurations evaluated and stored throughout the configuration run and compare their performance on the training and testing sets. This allows us to visualise the overtuning behaviour described in prior work (see *e.g.* Schneider et al., 2025; Eggensperger et al., 2019). The plots thus obtained are shown in Figure 3.3.

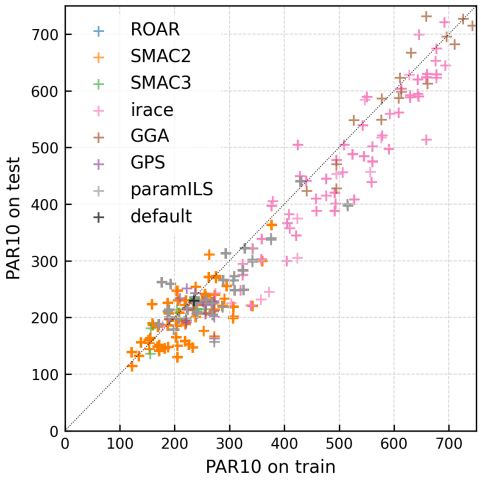
Figure 3.3 shows scenarios with the four types of behaviours we observed. Clasp on UNSAT (Figure 3.3a) is a case in which there is no sign of overtuning. The performance improvement on the training set correlates with the improvement on the testing set. Figure 3.3b shows an example for which the correlation is weaker, but still clearly visible. This type of behaviour is found more often than the previous one on the UNSAT dataset.On the other hand, CPLEX on CLS (Figure 3.3c) shows clear signs of overtuning from SMAC2, which kept many configurations with low PAR10 values on the training set and high PAR10 values on the testing set. Finally, Figure 3.3d shows a scenario for which the performance on the training and testing sets show lower correlation, though the PAR10 values still loosely follow the diagonal. This can indicate that the training and testing sets are related in a different way than the linear correlation detected with the Pearson correlation we used. Except for the outlier case on SMAC2 shown in Figure 3.3c, all configurators show a high correlation between the performances on the training and testing sets for our scenarios, from 0.6 on scenarios such as SpToRiss on LABS (Figure 3.3d) to above 0.9 for cases scenarios such as Clasp on UNSAT (Figure 3.3a)

3.3.4 Scenario features analysis

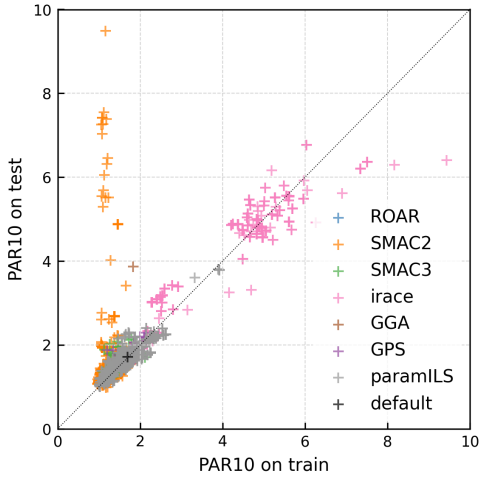
Finally, we investigate the impact of the features defined in Chapter 2 on the performance of the configuration approaches. To do so, we first compute the correlation between the features and normalised configuration performance. Then, we use the four CPLEX scenarios to investigate the impact of the choice of cutoff time for the target algorithm, since those scenarios have been used in the literature with two different cutoffs.



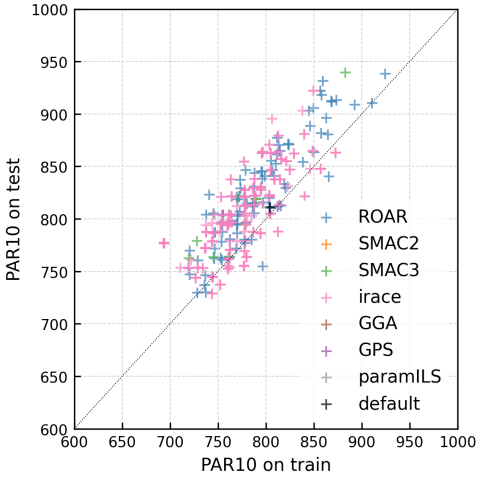
(a) Clasp on UNSAT



(b) LKH on rue-1000-3000



(c) CPLEX on CLS



(d) SpToRiss on LABS

Figure 3.3: PAR10 values on the train and test set for a subset of our scenarios

Table 3.2: Correlation between the normalised performance of the configurators and the scenario features

	ROAR	SMAC2	SMAC3	irace	GGA++	GPS	paramILS
Origin	0.21	-0.35	0.14	0.04	0.04	-0.07	0.12
Training size	-0.57	-0.43	-0.46	-0.52	-0.33	0.04	-0.34
Testing size	-0.56	-0.41	-0.44	-0.51	-0.23	0.01	-0.29
Features	-0.50	-0.40	-0.50	-0.38	-0.46	0.22	-0.42
Clusters	0.03	-0.07	0.14	-0.03	0.04	0.04	0.21
Deterministic	0.22	-0.42	-0.08	0.06	0.00	-0.10	-0.19
Total	0.16	0.22	0.16	0.34	-0.38	0.02	0.03
Categoricals	-0.05	0.10	0.02	0.10	-0.37	0.13	-0.06
Integers	0.37	0.31	0.29	0.52	-0.27	-0.12	0.13
Continuous	-0.58	-0.30	-0.42	-0.52	-0.29	0.35	-0.31
Conditionals	-0.36	0.06	-0.11	-0.32	-0.13	0.29	-0.03
Forbidden	-0.55	-0.11	-0.31	-0.44	-0.21	0.29	-0.19
Cutoff	0.21	-0.42	-0.08	0.06	-0.01	-0.09	-0.19
Timeouts	0.39	0.75	0.66	0.25	0.30	0.09	0.70
Budget	-0.07	-0.09	-0.12	-0.05	-0.82	0.36	-0.26

Correlation analysis

As a first high-level analysis regarding the impact of scenario features on the performance of the configurators, we look at the correlation between those features and the normalised performance of the configurators. We show those correlations in Table 3.2. We note that the normalisation depends on the performance of other configurators.

We observed that the size of the training and testing sets are negatively correlated with the performance of most configurators: a larger number of instances corresponds to a better performance (since lower is better). We notice that the overall number of parameters does not seem to impact the performance much, while the type of those parameters impacts the outcome. Continuous parameters seem easier to configure that integers and categoricals, except for GPS. Regarding timeouts, the more there are, the worse performance we reach for all but GPS. This aligns with expectations, since more timeouts means less information for the model to learn from and more time spent on running on those hard instances. Methods for spending less time on those challenging instances will be further developped in Chapters 5 and 6.

Table 3.3: Median expected performance of configurators on CPLEX scenarios, the best value for each configurator on a scenario is underlined

configurator	CLS		COR-LAT		RCW2		REG200	
	long	short	long	short	long	short	long	short
default	1.72		23.12		115.97		6.13	
ROAR	1.75	1.72	22.44	23.12	116.10	115.97	3.31	3.22
SMAC2	1.24	1.21	3.02	3.64	52.05	57.06	1.93	1.91
SMAC3	1.44	1.55	18.98	22.27	89.92	94.17	2.68	2.92
irace	2.15	1.57	7.74	6.46	149.58	120.60	2.40	2.25
GGA++	1.54	1.60	41.17	3.70	115.97	116.17	5.48	2.08
GPS	1.88	1.83	22.08	12.52	113.81	114.45	3.94	3.95
paramILS	1.19	2.92	22.43	8.62	86.11	83.35	1.73	1.75

Impact of the cutoff time

As previously discussed, the number of timeouts impacts the performance of the configurator. A characteristic related to the number of timeouts is the cutoff time, the maximum time given to the target algorithm before it is terminated. To investigate the impact of this characteristic, we ran the CPLEX scenarios with both the AClib based cutoff time and the one used by Cáceres et al. (2017). We compared the results of the configurators on those scenarios in Table 3.3. We note that the reduction of cutoff time did not impact the number of timeouts for CLS and REG200, while it brought it from 0.1 to 0.3 percent on COR-LAT and from 0 to 3 percent on RCW2.

Overall, we observed that the use of a shorter cutoff time does not have a clear impact, even when it leads to a higher number of timed out runs. Looking at the results per configurator, we observe that a shorter cutoff consistently benefits irace, while a longer cutoff time consistently benefits SMAC3.

3.4 Conclusion

In this chapter, we evaluated state-of-the-art configurators on a diverse set of scenarios for running time optimisation as introduced in Chapter 2. We showed that configurators typically find configurations better performing than the default value, when given the time to perform 500 or more target algorithm run. However, they do not always surpass ROAR, a simple random search with a racing mechanism. We found that on the configuration scenarios we considered paramILS was best on the most scenarios, while SMAC2 had the best average ranking. Though these advantages

are not statistically significant over all other configurators. Overall, we found that configurators show complementary strength and weaknesses. In particular, while irace did not perform strongly overall, it showed a significant contribution to a portfolio. This complementarity was already visible in the results, for example when configuring SpToRiss.

We analysed to what extend the performance of the best found configurations differ on the testing set to their performance on the training set to detect if the configurators are prone to overtuning, (*e.g.*finding a well performing configuration on the training set that does not generalise to the testing set). We did not find any strong evidence of it occurring on the scenarios we studied, with only one clear case with SMAC2 when optimising CPLEX for CLS.

Finally, we made an attempt at drawing high-level insights regarding the impact of scenario specific features such as the ones defined in Chapter 2. We found that the type of the parameters have more impact on the difficulty of a configuration scenario than their overall number, that scenarios with more instances to train on are typically easier and that the performance of GGA++ is strongly correlated with the configuration budget. Though we note that our scenarios did not include many variations of configuration budget and this correlation might be related to other elements of the small budget scenarios.

While we performed various analysis on the collected data, we believe that more can be done, such as defining new scenarios features to analyse or trying to predict which configurator should be used on a new scenario, and thus made it openly available for further exploration.

4

Default Value

General-purpose automated algorithm configuration procedures have enabled significant improvements in the state of the art for a wide range of challenging problems. This increasingly encourages algorithm designers to expose more parameters to the configuration process, leading to larger configuration spaces. To search these vast combinatorial spaces of parameter settings is challenging. Successful configurators combine techniques such as racing, estimation of distribution algorithms, Bayesian optimisation and model-free stochastic search (see Section 2.2.2). However, most methods disregard the default parameter values typically provided by the developers or merely use them as a starting point for the configuration process. This chapter[1] explores how and to what extent these default parameter values can be used as prior knowledge to guide the configuration search. First, we propose a simple method to reduce the size of the search space and restrict the configurator to parameter values close to a given default. Based on the encouraging results of this naïve approach, we introduce a principled method for integrating the default value into the configuration process by focusing the search around it.

[1] Parts of this chapter have been published as Anastacio et al. (2019); Anastacio and Hoos (2020a,b).

4.1 Introduction

The availability of effective automated algorithm configuration (AAC) procedures allows algorithm designers to expose design choices as parameters, following a design paradigm known as programming by optimisation (PbO) (Hoos, 2012b). By avoiding premature choices in algorithm design, automated methods can adapt the behaviour of algorithms to each specific use case, thereby achieving better performance. However, the size of the combinatorial configuration spaces encountered in this context grows exponentially with the number of exposed parameters. In the following, we will look in more depth into the configuration space and more closely to the default parameter settings and the knowledge hidden in it. These settings are typically set based on the algorithm designers' intuition on the inner workings of their methods, as well as experiments that were conducted to evaluate it. As such, they contain more information than random values would. Whilst state-of-the-art algorithm configurators combine sophisticated methods to search the parameter space effectively, we argue that they often overlook precious information that could help them focus their search on the most promising values.

4.1.1 Background

As explained in Chapter 2, the AAC problem consists of an algorithm A, a configuration space Ω, a set of problem instances $\mathcal{I}$ and a performance metric m. Ω contains all valid combinations of parameters $\mathcal{P}$ and their possible values $\mathcal{D}$. More precisely, each parameter $p_j \in \mathcal{P}$ has a domain $\mathcal{D}_j \in \mathcal{D}$ of possible values and typically a default value $D_j \in \mathcal{D}_j$. This default value would typically be used if a user is unsure of what would work best for their particular problem case or when they try a new algorithm. Algorithm developers usually provide a default parameter configuration that has been chosen to perform reasonably well across a broad range of problem instances, which involves substantial human intuition, experience and at least limited manual experimentation.

Because the configuration landscapes of numerical parameters tend to be benign (Pushak and Hoos, 2018), even limited manual tuning may produce valuable information about promising parameter values. Moreover, starting the model-based configurator SMAC (Hutter et al., 2011b) from a performance model learned on another set of benchmark instances allows considerable speedups (Lindauer and Hutter, 2018). This suggests that knowledge regarding high-quality configurations can be transferred between sets of problem instances. Together, these observations indicate that default

parameter configurations may contain valuable information that can be exploited for AAC. For scenarios with large configuration spaces in which high-performance configurations are complex to find, we conjecture that searching configurations inside a reduced search space, obtained by restricting the ranges of specific parameters, can be more effective than searching the whole configuration space. Despite the fact that pruning the search space could potentially exclude good configurations, we show that searching on this reduced search space improves the performance of irace, GGA++ and SMAC. Since this first method would require the user to define those smaller search spaces, we want to integrate a mechanism into a configurator that focuses the search around the default without leaving out parts of the search space. We propose focusing the search around the default value by modifying the sampling distribution of SMAC.

4.1.2 Related work

The idea of excluding part of the search space in optimisation algorithms is common and originates as far as the work of Megiddo (1983). It has been applied in the context of automated machine learning (autoML) by excluding areas deemed to be poorly performing (Wistuba et al., 2015), or by focusing on the hyperparameters found to have a significant impact on the performance of the target algorithm (Li et al., 2022). In the context of AAC, the configurator GPS (Pushak and Hoos, 2020) is also based on a pruning approach.

To focus the search around the default value, we use a similar intuition as the estimation of distribution included in irace. Moreover, it has been demonstrated that automatic configuration with irace (López-Ibáñez et al., 2016) can be sped up by focusing the search process on specific areas of a given configuration space. Franzin et al. (2018) applied transformations to a real-valued parameter of a simulated annealing algorithm and showed that the right transformation can improve the performance of irace. A similar kind of mechanism has been developed later for Bayesian Optimisation (Souza et al., 2021).

4.1.3 Research questions

In this chapter, we focus on two of the research questions introduced in Section 1.2.

RQ4 *How and to which extent do current configurators use the default values usually provided by algorithm developers?*

Considering that there is information in the default value, the first thing to question

is how much current methods benefit from this information. Oftentimes, this value is used as a starting point for the search. To evaluate the extent to which the default parameter values are exploited by state-of-the-art AAC procedures, we compare the performance of two prominent configurators, SMAC and irace, with or without using the default setting to initialise the configuration process.

RQ5 *How can we make better use of known good parameter values – in our case, the default value – to guide the search strategy?* We separate this question into two parts as follows:

RQ 5.1. Can we reach state-of-the-art performance by searching only around the default value?

A naïve approach to leverage the default value of continuous parameters is to exclude any value that is deemed too far from it. This also significantly reduces the size of the search space of the configurators. We define two neighbourhoods of the default values and evaluate their impact on the performance of several state-of-the-art configurators across a broad range of configuration scenarios.

RQ 5.2. What is the impact of focusing the search around the default value?

A more refined approach to focusing the search around the default value would not completely exclude values that are further away from it, but sample more values in the promising area around the default than in the least promising areas of the search space. To achieve this, we use a truncated normal distribution centred around the default value. We evaluate the impact of such a distribution on SMAC.

4.2 Protocol of experiments

To answer our three research questions, we will need to compare the performance of configurators. For each such comparison, we followed the same protocol.

Scenarios

All configuration scenarios were run according to the setup described in AClib (Hutter et al., 2014a). The considered configuration objective is the minimisation of PAR10 (average running time of the target algorithm, with the timed-out runs counted as ten times the cutoff time) for Boolean satisfiability (SAT), mixed integer programming (MIP) and automated planning (AI planning). More details regarding the benchmarks and algorithms can be found in Chapter 2 and the scenarios' characteristics

are described in Table 2.4. To this set, we add four autoML scenarios originating from AClib on which we optimise the cross-validated error rate. They all use the same machine learning system, autoweka (autoWK) (Thornton et al., 2013; Kotthoff et al., 2017), an autoML system based on, and distributed as part of, the WEKA machine learning and data mining workbench. We configure this system on four datasets (from Dua and Karra Taniskidou, 2017; Thornton et al., 2013): CAR contains 1728 instances of car evaluations, GC contains 690 instances classifying people as good or bad credit risks, WF contains 5000 generated waves, and WQW contains 4898 instances modelling wine quality based on physicochemical tests.

Protocol

Algorithm configurators are randomised, and their performance is known to vary substantially between multiple independent runs on the same scenario. Thus, we ran each configurator independently 24 times for each scenario and evaluated the resulting 24 parameter configurations on our training and testing sets.
Moreover, to leverage this variability, it is best practice to perform multiple independent runs of a configurator on a given scenario (usually in parallel) and to report the best configuration (evaluated on the training instances) as the final result of the overall configuration process (see the recommendations of Eggensperger et al., 2019). To capture the statistical variability of this standard protocol, we repeatedly sampled 8 runs uniformly at random and identified the best one according to its performance on the training set. We used 10 000 such samples to estimate the probability distribution of the quality of the result produced by each configurator on each configuration scenario. Note that the outcome of such a sampling approach should follow the same distribution as the one obtained following Section 3.2.1. We then compared the medians of these empirical distributions, using a one-sided Mann-Whitney U-test ($\alpha = 0.05$) to assess the statistical significance of observed performance differences.

Configurators

We included SMAC, irace and GGA++ in this comparison. More details on those configurators can be found in Chapter 2. We excluded paramILS since it requires a discrete search space, which might interfere with the methods we propose. GPS had not been proposed at the time of this study. For irace, when the estimated running time of the algorithm was too long and it refused to run within the given time budget, we applied the same protocol to the successfully completed runs. This happened in

particular for the CPLEX scenarios, where AClib prescribes a cutoff time of 10 000 seconds. In such cases, when the random seed (ranging from 1 to 24) results in the selection of a hard instance for evaluating the running time of the default configuration, irace determines that the running time is too high for the given configuration budget. The decision to apply the standard protocol to successful runs of irace leads to a positive bias in the irace results for CPLEX on CLS, where 19 runs were completed successfully. However, in cases such as the scenarios for CPLEX on RCW2 and on REG200, where only 7 and 9 runs, respectively, terminated successfully, we omit the results from our analysis, as application of the standard protocol would lead to extreme distortions from realistically achievable performance.

Code and execution environment

All experiments were performed on a computing cluster with CentOS using dual 16-core 2.10 GHz Intel Xeon E5-2683 CPUs with 40 MB cache and 94 GB of RAM. Our source code and results are available at `ada.liacs.nl/projects/smacps`.

4.3 Impact of the default

As mentioned previously, most configurable algorithms come with default parameter values that the developer has carefully chosen. Each AAC procedure handles those default values differently and thus is impacted differently by it. In this section, we explore the impact that the default configuration has on existing configuration procedures.

To answer RQ4 – *How and to which extent do current configurators use the default values usually provided by algorithm developers?* – let us look into the behaviour of two configurators. SMAC uses the default configuration as one of the starting configurations for building the model that guides its search process. irace not only uses the default as one of its initial racing configurations, but also as a way to estimate the running time of the target algorithm and thus the number of iterations.

For two of them, we compare their performance when provided with the default value given in AClib against their performance when no default value is given. The results obtained following the protocol described in Section 4.2 are reported in Table 4.1.

SMAC

SMAC finds better configurations for 5 out of 10 scenarios when starting from default parameter values, for 4 out of 10 scenarios starting from a random configuration produces better results, and for one scenario, no significant difference is observed. Overall, there seems to be no clear advantage in using the default configuration as a starting point. This is not surprising, considering that SMAC very quickly bases its search on the predictions of its random forest model and randomly chosen configurations. The latter, included to avoid stagnation of the model-based search process, likely limits the impact of using a specific starting configuration.

irace

When given a limited time budget (as in all our experiments), irace uses the default configuration of the given target algorithm to estimate the running time of the target algorithm. Consequently, when starting from a randomly chosen configuration, under which the target algorithm performs very poorly, irace may refuse to run, since it assumes there is insufficient time for the racing process to produce meaningful results within the given time budget. The results reported in Table 4.1 show only the results for scenarios for which at least 8 configurator runs finished within 5 times the given overall time budget for configuration. irace was unable to run on 2 of our 10 benchmark scenarios. For 5 others, it performs better when given access to the default configuration, and for the 3 remaining scenarios, starting from a random configuration produces better results. We note that irace systematically ran over time when starting from a random configuration.

What is the impact of the default value on current configurators?

To answer this question, we compared how widely used configurators perform with and without access to meaningful default parameter settings. We found that the performance of two state-of-the-art configurators, SMAC and irace, is affected very differently by default settings: While SMAC only rarely benefits from reasonable defaults, they are often crucial for irace.

4.4 Reduction of the search space

Automatic algorithm configurators are sophisticated search algorithms searching the configuration space of a given target algorithm. Due to the various types of parameters,

		Default	SMAC		irace	
			def	rand	def	rand
SpToRiss	CF	424.43	223.27	**196.62**	224.10	**223.05**
	LABS	885.78	**780.94**	**787.00**	**803.03**	815.06
	UNSAT	152.33	1.25	**1.13**	1.36	**1.20**
CPLEX	CLS	3.46	**2.14**	2.45	4.44	**3.99**
	COR-LAT	52.14	**7.75**	7.98	**10.64**	14.56
	REG200	10.83	**4.04**	4.13	**4.55**	4.86
AutoWK	CAR	0.500	**0.250**	0.270	**0.300**	-
	GC	0.590	0.330	**0.280**	**0.240**	-
	WF	1.97	0.340	**0.220**	**0.230**	0.300
	WQW	1.83	**0.350**	0.360	**0.370**	0.410

Table 4.1: Results for SMAC (left) and irace (right) for default and random initial configurations; median PAR10 (in CPU sec) for SAT and MIP, 10-fold cross-validated error rate for ML; best results are underlined, while boldface indicates results that are statistically tied to the best, according to a one-sided Mann-Whitney test ($\alpha = 0.05$).

the conditions linking them and the effects of their interaction, it can be hard to search the space of possible configurations. As argued earlier in this chapter, we have reasons to believe that the default value is located in a promising area of the search space. To test this hypothesis and answer RQ 5.1 – *Can we reach state-of-the-art performance by searching only around the default value?* – we restrict the search space of the configurator to the neighbourhood of the configuration.

4.4.1 Reduction methods

We will focus on integer- and real-valued parameters for our experiments, as their domains are often large intervals and reducing them can significantly impact the size of a given configuration space. Moreover, there is no straightforward way to reduce the range of a categorical parameter in a meaningful manner. We note that some solver have values in their integer-valued parameters with a specific meaning, different from the expected one. This is particularly true for CPLEX, which uses -1 as a special value, meaning that CPLEX will decide itself which value to use for this parameter. However, we did not treat those values differently in our reduction.

How to reduce parameter domains?

For $k \in \{1, 2, ..., n\}$, the domain of the integer- or real-valued parameter p_k is defined by its lower bound $D_{k,min}$, its upper bound $D_{k,max}$ and its default value D_k. The length of its range is denoted $D_{k,range} = D_{k,max} - D_{k,min}$.

We consider two different reduction techniques to calculate a new domain $\mathcal{D}_{k,sub}$. To avoid reducing small ranges, we apply domain reduction only to parameters with at least 10 possible values for integers and a range of 1 for real numbers. For parameters that vary on a logarithmic scale, we apply the reductions in the logarithmic domain.

