UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

MARCELO DE SOUZA

Automatic Algorithm Configuration: Methods and Applications

Thesis presented in partial fulfillment of the requirements for the degree of Doctor of Computer Science

Advisor: Prof. Dr. Marcus Rolf Peter Ritt

Porto Alegre
February 2022

CIP — CATALOGING-IN-PUBLICATION

Souza, Marcelo de

Automatic Algorithm Configuration: Methods and Applications / Marcelo de Souza. – Porto Alegre: PPGC da UFRGS, 2022.

203 f.: il.

Thesis (Ph.D.) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2022. Advisor: Marcus Rolf Peter Ritt.

1. Automatic algorithm configuration. 2. Automatic algorithm design. 3. Parameter tuning. 4. Capping methods. 5. Parameter regression models. 6. Unconstrained binary quadratic programming. I. Ritt, Marcus Rolf Peter. II. Título.

À minha amada família. Em especial, à minha esposa Taize, aos meus pais, Renato e Marlene, e à minha avó Clementina (in memoriam).

AGRADECIMENTOS

Apesar de desafiador e por vezes exaustivo, o doutorado tem sido uma incrível jornada de descobertas, aprendizado, crescimento e, por que não dizer, de alegria e diversão. Se assim foi, dou graças primeira e especialmente a Deus que, por intercessão de Maria, sempre cuidou de mim e da minha família, e guiou cada passo dessa caminhada. De maneira singular, agradeço por me abençoar com pessoas extraordinárias, que não somente contribuíram para o desenvolvimento dessa pesquisa, mas tornaram essa experiência muito mais agradável e especial. A essas pessoas, dedico aqui algumas palavras de gratidão e carinho.

Minha formação como pesquisador deve muito ao professor Marcus Ritt, que sempre acreditou na minha capacidade técnica e intelectual, despertou em mim o interesse pelo mundo da otimização e das meta-heurísticas, e me deu a oportunidade de trabalharmos juntos por tantos anos. Marcus foi exemplo de profissional e um grande orientador, cujos inestimáveis ensinamentos levarei sempre comigo. Sua experiência, competência e dedicação certamente elevaram a qualidade desta pesquisa. No mesmo espírito, tive a satisfação de trabalhar com o professor Manuel López-Ibáñez, que me recebeu e prestou todo o suporte durante o período em que estive na Universidade de Manchester. Nossas sempre frutíferas discussões e a colaboração em diferentes projetos me proporcionaram enorme aprendizado e moldaram a forma como entendo a pesquisa científica. Parte desta tese é resultado do seu trabalho e da sua criatividade ao discutir ideias, sugerir diferentes perspectivas e explorar novos métodos.

Minha família – o que de mais valioso tenho na vida – teve papel fundamental na minha trajetória pessoal e acadêmica. Tive a alegria de compartilhar dessa caminhada com minha esposa Taize, que esteve presente em cada momento, ainda que por vezes distante fisicamente. Seu amor, suas palavras de carinho e seu incentivo constante me deram forças para superar os desafios que se apresentaram. Meus pais, Renato e Marlene, grandes responsáveis pela minha formação humana, sempre me acolheram nos momentos difíceis e comigo celebraram cada vitória. Seu amparo, direcionamento e, principalmente, seu amor incondicional me trouxeram até aqui. Por me apoiarem na busca pelos meus sonhos, à minha família dedico este trabalho e com ela compartilho minha alegria.

Em vários trechos dessa jornada caminhei na presença de amigos especiais.

Pessoas essas que, ao dedicarem um pouco de si, alegraram meus dias e me transformaram como pessoa. Matthieu Balthazard, grande companheiro e exemplo de ser humano, acompanhou importantes etapas da minha vida e agora faz parte de mais uma delas. Suas palavras de apoio nos momentos desafiadores revelam o verdadeiro amigo que é. Ana Cruzat, amiga de longa data que tantas vezes nos recebeu em Porto Alegre, sempre acompanhada de uma refeição preparada com muito carinho, e de intermináveis e prazerosas conversas. Miguel Ángel Domínguez Ríos comigo compartilhou das aventuras por Manchester e da vida na universidade. Essa experiência não teria sido tão divertida sem sua presença.

Finalmente, o sucesso deste trabalho se deve às oportunidades e ao suporte dado por grandes instituições de ensino e pesquisa. A Universidade do Estado de Santa Catarina, instituição na qual sou professor do quadro efetivo, oportunizou dedicar-me integralmente às atividades do doutorado por meio de uma licença remunerada. A Universidade Federal do Rio Grande do Sul me deu a oportunidade de formação em uma instituição de excelência acadêmica, fornecendo a estrutura e as ferramentas para o desenvolvimento desta tese. A Universidade de Manchester me deu a oportunidade de atuar como pesquisador visitante em um ambiente único, e viver uma experiência de grande crescimento pessoal e intelectual.

A todos vocês, com carinho, muito obrigado!

Marcelo de Souza
Fevereiro de 2022

ABSTRACT

The performance of algorithms is often highly sensitive to the values of their parameters. Therefore, algorithm configuration plays a pivotal role when designing or adapting algorithms for a given problem domain. *Automatic algorithm configuration* methods automate this process, reducing human effort and potential biases involved in error-prone manual configuration approaches. A more general and ambitious research field, called *automatic algorithm design*, applies automatic configuration methods to select, combine and calibrate algorithm components, producing high-quality algorithms automatically for different problem domains. Despite the growing attention and substantial progress made in the last years, there are still open research directions on understanding, improving and exploring methods for the automatic design and configuration of algorithms.

We present a comprehensive study on automatic algorithm configuration with the following contributions. First, we improve the efficiency of the automatic configuration of optimization algorithms. In particular, we propose a set of capping methods that use previous executions to build a performance envelope, which is used to identify poor-performing executions and stop them early. These methods considerably reduce the configuration time without loss of quality. Second, we improve the quality of automatic algorithm configuration by exploring parameter regression models. Instead of searching for parameter values, we calibrate models that set these values according to the instance size of the instance being solved, leading to expressive gains in algorithm performance when compared to using fixed configurations. Third, we provide a visualization tool to analyze and understand the automatic algorithm configuration process. The visualizations allow to identify different types of flaws and improve configuration scenarios. Finally, we propose a component-wise heuristic solver for a general class of binary optimization problems. This solver implements a set of heuristic components that can be selected and combined to produce complete algorithms. Given a problem, automatic configuration methods explore this design space and search for the best heuristic algorithm. We automatically produce new state-of-the-art algorithms for different binary problems using this solver.

Keywords: Automatic algorithm configuration. Automatic algorithm design. Parameter tuning. Capping methods. Parameter regression models. Unconstrained binary quadratic programming.

Configuração Automática de Algoritmos: Métodos e Aplicações

RESUMO

O desempenho de algoritmos está geralmente associado aos valores dos seus parâmetros. Portanto, a configuração do algoritmo desempenha um papel fundamental ao projetar ou adaptar algoritmos para um dado domínio. Métodos de *configuração automática de algoritmos* automatizam esse processo, reduzindo o esforço humano e potenciais vieses envolvidos em abordagens de configuração manuais. Um campo de pesquisa mais geral e ambicioso, chamado *projeto automático de algoritmos*, aplica métodos de configuração automática para selecionar, combinar e calibrar componentes algorítmicos, produzindo algoritmos de alta qualidade automaticamente para diferentes problemas. Apesar da crescente atenção e substancial progresso feito nos últimos anos, ainda existem possibilidades de pesquisa em aberto relacionadas ao entendimento, melhoria e exploração de métodos de projeto e configuração automáticos de algoritmos.

Este trabalho apresenta um estudo abrangente sobre configuração automática de algoritmos com as seguintes contribuições. Primeiro, melhora-se a eficiência da configuração automática de algoritmos de otimização. Em particular, são propostos métodos de poda que usam execuções prévias para construir um envelope de desempenho, o qual é usado para identificar execuções de baixo desempenho e interrompê-las antecipadamente. Esses métodos reduzem consideravelmente o tempo de configuração sem perda de qualidade. Segundo, melhora-se a qualidade da configuração automática de algoritmos explorando modelos de regressão de parâmetros. Em vez de buscar por valores de parâmetros, são calibrados modelos que determinam esses valores de acordo com o tamanho da instância a ser resolvida, levando a um ganho expressivo no desempenho dos algoritmos quando comparado ao uso de configurações fixas. Terceiro, este trabalho disponibiliza uma ferramenta de visualização para analisar e entender o processo de configuração automática de algoritmos. As visualizações permitem identificar diferentes tipos de falhas e melhorar cenários de configuração. Finalmente, este trabalho propõe um *solver* heurístico baseado em componentes para a classe geral de problemas binários de otimização. Esse *solver* implementa um conjunto de componentes heurísticos que podem ser selecionados e combinados para a produção de algoritmos completos. Dado um problema, métodos de configuração automática exploram esse espaço de componentes e buscam pelo melhor algoritmo

heurístico. Foram produzidos novos algoritmos no estado-da-arte para diferentes problemas binários usando esse *solver*.

Palavras-chave: Configuração automática de algoritmos, projeto automático de algoritmos, ajuste de parâmetros, métodos de poda, modelos de regressão de parâmetros, programação quadrática binária irrestrita.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

Problem domains:

BPP	Bin Packing Problem	Chapter 3
COL	Graph COLoring	Chapter 3
CPP	Clique Partitioning Problem	Chapter 3
FS	Permutation FlowShop scheduling problem	Chapter 3
MaxCut	Maximum Cut on graphs	Chapters 3, 8
MIP	Mixed-Integer Programming	Chapter 3
SAT	Boolean SATisfiability	Chapter 3
TA	Test-Assignment problem	Chapter 8
TSP	Traveling Salesperson Problem	Chapters 3, 6
UBQP	Unconstrained Binary Quadratic Programming	Chapters 1, 3, 7, 8

Heuristic approaches:

ACO	Ant Colony Optimization	Chapter 3
BS	Bubble Search	Chapter 3
CC	CirCut method	Chapter 8
D^2TS	Diversification-Driven Tabu Search	Chapters 7, 8
HEA	Hybrid Evolutionary Algorithm	Chapter 3
HH	Hybrid Heuristic	Chapter 3
ILS	Iterated Local Search	Chapter 3
ITS	Iterated Tabu Search	Chapters 7, 8
PR	Path Relinking	Chapters 7, 8
SS	Scatter Search	Chapter 8
TS	Tabu Search	Chapter 8

Algorithm configuration scenarios:

ACOTSP	Ant Colony Optimization for TSP	Chapters 3, 4, 6
BSFS	Bubble Search for FS	Chapters 3, 5

HEACOL	Hybrid Evolutionary Algorithm for COL	Chapters 3, 4
HHBQP	Hybrid Heuristic for UBQP	Chapters 3, 4, 8
HHTA	Hybrid Heuristics for TA	Chapters 3, 5, 8
ILSBQP	Iterated Local Search for UBQP	Chapters 3, 5
LKH	Lin-Kernighan-Helsgaun heuristic	Chapters 3, 4
SCIP	Solving Constraint Integer Programs	Chapters 3, 4
TSBPP	Tabu Search for BPP	Chapters 3, 4
TSCPP	Tabu Search for CPP	Chapters 3, 5

Algorithms produced:

HHBQP	Hybrid Heuristic for UBQP	Chapters 3, 4, 7, 8
HHMC	Hybrid Heuristic for MaxCut	Chapter 8
HHPAL	Hybrid Heuristic for PALubeckis-like instances	Chapter 8
HHTA	Hybrid Heuristics for TA	Chapters 3, 5, 8
ILSBQP	Iterated Local Search for UBQP	Chapters 3, 5

Capping methods:

AD.2	Area + aDaptive + $a_g = 0.2$	Chapter 4
AD.4	Area + aDaptive + $a_g = 0.4$	Chapter 4
AD.6	Area + aDaptive + $a_g = 0.6$	Chapter 4
AD.8	Area + aDaptive + $a_g = 0.8$	Chapter 4
AEBB	Area + Elitist + Best + Best	Chapter 4
AEWW	Area + Elitist + Worst + Worst	Chapter 4
PD.2	Profile + aDaptive + $a_g = 0.2$	Chapter 4
PD.4	Profile + aDaptive + $a_g = 0.4$	Chapter 4
PD.6	Profile + aDaptive + $a_g = 0.6$	Chapter 4
PD.8	Profile + aDaptive + $a_g = 0.8$	Chapter 4
PEBB	Profile + Elitist + Best + Best	Chapter 4
PEMB.1	Profile + Elitist + Model + Best + $p = 0.1$	Chapter 4

PEMB.3	Profile + Elitist + Model + Best + $p = 0.3$	Chapter 4
PEMB.5	Profile + Elitist + Model + Best + $p = 0.5$	Chapter 4
PEMW.1	Profile + Elitist + Model + Worst + $p = 0.1$	Chapter 4
PEMW.3	Profile + Elitist + Model + Worst + $p = 0.3$	Chapter 4
PEMW.5	Profile + Elitist + Model + Worst + $p = 0.5$	Chapter 4
PEWW	Profile + Elitist + Worst + Worst	Chapter 4

Software produced:

ACVIZ	Algorithm Configuration VIsualiZations	Chapter 6
AutoBQP	Solver based on Automatic design for UBQP	Chapters 7, 8
CAPOPT	CAPping methods for OPTimization	Chapter 4

CONTENTS

Part I

Fundamentals

1 INTRODUCTION

We ourselves feel that what we are doing is just a drop in the ocean. But the ocean would be less because of that missing drop.

— Saint Teresa of Calcutta

In computer science, we design algorithms to solve a wide range of problems. We classify these problems in two main groups. *Decision* problems ask for a yes/no answer, e.g. determining whether it is possible to allocate forty-eight talks, each one with its predefined duration, to the eleven rooms available for the school event on Saturday morning. Instead, if we want to determine the allocation that uses the smallest number of rooms, we have an *optimization* problem. In this second type of problem, a measure of cost (or quality) indicates how good a solution is, and the goal is to find a best one, which is called an *optimal solution*. The search for the shortest route between two points is another example of optimization problem, where the cost of a route is simply its total distance and the optimal solution is the route with smallest distance.

The computational *algorithms* that manage and solve these problems are called *exact* (or *optimal*), when they guarantee finding the right answer for a decision problem, or an optimal solution in case of an optimization problem. For some optimization problems, exact algorithms take fast-growing time with respect to problem size and their use becomes impractical as problem size grows. To deal with these cases, (*meta*)*heuristic* algorithms apply (usually stochastic) search techniques to systematically explore the solution space and produce high-quality solutions in reasonable time, but there is no optimality or any approximation guarantee[1].

The progress of computer science relies on the advancement of these algorithms, either by developing new ones, improving existing algorithms, or adapting them to new problem domains. In such an algorithm design process, the developer must identify, from a wide range of possibilities, higher-level algorithmic strategies that perform well on the specific domain of interest. Then, he must decide over several lower-level choices regarding the internal behavior of such strategies to produce a complete algorithm. Some of these higher- and lower-level choices are left to the

[1]For some problems, there are also *approximation* algorithms, which explore the problem structure and guarantee a bound on the solution cost related to the cost of the optimal solution.

users as input parameters, allowing them to adapt the algorithm to new problems and get optimized performance on different domains. For example, the exact solver SCIP (ACHTERBERG, 2009) for mixed-integer programming has more than 2500 parameters[2] in its current version 7.0.2.

In this context, a *parameter configuration* is a valid assignment of values to the input parameters of an algorithm, and *algorithm configuration* is the search for the best parameter configurations. In general, configuring an algorithm involves testing several parameter configurations on different problem instances, in order to find those configurations that optimize the algorithm performance on the instances of interest. Therefore, algorithm configuration is a tedious and time-consuming process that usually takes a considerable effort from the researcher. Besides that, it is advisable to know the internal details of the algorithm to select promising combinations of parameter values to test. Even so, several configurations can be left out and the process is usually biased and non-replicable. Although using the algorithm's default configurations could avoid these complications, substantial performance improvements are usually observed when choosing appropriate parameter values relative to the specific problem domain or execution conditions (e.g. termination criteria). This emphasizes the importance of the algorithm configuration.

Automatic algorithm configuration is an alternative to this manual process and alleviate the aforementioned problems. Many tools, called *configurators*, have been proposed in the last years. They apply techniques specially designed for configuring algorithms, including heuristic search (HUTTER et al., 2009b; ANSÓTEGUI; SELLMANN; TIERNEY, 2009), model-based optimization (HUTTER; HOOS; LEYTON-BROWN, 2011), and racing algorithms (BALAPRAKASH; BIRATTARI; STÜTZLE, 2007; LÓPEZ-IBÁÑEZ et al., 2016). Such configurators have shown to be useful in improving the performance of existing algorithms by finding good configurations in several domains, like exact mathematical solvers (HUTTER; HOOS; LEYTON-BROWN, 2010; LÓPEZ-IBÁÑEZ; STÜTZLE, 2014), compilers (PÉREZ CÁCERES et al., 2017), decision algorithms (HOOS; HUTTER; LEYTON-BROWN, 2021) and optimization algorithms (YARIMCAM et al., 2014; PÉREZ CÁCERES; LÓPEZ-IBÁÑEZ; STÜTZLE, 2015). For example, Hutter, Hoos and Leyton-Brown (2010) report speed ups of exact mathematical solvers for mixed-integer programming up to a factor of 153, in comparison to default configurations.

[2]For a complete list of SCIP parameters see: https://www.scipopt.org/doc-7.0.2/html/PARAMETERS.php.

A more advanced use of such configurators is applying them to automate the whole algorithm design process. Instead of focusing on configuring the input parameters of a predefined algorithm, we define a set of algorithm components that can be combined to produce different and hybrid complete algorithms. We then apply a configurator to automatically explore the design space of components and search for the best algorithms for a problem domain. This approach is called *automatic algorithm design* and has produced state-of-the-art algorithms for different domains. Some examples include the automatic design of boolean satisfiability solvers (KHUDABUKHSH et al., 2016), stochastic local searches for permutation flowshop problems (MARMION et al., 2013; PAGNOZZI; STÜTZLE, 2019), and evolutionary algorithms for multi-objective optimization (BEZERRA; LÓPEZ-IBÁÑEZ; STÜTZLE, 2014a; BEZERRA; LÓPEZ-IBÁÑEZ; STÜTZLE, 2020b).

1.1 Motivation and Scope

Automatic configuration methods have been shown to be important both for configuring parameters of an existing algorithm and for automatically designing new ones. Automatic approaches improve the algorithm design process and contribute to the field in different ways by: (i) freeing researchers and practitioners from the tedious and time-consuming task of testing different algorithm configurations, which allows them to focus on creativity-related tasks, such as the development of new algorithm components; (ii) removing a potential bias from the configuration process, e.g. by better covering the configuration space when selecting parameter values to test; (iii) making it easier to design entirely new or adapt existing algorithms to new problem domains, even for users with no expert knowledge; (iv) allowing a fair evaluation and comparison of different algorithms; and (v) supporting the analysis of algorithms and problem domains by providing useful information about the algorithm performance on different problem instances and under different configurations.

Despite the recent efforts to understand and develop automatic techniques for algorithm configuration, there are still several open research directions related to the improvement of such methods. Given the wide applicability of automatic algorithm configuration to build and improve algorithms, the advancement of these methods contributes directly to the progress of various related research fields. In this work, we study the automatic configuration of algorithms and propose a set of methods to

improve and better understand the configuration process, as well as applications for designing algorithms automatically. In this sense, this work has four objectives:

Objective 1.
Increase efficiency in the automatic configuration of optimization algorithms.

When configuring algorithms with the objective of minimizing their running time, most state-of-the-art configurators implement capping methods to avoid spending time evaluating unpromising configurations (HUTTER et al., 2009b; HUTTER; HOOS; LEYTON-BROWN, 2011; PÉREZ CÁCERES et al., 2017a; PUSHAK; HOOS, 2020). These methods determine a running time bound based on previous executions of the best configuration found so far, and then stop early executions that exceed this bound. These approaches are not suitable for configuring optimization algorithms, since the objective is usually to minimize the cost of the best solution found after running the algorithm with a predefined termination criterion. Current configurators do not implement any method to save time when evaluating poorly-performing configurations of optimization algorithms. As a consequence, the configuration process of such algorithms is more costly, which sometimes prevents the appropriate use of automatic configuration in optimization scenarios with large configuration spaces. Given this situation, the first step of this thesis is to develop a set of capping methods specially designed to improve the efficiency when automatically configuring optimization algorithms.

Objective 2.
Improve the quality of automatic algorithm configuration.

Most algorithm configuration methods search for fixed parameter values for the whole set of problem instances, i.e. instance-oblivious configurations that do not consider their characteristics. For each parameter, such methods perform an equivalent to a constant regression, since the parameter value remains constant for any problem instance. However, optimal parameter values may vary depending on instance features, such as the instance size (BÖTTCHER; DOERR; NEUMANN, 2010; EL YAFRANI; AHIOD, 2018). In the second step of this thesis, we aim at improving the quality of the automatic algorithm configuration by exploring the

relation between instance size and optimal parameter values to produce non-constant configurations with optimized performance according to the instance being solved. In particular, focusing on the instance size makes this approach useful and easily applicable to any problem domain, since this feature is problem-independent, always available, and can be computed efficiently.

Objective 3.
Facilitate analyzing and understanding of the algorithm configuration process.

Despite the widespread use of automatic configuration methods in diverse domains, it is possible and even common to apply such techniques as black-box configuration methods, and a proper analysis and understanding of their operation is frequently missing. On the other hand, it is important to analyze the execution of the configurator being used and understand how it works, in order to obtain the best results from the configuration process. The data generated when executing the configurator provide useful information about the algorithm performance under different configurations and the decisions made during the configuration process. However, analyzing these data can be quite challenging, since they must be processed and interpreted carefully to get useful information, besides requiring knowledge about the internal behavior of the configurator, e.g how it selects configurations to evaluate. Therefore, our third objective aims at simplifying the analysis of the algorithm configuration process, by providing methods to process, interpret, and visualize the configuration data.

Objective 4.
Apply automatic algorithm design to build heuristic solvers for classes of problems.

There are several heuristic solvers for optimization problems. They commonly implement general-purpose algorithms based on metaheuristics that can be used to solve a class of problems. Examples include solvers based on local search (BENOIST et al., 2011) and scatter search (GORTAZAR et al., 2010). Although these approaches handle different problems, they implement fixed algorithmic strategies and their adaptation to different domains is limited. Therefore, the obtained performance across such domains is usually far from the one obtained by using problem-specific

algorithms.

An alternative idea is defining a variety of algorithm components and use automatic design techniques to search for the best algorithms for a given problem. This approach allows the generation of hybrid heuristic algorithms, by combining the available algorithm components and defining specific values for their input parameters. The primary goal of methods based on automatic algorithm design is not to outperform problem-specific algorithms, but provide a general-purpose approach that reaches competitive solutions in comparison to state-of-the-art algorithms, with little human effort and no need for expert knowledge in the problem domain. In the context of this thesis, we aim at using the above ideas to provide a heuristic solver for a wide range of binary problems.

1.2 Summary of Contributions

First, we propose a set of methods that improve the efficiency and quality of the configuration process, and facilitate its analysis and understanding. Below we briefly discuss our main contributions in these directions.

1. We introduce a set of capping methods for optimization scenarios. These methods define several strategies to (i) evaluate the quality of an execution, and (ii) determine a minimum required performance for new executions. Then, poorly-performing executions can be terminated early, saving the remaining running time. Our experiments show a reduction from about 5% to 78% of the configuration time, without loss of quality. We also provide evidence that capping can help to find better final configurations, when the saved time is used to further explore the configuration space, i.e. in scenarios using a configuration time budget.

2. We propose improved regression models for algorithm configuration. In contrast to existing configuration approaches, we represent parameters by non-constant models, which set the parameter values according to the instance size. Instead of searching for parameter values directly, the configuration process calibrates such models. We propose different approaches to approximate both linear and nonlinear relations between the instance size and optimal parameter values. The evaluation of the proposed methods on different configuration scenarios

show good performance gains in comparison to traditional instance-oblivious algorithm configuration with the same effort.

3. We present a visualization tool that helps to analyze the automatic configuration of algorithms. We focus on executions of the irace configurator (LÓPEZ-IBÁÑEZ et al., 2016) and provide a visual representation of the configuration process, allowing users to extract useful information, e.g. how the configurations evolve over time. When test data is available, this tool also shows the performance of each configuration on the test instances. Using this visualization, users can analyze and compare the quality of the configurations produced and observe their performance differences on training and test instances. We also present several exemplary case studies of how the visualizations can be used to analyze and improve configuration scenarios.

We also used automatic algorithm configuration techniques to automatically design heuristics for binary optimization problems. The resulting contributions are presented below.

1. We implement a component-wise framework of heuristics to handle binary problems. It allows instantiating different hybrid algorithms by selecting and combining its algorithm components. The framework is based on the Unconstrained Binary Quadratic Programming (UBQP) and since many problems can be reduced to UBQP (KOCHENBERGER et al., 2014; KOCHENBERGER et al., 2004), the framework can be used to solve a wide range of problems. Providing the implementation of such heuristic components and a unified framework that allows selecting and combining them into complete algorithms can be of high interest.

2. We demonstrate that automatic algorithm configuration techniques can be used to explore the design space of the framework and search for the best combinations of components to solve a given problem. Such automatic algorithm design approach uses a grammar to express the design space of components, and a parametric representation of this grammar to allow using automatic configuration tools to produce high-quality algorithms.

3. We provide a heuristic solver for binary problems that combines the proposed framework based on UBQP with the automatic algorithm design approach.

The user provides a problem description and a set of problem instances, and the solver automatically builds a heuristic algorithm to solve the problem. This solver can be quite useful, since it can be applied to a wide range of problems that can be modeled as binary quadratic programming, with no need of expert knowledge on the problem domain, heuristic algorithms or automatic design techniques.

4. By using the proposed solver, we produce hybrid heuristic algorithms for different optimization problems (namely UBQP, maximum cut on graphs, and test-assignment, a variant of the graph coloring problem). These algorithms outperformed state-of-the-art approaches, and found new best solutions for several instances of the maximum cut and test-assignment problems.

5. This work provides general guidelines for researchers and practitioners on the automatic design of algorithms.

Finally, we mention below some further side contributions of this research, which can also be useful when working with automatic algorithm configuration.

1. We demonstrate the practical relevance of the automatic design and configuration of algorithms. We test several (most optimization) algorithm configuration scenarios and show the improvements in algorithm performance when using automatic configuration approaches.

2. We provide a comprehensive set of configuration scenarios, with most of them being new ones, which can be useful either to adapting the underlying target algorithms to new problem domains, or testing and comparing different algorithm configuration techniques.

Additional information about the contributions of this research is given in Appendix A, including the resulting scientific publications, implementations of the proposed methods, and supplementary information about the experiments and results.

1.3 Thesis Outline

This thesis is organized in five parts. Part I introduces this research and gives some background information (Chapters 1, 2 and 3). Part II presents the proposed methods to improve and analyze the automatic configuration of algorithms (Chapters 4, 5 and 6). Part III presents applications of automatically building heuristics for a class of optimization problems (Chapters 7 and 8). Part IV concludes the thesis and presents some future research directions (Chapter 9). In addition to the main body, Part V comprises supplementary information that is of interest for future reference (Appendices A and B). Below we give an overview on the remaining chapters.

Chapter 2 formalizes and motivates the algorithm configuration problem. We present existing automatic configuration methods and detail the irace configurator. We review the literature and discuss several applications of automatic algorithm configuration on different problem domains. We also discuss the application of these approaches to the general concept of automatic algorithm design from components and present some closely related problems.

Chapter 3 introduces the algorithm configuration scenarios used in this work. We discuss the underlying problem, target algorithms, configuration budget, involved parameters and benchmark instances; we also classify these scenarios according to some characteristics (e.g. decision vs. optimization problems or exact vs. heuristic algorithms).

Chapter 4 presents a set of capping methods to reduce the configuration time of optimization algorithms. We discuss the proposed methods in detail and present extensive experiments on six configuration scenarios.

Chapter 5 explores parameter regression models for improving the quality of automatic algorithm configuration. We propose different models to map the instance size to optimal parameter values and evaluate them on four configuration scenarios.

Chapter 6 proposes a visual tool to analyze the automatic configuration process. We present a set of case studies, showing how the visualization techniques can be used to identify problems in the configuration scenarios based on the data generated during the configuration process.

Chapter 7 explores the use of automatic algorithm configuration and design techniques to build a component-wise heuristic solver for the general class of binary optimization problems. First, we propose an algorithm framework of heuristic components to tackle binary optimization problems. Then, we study the use of automatic configuration techniques to explore the design space defined by this framework and searches for the best combination of components to generate a complete algorithm for a given problem. We present and discuss the underlying methodology and internal behavior of the proposed solver.

Chapter 8 presents an experimental evaluation of the component-wise heuristic solver based on automatic algorithm design. We use it to solve different binary optimization problems and show that it is able to produce algorithms competitive to state-of-the-art approaches, improving them in some cases and finding new best solutions for different problems.

Chapter 9 concludes this thesis, discussing its main results and outlining directions to further extend our research.

Appendix A presents the main products of this thesis. We list the scientific publications arising from this research, the presentations of this work in workshops, doctoral consortia and Ph.D. schools, and software made available with the proposed approaches. We also provide the reader with supplementary material for the proposed methods and experiments.

Appendix B presents an extended abstract of this thesis in Portuguese.

2 THE ALGORITHM CONFIGURATION PROBLEM

> *If I had an hour to solve a problem I'd spend 55 minutes thinking about the problem and 5 minutes thinking about solutions.*
>
> — Albert Einstein

As stated in Chapter 1, many algorithms have parameters that allow to adapt their behavior to the problem being solved. Since most algorithms are sensitive to their parameter values, the selection of good configurations often improves their performance and is a fundamental step of the algorithm design. In this thesis, we deal with the *offline* algorithm configuration problem. Offline methods search, during a training phase, for the best configurations (i.e. those that lead to the best algorithm performance) by evaluating the target algorithm under several different configurations. When this phase ends, the best found configurations can be used in production. Usually, offline methods use a set of training instances to configure the algorithm, and then evaluate the resulting configurations on a different set of test instances. Despite the simple idea behind algorithm configuration, there are several difficulties involved in this problem (e.g. how to explore the configuration space efficiently), which make it challenging.

In this chapter, we formalize the algorithm configuration problem, introduce the corresponding mathematical notation, and present some approaches to automate the configuration process. We discuss the irace configurator in more detail, since we use it throughout the whole thesis to test the proposed methods. We present a summary of applications of automatic approaches for configuring algorithms for several different domains. We also present the ideas of automatic algorithm design from components, which use configurators to explore the design space of algorithms, automatically searching for the best algorithms for a given problem. Finally, we discuss some closely related problems.

2.1 Problem Formulation

The algorithm configuration problem can be formally defined as follows. Let $\mathcal{A}$ be a target algorithm with n parameters p_i, $i = 1, \ldots, n$, each one with domain Θ_i.

The space of all configurations Θ is the subset of $\Theta_1 \times \cdots \times \Theta_n$ of valid parameter combinations. Given a set of instances Π and a metric $c(\theta, \pi)$ that measures the cost of running $\mathcal{A}$ with configuration $\theta \in \Theta$ and instance $\pi \in \Pi$ as input, the algorithm configuration problem is to find a configuration $\theta^* \in \Theta$ that minimizes the cost of $\mathcal{A}$ over instances Π, i.e.

$$\theta^* \in \operatorname*{argmin}_{\theta \in \Theta} \mathcal{C}(\theta, \Pi), \tag{2.1}$$

where $\mathcal{C}$ is an aggregation function of $c(\theta, \pi)$ over all instances $\pi \in \Pi$. For example, we can use the average performance $\mathcal{C}(\theta, \Pi) = \sum_{\pi \in \Pi} c(\theta, \pi)/|\Pi|$. There are different choices for the cost metric c. For decision problems, usually the running time is used. For optimization problems, it is common to use the cost of the best found solution, after an execution with a given computational effort, such as running time or number of iterations[1]. Finally, if the target algorithm $\mathcal{A}$ is stochastic, then $c(\theta, \pi)$ is a random variable.

Parameters may have different types (TRIOLA, 2019). *Categorical* parameters can assume a fixed number of values, and are often used to model discrete choices such as algorithmic strategies, e.g. the acceptance criterion in a local search. *Ordinal* parameters have a natural ordering but no definite distance; an example would be the choice of a neighborhood in a local search when ordered by size. *Numerical* parameters represent real or integer values, e.g. the numerical tolerance in a mathematical solver, or the initial temperature in Simulated Annealing.

A parameter can be also conditional, i.e. it is only active when a second parameter assumes certain values. For example, setting a value to a tabu tenure parameter is only necessary when the parameter that defines the type of improvement procedure has been set to "tabu search". Apart of that, parameter values may have to satisfy constraints, either due to relations with other parameters, e.g. the number of individuals to be selected for crossover in a genetic algorithm can not be greater than its population size, or due to explicitly forbidden parameter combinations that cause undesired behavior or execution errors. These characteristics make some configurations invalid, which are excluded from the configuration space Θ.

Based on the above, we now formally define a configuration scenario.

[1] For other scenarios like the configuration of machine learning models, a performance metric can be used in place of the cost metric, e.g. the prediction accuracy of the model. In this case, the configuration task searches for configurations that maximize the algorithm performance.

Configuration scenario. *A configuration scenario is an instance of the algorithm configuration problem defined by a 6-tuple* $\langle \mathcal{A}, \Theta, \Pi, c, B, \mathcal{S} \rangle$*, where*

- $\mathcal{A}$ *is a parameterized target algorithm;*
- Θ *is the configuration space of* $\mathcal{A}$*;*
- Π *is a set of training instances;*
- c *is a cost metric used to evaluate an execution;*
- B *is a configuration budget;*
- $\mathcal{S}$ *is a set of additional settings.*

Budget B defines the computational resources available for the configuration process, e.g. the maximum number of executions of $\mathcal{A}$ or the total configuration time. We also define a set $\mathcal{S}$ of additional settings for the configuration process itself (e.g. default and initial configurations), for the training instances (e.g. instance specific arguments), and for the target algorithm (e.g. the termination criterion).

Algorithms can be configured manually. The so-called *manual algorithm configuration* is based on the previous experience and intuition of the algorithm designer, who tests different configurations and chooses those that seem more appropriate. This approach becomes difficult to replicate, requires specialized knowledge on the algorithm being configured and often demands a large amount of time, which is partially spent testing ineffective configurations. The configuration space is often not explored systematically, since sometimes configurations are chosen based only on previous experience, or evaluated using shortcuts to reduce configuration time, such as short test runs on few instances. This may ignore promising configurations and lead to biases in the configuration process.