Our first technique reduces the domain of a given parameter so it begins at one-tenth of the default value and ends at ten times the default value:

$$\mathcal{D}_{k,sub} = [0.1 \cdot D_k, 10 \cdot D_k]\,. \tag{R1}$$

This technique has the advantage of scaling up when the default value is big. For example, given the parameter p_k with a domain $D_k = [0, 10\,000]$, a default value $D_{k,\text{def}} = 500$ leads to a large range of $D_{k,\text{reduced}} = [50, 5\,000]$ while a default value of $D_{k,\text{def}} = 10$ leads to a large range of $D_{k,\text{reduced}} = [1, 100]$. However, it handles domains that span both positive and negative values poorly. For example if p_k has a domain $D_k = [-100, 100]$ and a default value $D_{k,\text{def}} = 5$ then the reduced range will become $D_{k,\text{reduced}} = [0.5, 50]$.

Our second technique reduces the range to a tenth of $\mathcal{D}_k$, centred around the default value:

$$\mathcal{D}_{k,sub} = \left[D_k - \frac{D_{k,range}}{20}, D_k + \frac{D_{k,range}}{20}\right]. \tag{R2}$$

Unlike the previous one, this technique does not have any issues with domains that span both positive and negative values.

For example if p_k has a domain $D_k = [-100, 100]$ and a default value $D_{k,\text{def}} = 5$, the reduced range will become $D_{k,\text{reduced}} = [-15, 25]$.

As $\mathcal{D}_{k,sub}$ thus defined could exceed the bounds of $\mathcal{D}_k$, we obtain the reduced range by taking the intersection of the two ranges. The final reduced range $\mathcal{D}_{k,red}$ is defined as

$$\mathcal{D}_{k,red} = \mathcal{D}_k \cap \mathcal{D}_{k,sub} \tag{4.1}$$

Moreover, each step of the reduction is made such that we keep at least 10 possible values for integers and a length of 1 for real numbers, meaning that this rule applies to $\mathcal{D}_{k,sub}$ and we add values to $\mathcal{D}_{k,red}$ as a final step if needed.

Solver	Parameters		Reduced parameters			
	Total	Numerical	R1		R2	
Clasp	75	37	25	(24)	29	(28)
Lingeling	322	185	154	(0)	163	(0)
SpToRiss	222	52	10	(9)	20	(16)
LPG	67	19	7	(4)	12	(8)
CPLEX	74	23	20	(2)	20	(2)
AutoWK	702	226	60	(60)	111	(111)

Table 4.2: Target algorithms parameter space description; The total number of numerical parameters; the number of parameters reduced by our techniques and, in parentheses, the number of these conditionally dependent on at least one other parameter.

How does the range reduction affect the configuration spaces of the considered algorithms?

We applied both reductions to the parameter space of the six algorithms appearing in our configuration scenarios. As seen in Table 4.2, for Clasp, CPLEX and Lingeling, the ranges of between 27 and 50% of the total number of parameters are reduced, which suggests that there is potential for a large impact when configuring them. We note that the parameters of Clasp to which reduction is applied almost all conditionally depend on at least one other parameter value, which may reduce the effect of domain reduction in this case. For SpToRiss, LPG and AutoWK, only 5 to 17% of the parameters are affected by domain reduction, and we thus expect the effect on configurator performance to be less pronounced.

4.4.2 Results

To evaluate the impact of search space reduction on the configuration process, we ran extensive experiments with SMAC, as well as more limited experiments with irace and GGA++. We then applied the protocol described in Section 4.2 to compare the results obtained for the full configuration space with those obtained using our two reduction techniques.

How do our reduction techniques impact the performance of SMAC?

As seen in Table 4.3, reducing the search space allowed SMAC to perform better for 15 out of the 20 studied scenarios, although for one of these, none of the optimised configurations reached the quality of the default configuration (on testing data). For the 5

		Quality			
		Default	full	R1	R2
Clasp	CF	174.05	**164.72**	173.67	173.60
	LABS	**718.15**	728.87	720.62	736.70
	UNSAT	0.847	0.380	**0.376**	0.383
Lingeling	CF	278.52	245.48	**187.11**	228.65
	LABS	808.70	830.63	**788.36**	849.25
	UNSAT	2.03	1.07	**0.984**	1.04
SpToRiss	CF	424.43	223.27	212.04	**204.21**
	LABS	885.78	780.94	**756.55**	805.10
	UNSAT	152.33	**1.25**	1.30	1.29
LPG	Depots	26.77	0.776	**0.709**	0.826
	Satellite	16.72	3.83	**3.70**	3.79
	Zenotravel	22.49	**1.67**	1.75	1.72
CPLEX	CLS	3.46	**2.14**	2.43	2.69
	COR-LAT	52.14	7.66	**6.01**	14.85
	RCW2	64.98	**52.64**	65.11	54.60
	REG200	10.83	4.04	**3.53**	3.73
AutoWK	CAR	0.500	0.250	**0.220**	0.280
	GC	0.590	0.330	0.280	**0.270**
	WF	1.97	0.340	0.370	**0.280**
	WQW	1.83	0.350	0.390	**0.340**

Table 4.3: Results for SMAC; median PAR10 (in CPU sec) for SAT, MIP and Planning, 10-fold cross-validated error rate for ML; best results are underlined, while boldface indicates results that are statistically tied to the best, according to a one-sided Mann-Whitney test ($\alpha = 0.05$).

remaining scenarios, significantly better results were obtained for the full configuration space. Reductions R1 and R2 lead to improved results for 12 and 8 scenarios, respectively. These improvements are observed across all AI problems and target algorithms that we studied.

Do our observations generalise to other configuration approaches?

Since each evaluation requires long and computationally expensive configuration runs, we ran limited experiments with irace and GGA++. We tested one target algorithm for each type of problem and chose among the benchmarks based on the results obtained by SMAC. We kept, for each problem type, one scenario on which each reduction technique improved the results of SMAC, as well as one for which it worsened them.

			irace quality			GGA++ quality		
		Default	full	R1	R2	full	R1	R2
SpToRiss	CF	424.43	**224.10**	**206.72**	246.99	233.11	235.83	**225.82**
	LABS	808.70	**803.03**	788.66	**787.47**	835.32	**804.31**	847.83
	UNSAT	152.33	**1.24**	1.28	**1.23**	**1.53**	1.61	1.54
LPG	Satellite	16.72	**4.94**	6.41	6.45	**3.69**	3.96	3.78
	Zenotravel	22.49	2.35	1.90	**1.84**	6.28	3.32	**3.07**
CPLEX	RCW2	64.98	69.62	68.16	**60.48**	63.87	63.69	**63.43**
	REG200	10.83	4.55	4.69	**3.86**	12.01	11.30	**10.57**
AutoWK	CAR	0.500	**0.210**	0.370	0.230	0.300	0.630	**0.260**
	WF	1.97	**0.230**	0.250	0.260	2.69	**2.40**	2.55

Table 4.4: Results for irace and GGA++; median PAR10 (in CPU sec) for SAT, MIP and Planning, 10-fold cross-validated error rate for ML; best results are underlined, while boldface indicates results that are statistically tied to the best, according to a one-sided Mann-Whitney test ($\alpha = 0.05$).

We kept three SAT, two AI planning, two MIP and two ML scenarios, resulting in a total of 9 scenarios. The results for irace and GGA++ are shown in Table 4.4.

irace performed better on our reduced spaces for 3 out of 9 scenarios, worse for 3 out of 9, and showed no significant performance differences for the remaining 3. We also notice that for CPLEX on RCW2, irace could not find a significantly better configuration than the default unless using our search space reduction techniques. Thus, exploiting the knowledge contained in the default parameter values through our reduction techniques benefited irace for these challenging scenarios. On the other hand, there are limited or no improvements for AutoWK and SpToRiss, which is unsurprising, considering that most of their reduced parameters are conditionally dependent on other parameter values (see Table 4.2). For SpToRiss on LABS, it is surprising that the Mann-Whitney test found evidence that the distribution for R1 is worse than that for R2, whereas this was not the case for the full configuration space. Further investigation revealed that the distribution of configuration quality obtained for R1 contains some significantly worse results than those for R2 and the full space, which causes the observed result. We note that irace implements a mechanism that resembles our range reduction techniques by focusing on sampling configurations around combinations of parameter values known to yield high performance. More precisely, when it generates a new configuration, it samples the values according to a Gaussian distribution centred around the best-known values and reduces the variance along the run to focus on promising parts of the configuration space. This focused sampling could explain

why the reduction did not significantly impact the performance of irace and is also the inspiration behind the work presented in section 4.5. GGA++ performed better on our reduced spaces for 7 out of 9 scenarios and worse for the remaining 2. There is no specific advantage to one or the other of the reductions; however, search space reduction consistently yields improvements.

Further investigation of our results.

A possible concern arising from the way we reduce the search space is that a strong reduction may exclude the global optimum from the search space. Unfortunately, there are no known methods for determining configurations that are guaranteed globally optimal for the kinds of scenarios we consider. However, by looking at the best-known configurations seen over all configurator runs for each scenario, we can at least develop some intuition. In Table 4.5, we show how many of the parameters have their best known value outside of the reduced ranges. We see that for Clasp, SpToRiss, LPG and AutoWK, the range reduced according to R1 contains the best known value for almost all the parameters, for Lingeling the range reduced according to R2 contains almost all the best known values, and for CPLEX the ranges reduced according to the two reductions contain almost all the best known values except for the CLS scenario. We could then expect to reach a better configuration with R1 for Clasp, SpToRiss, LPG and AutoWK and with R2 for Lingeling. However, the results presented in Tables 4.3 and 4.4 show that for AutoWK, Lingeling and LPG, the configuration results do not correspond to those expectations. Thus, for those three scenarios, excluding promising parts of the configuration space and possibly losing globally optimal configurations, we were still able to reach better configurations on average. This suggests that reducing the size of a given configuration space can make it easier for existing configuration procedures to find good local optima consistently.

In some cases, more particularly Lingeling on UNSAT and AutoWK on CAR and GC, the high number of parameter values that are outside of the reduced range might suggest that the default parameter value may not be a good focal point for the configuration process. However, for those three cases, one of the reductions actually contained the best-known configuration. Moreover, in preliminary experiments based on the idea of running a first short configuration run to find a better starting point than the default, we failed to observe better results than those obtained by reduction around the default.

Algorithm	Benchmark	R1	R2	Numerical Parameters
Clasp	CF	0	0	37
	LABS	0	0	
	UNSAT	2	7	
Lingeling	CF	0	0	185
	LABS	0	9	
	UNSAT	53	0	
SpToRiss	CF	1	0	52
	LABS	0	9	
	UNSAT	1	14	
LPG	Depots	0	6	19
	Satellite	0	9	
	Zenotravel	2	10	
CPLEX	CLS	7	7	23
	COR-LAT	0	2	
	RCW2	2	1	
	REG200	0	1	
AutoWK	CAR	0	42	226
	GC	0	40	
	WF	1	0	
	WQW	0	0	

Table 4.5: Number of parameters of the best known configuration (testing set) that take a value outside the reduced ranges (using R1 and R2, respectively)

Can we reach state-of-the-art performance by searching only around the default value?

Empirical results for well-known configuration scenarios for SAT, MIP, AI planning and autoML clearly indicate that using these reduction techniques, the state-of-the-art general-purpose configurators SMAC and GGA++ tend to find significantly better configurations within a given time budget. irace benefits from default-guided range reduction for scenarios with many numerical parameters that are not dependent on higher-level categorical design choices, such as CPLEX and LPG, while we did not observe benefits for scenarios with few numerical parameters, or numerical parameters that mainly depend on higher-level categorical choices, as for SpToRiss and AutoWK. This supports our hypothesis that configurators can benefit from the information contained in expert-determined default values for target algorithm parameters. Also, comparing our results on SparrowToRiss to those recently published for warm-

starting SMAC (Lindauer and Hutter, 2018), we find that the default-guided search space reduction gives similar, yet complementary benefits, in the sense that we obtain improvements where they do not, and vice versa. However, in contrast to warm-starting, our approach is applicable to a broader range of automatic configuration procedures.

The efficacy of our default-guided range reduction techniques suggests an interesting, largely unexplored direction for improving automated algorithm configurators, based on the idea of more strongly exploiting default configurations in the underlying search process, which we explore in the following section.

4.5 Probabilistic sampling

As shown previously, simply reducing the ranges of numerical parameters around the given default values can lead to significant performance improvements for SMAC (see Section 4.4 or Anastacio et al. (2019)). However, by pruning the search space, our previous approach completely excluded regions that, in specific cases, could be relevant to explore, and we did so in a configurator-agnostic, yet somewhat ad-hoc manner. Moreover, we saw that those reduction methods had a limited impact on the performance of irace since it already includes a mechanism to sample more configurations around its elite configurations. Building on those findings, our intuition is that sampling near good configurations will very likely lead to other good configurations, while sampling uniformly, as done in SMAC, will likely waste time on some underperforming parameter settings.

4.5.1 Extension with probabilistic sampling

We propose a principled approach for including the prior knowledge contained in the default value of parameters into the search process, by applying probabilistic sampling in the context of sequential model-based optimisation. As we will demonstrate, this can yield significant improvement for general-purpose AAC. To implement our approach, we extended SMAC with a simple mechanism for sampling new values according to a truncated normal distribution centred around the given default values, which replaces the uniform random sampling used in the original version of SMAC.

Algorithm 4.1 outlines at a high level our new configuration method, which we refer to as SMACPS. SMACPS differs from SMAC only in the sampling distribution

Algorithm 4.1 SMACPS

Require: Ω: configuration space, ω_d: the default configuration, Ω_{rand}, Ω_{prom} and Ω_{new}: sets of configurations.

1: $\mathcal{R}$: target algorithm runs performed. M: performance model.
2: $\omega_{\text{inc}} \leftarrow \omega_d$
3: $\mathcal{R} \leftarrow run\,(\omega_d)$
4: **while** Budget not exhausted **do**
5: $\quad M \leftarrow update\,(M, \mathcal{R})$
6: $\quad \Omega_{\text{prom}} \leftarrow uniform\_sample\_configurations\,(\Omega)$
7: $\quad \Omega_{\text{prom}} \leftarrow local\_search\,(M, \Omega_{\text{prom}})$
8: $\quad \Omega_{\text{rand}} \leftarrow normal\_sample\_configurations\,(\Omega)$
9: $\quad \Omega_{\text{new}} \leftarrow interleave\,(\Omega_{\text{rand}}, \Omega_{\text{prom}})$
10: $\quad inc \leftarrow intensify\,(\Omega_{\text{new}}, \omega_{\text{inc}})$
11: **end while**
12: **Return** ω_{inc}

used in line 7; further details of SMAC can be found in the original paper[2] (Hutter et al., 2011b). New configurations are sampled in two places: as starting points for the local search process (line 5) and for non-model-based diversification (line 7). The two sets of configurations thus obtained are then interleaved and raced against the current incumbent (line 9). We change the sampling distribution used for non-model-based diversification (line 7) – note that in Anastacio and Hoos (2020a) it was changed at both places. Each parameter is sampled independently from a distribution that depends on the parameter type and domain. For each numerical parameter p_n, we first normalise the range to $[0, 1]$ (if p_n is specified as "log scale", we first apply a log base 10 transformation). We then sample a value from a truncated normal distribution with mean equal to the default value of p_n and variance 0.05; this value was chosen based on preliminary experiments on configuration scenarios different from the ones used in our evaluation, namely the SAT solvers *Spear* and *Satenstein* on benchmark *SWGCP*; additional details can be found on AClib. For a categorical parameter p_c with k values, we sample the default value with probability 0.5, and each other value with probability $0.5/(k-1)$.

In cases where the default configuration is far away from the most promising areas of a given configuration space, this approach might be counterproductive. We note, however, that model-based local search, starting from uniformly sampled configurations, has the potential to counterbalance this effect.

[2]Recent versions of SMAC behave slightly differently, picking challengers from C_{prom} or C_{rand} with given probabilities instead of alternating.

Our probabilistic sampling approach is inspired by estimation of distribution algorithms, which are known to provide an effective way for leveraging prior knowledge when solving complex optimisation problems (Hauschild and Pelikan, 2011). It is also conceptually related to the approach taken by irace, which, unlike SMAC, does not use an empirical performance model to map parameter configurations to performance values, but instead uses the best-known configurations as a basis for estimating where promising parameters may be located. Indeed, irace employs an intensification mechanism that involves sampling new values for numerical parameters from truncated normal distributions, in conjunction with a racing mechanism for updating the incumbent configuration and the locations of the sampling distributions. We note that the probabilistic sampling process used in SMAC is simpler, as it maintains fixed sampling distributions throughout the configuration process. We decided on this design to evaluate the extent to which a simple probabilistic sampling mechanism solely focused on exploiting information from expert-chosen default values would already enable improvements over state-of-the-art algorithm configurators, such as SMAC and irace.

4.5.2 Results

In this section, we compare the algorithm described in Section 4.5.1 against the state-of-the-art configurations SMAC and irace following the protocol described in Section 4.2.

Comparison based on median PAR10 scores

We compare the performance obtained for the configuration scenarios we studied, following the protocol described in Section 4.2, which produces statistics on how state-of-the-art configurators are commonly used in practice. Table 4.6 shows the median PAR10 values we obtained; the missing results for CPLEX on RCW2 and REG200 are since irace refused to start more than half of the 24 runs, since it considered the configuration budget to be insufficient.

Comparing the results for SMAC and irace, we note that in most cases, SMAC achieves better performance than irace. SMAC achieved a statistically significantly lower median PAR10 score in 10 out of the 16 configuration scenarios we studied, while irace outperformed SMAC in the remaining 6 scenarios. This indicates complementary strengths of our two baseline configurators. While SMAC has an edge on most of the scenarios, there is at least one case where irace finds substantially better

Table 4.6: Results for SMAC, SMACPS and irace; median PAR10 (in CPU sec); best results are underlined, while boldface indicates results that are statistically tied to the best, according to a one-sided Mann-Whitney test ($\alpha = 0.05$). Right columns highlight the result of the pairwise comparison between SMACPS and the two baselines; ✓ if it is better, ✗ if not; parentheses indicate that the difference is not statistically significant.

Solver	Benchmark	Default	SMAC	irace	SMACPS	≻ SMAC	≻ irace
Clasp	CF	193.87	193.00	**174.00**	**173.61**	✓	(✓)
	LABS	745.74	**837.93**	847.10	**837.61**	(✓)	✓
	UNSAT	0.885	**0.362**	**0.359**	**0.359**	(✓)	(✓)
Lingeling	CF	327.00	**261.06**	328.214	300.40	✗	✓
	LABS	873.67	866.99	959.35	**863.73**	✓	✓
	UNSAT	2.41	**1.59**	2.36	1.65	✗	✓
SpToRiss	CF	472.78	**226.51**	236.61	253.72	✗	✗
	LABS	911.25	857.67	846.40	**805.46**	✓	✓
	UNSAT	222.26	1.56	1.56	**1.55**	✓	✓
LPG	Depots	34.68	1.14	**1.08**	1.25	(✗)	✗
	Satellite	22.40	5.43	8.04	**5.19**	✓	✓
	Zenotravel	29.64	2.61	2.93	**2.56**	✓	✓
CPLEX	CLS	4.06	3.36	4.06	**2.95**	✓	✓
	COR-LAT	24.81	21.84	**10.04**	21.26	✓	✗
	RCW2	82.51	**71.98**	–	78.10	✗	✓
	REG200	13.08	5.56	–	**5.09**	✓	✓

configurations (CPLEX on COR-LAT) – looking back at Chapter 3 allows us to see that the performance of SMAC3 on this scenario is poor.

Comparing SMACPS against SMAC, which is the strongest of our two baselines, and served as the starting point for our new configuration procedure, we notice that for 11 of the 16 scenarios, SMACPS reaches a lower median PAR10 score. In all but two of those cases (Clasp on LABS and UNSAT), the performance differences are statistically significant.

Finally, compared to irace, SMACPS achieves better performance on 13 of our 16 scenarios. In all but two cases (Clasp on CF and UNSAT), the differences are statistically significant. SpToRiss on LABS shows a case where SMAC could not achieve better result than irace, while SMACPS outperforms irace.

Overall, these results indicate clearly that SMACPS represents a significant improvement over both baselines.

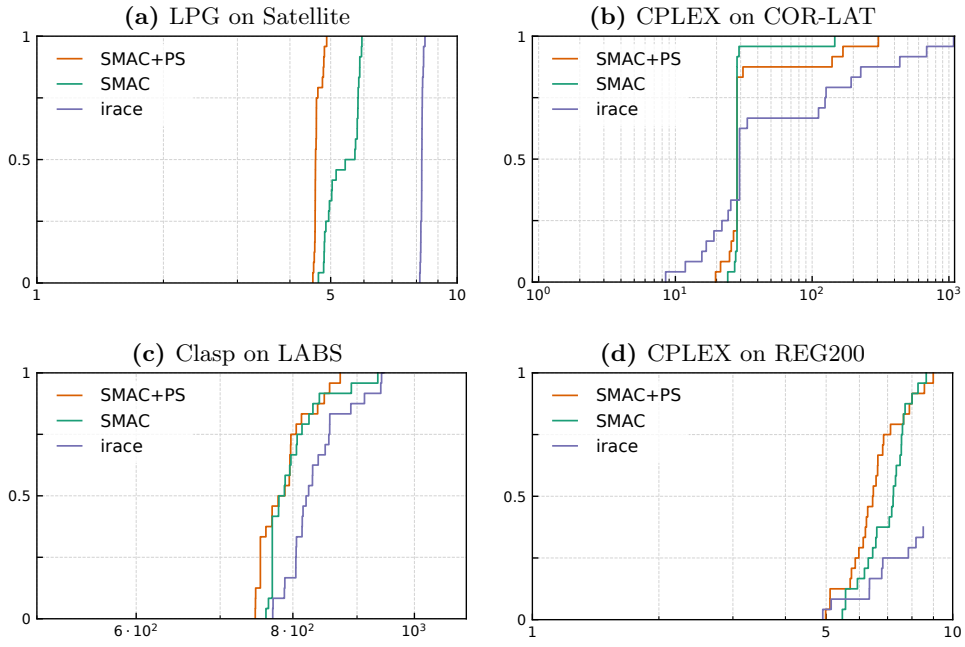


Figure 4.1: Cumulative distribution functions for PAR10 scores (in CPU seconds, x-axis) over independent configurator runs on the testing set.

Distributions of PAR10 scores over multiple configurator runs

To examine the performance of SMAC, irace and SMACPS in more detail, we studied the empirical cumulative distribution functions over individual configurator runs (without applying the standard protocol) – see Figure 4.1. As we minimise PAR10 scores, better performance is indicated by CDFs closer to the top left corner of the plots. We present results for four scenarios, each yielding a qualitatively distinct outcome in our comparison.

Figure 4.1a, LPG on Satellite, is a case in which the difference between the CDFs for SMAC and SMACPS is particularly pronounced. In this scenario, SMACPS clearly dominates the two other configurators.

Figure 4.1b, CPLEX on COR-LAT, is a case in which irace performs better than the two other configurators. SMAC and SMACPS give rise to similar performance distributions. irace has an edge, reinforced by the standard protocol, which leverages the left tail of the performance distributions.

Figure 4.1c, Clasp on LABS, a scenario on which SMAC and SMACPS are statistically tied, looks qualitatively similar, except that the difference in the left tail between SMAC and SMACPS is too small to be reliably exploitable using the standard protocol. irace, on the other side, is probabilistically dominated on this scenario.

Figure 4.1d shows a scenarios on which irace terminated prematurely (as described in Section 4.2). However, examining the partial CDFs for the successfully completed configurator runs, there is no reason to expect that irace would have performed significantly better than SMAC and SMACPS.

What is the impact of focusing the search around the default value?

We proposed a simple yet effective method to probabilistically bias the sequential model-based algorithm configurator (SMAC) (Hutter et al., 2014a) towards a given default configuration. To do so, we replaced its uniform random sampling mechanism with a probabilistic sampling approach for numerical parameters.