2.2 Automatic Algorithm Configuration

Automatic algorithm configuration methods apply techniques specially designed to use the available computational resources to efficiently explore the configuration space, minimizing the drawbacks discussed above. The tools implementing these methods, called *configurators*, follow the setup depicted in Figure 2.1. Note the interaction between the algorithm configurator and each component of the configura-

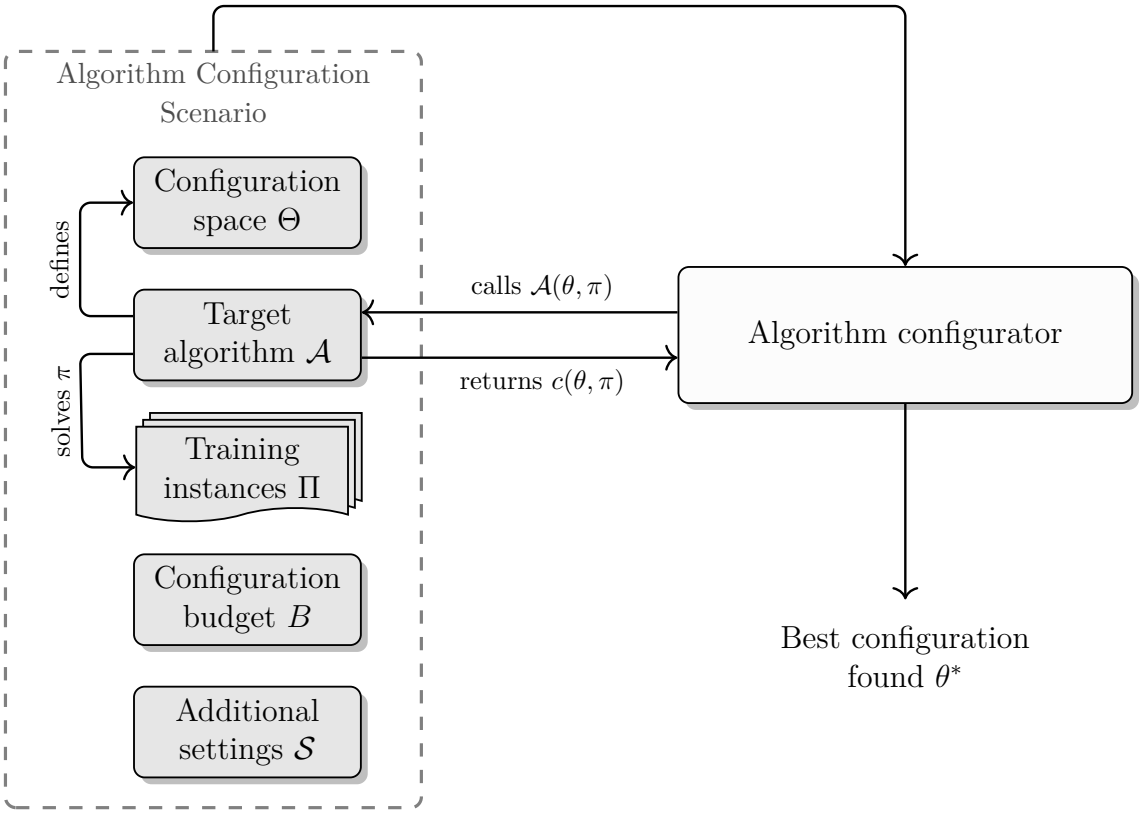


Figure 2.1 – General setup for algorithm configuration: the configuration space Θ, budget B, and the additional settings $\mathcal{S}$ are the inputs of the algorithm configurator; during the configuration process, the configurator calls the target algorithm $\mathcal{A}$ to solve instance π under configuration θ, and gets the resulting execution cost $c(\theta, \pi)$; when the budget B is consumed, the best configuration found θ^* is returned.

tion scenario (introduced in Definition 1). The inputs of the configurator are the configuration space Θ, defined by the parameters of the target algorithm $\mathcal{A}$ being configured, the configuration budget B, and the additional settings $\mathcal{S}$. The configurator explores the configuration space by repeatedly evaluating different configurations. It evaluates the cost of running $\mathcal{A}$ on instance π under configuration θ by calling $\mathcal{A}(\theta, \pi)$ and getting the resulting cost $c(\theta, \pi)$. When the budget B is consumed, the configurator returns one or a set of best configurations, possibly with information about the configuration process for further analysis.

The configuration of algorithms is itself an optimization problem, but the objective function is not given in closed form and we can not solve it analytically, but only evaluate it at given points. Besides that, it has several characteristics that make it a challenging problem. We discuss these characteristics in the following.

Multiple instances. The optimal configurations are those that minimize the cost of running the algorithm on a set of training instances (Eq. 2.1). Evaluating the

quality of a configuration involves aggregating the observed costs over instances. In case of costs with different scales across instances, additional processing might be necessary, e.g. normalization of costs or the use of ranks. Since it is usually not feasible (due to time restrictions) to test the configurations on all training instances, configuration costs have to be estimated through observations on a limited sampling of instances.

Stochastic nature. The first source of stochasticity is the sampling of the problem instances involved in the configuration process. Besides that, algorithm $\mathcal{A}$ can be stochastic itself. Then, $c(\theta, \pi)$ is a random variable even with fixed θ and π, and multiple tests of the same configuration-instance pair might be necessary.

Numerical, categorical and conditional parameters. Configuration scenarios usually have mixed numerical (real numbers or integers) and categorical parameters, and due to conditionality, some parameters may not exist in some configurations. As a consequence, typical algorithms must be adapted, e.g. the concepts of neighborhood or recombination, for local searches (HUTTER et al., 2009b) or evolutionary algorithms (ANSÓTEGUI; SELLMANN; TIERNEY, 2009), respectively. It also prevents using some approaches, like most of black-box optimization algorithms, since they usually handle only numerical variables with no conditionality (YUAN et al., 2012).

Parameter interaction. The algorithm performance might depend on second or higher order parameter interactions (HUTTER; HOOS; LEYTON-BROWN, 2014; FAWCETT; HOOS, 2016), i.e. the effect of changing these parameters simultaneously is different than the effect of changing them individually. This characteristic can be explored when searching for the best configurations.

Expensive evaluation. Evaluating a configuration is costly, since it involves running the target algorithm to solve one or more instances and observe the resulting execution cost. Therefore, configurators must carefully choose which configurations to evaluate on which instances, in order to minimize the number of algorithm executions, keep the configuration process feasible even on large configuration spaces, and ensure an efficient use of the available computational resources.

In addition to the characteristics discussed above, the configuration scenario must be carefully defined. The parameter domains, which determine the configuration

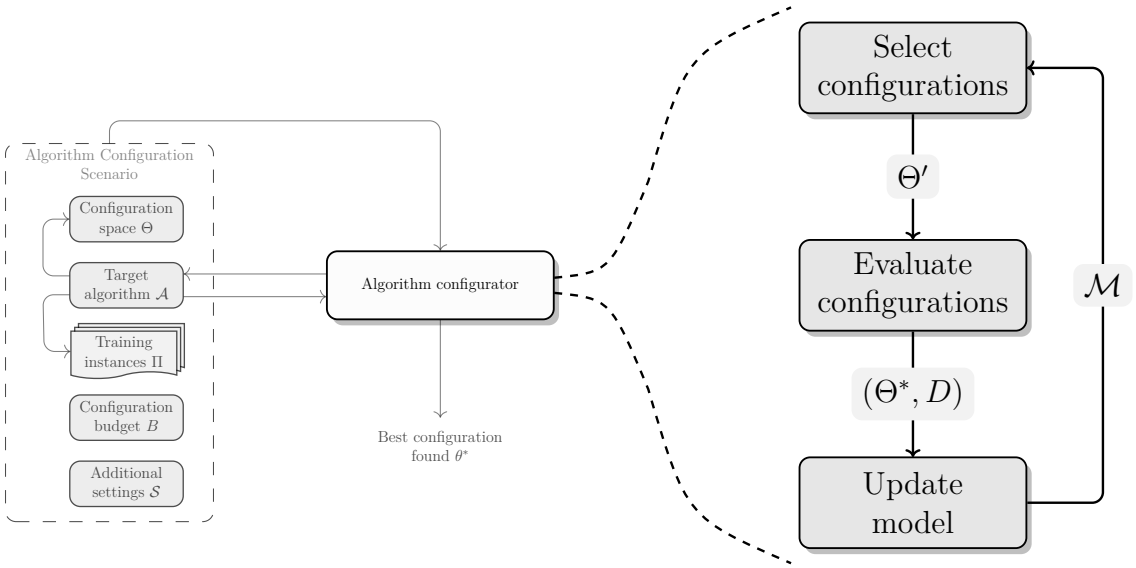


Figure 2.2 – Internal steps of general purpose algorithm configurators: first, a set of configurations $\Theta' \subset \Theta$ are selected based on a sampling model $\mathcal{M}$; second, configurations Θ' are evaluated on a subset of the training instances; finally, the best performing configurations Θ^* from the previous step and the evaluation data D are used to update the sampling model $\mathcal{M}$; the process is repeated until budget B is consumed.

space Θ, must be properly chosen. Large intervals or a lot of possible values, for numerical and categorical parameters, respectively, increase the size of the configuration space and, consequently, the complexity of the configuration task. On the other hand, reducing such domains too much might exclude promising values and lead to poor final configurations. Similarly, the configuration budget B must be properly defined. While low budgets might result in a poor exploration of the configuration space, potentially leading to underperforming algorithms, unnecessarily large budgets often imply waste of both time and computational resources.

In the offline algorithm configuration we search for the best configurations based on their performance on a set of training instances. Then, the best found configurations can be used in production or can be evaluated on a different set of test instances. Thus, it is important to choose training instances that are representative of the instances for which the target algorithm is configured to solve. Otherwise, the resulting configurations will perform poorly in production. Besides that, insufficiently diverse training instances may lead to overtuning (BIRATTARI, 2009), i.e. the observed performance on the training instances does not generalize to unseen test instances[2].

[2]In machine learning, this is named overfitting (HAWKINS, 2004), i.e. the resulting configuration (machine learning model) specializes on the training instances (training data) and does not generalize to unseen instances (test data).

A number of configurators were proposed in the last years to handle, at least partially, the particularities detailed above. These configurators follow the general schema given by Figure 2.2. First, a sampling model $\mathcal{M}$ is used to select one or more configurations ($\Theta' \subset \Theta$). The performance of the target algorithm $\mathcal{A}$ under these configurations are evaluated on a subset of the training instances Π. The best configurations Θ^* and the results of the evaluation step D are used to update the sampling model for the next iteration. These steps (selection, evaluation and model update) are repeated until the budget B is consumed. Some methods use a static model to guide the selection of configurations, therefore they do not require any model update.

The next sections discuss configurators and detail the underlying configuration techniques applied to select configurations, evaluate them and update the sampling models. We first present an overview of several configurators, then we discuss irace in further detail, since it is used in our experiments.

2.2.1 Review on Algorithm Configurators

We present here several automatic algorithm configuration methods developed in the past three decades. We focus on general purpose algorithm configurators, i.e. domain-independent methods that handle any target algorithm, and do not detail approaches like the MULTI-TAC system (MINTON, 1993) for constraint satisfaction problems, higher level genetic algorithms (GA) that configure lower level GAs (GREFENSTETTE, 1986; TERASHIMA-MARÍN; ROSS; VALENZUELA-RENDÓN, 1999), and the so-called adaptive problem solving methods, e.g. the COMPOSER system of (GRATCH; DEJONG, 1992; GRATCH; CHIEN, 1996). Nevertheless, most of the configurators described below have limitations, i.e. they often do not handle all particular characteristics of the algorithm configuration problem discussed above, but such methods were important to the development of the algorithm configuration field and proposed fundamental techniques that are currently used in the most prominent and widely used configuration methods (which are detailed at the end of this section). We classify the configurators according to the techniques they use to explore the configuration space, i.e. to select and evaluate

configurations and, in some cases, to fit and update a sampling model[3].

Several of these methods rely on *experimental design* techniques (MONTGOMERY, 2012), also called *Design of Experiments* (DoE), to select configurations to be tested, combined with additional approaches to increase efficiency and eliminate (or at least minimize) human intervention. Some approaches, e.g. Parsons and Johnson (1997) or Ridge and Kudenko (2006, 2007), rely on full or fractional design for selecting configurations to evaluate. The evaluation results are used to fit response surface models (RSM), which predicts the importance of each parameter and identify potentially optimal configurations. Coy et al. (2001) use fractional factorial design combined with gradient descent to select and evaluate configurations, identifying the best ones for each training instance. These best performing configurations are combined into the final configuration by setting each parameter to the average value taken from all of them. The CALIBRA configurator by Adenso-Díaz and Laguna (2006) selects configurations using a Taguchi experimental design (PEACE, 1993), and applies multiple runs of a local search procedure to refine the region of interest for the next selection. The main drawback of these approaches is the large number of algorithm evaluations required, which limits their application to scenarios with small configuration spaces (HOOS, 2012a).

Another research direction is the adaptation of *numerical optimization* methods to the algorithm configuration problem, with the main limitation of handling only (and usually few) continuous parameters with no dependencies. Numerical optimization methods already tested for the algorithm configuration problem are the mesh adaptive direct search (MADS) algorithms of Audet and Orban (2006), the bound optimization by quadratic approximation (BOBYQA) of Powell (2009), the Nelder-Mead Simplex approach of (NELDER; MEAD, 1965) (also used in Park and Kim (1998) and Muja and Lowe (2009) for algorithm configuration), and the covariance matrix adaptation evolution strategy (CMA-ES) described in Hansen and Ostermeier (2001) and Hansen (2006). Yuan et al. (2012) present a comprehensive study of this kind of methods and analyze their use in the context of algorithm configuration. They found that CMA-ES worked best and presented robust performance over different configuration scenarios. CMA-ES is an evolutionary algorithm that recombines configurations based on a multivariate Gaussian distribution, whose mean

[3]Note that many of the configurators discussed here could be classified in different groups at the same time, since they use various techniques simultaneously. We classify them according to the nature of the main inner approach.

is a linear combination of the best performing configurations from the population. The search trajectory is used to update the covariance matrix of the distribution, allowing to predict parameter influence and interactions.

Heuristic search techniques are also explored to configure algorithms, especially for scenarios involving categorical parameters. A widely used heuristic strategy is local search, applied in configurators like the dynamically dimensioned search (DDS) of Tolson and Shoemaker (2007), OpenTuner (ANSEL et al., 2014), and the optimization of algorithms (OPAL) of Audet, Dang and Orban (2014). Evolutionary approaches use heuristics to evolve a population of configurations, e.g. the evolutionary calibrator (EVOCA) of Riff and Montero (2013). Nannen and Eiben (2006, 2007) propose the relevance estimation and value calibration configurator (REVAC, further extended in Smit and Eiben (2009)), an evolutionary approach combined with an estimation of distribution algorithm (PELIKAN; GOLDBERG; LOBO, 2002) to configure evolutionary algorithms. REVAC maintains a population of configurations and evaluates them on the (single) training instance. The best performing ones are selected and recombined by a multi-parent crossover operator, generating a new configuration. After a mutation step, the newly generated configuration replaces the oldest configurator in the population. These steps repeat until the budget is consumed. Ansótegui, Sellmann and Tierney (2009) proposed a gender-based genetic algorithm (GGA) to evolve configurations. The population is divided into competitive and non-competitive genders. Competitive individuals are evaluated on the training instances, while non-competitive individuals are only used for diversity preservation. GGA recombines the best performing competitive individuals with non-competitive individuals, and selects the next generation using an age-based criterion.

The configurators discussed above present different strategies to select configurations to evaluate. None of them, however, present an explicit prediction-based model for that. In this line, *model-based optimization* approaches use the ideas behind Bayesian optimization methods (MOCKUS, 1989; SHAHRIARI et al., 2016) for algorithm configuration. They use robust surrogate models to predict the performance of new configurations based on the results of previous evaluations. These predictions guide the selection of configurations, and help to avoid spending computational resources evaluating unpromising configurations. Some examples of model-based configurators are the Learn-and-Optimize (LaO) approach of Brendel and Schoenauer (2011), the BONESA configurator of Smit and Eiben (2011), and GGA++ (AN-

SÓTEGUI et al., 2015), an extended version of GGA that uses a random forest model in an attempt to guide the search to high-performance regions of the configuration space. The sequential parameter optimization (SPO) is one of the main configuration methods of this nature and is described in Bartz-Beielstein, Lasarczyk and Preuss (2005) and Bartz-Beielstein and Preuss (2006). Further extensions of SPO were proposed by Hutter et el. (2009a, 2010), and its implementation is available in the SPOT configurator (BARTZ-BEIELSTEIN; LASARCZYK; PREUSS, 2010; BARTZ-BEIELSTEIN et al., 2011). At the beginning, SPOT selects initial configurations using Latin hyper-cube sampling. In subsequent iterations, it uses the prediction model for that, which is fitted using the results of previous evaluations. In its first version, SPO uses regression models based on design and analysis of computer experiments (DACE), like linear and quadratic models, handling numerical parameters only. The current version uses a more robust random forest model (BREIMAN, 2001) that also supports categorical parameters. The main limitation of SPOT is that it only considers a single training instance.

To the best of our knowledge, the most prominent configurators currently available are ParamILS, SMAC, GPS and irace. They consistently present robust performance over different configuration scenarios, and are the currently most used configurators in the literature. The general operation and underlying techniques applied in each of them are discussed below.

ParamILS. ParamILS performs an iterated local search in the configuration space (HUTTER et al., 2009b). It starts evaluating a set of initial configurations, including a user-defined default configuration and some additional random configurations selected uniformly from the configuration space; the best performing one is used as the first incumbent configuration. Then, ParamILS executes a local search modifying one parameter at a time. Every time the local search finds an improving configuration, it replaces the current configuration and the order in which parameters are modified is shuffled. Once the local search reaches a local optimum, the locally optimal configuration is compared to the incumbent and the best one is kept. The local search is followed by a perturbation method that changes the values of a subset of parameters of the incumbent configuration and applies a new local search starting from the resulting perturbed configuration. With a low probability, the perturbation is replaced by a restart mechanism that generates a random configuration for the next iteration. ParamILS implements *racing* strategies to compare configurations.

The BasicILS version evaluates configurations on a fixed number of instances, and selects the one with lower cost estimate. The FocusedILS version uses a dominance criterion to compare configurations and iteratively increases the number of instances on which they are evaluated. A configuration θ_1 dominates another configuration θ_2 when it has been evaluated on at least the same number of instances as θ_2, and presents lower cost estimate for the number of instances on which θ_2 has been evaluated. If no dominance is determined, FocusedILS increases the number of instances on which θ_2 is evaluated and repeats the dominance test. Increasing the number of instances used to evaluate configurations over the configuration process is a *sharpening* strategy. Since configurations produced in latter iterations tend to outperform those produced earlier, increasing the number of instances (or repetitions) helps comparing different configurations and identifying the best ones. Finally, ParamILS also implements a so-called adaptive *capping* method for configuration scenarios involving the minimization of the target algorithm's running time. This method defines a running time limit (i.e. a capping time) for new configurations as the maximum time they could take to be better than the incumbent configuration. Then, executions of poorly-performing configurations are terminated early and the configuration budget can be spent evaluating better configurations.

SMAC. SMAC uses random forest models to guide the search for good configurations (HUTTER; HOOS; LEYTON-BROWN, 2011). It defines an initial configuration, usually a default or a randomly generated configuration (alternatively, a set of initial configurations can be used instead of a single one). This configuration is evaluated on a subset of the training instances, and the results are used to learn the random forest model. Its predictions are used to determine the expected improvement (as used in Schonlau, Welch and Jones (1998) and Jones, Schonlau and Welch (1998)) of all previously seen configurations, and the best ones are used as starting points of a local search procedure that iteratively selects the best improving neighbor with respect to the expected improvement. The resulting locally optimal configurations, in addition to a set of randomly generated ones (to diversify the search process), are sorted according to their expected improvement and evaluated, in the given order, on a subset of the training instances. When a given time limit for the evaluation step is reached, it is terminated and the evaluation results are used to update the random forest model for the next iteration, when new configurations are generated and the process repeats. Analogous to ParamILS, SMAC also maintains an incumbent

configuration and its evaluation process implements a racing procedure based on configuration dominance, using sharpening and capping strategies. Besides that, SMAC includes instance features as variables for the random forest model. Thus, it is able to identify high-performing configurations related to classes of instances. However, such instance features might not be readily available for many practical problems.

GPS. GPS stands for golden parameter search, an algorithm that optimizes the value of each parameter semi-independently (PUSHAK; HOOS, 2020). In a recent work, Pushak and Hoos (2018) studied the configuration landscapes of several prominent algorithms and observed that most of the parameters appear to have uni-modal responses when changed individually; this is exploited in GPS. It maintains a bracket for each parameter, which defines an interval or set of values believed to contain the optimum parameter value. Iteratively, GPS samples a parameter using a multi-armed bandit procedure, which prioritizes parameters believed to be important by giving more probability to select parameters whose value was updated more in the configuration process. Then, GPS races each pair of values taken from the bracket of the selected parameter, using a dominance-based criterion to compare different configurations. The selected parameter is updated in the incumbent configuration when there is statistically sufficient evidence that it improves its performance. Additionally, the new value must have been evaluated on a minimum number of instances, as well as on a super-set of the instances the last incumbent was evaluated when it became incumbent. GPS also implements a procedure to update the brackets, i.e. expand or shrink the interval or set of values, when there is sufficient evidence for improvement. For numerical parameters, this procedure is similar to the golden section search algorithm (KIEFER, 1953), which uses end and interior points that follow a so-called golden ratio to control and update the brackets, in order to focus the search near the best performing values. For categorical parameters, the bracket update consists in simply removing or re-adding values according to their performance in comparison to the performance of the incumbent configuration. Finally, GPS uses sharpening to update the number of instances configurations must be evaluated on, and implements a capping method to early terminate the evaluation of unpromising configurations.

irace. The irace configurator iteratively samples configurations, evaluates them using

racing and uses the results to update the sampling models (LÓPEZ-IBÁÑEZ et al., 2016). It also uses sharpening and capping (PÉREZ CÁCERES et al., 2017a) when evaluating configurations. We devote Section 2.2.2 to discuss irace in further detail.

We summarize all configurators discussed above and indicate their main limitations and features in Table 2.1, following three indicators: whether the configurator handles multiple instances, allows mixed-type parameters, i.e. numerical and categorical, and handles conditional parameters. For additional details about these methods, we refer to Hoos (2012a) or Stützle and López-Ibáñez (2019).

2.2.2 The irace Configurator

The irace configurator (LÓPEZ-IBÁÑEZ et al., 2016) is an iterated racing method based on the Friedman-Race (F-Race) proposed by Birattari et al. (2002) and the iterated F-Race (I/F-Race) proposed by Balaprakash, Birattari and Stützle (2007) (see Birattari et al. (2010) for a detailed discussion of both methods). Algorithm 1 shows the main steps of irace, which follow the schema presented in Figure 2.2. The algorithm maintains an elite set $\Theta^{\text{elite}} \subseteq \Theta$ of the best found configurations and a sampling model $\mathcal{M}$, which guides the generation of new configurations Θ' (`sample` procedure in line 4). The newly generated configurations are evaluated against the elite ones by racing (`race` procedure in line 5) on a subset of the instances Π according to the performance metric $c(\theta, i)$. The best found configurations form the elite set Θ^{elite}, which is used to update the sampling model $\mathcal{M}$ for the next iteration. The sampling, racing and model update steps are repeated until budget B is consumed. The number of iterations is determined by irace at the beginning of the configuration process, based on the number of parameters to be configured. The budget of an iteration is determined at the start of the iteration, based on the remaining budget available and the number of iterations to be executed next.

The `sample` procedure (line 4) is based on model $\mathcal{M}$. At the beginning, $\mathcal{M}$ samples the configuration space Θ uniformly. In subsequent iterations, the elite configurations are ranked according to the observed quality in previous evaluations, and iteratively one of them is selected to generate each new configuration θ. Elite configurations of a higher rank have a higher probability of being selected. Then, a new value of each parameter in θ is determined based on a truncated normal distribution for

Table 2.1 – Summary of configuration methods and their main limitations. Note that some methods with limitations could be adapted to handle either multiple instances, mixed-type parameters, or conditional ones; we present here the characteristics found in the referenced papers. The numbered references are: [1] Parsons and Johnson (1997); [2] Ridge and Kudenko (2006); [3] Ridge and Kudenko (2007); [4] Coy et al. (2001); [5] Adenso-Díaz and Laguna (2006); [6] Yuan et al. (2012); [7] Audet and Orban (2006); [8] Powell (2009); [9] Nelder and Mead (1965); [10] Park and Kim (1998); [11] Muja and Lowe (2009); [12] Hansen and Ostermeier (2001); [13] Hansen (2006); [14] Tolson and Shoemaker (2007); [15] Ansel et al. (2014); [16] Audet, Dang and Orban (2014); [17] Riff and Montero (2013); [18] Nannen and Eiben (2006); [19] Nannen and Eiben (2007); [20] Smit and Eiben (2009); [21] Ansótegui, Sellmann and Tierney (2009); [22] Ansótegui et al. (2015); [23] Hutter et al. (2009b); [24] Pushak and Hoos (2020); [25] López-Ibáñez et al. (2016); [26] Brendel and Schoenauer (2011); [27] Smit and Eiben (2011); [28] Bartz-Beielstein, Lasarczyk and Preuss (2010); [29] Bartz-Beielstein et al. (2011); [30] Hutter, Hoos and Leyton-Brown (2011).

Configurator/approach	Multiple instances	Mixed-type parameters	Conditional parameters
Experimental design			
DoE + RSM [1, 2, 3]	✓	✓	–
DoE + gradient descent [4]	✓	–	–
CALIBRA [5]	✓	–	–
Numerical optimization			
MADS [6, 7]	✓	–	–
BOBYQA [6, 8]	✓	–	–
Nelder-Mead Simplex [6, 9, 10, 11]	✓	–	–
CMA-ES [6, 12, 13]	✓	–	–
Heuristic search			
DDS [14]	–	–	–
OpenTuner [15]	–	✓	–
OPAL [16]	–	✓	–
EVOCA [17]	✓	✓	–
REVAC [18, 19, 20]	–	–	–
GGA/GGA++ [21, 22]	✓	✓	✓
ParamILS [23]	✓	–	✓
GPS [24]	✓	✓	✓
irace [25]	✓	✓	✓
Model-based optimization			
LaO [26]	✓	–	–
BONESA [27]	–	–	–
SPOT [28, 29]	✓	✓	–
SMAC [30]	✓	✓	✓

continuous parameters, or a discrete probability distribution for discrete parameters. Each elite configuration has its associated probability distributions for each parameter, and newly generated configurations inherit this set of probability distributions from

Algorithm 1: Iterated racing procedure.

Input : Configuration scenario $\langle \mathcal{A}, \Theta, \Pi, c, B, \mathcal{S} \rangle$.
Output : Best configurations found Θ^{elite}.

1 $\Theta^{\text{elite}} \leftarrow \varnothing$
2 $\mathcal{M} \leftarrow \texttt{initialize}(\Theta^{\text{elite}})$
3 **repeat**
4 $\quad \Theta' \leftarrow \texttt{sample}(\Theta, \mathcal{M})$ ▷ *Selection step in Fig. 2.2*
5 $\quad \Theta^{\text{elite}} \leftarrow \texttt{race}(\Theta' \cup \Theta^{\text{elite}}, \Pi, c)$ ▷ *Evaluation step in Fig. 2.2*
6 $\quad \mathcal{M} \leftarrow \texttt{update}(\Theta^{\text{elite}})$ ▷ *Update step in Fig. 2.2*
7 **until** *budget B is consumed*
8 **return** Θ^{elite}

their parents. In the `update` procedure (line 6), model $\mathcal{M}$ is updated with the new elite set Θ^{elite}, and the parameters of their probability distributions are updated to focus the sampling process around the best parameters values as the configuration process evolves.

The `race` procedure evaluates the quality of the new and elite configurations (Θ' and Θ^{elite}, respectively) on a subset of the instances Π according to the cost metric c. After evaluating each configuration on a predefined number of initial instances, irace uses either the non-parametric Friedman test (associated with the post-hoc test described by Conover (1999)) or the paired t-test to identify worse configurations to be discarded. The racing method in irace extends I/F-Race (BALAPRAKASH; BIRATTARI; STÜTZLE, 2007) by reusing evaluations from previous races in the current race and by preventing elite configurations from being discarded without considering all their evaluations from previous races. The surviving configurations are evaluated on a new instance, and a new statistical test is performed. This process is repeated until the budget of the iteration is consumed, or a minimum number of surviving configurations remains. The surviving configurations become the elite set for the next iteration.

The updates of the probability distributions may lead to a premature convergence of the configuration process. In this case, the newly generated configurations are very similar to those already evaluated and the configuration process loses diversity. To avoid this, irace implements a convergence detection mechanism that compares each new configuration to the elite configuration used to generate it. The comparison is carried out by calculating their distance, based on the differences of the parameter values presented by both configurations. If this distance is less

than a threshold, irace performs a soft restart that updates the parameters of the sampling distributions associated to that elite configuration, in order to increase the probability of generating different configurations.

2.2.3 Some Applications of Automatic Algorithm Configuration

This section summarizes some applications of automatic configuration methods in different contexts and problem domains. It is not intended to be an exhaustive literature review, but aims at giving a notion of the applicability, contributions and significance of such techniques and, more generally, of this research field.

Automatic methods are widely applied to configure algorithms from a diversity of domains, reaching significant performance improvements when compared to default configurations or those obtained by a manual configuration process. We present some examples to give the reader an overview of such applications. In the case of configuring (exact and heuristic) optimization algorithms, several problems have been explored, like the traveling salesperson problem (LÓPEZ-IBÁÑEZ et al., 2013), machine reassignment (MALITSKY et al., 2013), online bin packing (YARIMCAM et al., 2014), graph coloring (BLUM; CALVO; BLESA, 2015), permutation flowshop scheduling (BENAVIDES; RITT, 2015), university course timetabling (MÜHLENTHALER, 2015), quadratic assignment (FRANZIN; STÜTZLE, 2019), multiobjective problems (DUBOIS-LACOSTE; LÓPEZ-IBÁÑEZ; STÜTZLE, 2011), and continuous function optimization (LIAO; MONTES DE OCA; STÜTZLE, 2013). In some cases, the improvements obtained by automatic configuration methods are even bigger when the target algorithms are applied to contexts different than those for which they were designed in the first place, like using different problem instances or changing the termination criterion. As an example, Pérez Cáceres, López-Ibáñez and Stützle (2015) configured ant colony optimization algorithms with a limit in the number of objective function evaluations. They found configurations with better performance than the one obtained by the best known configurations for the version of the algorithm without this modified termination criterion.

Regarding the configuration of decision algorithms, much effort has been spent in improving existing algorithms for boolean satisfiability problems (see Hoos, Hutter and Leyton-Brown (2021) for a recent survey). Additional examples include Hutter et al. (2007, 2017), Falkner, Lindauer and Hutter (2015), Hutter et

al. (2017), Pérez Cáceres et al. (2017a), Dang et al. (2017). We also include in the list of decision-based scenarios successful applications on the configuration of compilers (PÉREZ CÁCERES et al., 2017), mathematical solvers (HUTTER; HOOS; LEYTON-BROWN, 2010; LÓPEZ-IBÁÑEZ; STÜTZLE, 2014; PÉREZ CÁCERES; STÜTZLE, 2017), and AI planners (VALLATI et al., 2013). Another direction is the automatic configuration of machine learning algorithms (BERGSTRA et al., 2011; MIRANDA; SILVA; PRUDÊNCIO, 2014; BISCHL et al., 2016), also called *hyperparameter optimization* in the corresponding literature. Along this line, Hutter, Kotthoff and Vanschoren (2019) present a detailed review of existing techniques and open research directions related to the so-called automated machine learning (AutoML).

In addition to optimizing the algorithm performance, automatic configuration methods contribute in different ways to the algorithm design process. First, they ease the analysis of the contributions of particular parameters and design choices to the algorithm performance, which helps to understand such algorithms and guides their development. For example, Massen et al. (2013) use irace to generate high-quality configurations for a given hybrid heuristic; then a sensitivity analysis is performed to measure the significance of each parameter. Bezerra, López-Ibáñez and Stützle (2014b) analyze the contribution and interaction between different design choices and parameter values of multiobjective evolutionary algorithms. Franzin and Stützle (2019, 2021) explore similar ideas to study simulated annealing algorithms on different problem domains.

A second contribution of automatic configuration methods is to allow fairer comparisons between different algorithms. Once the configuration spaces of the algorithms have been defined, one can configure them using equivalent budgets prior to evaluation, in order to obtain high-quality configurations and avoid biases that could be inserted due to human expertise. Besides that, as stated above, configuration is necessary when applying an algorithm to different conditions than those for which it was designed. In Pellegrini and Birattari (2007) and Liao, Molina and Stützle (2015), authors compare the performance of different algorithms under the default and automatically produced configurations, showing the significant performance improves obtained after the configuration process. The latter work also shows how varying the benchmark instances affects the performance of different configurations.