We evaluated the resulting procedure, dubbed SMACPS, against two widely used and freely available state-of-the-art general-purpose algorithm configurators, SMAC and irace (López-Ibáñez et al., 2016). For this comparison, we used 16 running time optimisation scenarios from AClib, a widely used library of benchmarks for AAC (Hutter et al., 2014a).

We found that SMACPS performs better than SMAC on 11 of those 16 scenarios, and better than irace on 12 of them, and thus represents a significant improvement over the state of the art in AAC for running time minimisation. Whether similar results can be obtained for different performance metrics is an open question.

4.6 Conclusion

This chapter explored approaches to leverage the expert knowledge contained in the default value. We believe that for most state-of-the-art algorithms for challenging AI problems, default parameter values are chosen with care, in many cases not only based on limited experiments, but also on deep insights into the heuristics controlled by the parameters.

Answering RQ4 in Section 4.3, we demonstrated that each configurator relies on the default parameter values in different ways. As a proof of concept and to answer RQ 5.1, Section 4.4 introduces a naïve approach to prune the configuration space around the default value, thereby searching only among values near it. We obtained significant

improvement in configurator performance. However, our reduction techniques depend on the quality of the given default values and hold the risk of excluding promising areas of the search space if the default value was poorly chosen. It is also expected to have a greater impact when targeting numerical parameters with wide ranges. This is the case with many numerical parameters, because, in principle, extreme values produce valid behaviour of the algorithm, and the effort to manually restrict the range to good values can be considerable; thus, it is often avoided by algorithm designers. We also expect that reducing parameter ranges too aggressively will ultimately lead to degraded performance.

Section 4.5 introduced a more subtle incorporation of the default in the search strategy to answer RQ 5.2. By using a truncated distribution centred around the default value, we bias the search towards the promising area around the default while maintaining a probability of deviating from it if the default was poorly chosen. We compared our methods on a set of running time optimisation scenarios. We demonstrated that, in the majority of scenarios, the introduced methods outperform configurators that do not utilise the knowledge held in the default. We thus argue that the often-overlooked insights of experts can and should be included in configurators, as they hold the potential to guide the search. Our results are consistent with recent work showing that the configuration landscapes (*i.e.*, the functions relating parameter values to target algorithm performance) are far more benign than one might have expected (Pushak and Hoos, 2018), which suggests that sampling around known good parameter values provides an efficient way towards finding new good values, an assumption also leveraged by irace.

Our work also opens an interesting path towards richer mechanisms for allowing algorithm designers to express prior knowledge about parameter values. Note that, following the publication of results presented in this chapter, subsequent work explored this path further by allowing users (or algorithm designers) to define detailed priors on each parameter (see *e.g.* Souza et al., 2021; Hvarfner et al., 2022).

5

Sampling instances to compare the performance of algorithms

Empirical performance evaluation plays a critical role in algorithm configuration and performance optimisation, be it automated or manual. Whilst for a configurator we would need to compare two configurations of the same algorithm, this chapter[1] focus on approaches to compare two algorithms against each other. This problem arises in competitions and scientific publications aimed at improving the state of the art in solving many automated reasoning problems, such as Boolean satisfiability (SAT), constraint satisfaction problem (CSP) and Bayesian network structure learning (BNSL). We explore the intuition that the decision to keep or discard an algorithm can be taken earlier by carefully selecting on which instances to evaluate its performance. By performing runs on carefully chosen problem instances, we minimise the computational cost of running algorithms, whilst probabilistic tests allow us to control the desired statistical significance of observed performance differences. We describe a set of methods for this purpose and evaluate their efficacy on diverse datasets from SAT, CSP and BNSL. On all these datasets, most of our approaches were able to select the correct algorithm with approximately 95% accuracy, while using less than one-third

[1] Parts of this chapter have been published as Matricon et al. (2021)

of the CPU time required for a comparison on the full instance set. The best methods achieve this level of accuracy within less than 15% of the CPU time needed for a complete comparison. Intuitively, we expect the behaviour of two algorithms to be more distinct than that of two configurations of a single algorithm. The transfer of the evaluated methods to automated algorithm configuration (AAC) will be further explored in the subsequent chapter.

5.1 Introduction

From the evaluation of early algorithms against the human ability to solve given instances by hand (Davis and Putnam, 1960) to extensive competitions requiring CPU years to determine a winner (Heule et al., 2019; Pulina and Seidl, 2019; Sutcliffe, 2020), the amount of computational resources needed to assess empirically whether an algorithm exceeds state-of-the-art performance is growing along with the ability of state-of-the-art solvers to tackle larger instances.

Here, we address this issue by focusing on the instance space and on techniques for identifying instances that help discriminate between the compared algorithms. We argue that carefully selecting instances and avoiding long evaluations that provide only a limited amount of information allows us to decide earlier when to stop running a less promising algorithm. Despite similarities with existing problems, such as active learning and algorithm selection (AS), this problem has a different objective, and it is thus not possible to apply existing methods directly.

5.1.1 Background

To the best of our knowledge, the problem we address has not been previously studied in the literature. However, similar questions appear in other settings. Some early SAT Competitions (see *e.g.* the 2002 competition Simon et al., 2005) have been organised in two stages: first, they ran all the solvers on a subset of hand-picked instances to extract the top performers, which were then run on all problem instances. The subset was selected by experts, requiring extensive knowledge and understanding of the problem instances at hand, which is not readily available to an automated system.

To decide on which instance the algorithm should run, we assign a score to each instance (see Section 5.3) based on existing approaches from the literature. In the context of instance generation for CP problems, Gent et al. (2014) proposed a method for defining the discrimination power of an instance, enabling the generation of problem instances for model selection based on samples of running times. This method does not address our aim to reduce running time, but it could lead to selecting relevant instances. We included this method in our comparison, with minor adjustments to account for our objective of minimising running time. Note that after the publication of the work described in the current chapter, Bossek and Wagner (2021) developed an explicit-ranking method as a fitness function for evolutionary algorithms, in order to generate instances that follow a given ranking. Their ranking maximises the similarity between the ranking of the algorithms and the difference in their running times on

this instance.

Once we know which instances should be solved, the next decision is whether to stop or continue the comparison. This question also arises in other situations. In AAC, comparing the performance of two configurations is a key element (as explained in Section 2.2.2). SMAC and ROAR (Hutter et al., 2011b), as well as irace (López-Ibáñez et al., 2016), pick uniformly at random the instances on which they run it, without considering prior knowledge they gathered. irace is based on earlier work from Maron and Moore (1997), which aimed at comparing many machine learning models on a subset of test points to estimate their accuracy with a certain statistical confidence. In this line of work, irace requires evidence in the form of a statistical test to decide when to stop running a less promising configuration. SMAC and ROAR, on the other hand, compare the raw performance metric. We included the statistical test from irace in our experiments.

Our problem is also related to the per-instance algorithm selection (AS) problem (Kerschke et al., 2019, see *e.g.*) in which one tries to predict on which algorithm a specific instance should be run to be solved with the best possible performance. However, there are key differences that prevent us from using directly the methods developed for AS; typically, their use requires prior knowledge in the form of instance features, which we do not always assume to have, and the running time of the algorithms on other instances, which we do not have available for the new algorithm. Additionally, our primary goal is to reduce the time required to determine which one is the best.

Finally, there is a significant link with problems tackled by active learning methods (Sun and Wang, 2010), particularly the pool-based selective sampling problem, which seeks to choose an instance from a set on which the model should be trained next. The idea is that a relevant instance should have a high impact on the model, (*e.g.* increasing its accuracy or reducing its variance). This is closely related to our problem, but differs in that the chosen instance should also lead to low running times, which is not a common objective in active learning. Those methods are aimed at a machine learning model, and the choice of an instance is based on the impact it may have on the model, (*e.g.* reducing its variance or expected error).

In this chapter, we assume the accessibility to the empirical performance data of the studied algorithms. We focus on situations in which we cannot or do not want to learn a performance model for comparing the two algorithms, *e.g.* in the context of developing a new solver or running a competition. Chapter 6 will delve further into model-based approaches when incorporating these methods into a configurator.

5.1.2 Research question

RQ6 *How can we smartly select on which instances to run our evaluation to lower the time spent evaluating bad algorithms?*

We introduce the per-set efficient algorithm selection problem (PSEAS): Given two algorithms, an incumbent A_{inc} and a challenger A_{ch}, and a set of problem instances $\mathcal{I}$, how can we minimise the computational resources (here: CPU time) required to determine, at a required level of confidence, whether A_{ch} performs better than A_{inc} on $\mathcal{I}$? In the following, we describe five methods for selecting on which instances to run the competing algorithms, and two methods for deciding when to stop the evaluation. We compare the ten resulting approaches on four benchmarks for classic computational problems: SAT, CSP and BNSL. On these datasets, our approaches can determine the better-performing algorithm with up to 98% accuracy, while using less than a third of the CPU time required for a full comparison, and the best methods achieve this level of accuracy within less than 15% of the CPU time for an exhaustive comparison.

5.2 Model-free instance selection

To answer our research question RQ6, we place ourselves in the context of comparing the running time performance of two algorithms and define the problem at hand.

Definition of the per-set efficient algorithm selection problem (PSEAS)

We let $\mathcal{I}$ denote the set of instances, $T_{\text{cut}} \in \mathbb{R}^+$ the cutoff threshold, m the performance metric that evaluates an algorithm on an instance, and c the cost function that evaluates the cost of running an algorithm on an instance. We consider two algorithms: the incumbent A_{inc} and the challenger A_{ch}, and assume that the cost $c(A_{\text{inc}}, I)$ and performance $\mathcal{M}(A_{\text{inc}}, I)$ of running A_{inc} on an instance I is known for all instances, whereas these quantities are unknown on all instances for A_{ch}. This assumption is consistent with the fact that A_{inc} represents the state of the art, hence can be assumed to have been evaluated on many problems. The problem is to determine which of the two algorithms performs best according to $\sum_{I \in \mathcal{I}} \mathcal{M}(A_{\text{ch}}, I)$ while running A_{ch} only on a subset $\mathcal{I}_{\text{run}} \subset \mathcal{I}$ that minimises the cost $\sum_{I \in \mathcal{I}_{\text{run}}} c(A_{\text{ch}}, I)$.

Here, we pose $\mathcal{A}$ a set of algorithms, including A_{inc}, regarding which we have prior knowledge in the form of their costs and performances on (at least part of) the instances from $\mathcal{I}$. Unless stated otherwise, we write I for an instance in $\mathcal{I}$ and A for an algorithm in $\mathcal{A}$. For simplicity, we consider the algorithms to be deterministic; hence,

for an algorithm $A \in \mathcal{A} \bigcup \{A_{\text{ch}}\}$, we define the running time as $rt(A, I) \in [0, T_{\text{cut}}]$ for an instance I. We define $\mathcal{M}(A, I) = c(A, I) = rt(A, I)$: the running time of an algorithm is considered as a proxy for the energy cost of running it.

The performance of an algorithm defined as the sum of the running times over all instances, where timed out runs are penalised, following the Penalised Average Running time (PAR) typically used in configuration scenarios.

Each methods we describe relies on a different amount and type of background knowledge about the set of instances. This knowledge is similar to the one used in the AS problem and thus readily accessible. We consider the following ways of specifying the background knowledge:

- *Sample-based*: for each instance I and algorithm A we have the running time $rt(A, I)$ of A on I.
- *Feature-based*: for each instance I we have a feature vector f_I.
- *Statistics-based*: for each instance I, we have a prior in the form of a probability distribution δ_I over $[0, T_{\text{cut}}]$, expressing that $\delta_I(t)$ is the probability that A_{ch} solves the instance I at time t. In practice, we obtain this prior by fitting a distribution to the running times of A.

Note that above, $A \neq A_{\text{ch}}$, *i.e.* the background knowledge does not contain information about the challenger algorithm. The implicit assumption is that running times of algorithms from $\mathcal{A}$ and feature vectors of the instances are both predictive of the running times of A_{ch}: for instance, if all algorithms in $\mathcal{A}$ solve an instance I very quickly, then so should A_{ch}. In other words, A_{ch} is expected to have similar behaviour as the algorithms in $\mathcal{A}$. Similarly, if two feature vectors f_I and $f_{I'}$ are close for two instances I, I', then their running times should be close. These assumptions are prominently made in running-time prediction, such as in Hutter et al. (2014b) and per-instance AAC (see *e.g.* Kerschke et al., 2019). In other words, the key insights and mathematical formalisation of this section are based on the background knowledge described above to evaluate the expected performance of A_{ch}.

5.3 Methods

Our goal is to define a *strategy* that sequentially chooses the instances on which to run our challenger A_{ch} and decides if the evidence so far gives sufficient confidence to stop the comparative evaluation. Algorithm 5.1 formalises this iterative process using a

Algorithm 5.1 PSEAS solving strategy

Input A_{inc}: the incumbent algorithm, A_{ch}: the challenger algorithm, C_{thres}: the target confidence threshold, $\mathcal{I}$: the training instances.
Output A_{inc} or A_{ch}: the best performing algorithm.

1: set $\mathcal{I}_{\text{torun}} = \mathcal{I}$ and $C_{\text{current}} = 0$
2: compute score(I) for all $I \in \mathcal{I}$
3: **while** C_{current} ¡ C_{thres} **do**
4: pick $I^* \in \text{argmax}_{I \in \mathcal{I}_{\text{torun}}}\ score(I)$ and remove I^* from $\mathcal{I}_{\text{torun}}$
5: evaluate $rt(A_{\text{ch}}, I^*)$
6: update C_{current}
7: update score(I) for $I \in \mathcal{I}_{\text{torun}}$
8: **end while**
9: **return** best performing algorithm from (A_{inc}, A_{ch})

score-based approach: each instance is assigned a score, which may be updated along the comparison to – intuitively – reflect the interest in running this instance. C_{current} is the current confidence and depends on A_{inc}, $\mathcal{I}_{\text{torun}}$ are the instances on which A_{ch} has not been run There are two main components in this algorithm: one for score computation (lines 2, 4, 7; see Section 5.3.2) and one for confidence (lines 1, 3, 6; see Section 5.3.1). The score enables choosing the best instance to run, whereas the confidence tells when to stop the comparison. These two components will be explained in more detail later.

Strategy evaluation

We consider two metrics for evaluating strategies: the *cost* and the *accuracy*.

We measure the computational effort (which we want to minimise) as the ratio of the total running time for instances in $\mathcal{I}_{\text{run}}$, the set of instances on which A_{ch} has been run by the strategy, over the total running time over all instances; this results in a number between 0 and 1. Note that the goal is not to minimise the *number* of instances A_{ch} is run on, but rather the total running time of A_{ch} on these instances. To evaluate our strategy, we determine this cost over many ordered pairs of algorithms $(A_{\text{inc}}, A_{\text{ch}})$ and consider the median. Formally, for a set of ordered pairs $\mathcal{P}$:

$$cost(\mathcal{P}) = median\left[\left(\frac{\sum_{I \in \mathcal{I} \setminus \mathcal{I}_{\text{torun}}} rt(A_{\text{ch}}, I)}{\sum_{I \in \mathcal{I}} rt(A_{\text{ch}}, I)}\right)_{(A_{\text{inc}}, A_{\text{ch}}) \in \mathcal{P}}\right],$$

where $\mathcal{I}_{\text{torun}}$ are the instances that have not been run by the strategy during its execution, as defined in Algorithm 5.1. We note that $cost(\mathcal{P})$ only depends on A_{ch}, since A_{inc} is assumed to have already been run.

We measure the *accuracy* of a strategy (which we want to maximise), as the ratio of correct guesses made by the strategy when deciding which algorithm from an ordered pair of algorithms $(A_{\text{inc}}, A_{\text{ch}})$ performs best. Formally, for a set of ordered pairs $\mathcal{P}$:

$$accuracy(\mathcal{P}) = \frac{\sum_{(A_{\text{inc}}, A_{\text{ch}}) \in \mathcal{P}} 1_{\{\hat{A}_{\text{best}} = A_{\text{best}}\}}}{|\mathcal{P}|},$$

where A_{best} is the true best performing algorithm in $(A_{\text{ch}}, A_{\text{inc}})$, and $\hat{A}_{\text{best}}$ is the best algorithm given by the strategy. Our definition of *accuracy* uses the mean, since the median over the results of the indicator function would produce too limited a range of results to be useful for comparing strategies.

We note that the choice made in line 4 of Algorithm 5.1 aims at optimising two goals. The *instance selection component* tries to minimise the computational effort by deciding on which instances to run A_{ch}, based on a score given to each instance. The *discrimination component* decides, based on the data gathered so far, whether the expected accuracy, or confidence, is high enough to stop the comparison.

5.3.1 The discrimination component

The discrimination component aims at estimating the accuracy of the current decision of which among A_{inc} and A_{ch} performs best. However, this measure can never be accessed, since the complete data is not available. Hence, we look at the expected accuracy, or *confidence*, as a proxy for accuracy. The confidence is computed differently for each discrimination method and is thus not comparable among them. It provides a measure of the current state of the strategy. When the confidence reaches a threshold C_{thres} (line 3 of Algorithm 5.1), the strategy stops and returns the algorithm evaluated as being the best.

Baseline: Subset method

As a baseline, we use a fixed-size subset of instances: we fix $\gamma \in [0; 1]$ and decide to stop when A_{ch} has been run on $\lfloor \gamma |\mathcal{I}| \rfloor$ instances. Note that this does not ensure a ratio of γ for the total running time, as the total running time of A_{ch} over all instances is not available and therefore cannot be used for discrimination. The confidence for this method is 0 until all instances of the subset have been executed; then the confidence is 1.

Wilcoxon test

There is a large body of literature on statistical tests, and many of them can be used in the context of racing algorithms (Birattari et al., 2002). For instance, the F-Race (Birattari, 2009) algorithm uses a Friedman two-way analysis of variance by ranks. However, this test concerns a family of candidates, while here, we are interested in an ordered pair of algorithms. When only two configurations remain, the F-Race algorithm switches to a *Wilcoxon matched-pairs signed-ranks test* (Conover, 1998), because it is more powerful and data-efficient than the Friedman test in that scenario (Siegel and Castellan Jr, 1988).

The test we want to apply should satisfy the following requirements: it should be nonparametric, and it should apply to paired data. Such a test would *not need any background knowledge*. We chose the Wilcoxon test because it satisfies our requirement while exploiting other properties of our data: data is measured on an interval scale, the differences (between running times) are symmetric, and the magnitudes of the differences between our paired data are exploited. This test assumes that running times are independent and the two samples are mutually independent. While it is not truly the case, we find that assuming independence is a good first approximation. This test is only based on observed data; it does not take into account the remaining instances. Through hypothesis testing, we can find out when there is enough evidence to stop, at which point the best algorithm is the one with the lowest mean running time. In this case, our confidence threshold C_{thres} is compared to the p-value of the alternative two-sided hypothesis. Let us note that other statistical tests, such as the Mann-Whitney U test, the permutation test, the Kolmogorov-Smirnov test, or the paired t-test, do not satisfy our assumptions.

The distribution-based discrimination method

This method requires statistics-based background knowledge. Let us consider the following random variable computing the difference in performances:

$$\Delta_{\text{tot}} = \underbrace{\sum_{J \in \mathcal{I}_{\text{run}}} rt(A_{\text{ch}}, J) - rt(A_{\text{inc}}, J)}_{\text{constant}} + \sum_{I \in \mathcal{I}_{\text{torun}}} \underbrace{rt(A_{\text{ch}}, I)}_{\text{random variable}} - \underbrace{rt(A_{\text{inc}}, I)}_{\text{constant}}.$$

We are interested in determining the sign of Δ_{tot}, meaning which of the two algorithms performs best. For a fixed confidence threshold $C_{\text{thres}} = 1 - \varepsilon$, we estimate $\mathbb{P}(\Delta_{\text{tot}} > 0)$ and stop if:

- $\mathbb{P}(\Delta_{\text{tot}} > 0) \geq 1 - \varepsilon$, in which case A_{ch} performs worse than A_{inc},
- or $\mathbb{P}(\Delta_{\text{tot}} > 0) \leq \varepsilon$, meaning $\mathbb{P}(\Delta_{\text{tot}} \leq 0) \geq 1 - \varepsilon$, *i.e.* A_{ch} performs better than A_{inc}.

The confidence is $\mathbb{P}(\Delta_{\text{tot}} > 0)$ for the former case and $1 - \mathbb{P}(\Delta_{\text{tot}} > 0)$ for the latter. Looking at the definition of the random variable Δ_{tot}, its probability law can be described using translations and convolutions of the distributions $(\delta_I)_{I \in \mathcal{I}_{\text{torun}}}$. In practice, many natural classes of distributions (such as Gaussian and Cauchy distributions) are closed under translations and convolutions, so $\mathbb{P}(\Delta_{\text{tot}} > 0)$ can be effectively computed or approximated.

Because running times are positive and algorithms are stopped when they reach the cutoff time T_{cut}, the running times are bounded. A distribution matching this behaviour would be a truncated distribution. Still, most are not closed under convolution, which we have stated above as a necessary property, so they cannot be used directly. Nevertheless, the sum of the bounds on individual running times can be used as bounds for Δ_{tot}, which we can model as a truncated distribution. For heavy-tailed distributions, such as the Cauchy distribution, the confidence is higher with a truncated distribution than without, as impossible cases are not taken into account, enabling to stop earlier.

5.3.2 The instance selection component

With the aim of minimising the overall computational effort, our algorithm iteratively chooses the most relevant instance, according to a score (lines 2 and 7 in Algorithm 5.1). Instances with the highest score are expected to be the most relevant ones (*i.e.* intuitively giving the most information at the lowest cost).

Baseline: Uniform random sampling

As a baseline, we use a random sampling approach. In our algorithm, this corresponds to giving the same score to all instances, and thus to a uniform random choice at each iteration.

The discrimination-based selection method

This sample-based method is inspired by Gent et al. (2014); they developed it as a method to find optimal parameters for instances in an instance selection approach for

automated constraint model selection. The intuition is to choose the most discriminating instances first. Let $\rho > 1$ be a constant; an algorithm A is ρ-*dominated* on an instance I if there exists another algorithm A' such that $rt(A', I) \leq \rho \cdot rt(A, I)$. The *discrimination quality* of an instance I, denoted $G(I)$, is the fraction of algorithms that are ρ-dominated on this instance. Using this measure as-is would not take into account our goal of minimising the running time, so we divide the discrimination quality by the mean running time of the instance. The obtained score only needs to be computed once:

$$score(I) = \frac{G(I)}{mean[(rt(A, I))_{A \in \mathcal{A}}]}.$$

The variance-based selection method

This statistics-based method uses the intuition that the most interesting instances are the ones most likely to have very different running times for A_{inc} and A_{ch}. For each instance I we have a prior δ_I, which is the running time distribution of A_{ch}. We want to choose an instance with the highest variance $\text{argmax}_{I \in \mathcal{I}_{\text{torun}}} \mathbb{V}(\delta_I)$. As for the discrimination-based selection method, since we want to minimise the running time, we divide by the mean running time of the instance. The obtained score only needs to be computed once:

$$score(I) = \frac{\mathbb{V}(\delta_I)}{\mathbb{E}[\delta_I]}.$$

The information-based selection method

This statistics-based method is based on a similar intuition to the previous method. We are interested in instances from which we gain as much information as possible; the variance is only one (natural) indicator of this information. Following this approach, we can also estimate the information gained from a specific instance. The concrete information we are after is given by the discrete random variable stating that A_{ch} is better than A_{inc}, formally defined as $sign(\Delta_{\text{tot}})$. Let Δ_I be the random variable defined as $\Delta_I := rt(A_{\text{ch}}, I) - rt(A_{\text{inc}}, I)$, such that

$$\Delta_{\text{tot}} = \sum_{K \in \mathcal{I}} \Delta_K = \sum_{J \in \mathcal{I}_{\text{run}}} \Delta_J + \sum_{I \in \mathcal{I}_{\text{torun}}} \Delta_I.$$

We compute the expected information brought by $I \in \mathcal{I}_{\text{torun}}$; hence the information gain is defined as follows for I:

$$IG\left[\Delta_I\right] := \mathbb{E}_{rt(A_{\text{ch}},I)\sim\delta_I}[D_{\text{KL}}(Q_{+I} \mid\mid Q)] \text{ with}$$
$$Q = P\left(sign(\Delta_{\text{tot}})|\Delta_J = r_J, J \in \mathcal{I}_{\text{run}}\right)$$
$$Q_{+I} = P\left(sign(\Delta_{\text{tot}})|\Delta_J = r_J, J \in \mathcal{I}_{\text{run}}, \Delta_I = r_I\right),$$

where D_{KL} is the Kullback–Leibler divergence, δ_I is the distribution of running time on instance I, and r_J are the realisations of the Δ_J since the difference for the instances in $\mathcal{I}_{\text{run}}$ is known.