The widespread use of automatic algorithm configuration can be observed

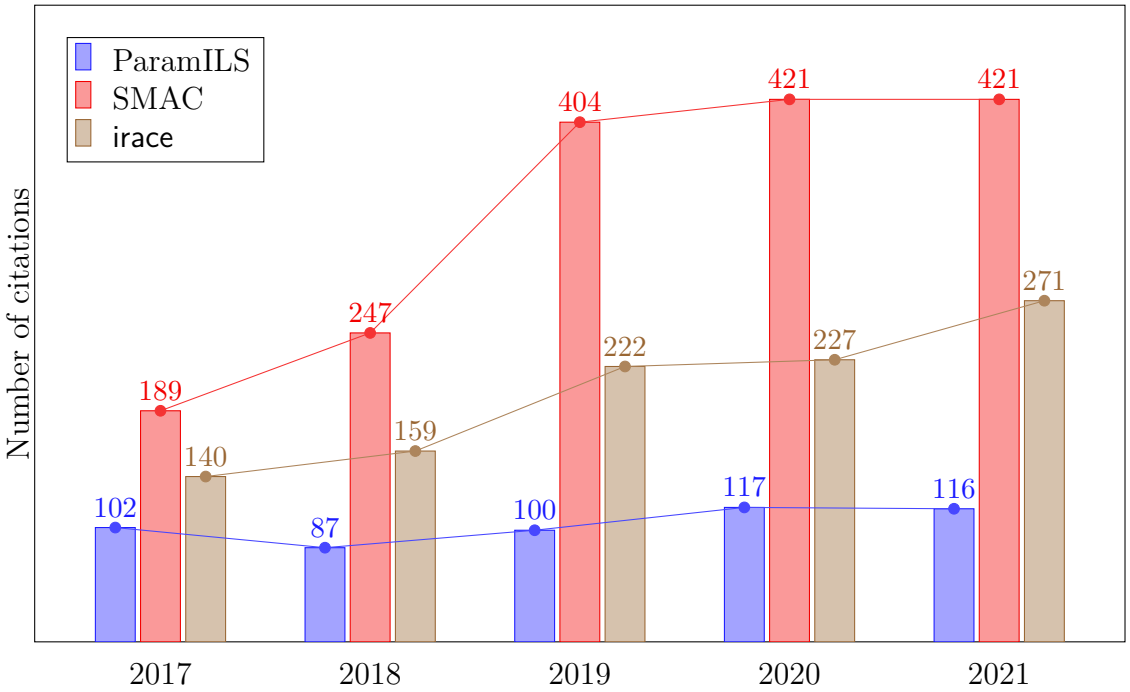


Figure 2.3 – Number of citations of ParamILS, SMAC and irace configurators in the last five years, according to Google Scholar; accessed in January 2022.

in the number of scientific publications related to this field. ParamILS, SMAC and irace have a total of 1024, 2108 and 1287 citations[4], respectively, according to Google Scholar (up to January 2022). Figure 2.3 shows the number of citations obtained by these configurators from 2017 to 2021. Considering the sum over the different configurators, we observe an increase in citations over the years, which indicates a notable interest of the scientific community, and also the relevance of automatic configuration in the algorithm design process. These observations justify the efforts in developing, improving and understanding automatic configuration methods.

2.3 Automatic Algorithm Design

Usually, algorithm configuration represents the last step of the algorithm design process, i.e. when the values of the parameters of a previously designed algorithm must be set. Here, automatic configuration helps not only by automating this task, but also by defining a more appropriate way of maintaining algorithms. Whenever a change is made in the algorithm or execution environment (e.g. including

[4]We consider the following publications: Hutter et al. (2009b) for ParamILS; Hutter, Hoos and Leyton-Brown (2011) for SMAC; and López-Ibáñez et al. (2011, 2016) for irace.

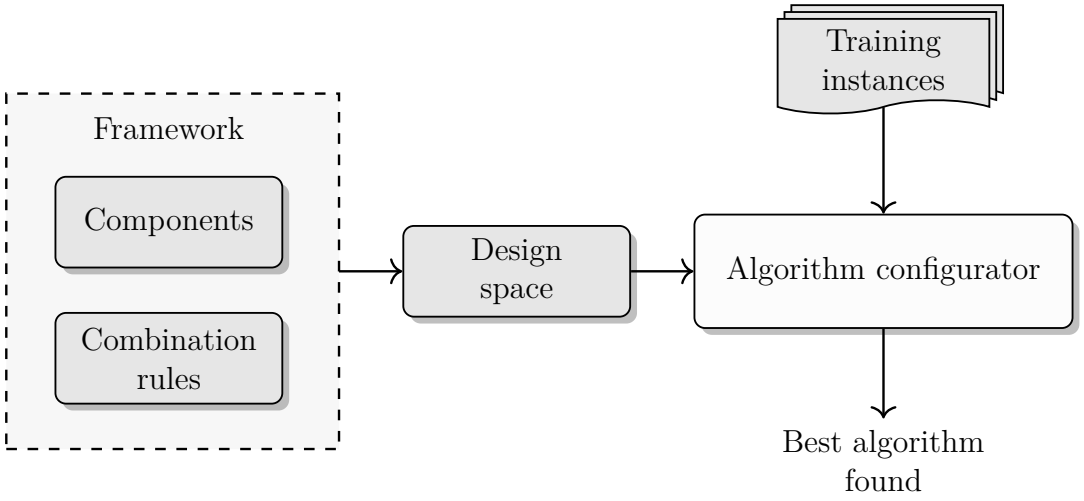


Figure 2.4 – General schema for the automatic algorithm design from components. The algorithm framework provides a set of components and the combination rules to determine how they can be selected and combined into complete algorithms. The framework defines the design space, which is explored by the automatic configurator to find the algorithm with optimized performance on a given set of training instances.

a new algorithm component or defining different termination criteria), the internal behavior of the algorithm or its execution conditions may change and the previously defined configurations may perform worse. Therefore, it is advisable to reconfigure the algorithm (STÜTZLE, 2009), and some works propose using automatic configuration in this iterative algorithm design process (MONTES DE OCA; AYDIN; STÜTZLE, 2011). A more general, sophisticated and even ambitious approach concerns in exposing not only the parameters of the algorithm, but also higher-level design choices (e.g. which search procedure or recombination method to use), and then applying existing configuration methods to make such choices, automating other steps, and even the entire algorithm design process. This research field is called *automatic algorithm design*[5], and its general idea is depicted in Figure 2.4. First, it uses a (preferably flexible) configurable algorithm framework of components, which allows selecting and combining such components to produce complete algorithms. The components, their input parameters and the combination rules form the configuration space, which we call *design space* in the context of automatic algorithm design. Now, we can apply automatic configuration methods to explore this design space and search for the best algorithms for a given set of (training) problem instances. Several approaches have been developed in this direction and applied to automatically design

[5]Other names for automatic algorithm design are commonly found in the literature, like *programming by optimization* (HOOS, 2012b), *automatic algorithm synthesis* (DI GASPERO; SCHAERF, 2007), or *automatic programming* (OLSSON; LØKKETANGEN, 2013).

complex algorithms for several different domains. We discuss automatic algorithm design below and give an overview of applications. For further details, we refer to Stützle and López-Ibáñez (2019).

The main task of the algorithm designer is to provide the design space, i.e. the set of algorithm components, and the implementation of the underlying algorithm framework that allows combining such components and instantiate complete algorithms. Once the design space, a set of training instances, and the configuration setup (i.e. budget, performance metric and other elements of the configuration scenario) are given, the rest of the design process can be fully automated. A possible approach to define the design space is by extracting the main components from state-of-the-art algorithms of a given domain of interest. The major benefit of this practice is that such components are usually high performing and well designed, at least when considering the problem domains from which they are being extracted. By defining a design space from such components, one may not only recreate algorithms from the literature, but also combine them with new components to produce hybrid and potentially better performing algorithms, in some cases reaching previously unexplored designs.

Existing approaches for automatic algorithm design differ from each other in the structure and representation of the design space. We classify them in *top-down* and *bottom-up* approaches.

2.3.1 Top-Down Algorithm Design

Top-down approaches define a static algorithm template in which specific points are left open, allowing one to choose between different alternative designs, i.e. selecting the desired algorithm components. Choosing specific components instantiates a particular version of the algorithm defined by the template. Algorithm 2 defines an exemplary template for an iterated local search: after generating an initial solution (line 1), the algorithm iteratively perturbs the current solution and applies an improvement step (lines 3 and 4), maintaining the best solution found during the search; an acceptance criterion determines whether the resulting solution replaces the current one (line 5). In this example, procedures initialize, perturb, improve and accept do not define any specific component. Instead, there are several alternative choices for each of them (examples are presented as comments after each procedure), e.g. we

Algorithm 2: Template for an iterated local search.

```
1 s ← initialize()                          ▷ random, greedy, ...
2 while termination criterion not met do
3     s' ← perturb(s)                       ▷ random, directed, ...
4     s'' ← improve(s')                     ▷ local search, tabu search, ...
5     s ← accept(s, s'')                    ▷ never, always, better, ...
6 return best solution found
```

can choose between a random or greedy strategy to produce the initial solution, or different search methods for the improvement step. Now, by fixing the desired component for each design choice, we instantiate different iterated local searches, and can search for specific versions with a good performance. The number of different iterated local search algorithms we can produce from this template is given by the total possible combinations of design choices.

The structure of the algorithm template defines the rules for combining the different components available. In this case, each design choice is represented by a categorical parameter. For example, the template of Algorithm 2 defines a design choice for the initialization procedure (line 1), which can be represented by a categorical parameter with the available components (e.g. *random*, *greedy*, ...). The choice of particular components can enable lower-level design choices, and even component-specific input parameters. For example, by choosing a *tabu search* as improvement procedure (line 4), we may need to choose between different neighborhood operators (i.e. a new design choice with possibly several alternative components) and a value for the tabu tenure parameter. In such cases, these choices are represented by conditional categorical or numerical parameters. Therefore, we can use a direct parametric representation for a given algorithm template, which enables us to use automatic configurators to perform the exploration of this design space.

There are several examples of top-down algorithm design in the literature. The SATenstein framework of Khudabukhsh et al. (2009, 2016) implements various local search heuristics extracted from the literature, and provides an algorithm template to guide their combination, allowing to produce different algorithms for the boolean satisfiability (SAT) problem. By applying automatic techniques to explore the design space of SATenstein, the authors could derive local search algorithms able to outperform the previous state of the art. Franzin and Stützle (2019) analyze the

literature of simulated annealing (SA), and extract both general and problem-specific components to build a configurable algorithm framework. They were able to study different variants of SA present in literature and improve their performance by automatically designing new SA implementations. Several works address the automatic design of heuristic algorithms for continuous optimization problems. They propose algorithm templates for different metaheuristics, including ant colony optimization (UACOR approach of Liao et al. (2014)), bee colony optimization (ABC-X approach of Aydın, Yavuz and Stützle (2017)), and particle swarm optimization (PSO-X approach of Villalón, Dorigo and Stützle (2021)). In all these examples, the resulting algorithm framework is built with components from the literature, and the application of automatic methods to explore the design space leads to new state-of-the-art designs. Finally, much effort has been spent in automatically designing algorithms to tackle multi-objective optimization problems. López-Ibáñez and Stützle (2010, 2012) automatically design multi-objective ant colony optimization (MOACO) algorithms to solve the bi-objective traveling salesperson problem. Bezerra et al. (2014a, 2016, 2020a, 2020b) study the automatic design of a more general class of multi-objective evolutionary algorithms (MOEA), considering components of various evolutionary approaches from the literature and producing new state-of-the-art algorithms.

2.3.2 Bottom-Up Algorithm Design

In the top-down approaches discussed above, the rules for combining components and building algorithms are given explicitly by the algorithm template. Considering all possible design alternatives and their interactions during the definition of such templates is increasingly hard as the number of design choices grows. Hence, an alternative bottom-up approach concerns providing algorithm components and allowing their composition in a more flexible way at the time the algorithm is instantiated. In this case, the algorithm template is not fixed and there is higher flexibility when combining components, which may lead to completely new designs and hybrid algorithms.

Since bottom-up approaches do not use a fixed algorithmic structure, the combination rules need to be defined. A common solution is using a context-free grammar in Backus-Naur form, which describes how an algorithm can be instantiated out of a set of algorithm components. Figure 2.5 presents a context-free grammar

```
1   ⟨search⟩      →  ls(⟨improvement⟩) | ts(⟨improvement⟩, ⟨tenure⟩)
2                    | sa(⟨improvement⟩, ⟨temp⟩) | ils(⟨search⟩, ⟨pert⟩)
3   ⟨improvement⟩ →  fi | bi | si
4   ⟨pert⟩        →  random(⟨size⟩) | directed(⟨size⟩)
5   ⟨tenure⟩      →  [10, 1000]
6   ⟨temp⟩        →  [10, 5000]
7   ⟨size⟩        →  {1, 10, 50, 100}
```

Figure 2.5 – An example of a context-free grammar of components. Note that the grammar is recursive in the rule of lines 1 and 2, allowing flexible combinations of components into hybrid algorithms.

for an exemplary design space of components to generate search-based heuristic algorithms. The grammar consists of a set of rules, each one describing a decision, i.e. a component to be chosen. The start symbol is the non-terminal ⟨search⟩ in line 1 (note that non-terminal symbols are presented between angular brackets). Starting from ⟨search⟩, four different search components can be chosen: local search (ls), tabu search (ts), simulated annealing (sa) and iterated local search (ils). Depending on the component chosen, new non-terminals are reached and additional decisions are needed. Besides that, the components' parameters can be also included in the grammar, e.g. those given by the non-terminals ⟨tenure⟩, ⟨temp⟩ and ⟨size⟩. A complete derivation of the grammar instantiates a concrete algorithm, and we can combine components in a flexible way into hybrid algorithms. For example, the grammar defines an iterated local search component (ils in line 2) with an internal search procedure, and allows us to freely choose any search component for this step, like tabu search, simulated annealing or even another iterated local search.

The design space defined by the grammar must be mapped into a parametric representation to allow using automatic configuration methods to explore it. A *codon-based mapping* is proposed in the context of grammatical evolution (RYAN; COLLINS; NEILL, 1998), an early grammar-based automatic design approach that uses evolutionary algorithms to explore the design space (grammatical evolution is discussed in Section 2.4). The codon-based mapping consists in defining a sequence of integer values (*codons*) to map the decisions for the rules of the grammar. To instantiate an algorithm, whenever a design choice must be made, the next codon c is consumed and the $c \bmod n$ alternative is selected, where n is the number of

alternatives for the corresponding design choice and `mod` is the integer modulo operation. Since the codon-based mapping is basically a sequential representation, usually expressed by a binary string, the use of evolutionary algorithms to explore the design space is straightforward. However, the codon-based mapping suffers from low locality and high redundancy problems. As discussed in Rothlauf and Oetzel (2006), a mapping is said to have a high locality when a small change in the choices representation leads to a small change in the produced algorithm. This is clearly not the case when using a codon-based mapping, e.g. by changing the value of the first codon from 0 to 1 we change the decision for the first rule in our exemplary grammar (line 1 in Figure 2.5), producing a completely different algorithm (a tabu search instead of a local search). Redundancy is a consequence of using integer codons with a fixed interval, say $[0, 255]$. If a rule has less than 256 alternative choices, which is usually the case, many different values maps to the same choice. Finally, when a derivation is finished but codons are still available, these approaches may ignore the remaining ones. In contrast, when there are still choices to be made but no codons, wrapping is usually allowed and the first codons are reused. As a consequence, changing values for one codon can produce different decisions for more than one design choice, increasing problems of low locality.

Lourenço, Pereira and Costa (2016) propose the structured grammatical evolution approach, addressing these problems. In their structured representation, each non-terminal is linked to a specific codon, increasing locality. This allows adjusting the interval of values of each codon according to the number of alternative choices of the corresponding non-terminal, eliminating redundancy. In case a non-terminal is expanded more than once, the corresponding codon stores a list of values, one for each possible expansion. Mascia et al. (2013, 2014b) propose to map the grammar rules directly as categorical and numerical parameters, and then use automatic configuration methods to explore the design space. This parametric mapping allows defining conditionality, i.e. since we know the conditions under which a non-terminal is expanded, we can represent such cases and require values for certain parameters only when needed. For example, in the grammar of Figure 2.5, a decision value for the non-terminal ⟨pert⟩ is only needed when ils is chosen for non-terminal ⟨search⟩. When representing such decisions by parameters, we define a categorical parameter pSearch to decide over non-terminal ⟨search⟩, and another categorical parameter pPert to decide over non-terminal ⟨pert⟩, which is required only when

pSearch = 'ts', i.e. pPert is conditional to pSearch. This reduces the size of the design space and, consequently, the complexity of the automatic design task. However, in case a non-terminal is expanded more than once, a set of parameters must be created. Thus, to keep a finite number of parameters, a maximum level of recursion has to be defined. Mascia et al. (2013, 2014b) show that their approach outperforms grammatical evolution when automatically designing heuristic algorithms for different problem domains.

We find several examples of bottom-up algorithm design in the literature. Mascia et al. (2013, 2014b) applied the approaches discussed above to automatically design iterated greedy algorithms for permutation flowshop scheduling and bin packing problems. Their framework was extended with additional components, allowing to produce hybrid stochastic local search algorithms. Different works used this approach to the automatic design of algorithms for different problem domains, including permutation flowshop scheduling, unconstrained binary quadratic programming and the traveling salesperson problem (MARMION et al., 2013; MASCIA et al., 2014a; LÓPEZ-IBÁÑEZ; MARMION; STÜTZLE, 2017). In more recent studies, these ideas were used to automatically design heuristic algorithms for permutation flowshop scheduling with different objective functions (BRUM; RITT, 2018a; BRUM; RITT, 2018b; PAGNOZZI; STÜTZLE, 2019; PAGNOZZI; STÜTZLE, 2021), and for hybrid flowshop problems (ALFARO-FERNÁNDEZ et al., 2020). These works present extensive algorithm frameworks with heuristic components from the associated literature, and produced algorithms with better performance than state-of-the-art approaches.

2.4 Related Problems

In previous sections, we discussed research efforts related to the general field of automatic design and configuration of algorithms, which are closely connected to the approaches we follow in the rest of this work. Nevertheless, there are a number of research fields extending the algorithm configuration problem, with active research communities working on and contributing to the goal of producing better algorithms while reducing human effort. We briefly introduce these topics below, making the proper distinction with the ideas previously discussed, presenting some application examples and indicating references for further reading.