As for the previous method, to balance information and running time, we divide by the expected running time, and therefore use the following score function, which we update at each iteration:

$$score(I) = \frac{IG\left[\Delta_I\right]}{\mathbb{E}[\delta_I]}.$$

The feature-based selection method

In this feature-based and statistics-based method, we assume that for each instance I, we have a feature vector $f_I \in \mathbb{R}^n$ in some dimension n. The implicit assumption is that features are predictive of the running times of A_{ch}. We proceed in two steps:

- Constructing a distance metric $d : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}_{\geq 0}$, such that if $d(f, f')$ is small, then two instances with features f and f' have similar running times.
- Assigning a score to each instance $I \in \mathcal{I}_{\text{torun}}$.

Constructing a distance metric. The objective is to define a distance predictive of the running times; to this end, we introduce a weight for instance features, represented by a weight vector $\theta \in \mathbb{R}^n$. Let us consider distances of the form:

$$d_\theta(f_I, f_J) = \sqrt{\sum_{x=1}^{n} \left(\theta(x) \cdot (f_I(x) - f_J(x))\right)^2}.$$

Intuitively, for a feature x, the parameter $\theta(x)$ determines the importance of x in predicting the running times. Let us write m_I for the median time over all algorithms on instance I. We optimise over θ by considering:

$$\theta^* \in \operatorname*{argmin}_{\theta\in\mathbb{R}^n} \sum_{I,J\in\mathcal{I}} \left(d_\theta(f_I, f_J)^2 - |m_I - m_J|\right)^2;$$

i.e., d_{θ^*} is the best distance in this family for predicting differences in median running

time. The parameter vector θ^* is the solution of a non-negative ordinary least square optimisation problem and can therefore be computed efficiently (Lawson and Hanson, 1995). Note that the space complexity is quadratic in the number of instances and linear in the feature space dimension.

Assigning a score. Given a distance metric d, we now define a score for a given problem instance. Here, it is convenient to minimise rather than maximise the following quantity with respect to d:

$$S(I) = \sum_{J \notin \mathcal{I}_{\text{torun}}} \frac{rt(A_{\text{ch}}, J)}{d(f_I, f_J)} + \sum_{J \in \mathcal{I}_{\text{torun}}} \frac{\mathbb{E}[\delta_J]}{d(f_I, f_J)} \quad \text{and} \quad score(I) = \frac{1}{S(I)}.$$

The score is updated at each iteration. In all previous methods, the score of an instance I only uses the information on I; the strength of this method is to gather and weight information over all instances. Indeed, the score of I is a weighted average over all running time predictions, meaning $\mathbb{E}[\delta_J]$ when $J \in \mathcal{I}_{\text{torun}}$ and $rt(A_{\text{ch}}, J)$ otherwise, and the prediction for J contributes to the prediction of I up to the multiplicative factor $\frac{1}{d(f_I, f_J)}$.

5.4 Setup of experiments

To empirically evaluate our approaches, we implemented it with Python, using Numpy and Scipy, and ran them on all ordered pairs of algorithms from well-known benchmark scenarios.

5.4.1 Datasets

We use ASlib (Bischl et al., 2016), a benchmark library for AS that contains datasets from competitions for various challenging problems, including Boolean satisfiability and constraint programming. It provides very relevant data on which our strategies can be tested, because such problems are the typical use-case scenario that we envisioned.

From ASlib, we use three datasets: the CSP MiniZinc 2016 dataset, which comprises performance data from the 2016 MiniZinc Challenge ('Free Search' Category) (Lindauer et al., 2017; Stuckey et al., 2014); the BNSL 2016 dataset (Malone et al., 2018) from Bayesian Network structure learning; the SAT18 dataset, which consists of performance data from the EXP track of the 2018 SAT Competition (Heule et al., 2019); and, to account for more recent advances in SAT, we created the SAT20 dataset from the results of the main track of the 2020 SAT Competition (Balyo et al., 2020). Those

Table 5.1: Characteristics of the used datasets

Dataset	CSP MiniZinc	BNSL	SAT 18	SAT 20
Algorithms	20	8	37	67
Instances	100	1178	353	400
Features	95	86	54	108
Mean difficulty	59.74	9.363	2458	78.88
Median difficulty	3.28	1.15	9.65	5.51
Top-3 mean difficulty	24.7	77.5	47.7	49.9

datasets were chosen to cover a broad range of prominent problems and instance sets.

For our feature-based approaches, we decided to replace missing features with the mean value, as done by Hutter et al. (2014b). Hence, no information can be extracted from such instances.

To get a sense of how difficult it is to discriminate between the algorithms from each dataset, we introduce a measure of difficulty based on how different the algorithms behave on our set of instances. We propose to use the following ratio:

$$\mathcal{D}_{\text{discr}}(A_{\text{inc}}, A_{\text{ch}}) = \frac{\sum_{I \in \mathcal{I}} median[(rt(A, I))_{A \in \mathcal{A}}]}{\left|\sum_{I \in \mathcal{I}} rt(A_{\text{ch}}, I) - rt(A_{\text{inc}}, I)\right|}.$$

This measure has been chosen because it grows when the two algorithms have similar performance, and it is invariant under scaling, so that the difficulty remains the same if running times are multiplied by a constant factor. It is also symmetric: exchanging A_{inc} and A_{ch} leads to the same result.

In Table 5.1, we report the characteristics of our datasets as well as their mean difficulty, median difficulty over all pairs and mean difficulty of the subset of the best 3 algorithms. Based on this measure, we expect it to be easy to discriminate between algorithms from BNSL, while SAT18 should provide a bigger challenge. The large discrepancy between the mean and median value, seen for SAT18 in particular, is caused by small groups of algorithms with very close performances. Pairs of algorithms from those groups usually have very high difficulty, reaching up to a million for SAT18, which affects the mean.

5.4.2 Implementation details

Our implementation is available on GitHub[2]. To estimate the parameter of running time distributions, we use maximum likelihood estimation, and we use a Cauchy distribution for the distribution-based discrimination method, as motivated in Section 5.4.3. For the timeout correction, the seed was set to 0.

For the random instance selection method, the seed was also set to 0. The parameter ρ for the discrimination-based selection method was set to 1.2. For the information-based method, we use the expression of Δ_{tot} defined for the distribution-based method, and to compute the expected value, which is an integral, we use Simpson's rule. For the Wilcoxon discrimination method, Conover (1998) recommends at least 20 samples; however, this would represent up to 20% of our instances for the CSP Minizinc dataset. Thus, we decided to follow irace (López-Ibáñez et al., 2016), which requires 5 samples in a context similar to ours. We found no significant performance change between different methods for managing zero differences, when paired data from both populations is equal, as such, we report the performance using Pratt's method (Pratt, 1959).

5.4.3 Estimation of the running time distribution

Our approach relies heavily on our ability to estimate the distribution of running times of algorithms on the instances. This distribution is used in 3 out of the 5 instance selection methods and one of our 3 discrimination methods. As such, the choice of the distribution could significantly impact the performance of those strategies. Fitting a distribution to our data requires us to decide how to handle the cutoff time and which distribution to use.

We note that when predicting a running time, a log transformation is typically used (Hurley and O'Sullivan, 2015; Hutter et al., 2014b). This transformation allows better performance for predicting running times, because running times distributions tend to be heavy-tailed as shown in the work of Gomes et al. (2000). Since in our case we are mostly interested in predicting the mean or the sum over instances, we do not apply this log transformation.

Handling censored running times

As explained in the problem definition, after a given cutoff time T_{cut}, the given algorithm is stopped. Running times are thus right-censored, which limits our ability to

[2] `github.com/Theomat/MPSEAS`

estimate the true distribution.

Our method for handling time-outs is based on the one proposed by Hutter et al. (2011a), which itself is based on a prior work from Schmee and Hahn (1979). The resulting algorithm is Algorithm 5.2 for instance I, with parameters $M \in \mathbb{N}$ and $t_{\max} \in \mathbb{R}_+$.

Algorithm 5.2 Correcting timeouts for a sample $(t_{I,A})_{A \in \mathcal{A}}$

```
fit Distribution on (t_{I,A})_{A∈A} without the timeouts
set N to the number of timeouts in (t_{I,A})_{A∈A} and n to 0
while not converged do
    set S to M · N + n samples from Distribution then increment n
    for k = 1 to N do
        set q_k to quantile k/(N+1) of S
        replace timeout k with min(q_k, t_max) in (t_{I,A})_{A∈A}
    end for
    fit Distribution on (t_{I,A})_{A∈A}
end while
return Distribution and (t_{I,A})_{A∈A}
```

There is a slight difference from the original algorithm in the use of a loop counter n to increment at each iteration the number of samples used to enable convergence when there is a majority of timeouts on an instance. The parameter M enables reducing the sampling variance; it is most important on instances with many timeouts. The parameter $t_{\max}$ prevents overly large variations of the samples. There are two steps in this algorithm: first, we estimate the parameters of the distribution, and second, we replace the timeouts in the sample. They are repeated until convergence, when the estimated parameters of the distribution are stable. We decided to stop when the squared difference between the parameters between two iterations is less than or equal to 1. Schmee and Hahn (1979) use the mean instead of the quantiles of a sample; however, heavy-tailed distributions such as the Cauchy distribution have an undefined mean. We chose to use the sampling approach used by Hutter et al. (2011a), which enabled them to translate the uncertainty and improve the likelihood for their random forest models.

Choosing a distribution

What is the distribution satisfying the imposed constraints that gives the best performance?

In practice, since only a set of running times is provided, the distribution parameters must be estimated. We explained how the parameters were estimated in practice

Table 5.2: Median log likelihood of Maximum Likelihood Estimation for Levy and Cauchy distributions over the instances of each dataset. The highest likelihood for each dataset is shown in boldface.

	CSP MiniZinc	BNSL	SAT 18	SAT 20
Levy	-129.6	**-58.08**	-299.7	-573.5
Cauchy	**-107.5**	-62.88	**-183.8**	**-364.9**

in Section 5.4, where here, we explain our choice of distribution. This choice can be motivated by choosing the best candidate distribution that has the lowest error on the set of all instances.

Since many running time distributions are heavy-tailed, we tested two heavy-tailed distributions on our four datasets. We report in Table 5.2 the median log likelihood for each distribution; the parameters of these distributions were estimated using maximum likelihood estimation. The Cauchy distribution provides a clear advantage over the Levy distribution. The only case in which the Levy distribution yields a higher likelihood shows a much smaller difference between the two distributions.

5.5 Experiments

We designed and conducted extensive experiments in order to answer our research question RQ6, which we divided into three parts as follows:

RQ 6.1. How much can our strategies reduce the CPU time required for evaluating a new algorithm?

RQ 6.2. How do the selection methods affect the accuracy of the strategies?

RQ 6.3. How well can our strategies discriminate between top-ranking algorithms?

A run consists of selecting an ordered pair $(A_{\text{ch}}, A_{\text{inc}})$ and running the strategy. On each run, all strategies have to compare the same A_{ch} and A_{inc}. In all of our experiments, we ran all of our strategies on each ordered pair of a given dataset.

General Performance Comparison

To answer question 6.1, we plotted our strategies in Figure 5.1, with a target confidence threshold $C_{\text{thres}} = 0.95$ (see Algorithm 5.1). For each of them, the y-axis shows accuracy (in percent) and the x-axis the median time used over all ordered pairs of

algorithms, as defined in Section 5.3. As this corresponds to a multi-objective setup, we highlight the Pareto fronts induced by our results. This does not imply that we can produce a strategy that follows the Pareto front between points; however, by changing the confidence threshold C_{thres}, we can obtain local curves around the performance of each strategy. Note that while we show the performance of our strategies without applying a penalty for timeouts, using penalty coefficients from $[|1; 10|]$ did not affect our findings.

On all datasets, we observe that our random baseline (random sampling a subset of 20% of the instances) shows rather strong performance, with 89% to 100% accuracy for about 20% of the running time. Further investigation (see Section 5.5) shows that the accuracy of the random baseline increases steeply as we add more instances, until reaching about 20% of the instances, after which the increase in accuracy is substantially slower. Thus, increasing the amount of instances does not lead to significantly higher accuracy. Moreover, more than half of the time, this strategy takes 17 to 22% of the running time, which means that the running times of the instances follow a distribution such that there are as many easy instances as hard ones.

We expect that this behaviour is linked to the nature of the competition datasets we are using; instances were gathered by experts to be representative and to show various levels of difficulty. We also note that the BNSL dataset, which is the one that gives the largest advantage to the random baseline, contains very few instances that are not solved within the cutoff time (about 10% of the instances, against about 50% for the other datasets). Choosing unsolved instances incurs a high penalty, because they offer no new information for deciding between the two algorithms while using up a large amount of running time.

On all datasets, we observe that the Wilcoxon method is superior and achieved the desired accuracy in less than 15% of the time; it thus represents the left-hand side of our Pareto front. The subset baseline uses consistently around 20% of the time but hardly reaches 90% accuracy on the hardest dataset; it contributes to the Pareto front only for BNSL. The distribution-based method tends to be more conservative and run longer but often reaches higher accuracies than the desired C_{thres} and thus marks the right-hand side of our Pareto front on our two SAT scenarios; however, it performs very poorly on BNSL, which is the scenario with the least background knowledge due to its low number of algorithms.

The instance selection methods do not show such a clear pattern. We notice, however, that the information-based method lies near or on the Pareto front when combined with Wilcoxon. In contrast, the discrimination-based and variance-based

methods show strong performance when used in combination with distribution-based discrimination.

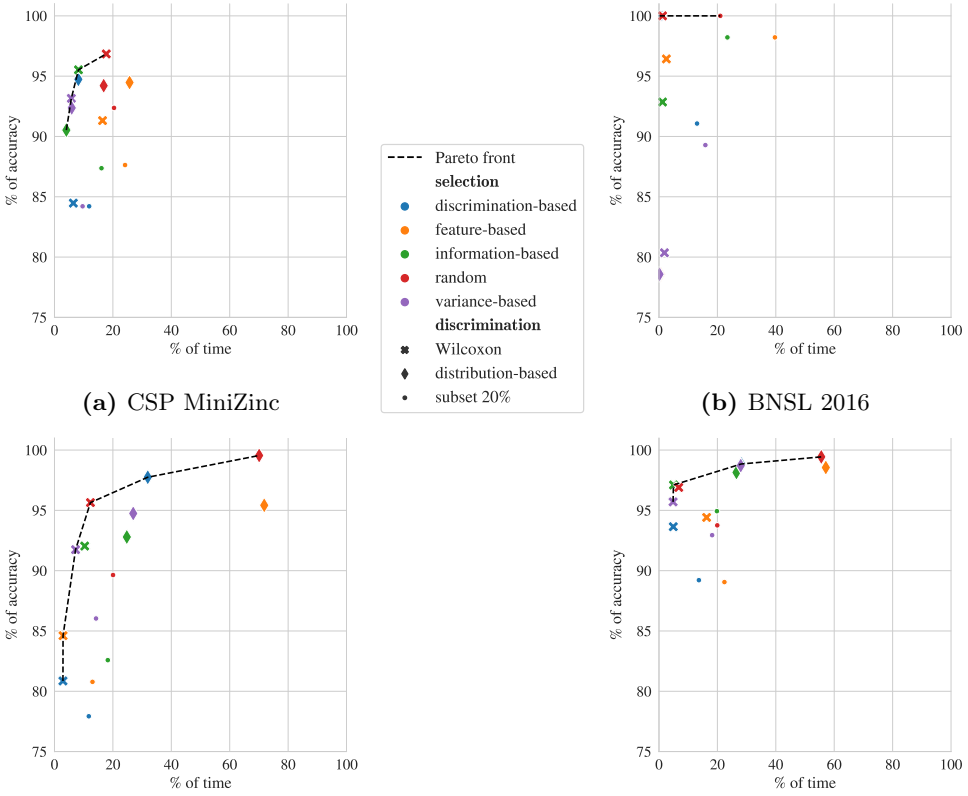
The evaluated strategies achieved up to 95.5% accuracy using 8.21% of the time on the MiniZinc dataset, 95.6% accuracy using 12.3% of the time on SAT18, and 97.1% accuracy using 4.96% of the time on SAT20. For the BNSL dataset, we observed a surprising 100% accuracy while using only 0.0001% of the time using the discrimination-based selection with Wilcoxon discrimination that is hidden behind on Figure 5.1b, running a median number of 6 instances. The observed performance of our strategies is consistent with the ranking of the datasets according to our difficulty metric (see Table 5.1 in Section 5.4) for the distribution-based methods, but not for Wilcoxon, where SAT20 should have been harder than MiniZinc. Overall, in the worst case, we managed to save 87.6% of CPU time while being 95.6% accurate, and in the best case, we saved 95.0% of CPU time while being 97.1% accurate.

Accuracy over time

To answer question 6.2, we ran our strategies without a stopping criterion, measuring regularly the percentage of accuracy and the time spent running A_{ch}. Figure 5.2 shows the accuracy (in percent) of the Wilcoxon and distribution-based discrimination methods on all our datasets.

Unlike Figure 5.1, which did not show any clear pattern regarding the instance selection methods, this analysis reveals two groups of methods. On all but the BNSL dataset, the information-based, variance-based and discrimination-based selection methods lead to a very high accuracy after 55 to 60% running time. This is consistent with the ratio of instances for which most algorithms time out, thus providing little discriminatory power. The feature-based method shows the lowest accuracy, and the random sampling comes in second to last after 40% of the running time.

The BNSL dataset is different, due to a low number of timeouts and large performance differences between the algorithms. In this case, randomly sampling instances offers high accuracy after a few instances. None of the selection methods offers a clear advantage, because all instances provide evidence towards the algorithm performing best. This suggests that the random method is a good choice for easy datasets, while more complex datasets containing instances that cannot be solved within the given cutoff time benefit from more sophisticated selection methods to save running time.



(a) CSP MiniZinc

(b) BNSL 2016

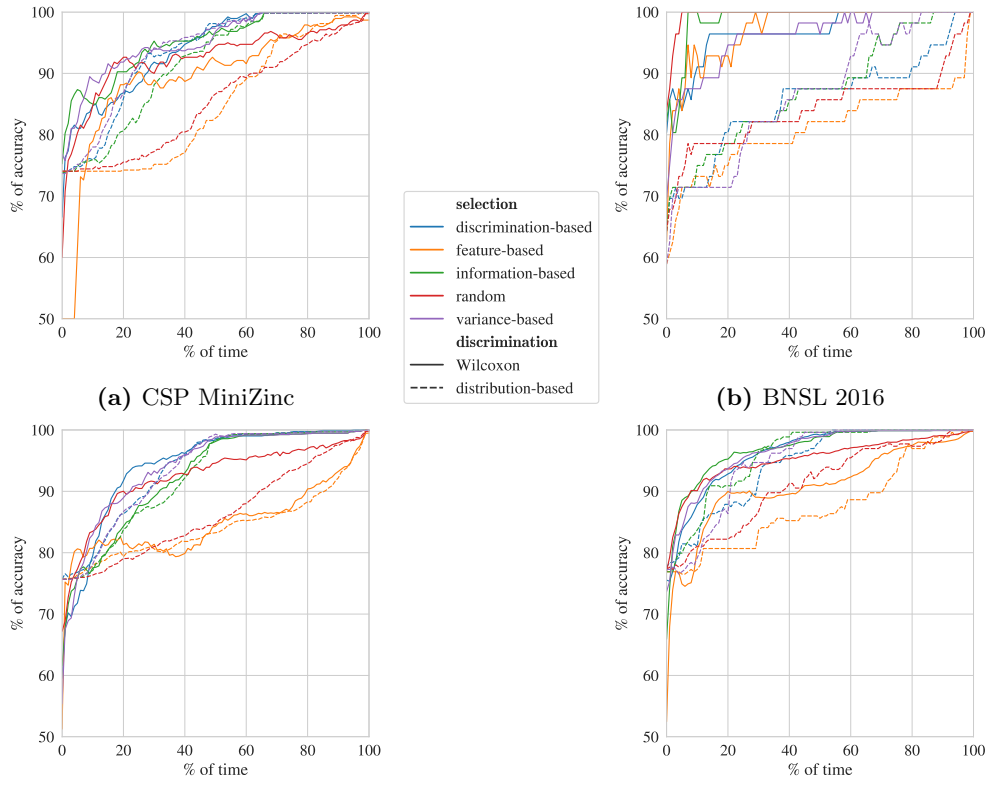
(c) SAT 18

(d) SAT 20

Figure 5.1: Accuracy over median running time. *y-axis*: percentage over all ordered pairs of algorithms in the dataset. *x-axis*: the time spent running the new algorithm.

Top ranking

To answer our last question 6.3, we decided to keep the top 10 algorithms according to their performance on the SAT20 dataset and use our strategies on this new dataset; this reflects the fact that often, the primary interest is in discriminating between top-ranking algorithms, be it to compare a new algorithm to the state of the art or to distinguish between the winners of a competition. As per our difficulty measure introduced in Section 5.4, the mean difficulty of the dataset thus obtained is 163, and the median is 22, which is higher than for any of our other datasets. Furthermore, the number of algorithms is reduced, which should reduce the performance of our methods based on prior knowledge. We report the results in Figure 5.3 analogous to what was



(a) CSP MiniZinc

(b) BNSL 2016

(c) SAT 18

(d) SAT 20

Figure 5.2: Accuracy over running time used. *y-axis*: percentage over all ordered pairs of algorithms in the dataset. *x-axis*: time spent running the new algorithm.

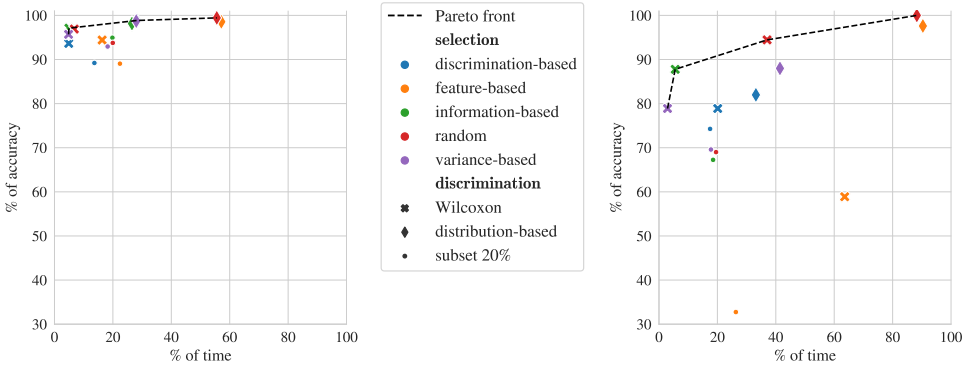
done in Section 5.5; for comparative purposes, we also plot the performances on the full SAT20 dataset. The performance of the subset method decreases by more than 10% in accuracy. The distribution-based discrimination method requires more time for this subset, and the discrimination-based selection method drops out of the Pareto front. Because they require prior knowledge, these methods encounter difficulties with this more challenging dataset. The Wilcoxon method is least affected, as it does not depend on prior knowledge; consequently, 3 out of the 4 strategies on the Pareto front use this method. The selection methods in combination with the Wilcoxon test are affected in different ways. The information-based and variance-based approaches yielded a quick but less accurate decision, while random sampling resulted in a slower decision, achieving 94.4% accuracy for 37.0% of running time.