Online algorithm configuration. Also known as *dynamic algorithm configuration* or *parameter control*, online configuration methods start with initial parameter values and adapt them during the algorithm execution. Therefore, they integrate the configuration process into the algorithm execution in order to not only find the best parameter values, but determine a strategy to update them throughout the algorithm execution, adapting to the observed execution conditions and the instance being solved. Historically, online configuration has been widely applied to evolutionary algorithms (EIBEN; HINTERDING; MICHALEWICZ, 1999; KARAFOTIAS; HOOGENDOORN; EIBEN, 2015; ALETI; MOSER, 2016) and several studies show that updating parameter values at different stages of the evolutionary process achieves better performance in comparison to using fixed parameter values (BÄCK, 1992; SMITH, 2001). Applications include the configuration of algorithms for AI planning (SPECK et al., 2021), computer vision (CHAU et al., 2014) and deep learning (DANIEL; TAYLOR; NOWOZIN, 2016).

Instance-specific algorithm configuration. The configuration approaches previously discussed try to find a configuration with the best performance over the whole set of training instances. However, as stated by the no free lunch theorem (WOLPERT; MACREADY, 1997) and observed in practice (MUJA; LOWE, 2009; SMITH-MILES et al., 2014), there is no single configuration (or algorithm) with optimized performance over all instances. Hence, increasing performance on certain instances usually leads to a corresponding loss of performance on others. Instance-specific algorithm configuration approaches rely on a set of features describing the problem instances, and determine the best performing configurations according to the feature values. A first approach uses empirical performance models to select the most promising configuration when solving a new instance (HUTTER et al., 2006). Kadioglu et al. (2010) cluster the training instances according to their features and determine the best configuration for each cluster. A new instance being solved is first associated to a cluster according to its feature values, and the corresponding configuration is used to solve it. Instance-specific algorithm configuration has been used in different problem domains, like boolean satisfiability (ANSÓTEGUI; MALITSKY; SELLMANN, 2014), mixed-integer programming (KADIOGLU et al., 2010) and black-box optimization (BELKHIR et al., 2017). We revisit instance-specific algorithm configuration in Chapter 5, in which we propose a set of models to determine parameter values as functions of the instance size. Hence, Chapter 5

discusses this topic in detail and reviews the corresponding literature.

Algorithm selection. In the algorithm selection problem (RICE, 1976) there is a set of problem instances and a set of algorithms, commonly called *algorithm portfolio*. The objective is to find the best algorithm for solving the problem instances, according to a given performance metric. Recent approaches define the selection rules on a per-instance basis, where a model predicts the most promising algorithm based on the features of the given instance (LEYTON-BROWN et al., 2003). Besides that, algorithm selection is commonly combined with algorithm configuration in more sophisticated approaches. In particular, several works propose an offline step to automatically construct the algorithm portfolio by means of instance-specific algorithm configuration, combined with an online strategy to select an algorithm when solving a new given instance. These approaches have been applied to different problem domains, including boolean satisfiability (XU et al., 2008; XU; HOOS; LEYTON-BROWN, 2010; MALITSKY; MEHTA; O'SULLIVAN, 2013), answer set programming (GEBSER et al., 2013), constraint solving (O'MAHONY et al., 2008), and the traveling salesperson problem (KERSCHKE et al., 2018). Kerschke et al. (2019) present a comprehensive review of automatic algorithm selection and discuss several applications.

Hyper-heuristics. Hyper-heuristics are higher-level search or learning methods that produce lower-level heuristics for solving computational search problems. Some hyper-heuristics select one or a set of heuristics to solve a given problem, while more sophisticated approaches generate new heuristics from a set of heuristic components using genetic programming (discussed below). Hence, hyper-heuristics can be seen as automatic algorithm selection or design approaches, but are usually applied to small scenarios and the selection or generation of simple heuristics. Burke et al. (2013) present a comprehensive literature review on hyper-heuristics, discuss a classification of the existing methods and present several applications of both selection and generation hyperheuristics. More recent articles from Burke et al. (2019) and Drake et al. (2020) also discuss hyper-heuristics and present advances in the field.

Genetic programming. Genetic programming is a bottom-up approach for automatic algorithm design, and can be seen as generation-based hyper-heuristic. However, genetic programming is applied to the design of not only heuristics, but a

number of types of algorithms. Unlike the approaches for automatic algorithm design previously discussed, the algorithm components tackled by genetic programming usually have a much smaller granularity, like simple operators, expressions or commands. Besides that, the design space exploration is performed by a genetic algorithm. The traditional tree-based genetic programming represents algorithms using syntax trees, while some extensions propose alternative representations, like using linear sequences of instructions (linear genetic programming, BRAMEIER; BANZHAF, 2007) or grammars (grammatical evolution, discussed below). For details on genetic programming and application examples, we refer the reader to Koza (1992), Poli, Langdon and McPhee (2008) and Poli and Koza (2014)

Grammatical evolution. Also called *grammar-based genetic programming*, grammatical evolution (RYAN; COLLINS; NEILL, 1998; O'NEILL; RYAN, 2001) is an automatic algorithm design approach whose design space is represented by grammars and explored by genetic algorithms[6]. As previously discussed, grammatical evolution first used a simple codon-based solution representation that leads to low locality and high redundancy (ROTHLAUF; OETZEL, 2006). More recent approaches follow the structured grammatical evolution proposed by (LOURENÇO; PEREIRA; COSTA, 2016) that use a direct association of codons to each decision of the grammar. Grammatical evolution was used in different problem domains, including the automatic generation of local search heuristics (BURKE; HYDE; KENDALL, 2012) and ant colony optimization algorithms (TAVARES; PEREIRA, 2012). For additional information about grammatical evolution and applications, we refer the reader to Ryan, O'Neill and Collins (2018).

[6]Some works propose alternative search algorithms to explore the design space in grammatical evolution, e.g. Hyde, Burke and Kendall (2013).

3 ALGORITHM CONFIGURATION SCENARIOS

There is no higher or lower knowledge, but one only, flowing out of experimentation.

— Leonardo da Vinci

This chapter introduces the algorithm configuration scenarios used to evaluate the methods proposed in this thesis (Section 3.1). For each scenario, we present the underlying problem, target algorithm, and problem instances. We also discuss characteristics of the configuration space (e.g. number and type of parameters) and additional details (e.g. configuration budget and termination criteria). We conclude the chapter with a summary of characteristics of each configuration scenario and a brief comparison among them in Section 3.2. All configuration scenarios are available at Souza and Ritt (2022a)[1], including algorithm implementations, execution scripts, problem instances, parameter descriptions and the configurator setup.

3.1 Description of Scenarios

We propose a set of configuration scenarios comprising different characteristics: heuristic and exact algorithms; mostly optimization, but also one decision problem, to evaluate the contributions of visualizations of configuring algorithms for both performance metrics (solution cost and solving time); deterministic and stochastic algorithms; small, medium and large configuration spaces; and different effort measures, especially to evaluate the proposed capping methods. In summary, the configuration scenarios described in this chapter were used in the following parts of this research:

- Chapter 4 (capping): ACOTSP, HEACOL, TSBPP, HHBQP, LKH and SCIP;
- Chapter 5 (regression models): ILSBQP, BSFS, HHTA and TSCPP;
- Chapter 6 (visualizations): ACOTSP and SPEAR;
- Chapters 7 and 8 define their own configuration scenario for the automatic design of heuristic algorithms from components; we describe it in detail in Chapter 7.

[1]The algorithm configuration scenarios can be downloaded at https://github.com/souzamarcelo/ac-scenarios.

For the scenarios using a running time limit as termination criterion, we define such time limit considering their execution on particular machine specifications. The running time limits of scenarios ACOTSP, HHBQP and LKH are based on their execution on a single core of a computer with an 8-core AMD FX-8150 processor running at 3.6 GHz and 32 GB main memory, under Ubuntu Linux. On the other hand, the running time limits of scenarios SCIP, ILSBQP, BSFS, HHTA and TSCPP are based on their execution on a single core of a computer with a 12-core AMD Ryzen 9 3900X processor running at 3.8GHz and 32GB main memory, under Ubuntu Linux.

3.1.1 Scenario ACOTSP

ACOTSP implements several ant colony optimization (ACO) algorithms for the symmetric traveling salesperson problem (TSP) (BONDY; MURTY, 1976). All algorithms are described in Dorigo and Stützle (2004), and the source code can be obtained in Stützle (2002). ACOTSP is part of the AClib benchmark library for algorithm configuration (HUTTER et al., 2014) and is widely used as a testbed for studying automatic algorithm configuration (see López-Ibáñez and Stützle (2014), Pérez Cáceres, López-Ibáñez and Stützle (2014, 2015), and López-Ibáñez, Stützle and Dorigo (2018) for some examples). We use ACOTSP version 1.03.

This scenario has 11 parameters, 5 of them being conditional. We use 60 seconds of wall clock time as termination criterion of ACOTSP and define 2000 executions as budget for the configuration process. We use Random Uniform Euclidean TSP instances of size 2000 (RUE-2000), as used in López-Ibáñez et al. (2016). We randomly generated 50 training instances and 200 test instances, using the *portgen* instance generator from the 8th DIMACS Implementation Challenge (JOHNSON et al., 2001). This generator distributes cities in a square uniformly at random, and computes the Euclidean distances between every pair of cities to determine the distance matrix.

3.1.2 Scenario HEACOL

HEACOL implements a hybrid evolutionary algorithm (HEA) for the graph coloring (COL) problem (BONDY; MURTY, 1976; LEWIS, 2016a). This algorithm

was proposed by Galinier and Hao (1999) and is described in Lewis (2016a). It keeps a population of solutions and combines an evolutionary approach based on a problem-specific recombination operator, with a local search procedure. The source code of HEACOL is provided by Lewis (2016b).

The HEACOL configuration scenario has 7 unconditional parameters. The termination criterion is the maximum number of constraint checks. A constraint check happens whenever the algorithm requests some information about the instance, e.g., whether two vertices are neighbors. We use a termination condition of 10^9 constraint checks and a configuration budget of 2000 executions. For the training, we generated 27 instances consisting of 3 randomly generated graphs for each combination of sizes $n \in \{250, 500, 1000\}$ and densities $d \in \{0.1, 0.5, 0.9\}$, where each of pair of vertices are made adjacent with probability d. For the test, we use the 79 well known graph instances available in Trick (2018), with sizes ranging from 11 to 1000 vertices and different structures, e.g. random and Latin square graphs.

3.1.3 Scenario TSBPP

This scenario concerns a tabu search (TS) algorithm for the two- and three-dimensional bin packing problems (BPP) (DELORME; IORI; MARTELLO, 2016) proposed by Lodi, Martello and Vigo (1999). The source code is described in Lodi, Martello and Vigo (2004a) and is available at Lodi, Martello and Vigo (2004b).

This scenario has 6 unconditional parameters, and the termination criterion is the number of iterations of the tabu search. We use a maximum of 5000 iterations. TSBPP is a deterministic algorithm and since it has only 6 parameters, we set a configuration budget of 500 executions only. We use the instances of the two-dimensional bin packing problem (2BPP) proposed by Berkey and Wang (1987) and Martello and Vigo (1998), which are divided in ten different classes with 50 instances each. A complete description of them can be found in Lodi, Martello and Vigo (1999). All 500 instances are used for the test phase, and 20 out of them were selected for the training phase (we randomly selected two instances of each class). Additional information about these and other instances, as well as other approaches to solve the BPP can be found in Delorme, Iori and Martello (2018).

3.1.4 Scenario HHBQP

This scenario consists in a hybrid heuristic (HH) algorithm to solve the unconstrained binary quadratic programming (UBQP). See Kochenberger et al. (2014) and Beasley (1998) for a review on UBQP and different approaches for solving it. This algorithm was automatically produced using our component-wise solver for binary problems (described in Chapters 7 and 8) that, in summary, uses irace to explore the grammar-based design space of heuristic components from the literature of UBQP (PALUBECKIS, 2006; GLOVER; LÜ; HAO, 2010; WANG et al., 2012). The source code is available at Souza and Ritt (2018d).

The scenario has 14 parameters, with 7 being conditional. We use a time limit of 20 seconds for the training executions, 30 seconds for the executions in the test phase, and a configuration budget of 2000 total executions. For the test phase, we use the 10 instances of size 2500 of Beasley (1998). They can be downloaded from Wiegele (2007b) and more details can be found in Wiegele (2007a). For the training phase, we randomly generated 9 instances with the same structure as Beasley's instances. We generated 3 instances for each size $n \in \{2000, 2500, 3000\}$, with a density of 0.1 and integer coefficients uniformly sampled within $[-100, 100]$.

3.1.5 Scenario LKH

This scenario concerns the configuration of the Lin-Kernighan-Helsgaun (LKH) algorithm for traveling salesperson problems (TSP). The LKH algorithm is an iterated local search based on the Lin-Kernighan heuristic (LIN; KERNIGHAN, 1973). The algorithm and an effective implementation are described in Helsgaun (2000, 2009, 2018a). The source code is available in Helsgaun (2018b). We use LKH version 2.0.9.

The LKH scenario has 21 unconditional parameters. We set a time limit of 10 seconds as termination criterion, and a configuration budget of 2000 total executions. We use Random Uniform Euclidean TSP instances of sizes 1000, 1500, 2000, 2500 and 3000 (RUE-1000-3000), randomly generated using *portgen* (JOHNSON et al., 2001). We generated 50 training instances (10 for each size) and 250 test instances (50 for each size).

3.1.6 Scenario SCIP

This scenario consists in the configuration of SCIP (Solving Constraint Integer Programs), an open-source exact solver for mixed integer programming (ACHTERBERG, 2009). We configure SCIP for solving the combinatorial auction winner determination problem (DE VRIES; VOHRA, 2003). SCIP was previously configured in López-Ibáñez and Stützle (2014). We use the same version of SCIP (2.0.2) together with the linear programming solver SoPlex 1.5.0. We also set the maximum memory to be used by SCIP to 350 MB.

We consider the same 207 unconditional categorical parameters used in López-Ibáñez and Stützle (2014). Although SCIP is an exact solver, we run it for a given running time limit and measure the cost of the best found solution, using it as performance metric for the configuration process. We set a time limit of 60 seconds for each execution of SCIP, and a configuration budget of 2000 total executions. We use the instances introduced in Leyton-Brown, Pearson and Shoham (2000). We randomly selected 50 training instances and 50 test instances with 200 goods and 1000 bids (Regions200).

3.1.7 Scenario ILSBQP

The ILSBQP scenario implements an iterated local search (ILS) algorithm to solve the unconstrained binary quadratic programming (UBQP). It iteratively executes a best improvement local search followed by a random perturbation step. ILSBQP has a single parameter that controls the perturbation size. The source code is available at Souza and Ritt (2022b).

We consider the UBQP instances of Beasley (1998) with sizes {250, 500, 1000, 2500}. There are 10 instances of each size, from which we use 3 for training, and the remaining 7 for testing. These instances are available in Wiegele (2007b) (for more details see Wiegele (2007a)). The running time limits vary according to the instance size, being 4, 8, 15 and 30 seconds of wall clock time for instances with sizes 250, 500, 1000 and 2500, respectively. We test configuration budgets of 300, 1000 and 2000 total executions.

3.1.8 Scenario BSFS

This scenario consists of the bubble search (BS) algorithm for the permutation flowshop scheduling (FS) problem (EMMONS; VAIRAKTARAKIS, 2013). The bubble search algorithm was proposed by Lesh and Mitzenmacher (2006) and extends priority-based greedy heuristics with applications in several problems. We use the implementation of Zubaran and Ritt (2013), who studied its application on FS and its performance under different values for the single parameter that controls the algorithm randomness.

We consider the instances proposed by Taillard (1993), which are divided in 12 groups with the number of jobs varying between 20 and 500, and the number of machines between 5 and 20. Each group has 10 instances, with processing times drawn uniformly at random from $[1, 99]$. We selected the first instance of each group for training, and all instances for testing. Following the literature, we use a running time limit of $30nm$ milliseconds for instances with n jobs and m machines. The configuration budget is set to 300, 1000 and 2000 total executions.

3.1.9 Scenario HHTA

The HHTA configuration scenario concerns a hybrid heuristic (HH) algorithm for the test-assignment (TA) problem (DUIVES; LODI; MALAGUTI, 2013). This algorithm solves the TA by first reducing it to UBQP, and then applying an evolutionary approach that keeps a set of elite solutions and iteratively recombines them with path relinking, followed by a tabu search improvement step. This algorithm was also automatically produced using our component-wise solver for binary problems (Chapters 7 and 8) and can be downloaded at Souza and Ritt (2018g).

In this scenario we adapted the HHTA algorithm, exposing three unconditional numerical parameters defined by constants: the tabu tenure, the size of the elite set, and the distance factor of the path relinking. We use a running time limit of 10 seconds of wall clock time and configuration budgets of 1000, 2000 and 5000 total executions. We use the 36 instances proposed in Duives, Lodi and Malaguti (2013), with 20 to 79 desks, 2 to 4 different tests, and between 0 and 20 unoccupied desks. We randomly selected 12 instances for training, and used all instances for testing.

3.1.10 Scenario TSCPP

This scenario consists in a three-phase tabu search (TS) for the clique partitioning problem (CPP). For details about the CPP we refer to the original works of Grötschel and Wakabayashi (1989, 1990). The TSCPP algorithm was proposed by Zhou, Hao and Goëffon (2016a) and its source code is available at Zhou, Hao and Goëffon (2016b). It uses a restricted neighborhood based on the problem structure, and iterates between three phases: an intensification by local search, an exploratory step using a tabu search, and a solution perturbation step.

We adapted the original TSCPP algorithm, exposing three unconditional numerical parameters defined by constants: the tabu tenure, the perturbation size, and the number of candidates for perturbation. We use a running time limit of 10 seconds of wall clock time and configuration budgets of 1000, 2000 and 5000 total executions. We use a set of graph instances proposed by Charon and Hudry (2006) and Brusco and Köhn (2009), with 100 to 500 vertices and edge weights selected uniformly at random from $[-100, 100]$ (5 instances) or $[-5, 5]$ (5 instances). We use 5 randomly chosen instances for training, and all 10 instances for testing. These instances (which we call "RAND") can be downloaded at Zhou, Hao and Goëffon (2016b).

3.1.11 Scenario SPEAR

SPEAR (BABIĆ; HUTTER, 2008) is an exact tree-based solver for (decision) boolean satisfiability (SAT) problems (BIERE et al., 2021). The SPEAR configuration scenario is also part of the Algorithm Configuration Library (HUTTER et al., 2014) and has been configured in several works (LÓPEZ-IBÁÑEZ et al., 2016; PÉREZ CÁCERES et al., 2017a; HUTTER et al., 2017). We use SPEAR version 1.2.1.

This scenario has 26 parameters, 9 of which are conditional. We use the SAT-encoded instances of graph coloring based on small world graphs (SWGCP) of Gent et al. (1999), which can be downloaded from Hutter (2007). We randomly selected 121 instances for training and no test instances, since we do not perform experiments with test phase for SPEAR (we only use this scenario as a test case for the visualizations proposed in Chapter 6). We set a limit of 10 seconds of wall-clock time for each execution of SPEAR, and a configuration time budget of 20000 seconds. Here, the

configuration process aims at minimizing SPEAR's solving time, where for evaluations in which the instance is not solved, a penalized performance value is returned. In particular, we return the running time limit multiplied by a constant penalty factor, which is in line with the penalized average running times (PAR) approach commonly used in the literature (LINDAUER et al., 2015; PÉREZ CÁCERES et al., 2017a).

3.2 Summary of Characteristics

We summarize the characteristics of each configuration scenario in Table 3.1. We indicate the target (heuristic or exact) algorithm, the associated (optimization or decision) problem, whether the algorithm is deterministic, and the configuration budget. We also present the configuration space, indicating the number of integer, categorical, real and conditional parameters, the effort type and limit used when executing the target algorithm, and the number of training and test instances.

We emphasize that the diverse characteristics presented by the configuration scenarios allow us to properly evaluate the methods we propose in this research. We have, for instance, scenarios with different configuration space sizes and various effort types to evaluate the capping methods under different conditions. We also use both optimization and decision scenarios to apply the proposed visualizations on different situations, i.e. when optimizing the target algorithm in terms of solution cost and running time, respectively.

Table 3.1 – Summary description of the configuration scenarios. Column *Problem* indicates the associated optimization ("opt") or decision ("dec") task; column *Det.* indicates whether the target algorithm is deterministic or stochastic; finally, column *Budget* is defined in terms of the maximum executions allowed, except for SPEAR, which uses a total configuration time budget. The performance metric of optimization scenarios is the cost of the best found solution, while for decision scenarios we use the solving time. The numbered references are: [1] Trick (2018); [2] Beasley (1998); [3] Taillard (1993); [4] Duives, Lodi and Malaguti (2013).

Scenario	Algorithm	Problem	Det.	Budget	Parameters				Effort		Instances		
					int	cat	real	cond	type	limit	description	train	test
ACOTSP	ACO (heuristic)	TSP (opt)	no	2000	4	3	4	5	time	60 s	RUE-2000	50	200
HEACOL	HEA (heuristic)	COL (opt)	yes	2000	4	2	1	0	checks	10^9	From [1]	27	79
TSBPP	TS (heuristic)	BPP (opt)	no	500	3	2	1	0	iterations	5000	2BPP	20	500
HHBQP	HH (heuristic)	UBQP (opt)	no	2000	10	3	1	7	time	20/30 s	From [2]	9	10
LKH	LKH (heuristic)	TSP (opt)	no	2000	12	9	0	0	time	10 s	RUE-1000-3000	50	250
SCIP	SCIP (exact)	MIP (opt)	yes	2000	0	207	0	0	time	60 s	Regions200	50	50
ILSBQP	ILS (heuristic)	UBQP (opt)	no	300 $\sim$ 2000	1	0	0	0	time	3 $\sim$ 300 s	From [2]	12	28
BSFS	BS (heuristic)	FS (opt)	no	300 $\sim$ 2000	1	0	0	0	time	4 $\sim$ 30 s	From [3]	12	120
HHTA	HH (heuristic)	TA (opt)	no	1000 $\sim$ 5000	2	0	1	0	time	10 s	From [4]	12	36
TSCPP	TS (heuristic)	CPP (opt)	no	1000 $\sim$ 5000	3	0	0	0	time	10 s	RAND	5	10
SPEAR	SPEAR (exact)	SAT (dec)	yes	20000 s	4	12	10	9	time	10 s	SWGCP	121	0

Part II

Methods

4 CAPPING METHODS FOR OPTIMIZATION SCENARIOS

A man who dares to waste one hour of time has not discovered the value of life.

— Charles Darwin

Automatic algorithm configuration free researchers from the tedious and error-prone task of manually searching the best configurations, which allows them to focus on other more creative activities of the algorithm design process. Besides that, the configuration time remains a bottleneck, since automatic approaches need to evaluate different parameter values on different (training) instances, and each evaluation often takes a considerable amount of time. As a result, researchers tend to wait from hours to weeks for the resulting configurations. As an example, López-Ibáñez et al. (2016) configure the 26 parameters of the SPEAR solver for boolean satisfiability problems using a cutoff time of 300 CPU-seconds for each execution, and a configuration budget of 10 000 executions. Without considering parallel evaluations, this configuration would take more than one month to finish (or almost a week considering parallel evaluations on a 6-core CPU).

There is a trade-off between the number of evaluations (or instances evaluated) and the quality of the final configurations found, and so reducing the effort (i.e. the configuration budget) will usually lead to worse configurations. A better approach to reduce the configuration time is to evaluate the quality of a configuration during its execution, and immediately stop it when poor performance is expected. This approach has been applied previously to the automatic configuration of decision algorithms (HUTTER et al., 2009b; PÉREZ CÁCERES et al., 2017a), where the performance of an algorithm is measured by its running time and the goal of configuration is to minimize it. The key idea behind existing methods is to *cap* the execution time, i.e. to determine a bound on the running time based on the best-performing configuration found so far and, if the execution reaches the running time bound, stop it and discard the current configuration.

For optimization problems, we usually execute an algorithm with a predefined stopping criterion, and measure its performance by the cost of the best found solution (assuming, without loss of generality, minimization of solution cost). Thus, existing capping methods are not suitable for these scenarios. In this chapter, we propose

capping methods for configuring optimization algorithms. The main idea is to use previously seen executions to determine a performance envelope for the current execution. We then monitor the evolution of the performance of each configuration, and stop it if at some point the conditions defined by the performance envelope are violated. We present and evaluate an implementation of such capping methods in the irace configurator.

This chapter details our contributions in the direction introduced above. First, we discuss some related work in Section 4.1. Then, we introduce several new capping methods for optimization problems in Section 4.2, and discuss the results of extensive computational experiments comparing the different methods in Section 4.3. Finally, we give some concluding remarks in Section 4.4.

4.1 Related Work

Hutter et al. (2009b) proposed capping methods for ParamILS to be used for configuration scenarios minimizing running time. The best configuration θ' of the current iteration of ParamILS is used to determine a cut-off running time for subsequent executions. A new configuration θ must be executed and present better performance on the same instances that θ' has been executed to replace it and become the new best configuration. If the time used by θ to solve a subset of those instances exceeds the time used by θ' to solve all of them, θ is discarded before the complete evaluation. Hutter et al. (2009b) call this method *trajectory-preserving capping*, since it discards only configurations that would be discarded after the complete evaluation. It reduces the configuration time, but does not change the search trajectory. A second approach, called *aggressive capping*, considers not the best found configuration of the current iteration, but the best configuration overall θ^*. In this case, the cut-off time when evaluating a new configuration θ is b times the mean running time of θ^*. Multiplier b defines the aggressiveness of the capping method. Although the aggressive capping method may change the search trajectory, it can also lead to large savings in the configuration time.

Based on the above ideas, Pérez Cáceres et al. (2017a) integrated a capping method for configuring decision algorithms into irace. Consider a new configuration θ that will be executed on instance π_i and was previously evaluated on instances $\pi_1, \pi_2, \ldots, \pi_{i-1}$. The cut-off time t^c is given by $t^c = i \cdot t_i^{\text{elite}} + t^{\min} - (i-1)t_{i-1}^{\theta}$, where t_i^{elite}

is the median running time of the elite configurations Θ^{elite} on instances $\pi_1, \pi_2, \ldots, \pi_i$, t^{θ}_{i-1} is the average running time of configuration θ on instances $\pi_1, \pi_2, \ldots, \pi_{i-1}$, and $t^{\min} > 0$ is the minimally measurable running time. The cut-off time can be seen as the maximum time available for θ to improve over the performance of the elite configurations. Pérez Cáceres et al. (2017a) also proposed the following dominance-based elimination criterion. A configuration θ is dominated if $t^{\text{elite}}_i + t^{\min} < t^{\theta}_i$. When all configurations have been evaluated on a new instance π_i, the dominated configurations are eliminated.

The capping methods proposed for ParamILS (HUTTER et al., 2009b) and irace (PÉREZ CÁCERES et al., 2017a) are designed for decision problems, when the goal is to minimize the running time of the target algorithm. Those capping methods cannot handle optimization scenarios, where the solution cost may be improved over time and, thus, there is information about the progress (or lack thereof) of the algorithm. In an optimization context, Karapetyan, Parkes and Stützle (2018) propose an approach to approximate the 1% best configurations of optimization algorithms based on short runs. They uniformly sample and evaluate 1% of the configuration space, determining a performance envelope, which is the hull defined by the worst solution cost obtained in those executions at each point in time. Then, previously untested configurations are executed. If, at some point in time, the cost of the best found solution is more than 20% worse than the one defined by the performance envelope, the execution is stopped and the configuration is discarded. If the configuration survives, it replaces the worst performing element of the 1% pool (according to the final solution cost). Among the capping methods proposed here, we include this definition of performance envelope as a particular case. Moreover, our capping methods are designed to work within the existing algorithm configuration techniques, e.g. the racing mechanism of irace, instead of relying on uniform sampling.

Capping methods for optimization scenarios analyze the algorithm performance under a given configuration during its execution. Thus, they are suitable when configuring anytime algorithms (ZILBERSTEIN, 1996; LÓPEZ-IBÁÑEZ; STÜTZLE, 2014), i.e. the algorithm has to produce feasible solutions prior to completion and continuously produce improved solutions until completion. Capping methods are not useful for configuring algorithms that do not satisfy these requirements, such as those that only return a single solution, that need a long exploratory phase or are based on random restarts. Fortunately, many practical optimization algorithms are anytime

(e.g. trajectory based heuristics that can return the best found solution if stopped at any time). Some techniques used to find configurations that present good anytime behavior are similar to those implemented by capping methods. For example, Branke and Elomari (2011) use a higher-level genetic algorithm to search for parameter settings of a lower-level genetic algorithm aiming to optimize its anytime behavior. To evaluate the population of configurations, they analyze their performance profiles. Configurations that present the best solution cost for some time point are ranked first. These configurations are removed from the population and the process is repeated, but now the best remaining configurations are ranked second, and so on. The ranks are used to guide the selection of configurations. Although their proposal is not a capping method, one of our methods similarly builds an envelope by considering the best solution cost found at any running time (effort) point.

López-Ibáñez and Stützle (2014) also explore the automatic configuration of algorithms to improve their anytime behavior. They model this configuration task as a bi-objective optimization problem, considering the performance profiles as a set of nondominated bi-objective points. They use irace to find configurations that maximize the hypervolume of such nondominated sets. The results show that the proposed techniques are effective in improving the anytime behavior of algorithms for two different scenarios. We propose here an area-based measure to determine the performance envelope (Section 4.2.2) that is equivalent to the hypervolume metric used in López-Ibáñez and Stützle (2014).

4.2 Capping for Optimization Scenarios

The capping methods we propose use the performance profiles of previously seen configurations to determine a minimum performance bound for new executions, and then stop poor performers early. The performance profile of a run is given by the solution cost as a function of the effort $P\colon \mathbb{R}_+ \to \mathbb{R}$, where $P(t) = c$ denotes the cost c of the best found solution after spending computational effort t. Any effort measure, e.g. the running time, number of iterations, or number of objective function evaluations, can be used. Assuming a minimization problem, a profile will always be monotonically decreasing and, for discrete problems, is a step function. Thus, in practice, we only need to collect the points (t, c) at efforts t where c decreases. Figure 4.1 presents an overview of the capping method, which is applied

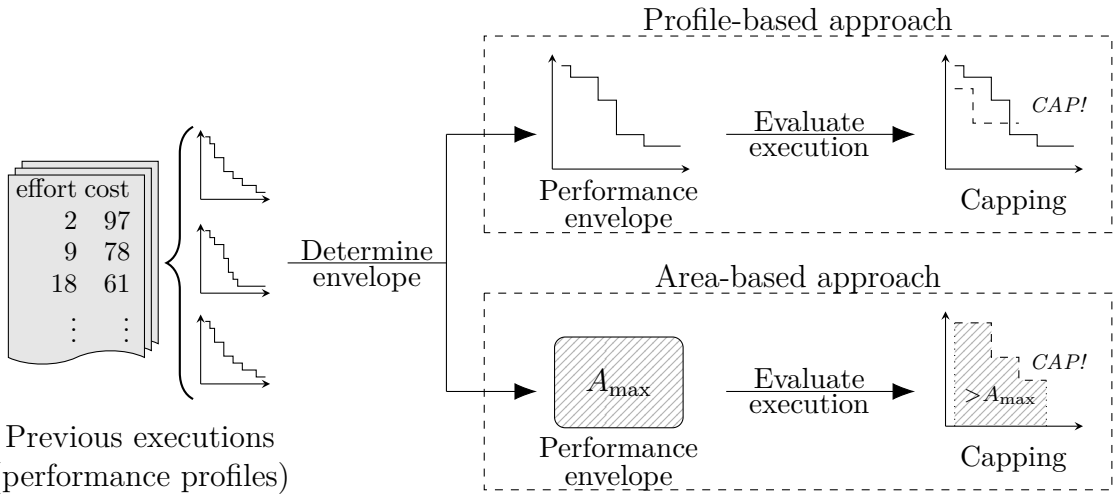


Figure 4.1 – Overview of the capping methods. Left: for each instance previous performance profiles are kept. Upper right: in the profile-based approach these performance profiles are combined into a performance envelope. If the current performance profile (dashed line) leaves the envelope (solid line) it is capped. Lower right: in the area-based approach profiles are combined into a maximum allowed area $A_{\max}$. If the total area of the current performance profile (dashed line) exceeds $A_{\max}$ it is capped.

before executing each configuration. The first step of this process is to obtain all performance profiles of the previous executions on the current instance. They are used to determine a performance envelope, which represents the minimum performance required for this instance. If at some point the observed performance profile is worse than the envelope, the execution is stopped. Unlike capping methods for decision problems, our methods do not cap after a fixed running time. Instead, they monitor the progress of the execution and terminate it when appropriate.

We propose two different types of envelopes. The *profile-based envelope* is represented by a performance profile, which determines the maximum allowed solution cost throughout the used effort. When using this method to evaluate an execution, as soon as its performance profile exceeds the envelope, i.e., the solution cost of the execution is worse than the one defined by the envelope for some value of effort, the execution is capped (see *Profile-based approach* in Figure 4.1). This approach not only defines the limits in terms of the expected solution cost, but also the acceptable behavior of the performance profile. For example, it is not allowed to be worse than the envelope in the beginning of the execution, e.g., by trying to diversify the search to find better solutions at a later time in the execution. Hence, this approach implies that the execution must show a clear good anytime behavior. In other cases, the final solution cost is the only performance criterion that matters, and allowing a

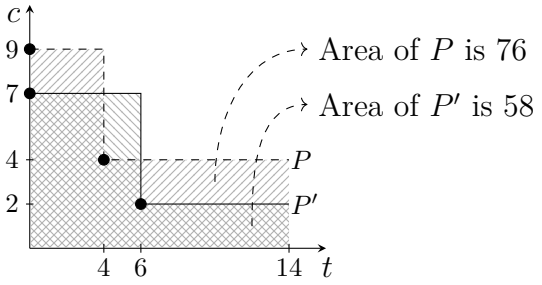


Figure 4.2 – Example of profile- and area-based capping behaviors. Note that the cost of P' is worse than the cost of P for $t \in [4, 6)$. The total area of P', however, is smaller than the area of P, since P' presents better cost during the execution, except for the mentioned interval. If P is the performance envelope, P' is capped by profile-based methods, but not capped by area-based methods.

worse performance in the beginning is not a problem, as long as a good solution is found in the end. Stützle et al. (2012), for example, studied parameter adaptation techniques for ant colony optimization algorithms. Their results on the traveling salesperson problem show that for some parameter settings, worse performance in the beginning of the execution leads to better final solutions.