In this experiment, which compares algorithms with similarly good performance,

the information-based method using the Wilcoxon test suffers less than the other strategies, both in terms of cost and accuracy. All other methods lead to either high cost or poor accuracy.

5.6 Conclusions and future work

In this chapter, we have investigated methods for reducing the computational effort required for comparing the performance of two automated reasoning algorithms, while gathering sufficient statistical evidence to correctly identify the solver that performs better on a given set of problem instances. We defined the per-set efficient algorithm selection problem (PSEAS) in Section 5.2. We studied the case in which the performance of a given algorithm is evaluated based on its running time on a set of instances. We described a set of strategies in Section 5.3, inspired by related problems from the literature and by novel considerations, and tested these on four datasets covering SAT, CSP and BNSL. Our experimental evaluation in Section 5.5 shows that on these datasets, some of our strategies consistently return the correct answer with at least 95% accuracy, while using less than 15% of the CPU time it would take to run the full comparison. In particular, using a Wilcoxon test to decide when to stop, while deciding the next instance to run based on the expected amount of information it can provide, is consistently near or among the best-performing approaches.



(a) SAT 20, full Dataset, 67 algorithms

(b) SAT 20, top-10 algorithms

Figure 5.3: Accuracy over running time used for the full and reduced SAT20 datasets. *y-axis*: percentage over all ordered algorithms' pairs in the dataset. *x-axis*: time spent running the new algorithm.

A finer-grained analysis of our instance selection methods (see Section 5.5) provides additional insights. We found that deciding on which instance to run based on its discrimination power, following the work of Gent et al. (2014), or simply on a notion of running time variance, has the potential to reduce the time required to make a decision when a significant fraction of the given instances are difficult.

Furthermore, we tested our methods on a smaller but more challenging set of algorithms, comprising the top 10 algorithms from the SAT20 competition. While the overall performance is lower than on the full dataset, the Wilcoxon method still reaches an accuracy of 94.4% in 37.0% of the overall running time. Overall, we found that for easy datasets, which discriminate between very different algorithms on instances that can be solved quickly, random sampling offers good performance. However, when facing hard instances or comparing well-performing algorithms, it is beneficial to use more sophisticated methods.

While the scope of our work presented here has been limited to comparing two algorithms, one interesting area of future work would be to extend it to many algorithms, to devise principled mechanisms for running competitions and other large-scale performance comparisons more efficiently.

Moreover, those methods can in theory be applied to comparing pairs of configurations of a single algorithm. This is the avenue studied in the following chapter.

6

Instance selection within automated algorithm configuration

In the previous chapter, we showed that selecting on which instances to run two algorithms and performing statistical tests allows us to make an accurate decision regarding which algorithm performs best on a set of instances in a fraction of the time it would require for running them on all instances.

Based on this evidence that selecting instances can unlock significant speed-ups in comparing the running times of algorithms, we extend this method to compare two configurations of the same algorithm and integrate it into an automated algorithm configurator, specifically, SMAC. In this chapter[1], we study the potential of this approach applied in the context of automated algorithm configuration (AAC). We adapt the previously studied selection methods to leverage the empirical performance models used in model-based configurators, and we also introduce two methods inspired by the active learning literature.

First, we test those methods on setup of experiments representing two situations requiring performance comparison that arise during the configuration process. Our empirical evaluation on six benchmarks shows that, depending on the problem in-

[1]Parts of this chapter have been published as Anastacio et al. (2022)

stances and their running time distributions, a decision can be reached 5 to 3000 times faster than with uniform random sampling, the method used in current state-of-the-art configurators. Then, we integrate the best-performing methods into the model-based configurator SMAC and evaluate the resulting configurator on five running time optimisation configuration scenarios from the literature. On two out of those, we almost double the performance improvement achieved by vanilla SMAC, allowing in one case to reach previously unseen performances (compared to Chapter 3). Additionally, we perform an ablation study to confirm that the instance selection mechanism is indeed responsible for this improvement and confirm that the selection of instances has very substantial potential in improving the state of the art in automated algorithm configuration.

6.1 Introduction

Comparing the performance of two configurations of a given algorithm is a key element of procedures for solving the AAC problem, since such comparisons are performed many times during the configuration process. However, in an automated algorithm configurator, the most computationally expensive task is to evaluate the quality of candidate parameter configurations. Executing time-consuming runs of the target algorithm on different problem instances to determine which parameter settings achieve the best performance requires substantial resources, and time is often wasted on less promising configurations as well as on instances that require a long running time to solve, regardless of the configuration utilised. With the increasing focus on sustainability, the computational resources and the environmental impact associated with the use of AI methods should be put under scrutiny, providing additional incentives to configure algorithms, but also to reduce the computational cost of AAC (see *e.g.* Tornede et al., 2023).

Several lines of research attempt to tackle this problem, mainly focusing on the idea of discarding configurations that are not sufficiently promising. For anytime algorithms, such as machine learning methods, there has been work on early stopping less promising runs based on learning curves (see *e.g.* Domhan et al., 2015; Luo et al., 2019), while adaptive capping mechanisms, such as the ones included in paramILS and irace (Hutter et al., 2009; López-Ibáñez et al., 2016), permit the early stopping of evaluations of configurations unlikely to be competitive with previously evaluated ones. Those lines of research are focused on the idea of discarding configurations deemed insufficiently promising.

On the other hand, in Chapter 5 (see also Matricon et al., 2021), inspired by the field of active learning, we explored the idea of selecting on which instances to compare two given algorithms. We introduced the per-set efficient algorithm selection problem (PSEAS) problem, which appears during AAC, albeit in a slightly different form. Rather than selecting an algorithm, the configurator needs to select a specific configuration of an algorithm among others.

In the following, Building on research from several areas, we aim to identify instances that help discriminate between the compared configurations. We argue that carefully selecting instances and avoiding long evaluations that provide only a limited amount of information allows the configurator to decide faster whether or not it should reject less promising configurations.

6.1.1 Background

In a configurator, the most expensive part is to evaluate the quality of numerous parameter configurations, executing time-consuming runs of the target algorithm on different problem instances to determine which parameter settings achieve the best performance. For anytime algorithms, such as machine learning methods, there has been work on early stopping less promising runs based on the learning curve (*e.g.* Domhan et al., 2015; Luo et al., 2019), *i.e.,* the curve representing the evolution of the performance of the model on the training (or validation) set during training. In contrast, adaptive capping, such as the mechanism included in paramILS (Hutter et al., 2009) and irace (López-Ibáñez et al., 2016), allows for early stopping the evaluation when a configuration is already deemed not to be competitive with the current incumbent (*i.e.,* , the best currently known). Those lines of research are focused on the idea of discarding configurations deemed insufficiently promising.

On the other hand, the per-set efficient algorithm selection problem (PSEAS) as defined in Chapter 5 appears during AAC, albeit in a slightly different form. Rather than selecting an algorithm, the configurator needs to select one of many configurations of a given target algorithms. In a configurator, the prior information on which we can base our instance selection comes in the form of prior runs of other configurations on the instances and of instance features. The latter are used in particular when learning a surrogate model, which is why we limit ourselves to model-based configurators. Because we have a model, our work relates to the questions addressed by active learning methods, which we therefore incorporate into our study.

6.1.2 Research questions

The work presented in this chapter addresses two main research questions, each subdivided into two questions addressed through our experiments.

RQ7 *How can we smartly select on which instances to run our evaluation to lower the time spent evaluating bad configurations?*

We evaluated our methods outside the configurator on artificially generated running time data. We defined two phases of comparison within the configuration process. The first phase takes place on a subset of instances on which configurations have been run before, and the second on instances never seen before. We conducted separate experiments to evaluate the performance of the selection methods within these two phases, aiming to answer the following questions:

RQ 7.1. How does the selection method perform when comparing a new configuration to the incumbent on the subset of instances for which we already collected information throughout the configuration run, as seen in phase 1?

RQ 7.2. How does the selection method perform when comparing a new configuration to the incumbent on all instances, selecting instances for which we did not collect information throughout the configuration run, as seen in phase 2?

To answer those questions, we adapted the two best-performing selection methods from Chapter 5 (also Matricon et al., 2021) with the performance model used in model-based configurators, added two methods inspired by the active learning literature (Gu et al., 2014), and evaluated them on five benchmarks. We designed two sets of experiments, showing that, depending on the problem instances and their running time distribution, the decision to stop evaluating a less promising configuration could be reached 5 to 3000 times faster than with random sampling, the method currently used in most state-of-the-art configurators.

RQ8 *How can instance selection boost the configurator performance or speed?*

We included the best-performing methods in a state-of-the-art configurator and evaluated their performance. Since SMAC (Hutter et al., 2011b) does not include any statistical test to decide when to discard configurations as soon as there is sufficient evidence for doing so, we implemented this test, as well as the selection mechanisms that showed the best result on our initial evaluation. We conducted an ablation study to evaluate the impact of the test we newly introduced without instance selection. This allows us to answer the following questions:

RQ 8.3. Do sophisticated instance selection mechanisms allow us to improve over picking instances uniformly at random?

RQ 8.4. How does the introduction of a statistical test during the comparison impact the performance of the configurator – in our case SMAC?

We evaluated the resulting configurator on five configuration scenarios from the literature and found that the method shows great potential, almost doubling the improvement reached by vanilla SMAC on two out of five scenarios and reaching previously unseen performances (compared to Chapter 3) on one of them (EAX solver on `rue-1000-3000`). Our ablation study confirmed that the origin of the improvement lies in the selection method or in the combined effect of both new mechanisms.

6.2 Comparison of two configurations

We want to efficiently compare two configurations of a single algorithm. This problem is similar to the comparison of two algorithms studied in Chapter 5. However, when comparing two algorithms, the distribution of running times for each algorithm is potentially completely different, with each algorithm taking advantage of different elements or structures in the problem instances. Comparing configurations of a single algorithm requires to detect smaller changes, potentially simple shifts of the distribution of running time. It is this important to evaluate if the methods are able to detect those changes. To do so, we need to gather enough statistical evidence while using the least possible amount of computing time.

6.2.1 Instance selection

Following the definition of AAC in Chapter 2, $\mathcal{I}$ is the finite set of instances and Ω is the set of valid configurations of the algorithm at hand. At a given step, we have partial running time information on $\mathcal{I}_{\text{known}} \subseteq \mathcal{I}$ for configurations in $\Omega_{\text{known}} \subseteq \Omega$, which means that for $\omega \in \Omega_{\text{known}}$, there exists information about the performance of ω on at least one instance of $\mathcal{I}_{\text{known}}$.

When comparing a challenger configuration ω_{ch} to the incumbent configuration ω_{inc}, instance selection appears in two forms. In Algorithm 6.1, a high-level description of how the comparison is conducted in SMAC (corresponding to line 10 of Algorithm 2.4 in Section 2.2.2), these are found in lines 6 and 10 (coloured purple), but the same mechanisms arises in any configurator. The first of these, which we name *phase 1*, corresponds to the PSEAS problem (Chapter 5), where we already know the performance of ω_{inc} on a set of instances $\mathcal{I}_{\text{known}}$ and want to determine whether ω_{ch} performs better on this set. The second, which we name *phase 2*, corresponds to a case where we know the performance of both ω_{inc} and ω_{ch} on $\mathcal{I}_{\text{known}}$ and want to evaluate both configurations on additional instances from $\mathcal{I} \setminus \mathcal{I}_{\text{known}}$. This can happen, for example, by steadily increasing the size of $\mathcal{I}_{\text{known}}$ at each iteration of the configuration process, or only when we do not have sufficient information to decide which one is the best. Since with our method we would have already discarded ω_{ch} if it was worse than ω_{inc}, and considering our goal of lowering the number of evaluations of the target algorithm, we will focus on this second case in the following.

In both cases, we seek to iteratively choose an instance $I \in \mathcal{I}_{\text{choose}} \subseteq \mathcal{I}$ and gather performance information on it until we satisfy a stopping condition.

Algorithm 6.1 Intensification for one challenger (based on SMAC Hutter et al. (2011b))

Input ω_{inc}: the incumbent configuration, ω_{ch}: the challenger configuration, $n_{\max}$: maximum number of new instances on which to run ω_{inc}, $\mathcal{I}$: the training instances, $\mathcal{I}_{\text{known}}$: the instances on which ω_{inc} was run.
Output ω_{ch}: the best found configuration.

if Insufficient runs for configuration ω_{inc} **then**
    execute a run of ω_{inc} on an instance not run yet, sampled uniformly at random
end if
$N \leftarrow 1$
while there are instances on which ω_{inc}, but not ω_{ch}, has been run **do**
    execute runs of ω_{ch} **on N instances from $\mathcal{I}_{\text{known}}$ on which ω_{ch} has not been run**
    if ω_{ch} is worse than ω_{inc} **then**
        return ω_{inc}
    else $N \leftarrow N \cdot 2$
    end if
end while
while ω_{inc} and ω_{ch} cannot be distinguished **do**
    if $N_{\text{run}} < n_{\max}$ **then**
        run ω_{inc} and ω_{ch} on **an instance from $\mathcal{I} \setminus \mathcal{I}_{\text{known}}$**
        $N_{\text{run}} \leftarrow N_{\text{run}} + 1$
    else
        return ω_{inc}
    end if
end while
return best of ω_{ch}, ω_{inc}

In phase 1, $\mathcal{I}_{\text{known}}$ is the subset of instances on which we have run our incumbent ω_{inc} so far, and ω_{inc} is the best performing configuration known to us on $\mathcal{I}_{\text{known}}$. At the first step, we have no performance information regarding ω_{ch}. At each step, we select an instance I from $\mathcal{I}_{\text{choose}}$, run ω_{ch} on I and add it to $\mathcal{I}_{\text{selected}}$. At any step, $\mathcal{I}_{\text{selected}} \subseteq \mathcal{I}_{\text{known}}$ and $\mathcal{I}_{\text{choose}} = \mathcal{I}_{\text{known}} \setminus \mathcal{I}_{\text{selected}}$. During this phase, we want to discard ω_{ch}, given sufficient evidence that it performs worse than ω_{inc}.If ω_{ch} performs as well or better than ω_{inc}, we would need to run it on all instances of $\mathcal{I}_{\text{known}}$ before applying the second phase or continuing the configuration process, thus we do not discard ω_{inc} early. Moreover, ω_{inc} already showed evidence that it performs better than previously tested challengers and thus comes with stronger evidence of good performance. Thus, our stopping criterion is either to have $\mathcal{I}_{\text{choose}} = \emptyset$, or to be confident that ω_{ch} is worse than ω_{inc}. We consider that, to select instances, we have access to an empirical prediction model trained on all pairs $(\omega, I) \in \Omega_{\text{known}} \times \mathcal{I}_{\text{known}}$, such that $\mathcal{M}(\omega, I)$ is known, and predicting the performance for any pair of instance and configuration.

In phase 2, we also have a subset $\mathcal{I}_{\text{known}} \subset \mathcal{I}$, but unlike in phase 1, there is no asymmetry between ω_{inc} and ω_{ch}. We know their running time on all instances of

$\mathcal{I}_{\text{known}}$ and both can be discarded given sufficient evidence. The goal is to be able to decide which of ω_{inc} and ω_{ch}, whose performance on $\mathcal{I}_{\text{known}}$ cannot be distinguished reliably, actually is to be preferred; to achieve this, we can select instances from $\mathcal{I}_{\text{choose}} = \mathcal{I} \backslash \mathcal{I}_{\text{known}}$ and iteratively add them to $\mathcal{I}_{\text{known}}$. Since no configuration has been run on any of the instances in $\mathcal{I}_{\text{choose}}$, we predict the performance of ω_{inc} and ω_{ch} with a predictive model trained on the performance of the configurations from Ω_{known} on the instances from $\mathcal{I}_{\text{known}}$. To do so, we require instance features, as defined in previous work for a broad range of problems, *e.g.* for Boolean satisfiability (SAT) (Xu et al., 2008), mixed integer programming (MIP) (Xu et al., 2011) or traveling salesperson problem (TSP) (Mersmann et al., 2013). In this phase, the stopping criterion is either to be able to clearly separate the performance of ω_{inc} and ω_{ch} on $\mathcal{I}_{\text{known}}$, or to reach a predefined maximum number $n_{\max} \in \mathbb{N}$ of instances added during the process.

In Chapter 5, we have used selection methods to decide which of two given solvers for an NP-hard problem performs best. To do so, each instance is assigned a score designed to reflect the relevance of choosing that instance both in terms of information obtained and cost incurred. The highest-scoring instance is chosen iteratively until one solver is deemed to have shown better performance than the other. Since we are working with similar types of solvers, we expect that a similar approach would be promising in our situation. We assign scores to instances from $\mathcal{I}_{\text{choose}}$ and select iteratively the highest scoring instance $I^* \in \operatorname{argmax}_{I \in \mathcal{I}_{\text{choose}}} score(I)$. We adapted two of the best methods tested on PSEAS to support the partial-information context. Note that these methods do not take advantage of the model in phase 1, while in phase 2, they are using the predictions given by the model as if they were ground truth. We did not adapt the information-based method, as it relies on assumptions regarding the performance distribution that could not be made in the current context. For PSEAS, this method relied on distributions estimated on the runs of Ω_{known} on all the instances from $\mathcal{I}$. In our current context, we only have information on $\mathcal{I}_{\text{known}}$, which is influenced heavily by the selection method itself; the estimated distributions would hence be strongly biased. Considering work from the active learning literature applicable to a random forest regression model – the model used in SMAC and previously demonstrated to be most effective for empirical performance prediction (Hutter et al., 2014b) –, we chose to adapt the work of Gu *et al.*, which considers active learning for terrain classification using random forests (Gu et al., 2014). Other works (*e.g.* Bhosle and Kokare, 2020; Ayerdi and Graña, 2015) have used similar ideas, focusing on the uncertainty of the model, so we also include a measure solely based on uncertainty.

Baseline: Uniform random sampling

This is equivalent to assigning every instance the same score, and thus sampling an instance uniformly at random.

Discrimination

This method, originally inspired by the work of Gent et al. (2014), aims to choose the instance that best discriminates between the best and other configurations. We say that a configuration ω is ρ-*dominated* on an instance I, for a given $\rho > 1$, if there exists another configuration ω' such that $\mathcal{M}(\omega', I) \leq \rho \cdot \mathcal{M}(\omega, I)$. Thus, we define the *discrimination quality* of an instance I, denoted $Q(I)$, as the fraction of known configurations that are ρ-dominated on this instance. The score is then defined as the discrimination quality divided by the mean running time of the instance:

$$score(I) = \frac{Q(I)}{Mean(I)}.$$

Variance

This approach is based on the intuition that an instance with high variance is likely to discriminate between two configurations. To also take into account the cost of running this instance, we divide the variance by the mean running time of the instance. Note that, according to Section 5.4.3, the underlying distribution of running times follows a Cauchy distribution and would thus not have a well-defined mean or variance. However, due to running times being bounded by 0 and the cutoff time, it is a truncated Cauchy distribution, which is well-behaved and has a mean and a variance. Our score is thus the relative variance

$$score(I) = \frac{Var(I)}{Mean(I)}.$$

Uncertainty-Diversity-Density (UDD)

This method is inspired by the work of Gu et al. (2014) from the active learning literature mentioned earlier. We decided to take the core ideas for their classification model and adapt it to our regression model. We named this approach UDD, because it is based on a combination of three scores: uncertainty, diversity and density. All three scores are scaled and translated to the interval $[0; 1]$ before computing $score(I)$

as

$$\begin{aligned} score(I) = &\ Uncertainty(I) \\ &+ \alpha \cdot Diversity(I) \\ &+ \beta \cdot Density(I). \end{aligned}$$

$Uncertainty(I)$ is the variance of the random forest on running time predictions for instance I and $Diversity(I) = -min_{I' \in \mathcal{I}_{\text{known}}} \mathbb{D}(I, I')$, where $\mathbb{D}$ is a distance function over instances. Intuitively, the closer I is to instances from $\mathcal{I}_{\text{known}}$, the more unlikely it is to provide additional information. Finally, $Density(I) = \frac{1}{k} \cdot \sum_{I' \in \mathcal{N}_k(I, \mathbb{D})} \mathbb{D}(I, I')^2$ where $k \in \mathbb{N}$ is a parameter, $\mathbb{D}$ is a the same distance function as for diversity and $\mathcal{N}_k(I, \mathbb{D}, \mathcal{I}_{\text{choose}})$ returns the k closest neighbours of I in $\mathcal{I}_{\text{choose}} \setminus \{I\}$ according to $\mathbb{D}$. Intuitively, if an instance I is close to many instances from $\mathcal{I}_{\text{choose}}$, then running I should also provide information about these other instances.

Uncertainty

This corresponds to UDD with α and β set to zero, which is reminiscent of the variance method applied to the predictions of a model instead of measured performance values.

6.2.2 Stopping criterion

At each phase, we need to decide when we consider that sufficient statistical evidence has been gathered. In Algorithm 6.1, this decision appears in lines 7 and 12. Based on previous work (Matricon et al., 2021; López-Ibáñez et al., 2016), we use a *Wilcoxon matched-pairs signed-rank test* (Conover, 1998) with a significance level of 0.05.

6.3 Setup of experiments

The final goal of this chapter is to include the instance selection methods presented in the previous section in a configurator at both phases and to assess the impact of these modifications on the performance. However, directly including them without decomposing the mechanism into smaller, more easily analysed components would give us little to no information about which of the components shows the desired impact.

Following our research questions, our evaluation is divided into two main sections, each subdivided into two research questions. We evaluated our methods outside the configurator on artificially generated running time data, conducting experiments to

evaluate the performance of the selection methods, separately for phase 1 and phase 2 defined earlier. Then, we included the best performing methods in the state-of-the-art configurator SMAC and conducted an ablation study to evaluate the impact of the statistical test we newly introduced with and without instance selection.

6.3.1 Datasets

We used configuration scenarios taken from the Algorithm configuration library AClib (Hutter et al., 2014a) or derived from these. Table 6.1 reminds the features of the considered scenarios; more details are shown in Chapter 2.

Evaluation outside of a configurator

We evaluated our method on two NP-hard problems that have been well-studied in the algorithm configuration literature: SAT and MIP. For each, we chose two widely used datasets from AClib and added a more recent and harder dataset to test the limits of our methods.

For SAT, we used CF and IBM from AClib and generated a new set of cryptography instances based on the work of Nejati and Ganesh (2019). Specifically, we used the sha256 encoding, 16 to 60 rounds and an input size of 2^n with $n \in \mathbb{N}, n \leq 10$. For this last dataset, we set the cutoff time to 5000 seconds, such that 70% of the instances can be solved by the default configuration before reaching this time limit. Based on the results of the SAT competition 2020 (Balyo et al., 2020), we decided to configure Kissat (Biere et al., 2020), the best SAT solver currently available.

For MIP, we used RCW2, REG200 from AClib and added a more difficult dataset based on the work of König et al. (2021), which is comprised of challenging neural

Table 6.1: Benchmark instance sets characteristics.

Name	Origin	Training size	Testing size	Features	Clusters
CF	generated	298	301	113	14
IBM	real-world	382	302	113	21
Crypto	generated	225	225	103	8
CLS	generated	50	50	148	3
RCW2	real-world	495	495	148	6
REG200	generated	999	999	148	2
MIPverify	generated	92	92	206	5
rue-1000-3000	generated	50	250	64	9

network verification problems. For this last dataset, we set the cutoff-time to 9000 seconds, such that 70% of the instances can be solved by the default configuration before reaching this time limit . For these scenarios, we chose CPLEX, since it is well known in the literature and also prominently used in AClib.

Evaluation inside a configurator

We chose 2 TSP scenarios and 3 MIP scenarios. Because we had to run many different versions of the configurators, we selected well-studied scenarios with a relatively low time budget. *For TSP,* we used two datasets from AClib (EAX and LKH on rue-1000-3000) due to their short configuration time (see *e.g.* Pushak and Hoos, 2020). *For MIP,* we used three datasets from AClib (REG200, CLS and RCW2) well-known from the literature (see *e.g.* Hutter et al., 2010b). For this scenario, we use a cutoff time of 300 seconds (following Cáceres et al. (2017)), since our method is more suited for situations in which runs might be cut off upon reaching the time limit. Using a cutoff time of 10 000 seconds results in all runs completing before the cutoff is reached.

6.3.2 Implementation details

Our implementations are available on GitHub[2]. The UDD method requires a distance function $\mathbb{D}$ in the instance space; we compute this using the same procedure as in Chapter 5, where we find weights for instance features and compute a weighted feature distance between instances. Since the discrimination and UDD methods have parameters, we tuned those with a simple grid search on a separate scenario (Kissat with the SWGCP dataset from AClib). For discrimination, we evaluated values in $[1.01; 2]$ with a step size of 0.11 and found that $\rho = 1.12$ performed well on both phases. For UDD, we evaluated values in $[0; 2]$ with a step size of 0.21 for both values independently and found that $\alpha = 0.2$ and $\beta = 1.4$ performed well on both levels.

Evaluation outside the configuration process

To carry out our empirical investigation, a dataset of configurations and their associated performance scores were required. To obtain such a set, we generated 100 random configurations uniformly at random for each solver and ran them on all instances of the datasets included in the respective configuration scenarios. This allowed us to

[2]The implementation of the first set of experiments is found at `github.com/Theomat/MPSEAS` and that of the second set of experiments is found at `github.com/ADA-research/SMACIS`

collect performance data on many pairs of problem instances and algorithm configurations. We used the same random forest model as in SMAC (Hutter et al., 2011b) as an empirical performance model (EPM). We trained this EPM on the previously described performance dataset. To evaluate how efficient our methods will be along a configuration run, we trained the EPM on various amounts of performance data: the number of known configurations is in $[10, 20, 30, 40, 50]$ and the amount of known instances is a fraction of $[0.1, 0.2, 0.3, 0.4, 0.5]$ of the full dataset.

Evaluation inside the configuration process

This evaluation required us to include the selection method in the configurator. As our first evaluation is based on the inner working of SMAC, we included the methods in SMAC3 version 1.1.1. However, in principle, a similar mechanism could be used in any configuration procedure.

We included in SMAC a *Wilcoxon matched-pairs signed-ranks test* (Conover, 1998) with a significance level of 0.05 between the runtime of the incumbent ω_{inc} and the challenger ω_{ch} to decide if the challenger can be discarded. Remember that in phase 1, we only discard the challenger in the presence of sufficient evidence that it performs worse than the incumbent and never decide to replace the incumbent based on the test. This means that we take the risk of discarding good configurations in case of error, but would not risk replacing the incumbent with a worse configuration (on known instances).

Due to the large number of statistical tests involved, we need to account for the problem of multiple testing. First, we do not perform tests before running ω_{ch} on at least 5 instances based on the recommended smallest number of samples for the statistical test to be effective (Conover, 1998). Moreover, we use batches to lower the number of tests performed. For each test, we apply a Bonferroni correction (Dunn, 1961), which means that we divide the significance threshold by the number of tests to be performed (given by the size of $\mathcal{I}_{\text{known}}$ divided by our batch size) before comparing it to our confidence threshold. Note that if we use a fixed batch size, the larger the number of instances in $\mathcal{I}_{\text{known}}$, the lower the p-value would need to be to reject the null-hypothesis. Along the configuration process, more instances are added to $\mathcal{I}_{\text{known}}$, and it would become very unlikely to reject a new incumbent, whilst the time to compare configurations will become larger due to the number of runs required. This phenomenon would counteract our goal to lower the comparison computation cost. Thus, we decide to set our batch size relatively to the size of $\mathcal{I}_{\text{known}}$. Based on the results of Matricon et al. (2021), we decided to test every 20% of $\mathcal{I}_{\text{known}}$;

this corresponds to an amount of instances above which the Wilcoxon test had high accuracy in their reported results for most of the selection methods.

Due to the large computation time required to evaluate every possible combination of methods between phase 1 and phase 2, we had to carefully select a subset of possible experiments; specifically, we only considered the best-performing methods from the first set of experiments at each phase of the configuration. Because our method involves adding a Wilcoxon test to stop comparisons early, we also evaluate its impact separately, to gain further insights into the observed behaviour. To compare the performance of two versions of the configurator, we want to look at the expected best performance of the best found configuration. Since a user would typically run the configurator several times and select the configuration found to perform best on the given training data (which corresponds to the so-called standard protocol), we apply the following protocol: we run the configuration 8 times with seeds from 1 to 8, repeatedly sample 5 runs uniformly at random from that set of 8, and identify the best of these according to performance on the training set. We used 1000 such samples to estimate the probability distribution of the quality of the result produced by each configurator on each configuration scenario. We then compared the medians of these empirical distributions. This is similar to procedures used in the literature (see *e.g.* Pushak and Hoos, 2020; Anastacio and Hoos, 2020b).

6.3.3 Execution environment

The first set of experiments was run on the high-performance compute cluster *Grace*, hosted by Leiden University, running CentOS Linux operating system version 8.5. Each node is equipped with two Intel Xeon E5-2683 CPUs with 16 cores and 40MB cache each, as well as 94GB RAM. The second set of experiments was run on the high-performance cluster *Kathleen*, hosted by RWTH Aachen University, running Rocky Linux operating system version 9.3. Each node is equipped with two AMD EPYC 7543 CPUs with 32 cores and 256 MB of cache each, as well as 1 TB of RAM.

6.4 Evaluation outside the configuration process

This section describes the results obtained from our first set of experiments. The goal of these experiments was to evaluate how well the selection methods perform at both phases described in Section 6.2, independently of the whole configuration procedure around. We show aggregated results here, but the raw results and scripts to generate

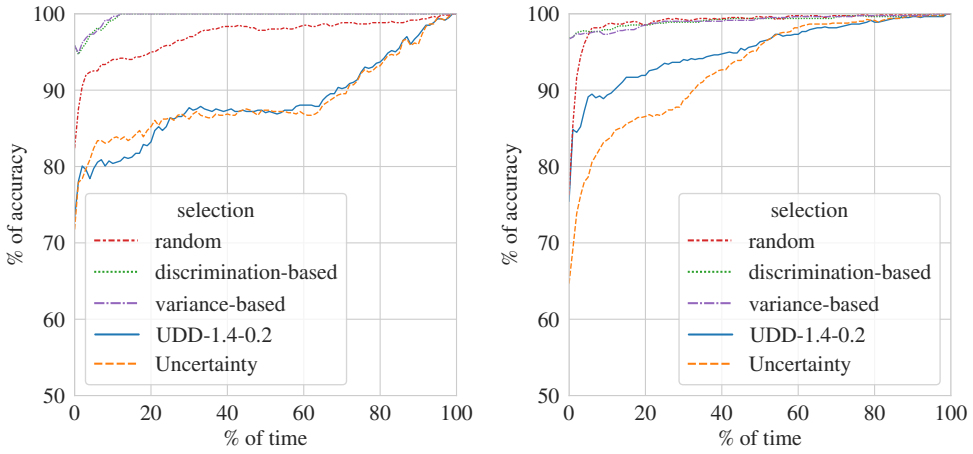
more visualisations are available in our git repository.

6.4.1 Compare configurations on known instances

To answer the first question – *How does the selection method perform to compare a new configuration to the incumbent on the subset of instances for which we already collected information throughout the configuration run, as seen in phase 1*? – we consider phase 1 (see In phase 1,). We populate $\mathcal{I}_{\text{known}}$ and Ω_{known} with instances and configurations, respectively, selected uniformly at random. We note that this does not reflect accurately the way these sets are populated during a configuration run since each configurator will bias their search according to their configuration sampling mechanisms. We choose $\omega_{\text{inc}} \in \text{argmin}_{\omega \in \Omega_{\text{known}}} \mathcal{M}(\omega, \mathcal{I}_{\text{known}})$ and train the random forest model on all the available data. Then we pick configurations from $\Omega \setminus \Omega_{\text{known}}$ as ω_{ch} and run our iterative process; we refer to this as one run. We stop when we have run all instances of $\mathcal{I}_{\text{selected}}$. After each new instance is added, we report the percentage of time that has been spent up to that point to evaluate $\mathcal{M}(\omega_{\text{ch}}, \mathcal{I}_{\text{selected}})$ compared to running it on all instances of $\mathcal{I}_{\text{known}}$, and we perform a *Wilcoxon matched-pairs signed-ranks test* (Conover, 1998) with a significance level of 0.05 to decide if the challenger can be discarded. If $\mathcal{M}(\omega_{\text{ch}}, \mathcal{I}_{\text{selected}}) > \mathcal{M}(\omega_{\text{inc}}, \mathcal{I}_{\text{selected}})$ and the statistical test indicated statistical significance, ω_{ch} is discarded. We compare the resulting decision to the ground truth given by comparing $\mathcal{M}(\omega_{\text{ch}}, \mathcal{I}_{\text{known}})$ to $\mathcal{M}(\omega_{\text{inc}}, \mathcal{I}_{\text{known}})$ to assess the accuracy of the decision. For a given pair of $(\mathcal{I}_{\text{known}}, \Omega_{\text{known}})$, we performed 10 independent runs, using different pseudo-random number seeds, and report the average over those runs.

Figure 6.1 shows the collected accuracy over the time spent to make the comparison for two examples. Figure 6.1a is a case in which the discrimination and variance methods are significantly more accurate than the three others at any given time, while UDD and uncertainty show lower accuracy than random sampling. Figure 6.1b is a case in which discrimination and variance methods start with an advantage over random but quickly reach the same accuracy; once again, UDD and uncertainty perform substantially worse.

Figure 6.2 summarises the previously described curves by computing the area under the curve (AUC) for all tested amounts of prior data; the higher the AUC, the faster and more accurately the decision can be taken. This visualisation allows us to examine how the methods compare, but also illustrates the impact of the prior data used on the empirical performance model. In all our scenarios, we can see a clear correlation



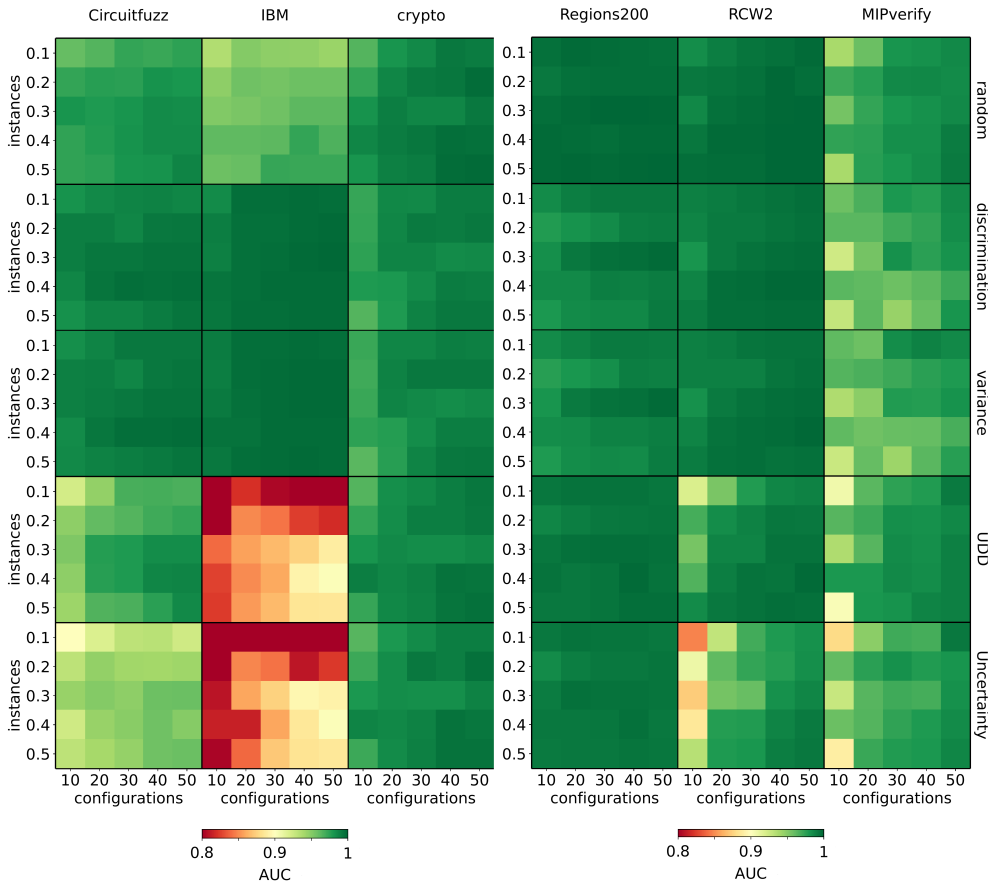
(a) Kissat on IBM, 50% instances, 50 configurations

(b) Cplex on RCW2, 10% instances, 20 configurations

Figure 6.1: Mean accuracy of the Wilcoxon test ($p < 0.05$) on which among ω_{ch} and ω_{inc} performs best *vs* the percentage of time spent on evaluations (100% means that all instances of $\mathcal{I}_{\text{known}}$ have been run)

between the amount of configurations in Ω_{known} and the AUC. This would allow the selection method to become increasingly efficient over the course of the configuration run, thereby avoiding wasted time in the final steps. On the other hand, adding more instances does not seem to improve performance consistently. This is in line with the expectation that our instance sets are built to be homogeneous; thus, adding more instances will be unlikely to improve the model substantially.

Regarding the selection methods, randomly sampling instances performs well, but in most cases, the discrimination and variance approaches are superior. The IBM dataset is unusual in this context, in that the UDD and uncertainty methods perform notably worse than random sampling. This could be explained by the large variation in the running time required to solve the respective problem instances: There are many instances requiring long running times, but also many that are solved within a second or less. This means that selecting the wrong instance can have a dramatic effect on overall running time. This would explain why random sampling does not perform as well on this scenario as on the others, but also why adding more instances in the prior data improves the performance of our selection methods for this scenario.

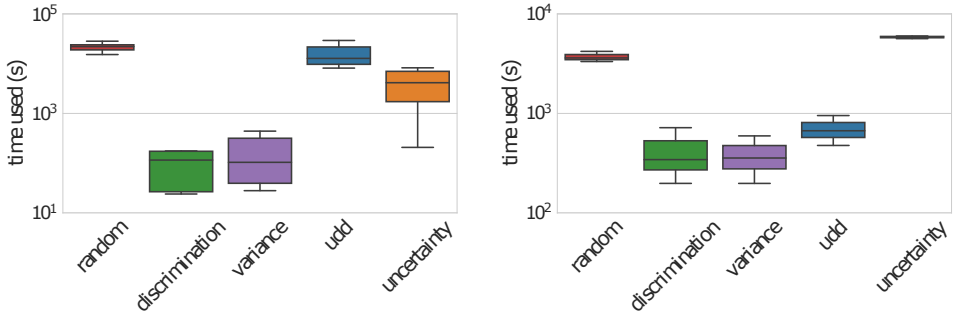


(a) Kissat scenarios **(b)** Cplex scenarios

Figure 6.2: Area under the curve of the mean accuracy of the Wilcoxon test ($p < 0.05$) on which among ω_{ch} and ω_{inc} performs best against the time spent on evaluations.

6.4.2 Compare configurations on unknown instances

To answer the second question – *How does the selection method perform to compare a new configuration to the incumbent on all instances, selecting instances for which we did not collect information throughout the configuration run as seen in phase 2?* – we consider phase 2 (see In phase 2,). We populate $\mathcal{I}_{\text{known}}$ and Ω_{known} with instances and configurations, respectively, selected uniformly at random. The random forest model is trained on all the performance data regarding all pairs of instance and configuration from those two sets. We choose $\omega_{\text{inc}} \in \text{argmin}_{\omega \in \Omega_{\text{known}}} \mathcal{M}(\omega, \mathcal{I}_{\text{selected}})$ and collect all $\omega_{\text{ch}} \in \Omega \setminus \Omega_{\text{known}}$, such that the performance of ω_{inc} and ω_{ch} cannot be distinguished



(a) Kissat – crypto, 30% instances, 20 configurations

(b) cplex – RCW2, 40% instances, 40 configurations

Figure 6.3: Time used (in seconds) before deciding that one configuration is better than the other based on a Wilcoxon test ($\alpha = 0.05$) or reaching a maximum of 10 instance selected

on the instances of $\mathcal{I}_{\text{known}}$ by a Wilcoxon test with a significance level of 0.05. We then apply our selection methods to select up to $n_{\max} = 10$ instances on which we run both configurations until they can be distinguished using the previous test.

For each method and each considered pair of $(\mathcal{I}_{\text{known}}, \Omega_{\text{known}})$, we gather the time used to decide between the two configurations at hand, *i.e.* the sum of the running times of ω_{inc} and ω_{ch} on $\mathcal{I}_{\text{selected}}$. Figure 6.3 shows the running times obtained for two example scenarios.

To evaluate the performance of the selection methods, we computed the median time used to run the instances selected by each of the methods for each prior data, see Table 6.2. The statistical significance of differences in the medians was tested with a permutation test (significance level of 0.05). In most cases, random is outperformed by all other methods, with some exceptions (uncertainty performs worse on RCW2, and random is best on MIPverify). The data shows that discrimination and variance

Table 6.2: Median time in seconds for each method over every tested prior data, with lowest medians boldfaced (statistical significance according to a permutation test with $\alpha = 0.05$).

	ibm	kissat cf	crypto	reg200	cplex RCW2	MIPverify
random	1557	979.7	21243	576.8	4138	**29470**
discrimination	0.086	143.6	419.3	**96.66**	364.7	44390
variance	0.776	**95.16**	**372.2**	109.5	**342.0**	41365
udd	880.9	393.2	13483	379.7	1299	**28845**
uncertainty	**0.033**	330.8	2361.9	152.7	5974	39801

outperform the other methods in almost all cases, with variance providing a speedup ranging from a factor of 5.8 up to 3000 compared to random. We note that the high speedups observed for the IBM dataset are linked to high variance in the running time distribution of the instances, which range from milliseconds to the timeout of 300 seconds.

6.4.3 Discussion

The results shown in this section indicate that the best-performing methods to discriminate between two configurations of the same algorithm within a limited amount of time are the ones based on the running time variance on each instance and on their discrimination power. We notice that both methods inspired by the active learning literature are not performing as well. Whilst we wanted to assess these methods on our problem, this was to be expected, since they were designed with a different goal in mind. Indeed, the field of active learning focuses on improving the accuracy of the model, whereas we only use a model to avoid having to run each configuration on each instance. Improving the accuracy of this model can serve our purpose, but it is not our final goal. We note that the experiments reported here made use of randomly chosen configurations of a given algorithm. As a result, the variation in running times between these configurations is much larger than that expected during an actual configuration run, which focuses on high-performance configurations. While this certainly does not invalidate our results, it implies that we should not expect speedups as large as the ones observed in Table 6.2 when including our methods inside a configurator.

6.5 Evaluation inside the configuration process

As previously shown, applying instance selection and performing a statistical test allows us to spend less time on comparing the performance of two configurations through two expected mechanisms: early stopping of the evaluation of less promising configurations and performance comparisons on less time-consuming instances. In this section, we include the instance selection mechanism inside a model-based configurator in order to evaluate whether the previously observed results can be translated to the performance of the configurator itself. To do so, we expanded the prominent sequential model-based configurator SMAC. However, since SMAC does not include a statistical test, the two aspects of our methods have to be evaluated separately. First, we evaluate SMAC-IS (SMAC with Instance Selection), a version of SMAC3 in which

we added at both phases of the both parts of the instance selection method, namely a *Wilcoxon matched-pairs signed-ranks test* (Conover, 1998) with a significance level $\alpha = 0.05$ to decide if the challenger configuration should be dropped earlier, and an instance selection method to decide on which instance the next run should happen. To compare the performance of SMAC3 to SMAC-IS, we followed the procedure described in Section 6.3 and obtained for each scenario and configurator a distribution of best configurations. The following results are based on those distributions.

6.5.1 Impact of the instance selection methods

To answer our first question, "*Do sophisticated instance selection mechanisms allow us to improve over picking instances uniformly at random?*", we implemented the instance selection mechanisms inside SMAC at the two phases identified earlier and named this new version SMAC-IS. Table 6.3 shows the median performance values of the best configurations distribution. We validate the statistical significance of the differences with a Mann-Whitney U-test with a significance level $\alpha = 0.05$. Moreover, we show in bold methods that perform better than SMAC, our baseline (the performance of SMAC can bee seen in Table 6.4).

Compared to vanilla SMAC, SMAC-IS showed improved behaviour for three of our five scenarios. In particular, the EAX on rue-1000-3000 scenario (Table 6.3a), displays improvements with most instance selection methods; at best, from a default performance of 120.82 seconds, SMAC-IS reaches a median of 65.68 seconds, while SMAC could only reach 92.93 seconds. A similarly impressive improvement was achieved for CPLEX on RCW2 (Table 6.3d), on which, from a default value of 115.95 seconds, SMAC-IS reaches a median of 57.63 seconds, while SMAC could only reach 83.96 seconds.

For CPLEX on REG200 (Table 6.3e), SMAC-IS improves slightly over SMAC, but for CPLEX on CLS (Table 6.3c) it does not, despite being able to find a better configuration than the default. At the other end of the spectrum, for LKH on rue-1000-3000 (Table 6.3b) SMAC-IS returns configurations that perform even worse than the default values in half of the cases.

6.5.2 Impact of the statistical test

To evaluate the impact of the statistical test, we examined the performance of SMAC-IS with random sampling at both phases, which corresponds to vanilla SMAC with a Wilcoxon test to discriminate between the performance of the incumbent and of

Table 6.3: Median performance of SMAC-IS with the selection methods random (rand), variance-based (var) and discrimination-based (disc) at both phases. Boldfaced values are better than those for vanilla SMAC. The lowest median is underlined. All underlined medians are significantly different from others based on a Mann-Whitney U-test ($\alpha = 0.05$).

(a) eax rue-1000-3000

		phase 2		
		rand	var	disc
phase 1	rand	**89.87**	**71.87**	**72.72**
	var	121.69	**87.14**	95.55
	disc	**89.80**	**87.34**	**65.68**

(b) lkh rue-1000-3000

		phase 2		
		rand	var	disc
phase 1	rand	233.13	228.74	229.48
	var	229.39	229.00	243.04
	disc	228.76	185.62	229.19

(c) cplex cls

		phase 2		
		rand	var	disc
phase 1	rand	1.79	1.73	1.67
	var	1.61	1.66	1.68
	disc	1.66	1.69	1.65

(d) cplex RCW2

		phase 2		
		rand	var	disc
phase 1	rand	113.73	113.54	113.94
	var	**57.63**	113.98	114.27
	disc	86.06	114.86	114.48

(e) cplex regions200

		phase 2		
		rand	var	disc
phase 1	rand	3.68	2.95	3.35
	var	3.25	3.77	3.74
	disc	**2.79**	**2.78**	3.05

the challenger configuration at both phases of the configuration. We dub this variant SMAC-W (SMAC with Wilcoxon test). We show the median of those distributions in Table 6.4a. Similarly to the previous results, we validated the statistical significance of the differences using a Mann-Whitney U-test (with $\alpha = 0.05$) and detected statistical significance for all observed differences.

In all except one scenario, the use of a statistical test for early stopping of the comparison has an adverse effect. To further investigate these results, we examined the frequency at which the challenger replaces the incumbent and the number of instances on which the configurations are evaluated. These results are shown in Table 6.4b. We note that the number of accepted incumbents during a run is significantly lower when using the test. Vanilla SMAC accepts the incumbent up to twice as often than SMAC-W for CPLEX on CLS. Moreover, since incumbents get rejected more quickly, the number of instances on which the configurations are evaluated does not increase

Table 6.4: Comparison of SMAC and SMAC-W, respectively, without and with a Wilcoxon test to decide whether a challenger configuration should be kept longer.

(a) Median PAR10 of the best found configurations. The lowest medians are underlined, all are statistically significantly lower according to a Mann-Whitney U-test (with $\alpha = 0.05$).

scenario		default	SMAC	SMAC-W
CPLEX	CLS	1.72	<u>1.31</u>	1.79
	RCW2	115.97	<u>83.96</u>	113.73
	REG200	6.13	<u>2.84</u>	3.68
EAX	rue-1000-3000	120.82	92.93	<u>89.87</u>
LKH		229.22	<u>157.83</u>	233.13

(b) Mean number of changes in incumbent and number of instances in $\mathcal{I}_{\text{known}}$ at the end of the configuration procedure

scenario		changes		instances	
		SMAC-W	SMAC	SMAC-W	SMAC
CPLEX	CLS	3.0	7.1	50	50
	RCW2	3.1	4.2	495	495
	REG200	4.5	5.6	816	823
EAX	rue-1000-3000	4.9	7.6	332	294
LKH		3.1	3.5	432	685

as quickly in SMAC-W as in SMAC. We can expect that running on a smaller number of instances prevents the configurator from seeing the full range of instances on which the algorithm should perform well, leading to overfitting. This is especially evident for the LKH scenario, on which the expected performance of SMAC-W is worse than the default on the test set. We also noticed that the only case in which the number of instances seen during configuration is higher for SMAC-W corresponds to the only scenario in which SMAC-W performs better than SMAC. Based on those results, we can answer our research question and state that in most cases, adding a Wilcoxon test to SMAC hinders its performance.

6.5.3 Discussion

Since the instance selection mechanism did not allow us to improve over SMAC on all scenarios, we looked into the characteristics of each scenario to better understand what might allow instance selection to reach its full potential.

When we look at each selection phase separately, there is no clear trend in terms of

which method performs best at any of those. One expectation was that for scenarios with a low running time, the overhead induced by our methods would hinder the process. Still, SMAC-IS performed slightly better than SMAC on one out of our two scenarios with short running times (CPLEX on CLS and REG200), so this hypothesis does not hold in our experiments. Another expectation was that the homogeneity of the dataset would strongly impact the ability to select the right instances and to decide accurately which configurations to drop. However, the best and worst outcomes were obtained on the same dataset, rue-1000-3000, on which we found 9 clusters of instances when applying a simple mean shift algorithm, which is the highest number among our datasets. Moreover, two seemingly homogeneous datasets, namely CLS and REG200, show very different outcomes. However, the number of clusters does not capture how far those clusters are from each other, which would impact the difficulty of selecting representative instances.

Thus, based on our results, we do not see a clear trend regarding what kinds of scenarios would benefit (or not) from our instance selection mechanism. We note, however, that in two out of five scenarios, we were able to nearly double the improvement obtained by SMAC. This improvement demonstrates that in some scenarios, selecting the instances on which to run the configurations at hand can significantly improve the performance of a general-purpose algorithm configurator.

6.6 Conclusion

Inspired by the success of instance selection when comparing algorithms (see Chapter 5), we adapted four methods from several fields (Matricon et al., 2021; Gu et al., 2014) that could be applied to select instances in the context of automated algorithm configuration (AAC). We identified two steps of AAC procedures at which the selection mechanism could be applied and designed two sets of experiments to assess the performance gains that are thus obtainable. In the first, we considered a situation in which the performance of an incumbent configuration on a set of instances is known, and we want to determine whether the challenger configuration, whose performance is unknown, performs better on this set. In the second, two similarly performing configurations have to be evaluated on unknown instances. Our results show that in both cases, there is considerable potential in the use of those methods, in particular the ones based on the variability in running time or on discrimination power.

Based on those encouraging results, we included the two best selection mechanisms identified in the first phase of our study at both identified steps of the configuration

process within the prominent and state-of-the-art SMAC3 configuration system. On half of the considered scenarios, selecting on which instances to run the first and second phase, on top of performing a Wilcoxon test to decide when to stop the comparison between the current incumbent and a challenger configuration, makes it possible to reach better performing configurations within the same configuration budget, sometimes reaching major improvement compared to SMAC and all previously evaluated configurators according to the results shown in Chapter 3. However, we have not yet found a straightforward way to decide which instance selection method to apply or which scenarios have the potential to benefit from them. Moreover, we studied the impact of solely adding the Wilcoxon test and found that, in most scenarios, using the test degrades the configuration process of SMAC. We note that on the scenarios we have studied, use of the test lowers the number of accepted challengers, likely discarding well-performing configurations by mistake, and tends to slow down the addition of new instances to the pool of instances on which configurations are evaluated. This second point could potentially lead to a form of over-fitting. Those observations confirm that the selection mechanism is responsible for the observed improvements.

This work opens the door to a more principled approach for deciding on which instances the configurations should be evaluated. While more research is needed to determine which specific method to apply in practice, selecting instances during automated algorithm configuration shows great potential.

7

Conclusions

In this thesis, we conducted a detailed investigation of the sampling strategies in the context of general-purpose algorithm configurators for running time optimisation tasks (as defined in Chapter 2).

7.1 The work achieved so far

Before developing our own methods, it was essential to gain a deeper understanding of the current state of the art. We compiled a set of widely used AAC scenarios for running time optimisation and used them to evaluate state-of-the-art algorithm configurators. Current configuration approaches are based on various search algorithms such as racing (Birattari et al., 2010), genetic algorithm (Ansótegui et al., 2009), Bayesian optimisation (Hutter et al., 2011b) and golden search (Pushak and Hoos, 2020). Each of those methods demonstrated its own strengths in a set of scenarios when introduced to the research community. However, we are not aware of any extensive comparison between them for running time scenarios. In Chapters 2 and 3, we addressed this gap (through RQ1, RQ2 and RQ3). We ran a carefully selected set of configurators – ROAR, SMAC2 and 3, irace, GGA++, GPS and paramILS – on a set of 20 scenarios spanning four different NP-hard problems, eight target algorithms and ten datasets of problem instances. Additionally, in Chapter 2, we identified 16 general characteristics of the configuration scenarios, highlighting in the process that the commonly used benchmark AClib lacks variety in some of those characteristics. Specifically, it is

limited to one default configuration, whereas some configurators can handle multiple of them, and it contains mostly non-deterministic algorithms and randomly generated datasets. In Chapter 3, we analysed the performance of the configurators in general and with regard to the scenario characteristics specifically. This allowed us to draw high-level conclusions regarding their strength and weaknesses. Specifically, we found that, despite learning on a discretised search space, paramILS is the best performing configurator on almost half of the considered scenarios. We further described the average ranking of the configurators, showing that SMAC2 has the best average ranking, statistically tied with paramILS, SMAC3 and ROAR. We exposed good complementarity between the top performing configurators using Shapley values and attempted to find which characteristics of a scenario impact the performance of the configurators the most. However, it remains challenging to understand what makes some scenarios more difficult than others or which configurator should be used for specific scenarios.

We observed that in any configurator, there are two key elements to sample: new target algorithm configurations to evaluate and instances on which to run them. In the subsequent chapters of this thesis, we explored whether there are simple, yet effective ways to refine the sampling strategies at those points.

First, we examined the sampling strategies for new configurations. Each configurator has its own approach to generating new configurations based on insights gained during the configuration run. However, most overlook a prior typically given with an algorithm: the default configuration. This configuration is typically set based on the algorithm designers' insight regarding their method, coupled with prior experiments results. In Chapter 4, we hypothesised that the default contains meaningful information. To test this, we first explored the impact and usage of these default parameter values in current configurators (RQ4). We compared the performance of configurators depending on the given default configuration and showed that it has a large impact on irace but a limited one on SMAC. We then proposed approaches to either reduce the search space to keep only configurations close to the given default or sample following a truncated normal distribution centred on the default value (RQ5). We tested these approaches in a state-of-the-art configurator on 20 configuration scenarios, showing that they improve performance for a majority of the studied scenarios.

Second, we examined how configurators decide on which instances to run the next configuration during its evaluation. Most configurators simply select an instance uniformly at random. We hypothesised that two algorithms or two configurations thereof could be accurately and efficiently compared on a smaller set of well-chosen instances. Chapter 5 provided evidence in support of that intuition in the context of the com-

parison of algorithms (RQ6). We evaluated 5 selection methods including random sampling, and two statistical tests to stop the comparison of two algorithms. We showed that, by selecting instances with either a high variance in their running time or a high discrimination power, statistical tests can decide which algorithm is best after collecting algorithm performance data for less than 15% of the CPU time required for a complete comparison. Then, Chapter 6 translated the previously developed methods for the comparison of two configurations of a single algorithm to study how they could help in the context of automated algorithm configuration. First, we designed experiments to evaluate how efficiently two configurations of the same algorithm can be compared and how quickly we can abandon bad configurations using the two best performing methods for instance selection in the context of algorithm comparison (RQ7) and two methods inspired from the active learning literature. We tested the selection methods on randomly generated configurations, and obtained significant speed-ups, from 5 to 3000 folds, which encouraged us towards the next step. We implemented the best-performing selection methods – based on variance and discrimination power – within the state-of-the-art model-based configurator SMAC and evaluated their impact on the configuration process (RQ8). Although there is no clear guideline allowing us to determine which selection method to apply when, we found that including these mechanisms into algorithm configurators could unlock previously unseen performance. In particular for EAX on `rue-1000-3000`, one of the five studied scenarios, our modified version of SMAC3 nearly doubled the improvement of vanilla SMAC3 and performed better than the best performing method from Chapter 3.

7.2 Future work

The continuation of this line of research could involve implementing our methods in other configurators such as paramILS and SMAC2 – the top performing according to Chapter 3 – and evaluating them according to the framework used in Chapter 3 to assess in more detail which kinds of scenarios benefit from using them and which are not. For the latter, identifying the reasons for such a discrepancy could lead to a better understanding of what makes a scenario challenging and how to handle those challenging scenarios more effectively. For example, the instance selection methods from Chapters 5 and 6 would likely have a larger impact on datasets containing problem instances with large variations in running time, whilst the sampling strategies from Chapter 4 would benefit scenarios with a large search space and well-thought default values based on long-standing research.

The work done in Chapter 3 could also be extended to more types of algorithm configuration problems, such as multi-objective configuration or configuration for other performance metrics besides running time optimisation (see *e.g.* Blot et al., 2016). It also opens the door to the development of a configurator selector, which would predict for a specific scenario which configurator should be applied based on its characteristics. However, for this last point, we conducted preliminary experiments to learn a multi-class random forest model on the data collected in Chapter 3 and did not reach any promising results. We might need more expressive characteristics than those listed in Chapter 2 or design scenarios with more variations in those characteristics to allow the model to learn more from them.

The work conducted in Chapter 5 opens up ways to make the development and evaluation of solvers for hard problems more sustainable. For example, instance selection methods would allow a developer to receive quick feedback on new ideas without running their solver on too many problem instances. These methods could also reduce the amount of computation needed to declare the winners of a competition by only running the algorithms on more instances if there is a probability for them to land in the top. Since the less performing algorithms are also the ones consuming the most resources, this would have a large impact on the required computing time, *e.g.* in 2020 the top 10 algorithms of the SAT competition ran for about a tenth of the time required for the 49 others. Compared to the 2-round design used in the early instalment of the competition (see *e.g.* Simon et al., 2005), the choice of benchmarks would be based on statistics rather than be hand-picked by the organisers.

Following up on our work regarding instance selection in Chapter 6, similar mechanisms could be designed in a machine learning context to focus the learning effort on relevant data rather than processing as much data as possible. The challenge here would be to decide what is applicable in that context. Further work is already ongoing to apply similar techniques for automated algorithm selection (Kuş et al., 2024). The key difference between our methods and the ones found in active learning is that the active learning community often assumes that acquiring a data point has a fixed cost which is the same for each data point. On the other hand, we account for the price of each point on top of how informative they are. This problem has also been studied with cost-aware active learning (see *e.g.* Tomanek and Hahn, 2010; Guillory and Bilmes, 2009) and more work can be done at the intersection of our fields.

Another element that we did not explore is the target algorithm itself. It could be interesting, following our work on algorithm selection (Pulatov et al., 2022), to define algorithm features for algorithm configuration that would inform the model used by

configurator about the meaning and impact of a specific parameter value on the inner working of the target algorithm. The challenge in this would be to describe features that represent the different configurations and are dependent on the configuration. Such feature could for example represent the amount and complexity of code activated by a Boolean parameter. There might also be a way, through source code inspection (such as *e.g.* program slicing Gallagher and Kozaitis, 2025), to find out relationships between parameters or predict how much they impact the performance. However, the differences in programming languages and code style might have a significant impact on this kind of research, for example, the syntax trees we built for the features analysed in our prior work (Pulatov et al., 2022) were largely different between the solvers in C and the ones in Java.

7.3 Final word

In this thesis, we addressed key challenges in the design of general-purpose automated algorithm configurators for running time optimisation, focusing on their sampling strategies. We provided a deeper understanding of the strength and weaknesses of current configurators through empirical analysis and developed methods to improve them. Our contributions show the potential of simple refinement to substantially improve the state of the art in AAC.

This work paves the way towards more flexibility in algorithm configuration, where the right approach would be applied to each configuration scenario. The methods we developed, in particular the ones related to instance selection, also offer the opportunity to be applied to other domains with costly evaluations and a focus on sustainability.

Our hope is that this thesis encourages further research into the underlying question: *How to design flexible systems that adapt to the problem at hand and learn more from less?*

Bibliography

Adriaensen, S. and Nowé, A. (2016). Towards a white box approach to automated algorithm design. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI*, pages 554–560. IJCAI/AAAI Press.

Ahmadizadeh, K., Dilkina, B., Gomes, C. P., and Sabharwal, A. (2010). An empirical study of optimization for maximizing diffusion in networks. In *Proceedings of Principles and Practice of Constraint Programming, CP*, volume 6308 of *Lecture Notes in Computer Science*, pages 514–521. Springer.

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the Association for Computing Machinery ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD*, pages 2623–2631. Association for Computing Machinery ACM.

Anastacio, M. (2021). Greybox algorithm configuration. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI*, pages 4875–4876.

Anastacio, M. and Hoos, H. H. (2020a). Combining sequential model-based algorithm configuration with default-guided probabilistic sampling. In *GECCO 2020: Genetic and Evolutionary Computation Conference 2020, Companion Volume*, pages 301–302. ACM.

Anastacio, M. and Hoos, H. H. (2020b). Model-based algorithm configuration with default-guided probabilistic sampling. In *Proceedings of Parallel Problem Solving from Nature - PPSN XVI, Part I*, volume 12269 of *Lecture Notes in Computer Science*, pages 95–110. Springer.

Anastacio, M., Luo, C., and Hoos, H. (2019). Exploitation of default parameter values in automated algorithm configuration. In *Workshop Data Science meets Optimisation, DSO, in conjunction with IJCAI*.

Anastacio, M., Matricon, T., and Hoos, H. H. (2022). Instance selection for configuration performance comparison. In *Meta-knowledge transfer workshop, in conjunction with ECML-PKDD*.

Ansótegui, C., Malitsky, Y., Samulowitz, H., Sellmann, M., and Tierney, K. (2015). Model-based genetic algorithms for algorithm configuration. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI*, pages 733–739. AAAI Press.

Ansótegui, C., Sellmann, M., and Tierney, K. (2009). A gender-based genetic algorithm for the automatic configuration of algorithms. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming, CP*, volume 5732 of *Lecture Notes in Computer Science*, pages 142–157. Springer.

Atamtürk, A. and Muñoz, J. C. (2004). A study of the lot-sizing polytope. *Mathematical Programming*, 99(3):443–465.

Ayerdi, B. and Graña, M. (2015). Random forest active learning for retinal image segmentation. In *Proceedings of the International Conference on Computer Recognition Systems, CORES*, volume 403 of *Advances in Intelligent Systems and Computing*, pages 213–221. Springer.

Balcan, M.-F., Sandholm, T., and Vitercik, E. (2018). A general theory of sample complexity for multi-item profit maximization. In *Proceedings of the ACM Conference on Economics and Computation*, pages 173–174.

Balint, A. and Manthey, N. (2014). Sparrowtoriss. In *Proceedings of SAT Competition 2014*, volume B-2014-2 of *Department of Computer Science Series of Publications B*, page 77. University of Helsinki, Helsinki, Finland.

Balyo, T., Froleyks, N., Heule, M., Iser, M., Järvisalo, M., and Suda, M., editors (2020). *Proceedings of SAT Competition 2020: Solver and Benchmark Descriptions*. Department of Computer Science Report Series B. Department of Computer Science, University of Helsinki.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(1):281–305.

Bhosle, N. P. and Kokare, M. (2020). Random forest-based active learning for content-based image retrieval. *International Journal of Intelligent Information and Database Systems*, 13(1):72–88.

Biere, A. (2014). Yet another local search solver and lingeling and friends entering the sat competition 2014. In *Proceedings of SAT Competition 2014: Solver and Benchmark Descriptions*, volume B-2014-2 of *Department of Computer Science Series of Publications B*. University of Helsinki 2014.

Biere, A., Cimatti, A., Clarke, E. M., and Zhu, Y. (1999). Symbolic model checking without bdds. In *Proceedings of Tools and Algorithms for Construction and Analysis of Systems, 5th International Conference, TACAS '99, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS*, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207. Springer.

Biere, A., Fazekas, K., Fleury, M., and Heisinger, M. (2020). CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In *Proceedings of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53. University of Helsinki.

Birattari, M. (2009). *Tuning Metaheuristics - A Machine Learning Perspective*, volume 197 of *Studies in Computational Intelligence*. Springer.

Birattari, M., Stützle, T., Paquete, L., and Varrentrapp, K. (2002). A racing algorithm for configuring metaheuristics. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference 2002*, pages 11–18. Morgan Kaufmann.

Birattari, M., Yuan, Z., Balaprakash, P., and Stützle, T. (2010). F-Race and iterated F-Race: An overview. In *Experimental Methods for the Analysis of Optimization Algorithms*, pages 311–336. Springer.

Bischl, B., Kerschke, P., Kotthoff, L., Lindauer, M., Malitsky, Y., Fréchette, A., Hoos, H. H., Hutter, F., Leyton-Brown, K., Tierney, K., and Vanschoren, J. (2016). Aslib: A benchmark library for algorithm selection. *Artificial Intelligence*, 237:41–58.

Blot, A., Hoos, H. H., Jourdan, L., Kessaci-Marmion, M., and Trautmann, H. (2016). Mo-paramils: A multi-objective automatic algorithm configuration framework. In *Learning and Intelligent Optimization - 10th International Conference, LION 10, Revised Selected Papers*, volume 10079 of *Lecture Notes in Computer Science*, pages 32–47. Springer.

Boddy, M. S., Gohde, J., Haigh, T., and Harp, S. A. (2005). Course of action generation for cyber security using classical planning. In *Proceedings of the International Conference on Automated Planning and Scheduling, ICAPS*, pages 12–21. AAAI.

Bossek, J. and Wagner, M. (2021). Generating instances with performance differences for more than just two algorithms. In *GECCO 2021: Genetic and Evolutionary Computation Conference 2021, Companion Volume*, page 1423–1432. Association for Computing Machinery ACM.

Brown, C. E. (2013). Reducing higher-order theorem proving to a sequence of SAT problems. *Journal of Automated Reasoning*, 51(1):57–77.

Brummayer, R., Lonsing, F., and Biere, A. (2010). Automated testing and debugging of SAT and QBF solvers. In *Proceedings of Theory and Applications of Satisfiability Testing, SAT*, volume 6175 of *Lecture Notes in Computer Science*, pages 44–57. Springer.

Burch, J. R., Clarke, E. M., Long, D. E., McMillan, K. L., and Dill, D. L. (1994). Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(4):401–424.

Cáceres, L. P., López-Ibánez, M., Hoos, H., and Stützle, T. (2017). An experimental study of adaptive capping in irace. In *Proceedings of the International Conference on Learning and Intelligent Optimization, LION*, volume 10556 of *Lecture Notes in Computer Science*, pages 235–250. Springer.

Comaniciu, D. and Meer, P. (2002). Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence TPAMI*, 24(5):603–619.

Conover, W. (1998). *Practical nonparametric statistics*, volume 350 of *Wiley series in probability and statistics*. John Wiley & Sons.

Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Proceedings of the Annual Association for Computing Machinery ACM Symposium on Theory of Computing,*, pages 151–158. Association for Computing Machinery ACM.

Davis, M. and Putnam, H. (1960). A computing procedure for quantification theory. *Journal of the Association for Computing Machinery ACM*, 7(3):201–215.

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

Domhan, T., Springenberg, J. T., and Hutter, F. (2015). Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI*, pages 3460–3468. AAAI Press.

Dua, D. and Karra Taniskidou, E. (2017). UCI machine learning repository.

Dunn, O. J. (1961). Multiple comparisons among means. *Journal of the American Statistical Association*, 56(293):52–64.

Eggensperger, K., Lindauer, M., and Hutter, F. (2019). Pitfalls and best practices in algorithm configuration. *Journal of Artificial Intelligence Research JAIR*, 64(1):861–893.

Falkner, S., Klein, A., and Hutter, F. (2018). BOHB: robust and efficient hyperparameter optimization at scale. In *Proceedings of the International Conference on Machine Learning, ICML 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1436–1445. PMLR.

Fawcett, C. and Hoos, H. H. (2016). Analysing differences between algorithm configurations through ablation. *Journal of Heuristics*, 22(4):431–458.

Fokkinga, D., Latour, A. L. D., Anastacio, M., Nijssen, S., and Hoos, H. (2019). Programming a stochastic constraint optimisation algorithm, by optimisation. In *Workshop Data Science meets Optimisation, DSO, in conjunction with IJCAI*.

Franzin, A., Cáceres, L. P., and Stützle, T. (2018). Effect of transformations of numerical parameters in automatic algorithm configuration. *Optimization Letters*, 12(8):1741–1753.

Fréchette, A., Kotthoff, L., Michalak, T., Rahwan, T., Hoos, H., and Leyton-Brown, K. (2016). Using the shapley value to analyze algorithm portfolios. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1).

Gallagher, K. B. and Kozaitis, S. J. (2025). Program slicing: A brief retrospective. *IEEE Transactions on Software Engineering*, 51(3):720–724.

Gebser, M., Kaufmann, B., and Schaub, T. (2012). Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence*, 187:52–89.

Gent, I. P., Hussain, B. S., Jefferson, C., Kotthoff, L., Miguel, I., Nightingale, G. F., and Nightingale, P. (2014). Discriminating instance generation for automated constraint model selection. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming, CP*, volume 8656 of *Lecture Notes in Computer Science*, pages 356–365. Springer.

Gerevini, A., Saetti, A., and Serina, I. (2003). Planning through stochastic local search and temporal action graphs in LPG. *Journal of Artificial Intelligence Research*, 20:239–290.

Gerevini, A., Saetti, A., and Serina, I. (2008). An approach to efficient planning with numerical fluents and multi-criteria plan quality. *Artificial Intelligence*, 172(8-9):899–944.

Gerevini, A., Saetti, A., and Serina, I. (2011). An empirical analysis of some heuristic features for planning through local search and action graphs. *Fundamenta Informaticae*, 107(2-3):167–197.

Gerevini, A. and Serina, I. (2002). Lpg: a planner based on local search for planning graphs with action costs. In *Proceedings of the International Conference on Artificial Intelligence Planning Systems AIPS*, AIPS, page 13–22. AAAI Press.

Gerlach, B., Anastacio, M., and Hoos, H. H. (2025). On the efficiency of training robust decision trees. In *Poster at the Symposium on AI Verification SAIV, adjunct to the International Conference on Computer-Aided Verification CAV*.

Gomes, C., Selman, B., Crato, N., and Kautz, H. (2000). Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24(1/2):67–100.

Gomes, C. P., van Hoeve, W. J., and Sabharwal, A. (2008). Connections in networks: A hybrid approach. In *Proceedings if Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 5th International Conference, CPAIOR*, volume 5015 of *Lecture Notes in Computer Science*, pages 303–307. Springer.

Gu, Y., Zydek, D., and Jin, Z. (2014). Active learning based on random forest and its application to terrain classification. In *Progress in Systems Engineering - Proceedings of the International Conference on Systems Engineering, ICSEng*, volume 366 of *Advances in Intelligent Systems and Computing*, pages 273–278. Springer.

Guillory, A. and Bilmes, J. A. (2009). Average-case active learning with costs. In Gavaldà, R., Lugosi, G., Zeugmann, T., and Zilles, S., editors, *Proceedings of the International Conference on Algorithmic Learning Theory, ALT*, volume 5809 of *Lecture Notes in Computer Science*, pages 141–155. Springer.

Hauschild, M. and Pelikan, M. (2011). An introduction and survey of estimation of distribution algorithms. *Swarm and Evolutionary Computation*, 1(3):111–128.

Heins, J., Schäpermeier, L., Kerschke, P., and Whitley, D. (2024). Dancing to the state of the art? how candidate lists influence lkh for solving the traveling salesperson problem. In *Proceedings of Parallel Problem Solving from Nature, PPSN XVIII, Part I*, page 100–115. Springer-Verlag.

Helsgaun, K. (2000). An effective implementation of the lin-kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130.

Heule, M., Iser, M., Järvisalo, M., and Suda, M., editors (2024). *Proceedings of SAT Competition 2024: Solver, Benchmark and Proof Checker Descriptions*. Department of Computer Science Report Series B. Department of Computer Science, University of Helsinki.

Heule, M., Järvisalo, M., and Suda, M. (2019). Sat competition 2018. *Journal on Satisfiability, Boolean Modeling and Computation*, 11:133–154.

Hoos, H. H. (2012a). Automated algorithm configuration and parameter tuning. In *Autonomous Search*, pages 37–71. Springer.

Hoos, H. H. (2012b). Programming by optimization. *Communication of the Association for Computing Machinery ACM*, 55(2):70–80.

Hurley, B. and O'Sullivan, B. (2015). Statistical regimes and runtime prediction. In *Proceedings of the International Conference on Artificial Intelligence, IJCAI*, page 318–324. AAAI Press.

Hutter, F., Hoos, H., and Leyton-brown, K. (2011a). Bayesian optimization with censored response data. In *workshop on Bayesian Optimization, Sequential Experimental Design, and Bandits, in conjunction with NIPS*.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2010a). Automated configuration of mixed integer programming solvers. In *Proceedings of Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 7th International Conference, CPAIOR*, volume 6140 of *Lecture Notes in Computer Science*, pages 186–202. Springer.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2010b). Automated configuration of mixed integer programming solvers. In *Proceedings of Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 7th International Conference, CPAIOR 2010*, volume 6140 of *Lecture Notes in Computer Science*, pages 186–202. Springer.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011b). Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization - 5th International Conference, LION 5, Selected Papers*, volume 6683 of *Lecture Notes in Computer Science*, pages 507–523. Springer.

Hutter, F., Hoos, H. H., Leyton-Brown, K., and Stützle, T. (2009). ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research JAIR*, 36:267–306.

Hutter, F., Hoos, H. H., and Stützle, T. (2007). Automatic algorithm configuration based on local search. In *Proceedings of the AAAI Conference on Artificial Intelligence,*, pages 1152–1157. AAAI Press.

Hutter, F., Kotthoff, L., and Vanschoren, J., editors (2019). *Automated Machine Learning - Methods, Systems, Challenges*. The Springer Series on Challenges in Machine Learning. Springer.

Hutter, F., Lindauer, M., Balint, A., Bayless, S., Hoos, H. H., and Leyton-Brown, K. (2017). The configurable SAT solver challenge (CSSC). *Artificial Intelligence*, 243:1–25.

Hutter, F., López-Ibáñez, M., Fawcett, C., Lindauer, M., Hoos, H. H., Leyton-Brown, K., and Stützle, T. (2014a). Aclib: A benchmark library for algorithm configuration. In *Learning and Intelligent Optimization - 8th International Conference, Lion 8, Revised Selected Papers*, volume 8426 of *Lecture Notes in Computer Science*, pages 36–40. Springer.

Hutter, F., Xu, L., Hoos, H. H., and Leyton-Brown, K. (2014b). Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111.

Hvarfner, C., Stoll, D., Souza, A. L. F., Lindauer, M., Hutter, F., and Nardi, L. (2022). πBO: Augmenting acquisition functions with user beliefs for bayesian optimization. In *Proceedings of the International Conference on Learning Representations ICLR*. OpenReview.net.

Jamieson, K. and Talwalkar, A. (2016). Non-stochastic best arm identification and hyperparameter optimization. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 240–248. Proceedings of Machine Learning Research PMLR.

Kalkreuth, R., de França, F. O., Dierkes, J., Anastacio, M., Jankovic, A., Vasicek, Z., and Hoos, H. (2025). Tinyversegp: Towards a modular cross-domain benchmarking framework for genetic programming. In *GECCO 2025: Genetic and Evolutionary Computation Conference, Companion Volume*. ACM.

Kerschke, P., Hoos, H. H., Neumann, F., and Trautmann, H. (2019). Automated algorithm selection: Survey and perspectives. *Evolutionary Computation*, 27(1):3–45.

Kiefer, J. (1953). Sequential minimax search for a maximum. *Proceedings of the American mathematical society*, 4(3):502–506.

König, M., Hoos, H. H., and van Rijn, J. N. (2021). Speeding up neural network verification via automated algorithm configuration. In *Workshop on Security and Safety in Machine Learning Systems, in conjunction with ICLR*.

Kotthoff, L., Thornton, C., Hoos, H. H., Hutter, F., and Leyton-Brown, K. (2017). Auto-weka 2.0: Automatic model selection and hyperparameter optimization in WEKA. *Journal of Machine Learning Research*, 18:25:1–25:5.

Kuş, E., Akgün, O., Dang, N., and Miguel, I. (2024). Frugal Algorithm Selection. In Shaw, P., editor, *30th International Conference on Principles and Practice of Constraint Programming, CP*, volume 307 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 38:1–38:16, Dagstuhl, Germany. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Latour, A. L., Babaki, B., Fokkinga, D., Anastacio, M., Hoos, H. H., and Nijssen, S. (2022a). Exact stochastic constraint optimisation with applications in network analysis. *Artificial Intelligence*, 304:103650.

Latour, A. L. D., Babaki, B., Fokkinga, D., Anastacio, M., Hoos, H. H., and Nijssen, S. (2020). Stochastic constraint optimisation with applications in network analysis (extended abstract). In *International Workshop on Model Counting, MCW, in conjunction with SAT*.

Latour, A. L. D., Babaki, B., Fokkinga, D., Anastacio, M., Hoos, H. H., and Nijssen, S. (2022b). Stochastic constraint optimisation with applications in network analysis (extended abstract). In *Workshop on Counting and Sampling 2022, in conjunction with FLoC 2022 and SAT 2022*.

Lawson, C. L. and Hanson, R. J. (1995). *Solving least squares problems*, volume 15 of *Classics in applied mathematics*. Society for Industrial and Applied Mathematics SIAM.

Leyton-Brown, K., Pearson, M., and Shoham, Y. (2000). Towards a universal test suite for combinatorial auction algorithms. In *Proceedings of the Association for Computing Machinery ACM Conference on Electronic Commerce*, EC, page 66–76. Association for Computing Machinery ACM.

Li, H., Liang, Q., Chen, M., Dai, Z., Li, H., and Zhu, M. (2022). Pruning smac search space based on key hyperparameters. *Concurrency and Computation: Practice and Experience*, 34(9):e5805.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2018). Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52.

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

Lindauer, M., Hoos, H. H., Hutter, F., and Schaub, T. (2015). Autofolio: An automatically configured algorithm selector. *Journal of Artificial Intelligence Research JAIR*, 53:745–778.

Lindauer, M. and Hutter, F. (2018). Warmstarting of model-based algorithm configuration. In *Proceedings of the AAAI Conference on Artificial Intelligence, AAAI, the innovative Applications of Artificial Intelligence IAAI, and the AAAI Symposium on Educational Advances in Artificial Intelligence EAAI*, pages 1355–1362. AAAI Press.

Lindauer, M., van Rijn, J. N., and Kotthoff, L. (2017). Open algorithm selection challenge 2017: Setup and scenarios. In *Proceedings of the Open Algorithm Selection Challenge*, volume 79. Proceedings of Machine Learning Research PMLR.

Lis, J. and Eiben, A. E. (1997). A multi-sexual genetic algorithm for multiobjective optimization. In *Proceedings of 1997 IEEE International Conference on Evolutionary Computation, ICEC 1997*, pages 59–64. IEEE.

Long, D. and Fox, M. (2003). The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research JAIR*, 20:1–59.

Lourenço, H. R., Martin, O. C., and Stützle, T. (2003). *Iterated Local Search*, pages 320–353. Springer US.

Luo, C., Hoos, H. H., Cai, S., Lin, Q., Zhang, H., and Zhang, D. (2019). Local search with efficient automatic configuration for minimum vertex cover. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI*, pages 1297–1304. International Joint Conferences on Artificial Intelligence Organization.

López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., and Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58.

Malone, B., Kangas, K., Järvisalo, M., Koivisto, M., and Myllymäki, P. (2018). Empirical hardness of finding optimal bayesian network structures: algorithm selection and runtime prediction. *Machine Learning*, 107:247–283.

Maron, O. and Moore, A. W. (1997). The racing algorithm: Model selection for lazy learners. *Artificial Intelligence Review*, 11(1-5):193–225.

Matai, R., Singh, S. P., and Mittal, M. L. (2010). Traveling salesman problem: an overview of applications, formulations, and solution approaches. *Traveling salesman problem, theory and applications*, 1(1):1–25.

Matricon, T., Anastacio, M., Fijalkow, N., Simon, L., and Hoos, H. H. (2021). Statistical comparison of algorithm performance through instance selection. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming, CP*, volume 210 of *LIPIcs*, pages 43:1–43:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Megiddo, N. (1983). Linear-time algorithms for linear programming in r^3 and related problems. *SIAM Journal on Computing*, 12(4):759–776.

Mersmann, O., Bischl, B., Trautmann, H., Wagner, M., Bossek, J., and Neumann, F. (2013). A novel feature-based approach to characterize algorithm performance for the traveling salesperson problem. *Annals of Mathematics and Artificial Intelligence*, 69(2):151–182.

Minton, S. (1993). An analytic learning system for specializing heuristics. In *Proceedings of the international joint conference on Artifical intelligence , IJCAI 1993*, volume 2 of *IJCAI*, pages 922–928, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Mockus, J. (1989). *Bayesian Approach to Global Optimization: Theory and Applications*, volume 37 of *Mathematics and Its Applications*. Springer Netherlands.

Moeini, E., Vox, C., Anastacio, M., Skaf, W., Barachi, M., and Hoos, H. H. (2026). Neural architecture and hyperparameter selection through meta-learning on time series. In *Proceedings of the AAAI Conference on Artificial Intelligence*. AAAI Press.

Mugrauer, F. and Balin, A. (2013). Sat encoded low autocorrelation binary sequence (labs) benchmark description. In *Proceedings of SAT Competition 2013: Solver and Benchmark Descriptions*, volume B-2013-1 of *Department of Computer Science Series of Publications B*, page 117–118. University of Helsinki.

Nagata, Y. and Kobayashi, S. (2013). A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. *INFORMS Journal on Computing*, 25(2):346–363.

Nannen, V. and Eiben, A. E. (2007). Relevance estimation and value calibration of evolutionary algorithm parameters. In *Proceedings of the International Joint Conference on Artifical Intelligence, IJCAI*, page 975–980. Morgan Kaufmann Publishers Inc.

Nejati, S. and Ganesh, V. (2019). Cdcl(crypto) SAT solvers for cryptanalysis. In *Proceedings of the Annual International Conference on Computer Science and Software Engineering, CASCON*, pages 311–316. Association for Computing Machinery ACM.

Packwood, D. et al. (2017). *Bayesian optimization for materials science*, volume 3. Springer.

Penberthy, J. S. and Weld, D. S. (1994). Temporal planning with continuous change. In *Proceedings of the 12th National Conference on Artificial Intelligence*, volume 2, pages 1010–1015. AAAI Press / The MIT Press.

Pratt, J. W. (1959). Remarks on zeros and ties in the wilcoxon signed rank procedures. *Journal of the American Statistical Association*, 54:655–667.

Pulatov, D., Anastacio, M., Kotthoff, L., and Hoos, H. H. (2022). Opening the black box: Automated software analysis for algorithm selection. In *International Conference on Automated Machine Learning, AutoML*, volume 188 of *Proceedings of Machine Learning Research*, pages 6/1–18. Proceedings of Machine Learning Research PMLR.

Pulina, L. and Seidl, M. (2019). The 2016 and 2017 qbf solvers evaluations (qbfeval'16 and qbfeval'17). *Artificial Intelligence*, 274:224–248.

Purucker, L. O., Schneider, L., Anastacio, M., Beel, J., Bischl, B., and Hoos, H. H. (2023). Q(D)O-ES: population-based quality (diversity) optimisation for post hoc ensemble selection in automl. In *International Conference on Automated Machine Learning, AutoML*, volume 224 of *Proceedings of Machine Learning Research*, pages 10/1–34. Proceedings of Machine Learning Research PMLR.

Pushak, Y. and Hoos, H. H. (2018). Algorithm configuration landscapes: - more benign than expected? In *Proceedings of Parallel Problem Solving from Nature - PPSN XV , Part II*, volume 11102 of *Lecture Notes in Computer Science*, pages 271–283. Springer.

Pushak, Y. and Hoos, H. H. (2020). Golden parameter search: exploiting structure to quickly configure parameters in parallel. In *GECCO 2020: Genetic and Evolutionary Computation Conference 2020*, pages 245–253. ACM.

Ramos-Gutiérrez, B., Varela-Vaca, A. J., Galindo, J. A., Gómez-López, M. T., and Benavides, D. (2021). Discovering configuration workflows from existing logs using process mining. *Empirical Software Engineering*, 26(1).

Rogers, J., Anastacio, M., Bernard, J., Chakhchoukh, M., Faust, R., Kerren, A., Koch, S., Kotthoff, L., Turkay, C., and Wall, E. (2024). Visualization and automation in data science: Exploring the paradox of humans-in-the-loop. In *Workshop on Visualization in Data Science VDS, in conjunction with IEEE Visualization and Visual Analytics Conference VIS*.

Schmee, J. and Hahn, G. J. (1979). A simple method for regression analysis with censored data. *Technometrics*, 21(4):417–432.

Schneider, L., Bischl, B., and Feurer, M. (2025). Overtuning in hyperparameter optimization. In *AutoML 2025 Methods Track*.

Siegel, S. and Castellan Jr, N. J. (1988). *Nonparametric statistics for the behavioral sciences*. Mcgraw-Hill Book Company.

Simon, L., Leberre, D., and Hirsch, E. A. (2005). The SAT 2002 Competition. *Annals of Mathematics and Artificial Intelligence*, vol.43, Issue 1-4:307–342.

Souza, A. L. F., Nardi, L., Oliveira, L. B., Olukotun, K., Lindauer, M., and Hutter, F. (2021). Bayesian optimization with a prior for the optimum. In *Proceedings of Machine Learning and Knowledge Discovery in Databases. Research Track - European Conference, ECML PKDD 2021, Part III*, volume 12977 of *Lecture Notes in Computer Science*, pages 265–296. Springer.

Stuckey, P., Feydy, T., Schutt, A., Tack, G., and Fischer, J. (2014). The minizinc challenge 2008-2013. *AI Magazine*, 35(2):55–60.

Sun, L.-L. and Wang, X.-Z. (2010). A survey on active learning strategy. In *2010 International Conference on Machine Learning and Cybernetics*, volume 1, pages 161–166.

Sutcliffe, G. (2020). Proceedings of the international joint conference on automated reasoning IJCAR automated theorem proving system competition CASC-J 10). *AI Communications*, 34:163–177.

Taitler, A., Alford, R., Espasa, J., Behnke, G., Fiser, D., Gimelfarb, M., Pommerening, F., Sanner, S., Scala, E., Schreiber, D., Segovia-Aguas, J., and Seipp, J. (2024). The 2023 international planning competition. *AI Magazine*, 45(2):280–296.

Thornton, C., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2013). Auto-weka: combined selection and hyperparameter optimization of classification algorithms. In *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013*, pages 847–855. ACM.

Tomanek, K. and Hahn, U. (2010). A comparison of models for cost-sensitive active learning. In Huang, C. and Jurafsky, D., editors, *Proceedings of the International Conference on Computational Linguistics, COLING, Posters Volume,*, pages 1247–1255. Chinese Information Processing Society of China.

Tornede, T., Tornede, A., Hanselle, J., Mohr, F., Wever, M., and Hüllermeier, E. (2023). Towards green automated machine learning: Status quo and future directions. *Journal of Artificial Intelligence Research JAIR*, 77:427–457.

Vallati, M., Fawcett, C., Gerevini, A., Hoos, H. H., and Saetti, A. (2013). Automatic generation of efficient domain-optimized planners from generic parametrized planners. In *Proceedings of the Annual Symposium on Combinatorial Search, SOCS 2013*, pages 184–192. AAAI Press.

Verbert, K., Schutter, B. D., and Babuska, R. (2017). Timely condition-based maintenance planning for multi-component systems. *Reliability Engineering & System Safety*, 159:310–321.

Wahab, H., Jain, V., Tyrrell, A. S., Seas, M. A., Kotthoff, L., and Johnson, P. A. (2020). Machine-learning-assisted fabrication: Bayesian optimization of laser-induced graphene patterning using in-situ raman analysis. *Carbon*, 167:609–619.

Weber, T., Conchon, S., Déharbe, D., Heizmann, M., Niemetz, A., and Reger, G. (2019). The SMT competition 2015-2018. *Journal on Satisfiability, Boolean Modeling and Computation*, 11(1):221–259.

Wistuba, M., Schilling, N., and Schmidt-Thieme, L. (2015). Hyperparameter search space pruning – a new component for sequential model-based hyperparameter optimization. In *Machine Learning and Knowledge Discovery in Databases*, pages 104–119. Springer International Publishing.

Xu, L., Hutter, F., Hoos, H., and Leyton-Brown, K. (2012). Evaluating component solver contributions to portfolio-based algorithm selectors. In *Proceedings of Theory and Applications of Satisfiability Testing, SAT*, pages 228–241. Springer Berlin Heidelberg.

Xu, L., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2008). SATzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research JAIR*, 32:565–606.

Xu, L., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). Hydra-MIP: Automated algorithm configuration and selection for mixed integer programming. In *Proceedings of the Knowledge Representation and Automated Reasoning RCRA Workshop, in conjunction with IJCAI*, pages 16–30.

Acknowledgements

Between the time I began working on the content of this thesis and the moment I write these lines, almost 9 years have passed. This has been in the work for about a fourth of my entire life, and many people passed through this life during those years.

Let me start with those who had to support me daily, my partner *Jérémie* and my daughter *Sedna*. In French, "supporter" someone also means "to endure" or "to tolerate" someone. And how much had they to endure! We went together through several weddings – ours included – and the birth of the young *Orion*, who brightened the last years of the journey. On the other hand, there has been a global pandemic and the mourning of several loved ones, parents and grand-parents. As I talk about family, I should mention my parents, brothers, and sister. The former, I am sure, had doubts about my choice in the beginning, but kept those to themselves. My older brother genuinely tried to understand what I was working on. Thank you for that.

I know that one is supposed to start with their supervisors, but I am convinced *Holger* will not mind. He has always supported me in my decisions to keep my family first. Thank you for caring about making a researcher out of me, encouraging me to explore my own ideas and holding me up to high standards. Thank you for your expectations and your trust.

At Leiden University, I met people who also shaped this work. The first friendly face was my office mate, *Chuan*. He's been guiding me like an older brother in my first year in the world of research, whilst I felt like the older sister when talking about Europe customs and work habits. During the first week, I also met *Koen* who quickly became a friend and a mentor to me. He was part of another research group, nearly done with his own PhD, and through him I met the rest of those who accompanied my everyday life for 3 years: *Thodoris* and *Furong*, the office-mates constantly teasing each

other; *Sander* and *Lieuwe*, the locals who knew each nook and cranny of Leiden. Thank you for the drinks and evenings, for the endless discussions, laughs and friendship. I also attempted to make our work environment more inclusive with *Anna*, *Daniela* and *Anna-Lena*. Without *Anna* in particular, I would not be the person I am now. Thank you for supporting me, disagreeing with me, fighting alongside me.

I learnt about the cold analytical side of research, but also about the soft skills and social understanding required to be a full fledged researcher. *Lars* reminded me that the most important was not to work at a highly-ranked university or get hundreds of citations, but to feel that I am at the right place with the right people. *Jan*, always open to working with me though we still have not written a single paper together, helped me more than he knows. In times of doubt, his appreciation of my work and ideas lifted me up. *Laurent* allowed me to appreciate serendipity as part of research and provocative statements as starters for novel ideas. I admit that the latter still frightens me. In the later years of my PhD, *Roman* impressed upon me that the responsibility of a researcher also lies in making sure that everyone in their research environment feels at ease. I had been advocating for it for several years already, but his earnestness in applying it led me to consider how I was interacting with students and supervisees myself. *Tom* taught me to stop worrying about everything and enjoy the moment.

Outside of family and work, more friends are gravitating. The friends with whom I made crazy plans often aborted to travel the world — *Moalejon*, I think we both know the Trans-Siberian train trip is not gonna happen anytime soon — and those with whom I spent festivals and holidays were an important anchor in my mental health and happiness. The FIMU was the only weekend I spent without my work computer. Online gaming evenings during the pandemic held me together in trying times. *Mowgly*, *Mouman*, *Kaal*, *Couette*, *Six*, and I know I am forgetting people; we always do. You were like bubbles of warmth and emotional security in a path of hardship.

From those who have been listening to my rambling to those who made my holidays lighter: you've all been important in your own way. Thank you.

Curriculum Vitae

Marie Anastacio was born in France on the 10th of September 1989. She grew up near Bourges, were she went to classes préparatoires in maths and physics. She moved to Belfort (France) to acquire a Computer science engineering degree from the Université Technologique de Belfort-Montbéliard in 2012. She concluded her engineering studies as an exchange student in Trois-Rivières (Canada), which extended into an internship followed by a master degree in Applied Mathematics and Computer Science at the Université du Québec à Trois-Rivières obtained in 2015. Marie came back to Europe in 2015 with a fiancé and a child to be born. In 2017, she started her PhD studies under the supervision of Professor Holger H. Hoos in the Leiden Institute of Computer Science at Leiden University (the Netherlands), the most tangible result of which you can read in the present thesis. Since the goal is the journey, Marie got involved in the community by starting monthly seminars to share research among PhD candidates, sitting on the diversity committee of the research institute and co-founding the CAIRNE Rising Researcher Network – a European network of early career researchers in artificial intelligence. She also got involved in outreach towards students at various stages of their studies, from talking about research in a primary school, to giving a first taste of research to high-school students in the Hague. During her PhD studies, she took courses in time management, scientific writing, effective communication and scientific conduct, among others.

Marie now works as a researcher at RWTH Aachen university in Germany, where she explores further her research interests in optimisation and their application to problems she cares about, such as trustworthiness of AI and sustainability.