In the *area-based envelope*, instead of dealing directly with the performance profiles, we consider the area defined by them. In this case, the area under the performance profiles of previous executions is used to determine a maximum area for the current execution. This maximum area value is the envelope. The current execution will be capped if the area of its performance profile exceeds the envelope, i.e. the maximum area available (see *Area-based approach* in Figure 4.1). As long as the maximum area is not exceeded, the performance profile can present different behaviors. For example, a configuration can produce worse solutions in the beginning of its execution in comparison to previously seen configurations, but produce better solutions towards the end of the execution, maintaining the total area within the envelope. Figure 4.2 illustrates this situation for the performance profile P' of a running execution. With a profile-based envelope P, execution P' would be capped at effort $t = 4$, since the solution cost of P' at that point exceeds the corresponding solution cost of envelope P. However, when defining the envelope as the area under P at $t = 14$, then P' would not be capped, since its area is always smaller than the envelope.

It is important to consider that an unsatisfactory performance of a capped configuration on the current instance does not imply a similar performance on other instances. Therefore, instead of directly discarding the configuration, we just stop

the execution and return to irace the cost of the best solution found. In other words, the configuration is penalized by having a reduced execution effort, but it can compensate this by presenting a better performance on other instances. The decision of discarding configurations is delegated to irace.

4.2.1 Profile-based Envelope Generation

The profile-based methods define the envelope as a performance profile. We propose two different strategies to compute the envelope: *elitist* and *adaptive*. For both approaches, we first select a subset of the non-capped previous executions on the current instance. Then, we aggregate their performance profiles into a performance envelope. The elitist envelope generation is based on the executions of configurations from an elite set, while the adaptive strategy considers all previous executions and discards some of them according to an aggressiveness value. Another difference between elitist and adaptive strategies is the behavior when evaluating elite configurations. Elitist strategies use those configurations to determine the performance envelope, and never cap executions of elite configurations. Adaptive methods, on the other hand, do not differentiate configurations when determining the envelope and when evaluating them. Therefore, adaptive strategies can cap both elite and non-elite configurations.

4.2.1.1 Elitist Profile-based Envelope Generation

The elitist strategy uses only the best performing (elite) configurations to define the envelope. In irace, the elite configurations are those that survived the previous race. We divide the aggregation of the previous executions for a fixed instance in two steps, as illustrated in Figure 4.3. Let us consider a set of n configurations $\theta_1, \ldots, \theta_n$, where each configuration θ_i has been evaluated m_i times on the same instance. Let $\mathbb{P}$ be the space of all possible performance profiles and $P_{ij} \in \mathbb{P}$ be the performance profile of the jth replication of θ_i on that instance. We define the aggregated performance profile for configuration θ_i as $P_i = AR(P_{i1}, P_{i2}, \ldots, P_{im_i})$, where $AR\colon 2^{\mathbb{P}} \to \mathbb{P}$ is an function that aggregates the performance profiles of multiple runs of a configuration. The profile-based envelope is defined as $P = AC(P_1, P_2, \ldots, P_n)$, where $AC\colon 2^{\mathbb{P}} \to \mathbb{P}$ is a function that aggregates the performance

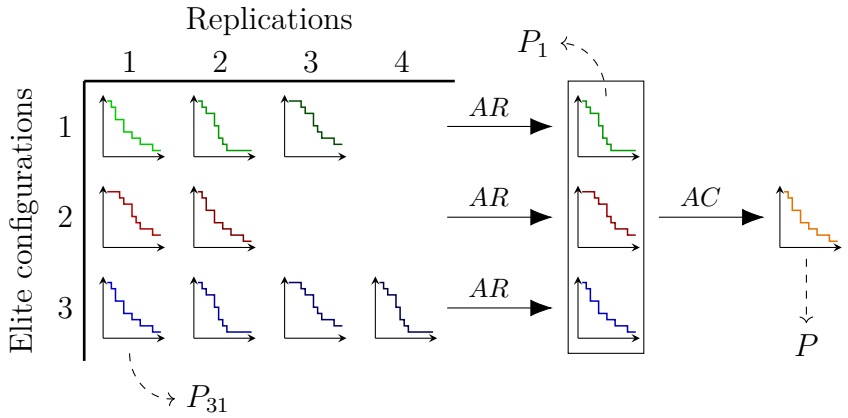


Figure 4.3 – Aggregation scheme for elitist capping methods. First, all replications of a given configuration i are aggregated by function AR into a single performance profile P_i. Then, function AC aggregates all resulting performance profiles into the performance envelope P.

profiles of multiple configurations.

Two possible approaches for defining AR and AC are the *worst* aggregation function W and the *best* aggregation function B. Function $W\colon 2^{\mathbb{P}} \to \mathbb{P}$ generates a performance profile that selects, for each effort t, the pointwise maximum solution cost among all input performance profiles. Given a set of performance profiles $P_1, P_2, \ldots, P_k$, the performance profile generated by W is given by

$$W(P_1, \ldots, P_k)(t) = \max\{P_1(t), \ldots, P_k(t)\}. \tag{4.1}$$

Analogously, the best aggregation function $B\colon 2^{\mathbb{P}} \to \mathbb{P}$ generates a performance profile that selects, for each value of effort t, the pointwise minimum solution cost among all performance profiles, then

$$B(P_1, \ldots, P_k)(t) = \min\{P_1(t), \ldots, P_k(t)\}. \tag{4.2}$$

Aggregation methods W and B are illustrated in Figure 4.4. There are three different performance profiles to be aggregated (left side). The aggregated performance profiles given by W and B functions are shown on the right side of the figure. W and B form the hull of the input performance profiles. Function W produces the pessimistic performance (least aggressive profile), while function B produces the optimistic performance (most aggressive profile).

Aggregation functions AR and AC may be different or the same. For example,

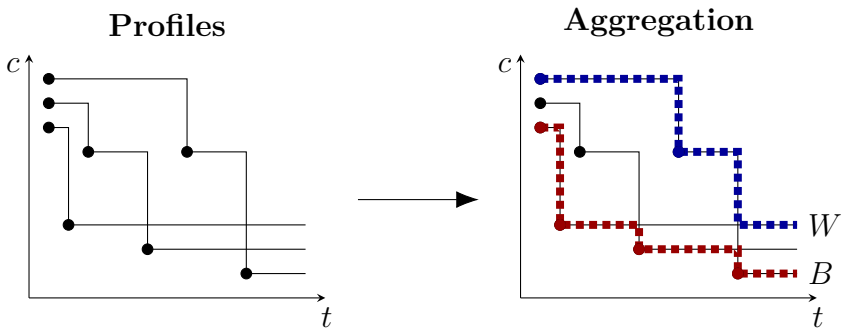


Figure 4.4 – Best (B) and worst (W) aggregation methods for profile-based envelopes. Methods W and B select the pointwise maximum and minimum solution cost, respectively, for each effort t, among all input performance profiles.

if $AR = W$ and $AC = B$, then the envelope will be

$$\begin{aligned} P(t) &= AC(AR(P_{11}, \ldots, P_{1m_1}), \ldots, AR(P_{n1}, \ldots, P_{nm_n}))(t) \\ &= B(W(P_{11}, \ldots, P_{1m_1}), \ldots, W(P_{n1}, \ldots, P_{nm_n}))(t) \\ &= \min\{\max\{P_{11}(t), \ldots, P_{1m}(t)\}, \ldots, \max\{(P_{n1}(t), \ldots, P_{nm}(t)\}\} \end{aligned} \tag{4.3}$$

We also propose a model-based aggregation function as follows. Given the performance profile P of a single run and a maximum cut-off effort $t_{\max}$ (maximum budget for any run) such that $P(t) \geq P(t_{\max})$, $\forall t \geq 0$, let $T(P, c)$ be the smallest effort to reach target c defined as

$$T(P, c) = \begin{cases} \min\{t \geq 0 \mid P(t) \leq c\} & \text{if } P(t_{\max}) \leq c, \\ \alpha t_{\max} & \text{otherwise,} \end{cases} \tag{4.4}$$

where $\alpha \geq 1$ is a penalty factor applied when the performance profile does not reach the target, similar to the PARX penalization approach (PÉREZ CÁCERES et al., 2017a). We now assume that the value of $T(P, c)$ for any randomly selected run P of an algorithm is a random variable $\mathcal{T}(c)$ that follows an exponential distribution, which is often a reasonable assumption for optimization algorithms (HOOS; STÜTZLE, 2005). Its empirical cumulative distribution function is given by $F(t; \lambda) = \Pr(\mathcal{T}(c) \leq t) = 1 - e^{-\lambda t}$, where λ is the parameter of the distribution. Given the performance profiles $P_1, \ldots, P_k$ of k runs of the algorithm, we estimate the mean effort to reach target c as $\bar{T}(c) = \sum_i^k T(P_i, c)/k$. Then, the maximum likelihood estimator for parameter λ is $\hat{\lambda}(c) = 1/\bar{T}(c)$ and we can determine the effort $T_p(c)$ required for a fraction $p \in [0, 1]$ of the executions not reaching target value c by setting $F(T_p(c); \lambda) = 1 - p$,

which holds for $T_p(c) = -\ln(p) \cdot \bar{T}(c)$. Since $T_p(c)$ is monotone, it has an inverse $T_p^{-1}(t)$ that gives the expected c reached after effort t by at most a fraction $1 - p$ of executions. We can now define a model-based aggregation as

$$M(P_1, \ldots, P_k; p)(t) = T_p^{-1}(t). \tag{4.5}$$

If there is only one performance profile to be aggregated i.e. $k = 1$, methods W and B produce the same performance profile, since they simply select cost values for each value of effort from the available profile. In contrast, method M computes the aggregated performance profile based on the exponential model, whose parameter can be estimated even with a single sample. For example, given a single performance profile P to be aggregated with $p = 0.1$ and $T(P, c) = 20$ for a particular target cost c, the mean $\bar{T}(c) = 20$ is multiplied by $-\ln(0.1) \approx 2.3$, leading to an effort equal to 43 for the aggregated performance profile. The effort values required for the fraction p not reaching all target costs are computed, producing an aggregated performance profile different from the one used as input.

4.2.1.2 Adaptive Profile-based Envelope Generation

The adaptive strategy determines the envelope based on all previous executions on the current instance, not only on the executions of elite configurations. These performance profiles are ordered by the cost of the best solution found, and the envelope is determined in such a way that only the best ones would not be capped. The number of such non-capped performance profiles is determined by an aggressiveness parameter $a \in [0, 1]$. Given k performance profiles sorted by the cost of their best solution found, we determine the most aggressive envelope that would cap a fraction a of them. That is, the adaptive profile-based envelope generation is equivalent to the aggregation function

$$D(P_1, \ldots, P_k)(t) = W(P_1, \ldots, P_{\lceil (1-a)k \rceil})(t), \tag{4.6}$$

given that $P_1, P_2, \ldots, P_k$ are ordered by the cost of their best solution found. Figure 4.5 shows an example with 4 performance profiles P_A, P_B, P_C, P_D. When sorting them, we get the ordered list $L = (P_D, P_A, P_C, P_B)$. Given $a = 0.5$, we select only the 2 best performance profiles and compute the most aggressive envelope that would

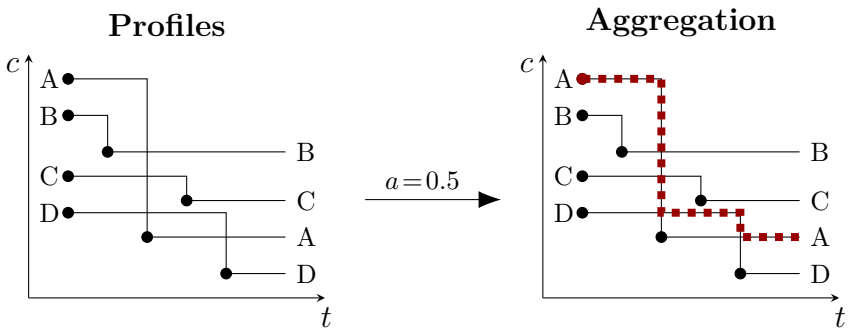


Figure 4.5 – Adaptive aggregation for profile-based envelope. A profile that would be capped $\lceil (1-a)k \rceil$ of the k previous executions is determined.

not cap them by applying W function (Eq. 4.1). The resulting envelope is shown on the right side of Figure 4.5.

Given a value of a for the current iteration, we create an envelope before each execution, and use it to evaluate (and possibly cap) that execution. However, we cannot ensure that this value of a will produce envelopes that cap exactly a fraction a of those executions. Therefore, once the current iteration is finished, we adapt the value of a based on the number of executions capped, and then use the updated value to compute the envelopes in the next iteration, in an attempt to reach a user-defined aggressiveness goal a_g.

Let a_c be the fraction of executions capped in the previous iteration of irace. Since the goal was to cap a fraction a_g of those executions, we

- increase a to the lowest value that would have capped a_g, if $a_c < a_g - \varepsilon$,
- decrease a to the highest value that would have capped a_g, if $a_c > a_g + \varepsilon$,
- maintain a, otherwise,

where $\varepsilon \in [0, 1]$ is a user-defined parameter that specifies the tolerance of the observed aggressiveness deviation from the aggressiveness goal a_g. For example, given $a_g = 0.3$ and $\varepsilon = 0.05$, if in the previous iteration we capped 18 out of 100 executions, then the adaptation procedure would increase a, since $a_c = 18/100 < 0.3 - 0.05$, such that Eq. (4.6) would cap exactly $0.3 \cdot 100 = 30$ executions.

When increasing a, we can easily determine which value of a would cap the desired number of executions from the previous iteration by adding performance profiles one at a time to those used by Eq. (4.6). The case of decreasing a is more complicated since we want to find the value a that would not cap the desired number of executions. Because the executions were capped, we do not know the performance

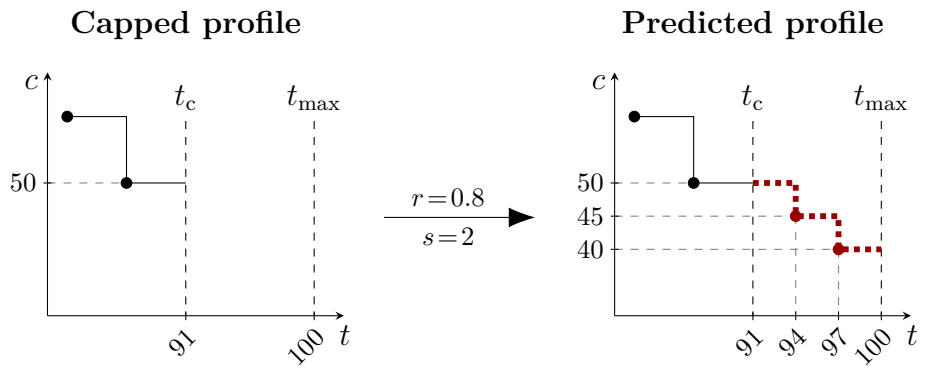


Figure 4.6 – Example of the performance profile prediction for adaptive profile-based methods. Given the median of improvement ratios r and the number of improvements s, the missing part of the capped performance profile (left) is predicted (right).

profile until the cut-off effort and, therefore, we cannot compute exactly the value of a that would not ever cap them. Instead, we predict the unknown part of the performance profile and combine it with the known part when calculating the value of a that would not cap it.

To predict how a capped performance profile P_c would behave from the effort at which it was capped t_c until the cut-off effort t_{max}, we consider all previous non-capped performance profiles on the same instance, and compute from them a simple extrapolation. First, we estimate the number of improvements s of the objective function from t_c to t_{max} as the median number of improvements in the same range over all non-capped profiles. Next, we estimate the final solution cost $\hat{P}_c(t_{max}) = r \cdot P_c(t_c)$, where r is the median of the improvement ratios $P(t_{max})/P(t_c)$ between the solution cost at t_c and the final cost at t_{max} over all non-capped performance profiles. Finally, we estimate the solution cost at each effort $t_i = t_c + i(t_{max} - t_c)/(s+1)$, with $i = 1, \ldots, s$, as

$$\hat{P}_c(t_i) = P_c(t_c) + i(\hat{P}_c(t_{max}) - P_c(t_c))/s. \tag{4.7}$$

Figure 4.6 presents an example of the performance profile prediction. On the left side we can see the performance profile P_c capped at $t_c = 91$ with cost $P_c(t_c) = 50$. Let us assume that $r = 0.8$ and $s = 2$, thus the predicted final cost at $t_{max} = 100$ is $\hat{P}_c(t_{max}) = 50 \cdot 0.8 = 40$. Then, we build $s = 2$ intermediary points until t_{max}. The first will be at $t_1 = t_c + 1 \cdot (t_{max} - t_c)/(s+1) = 91 + 9/3 = 94$ with cost $\hat{P}_c(t_1) = P_c(t_c) + 1 \cdot (\hat{P}_c(t_{max}) - P_c(t_c))/s = 50 + 1 \cdot (-10/2) = 45$. The second point will be at $t_2 = t_c + 2 \cdot (t_{max} - t_c)/(s+1) = 91 + 2 \cdot (9/3) = 97$ with cost $\hat{P}_c(t_2) = P_c(t_c) + 2 \cdot (\hat{P}_c(t_{max}) - P_c(t_c))/s = 50 + 2 \cdot (-10/2) = 40$. The predicted

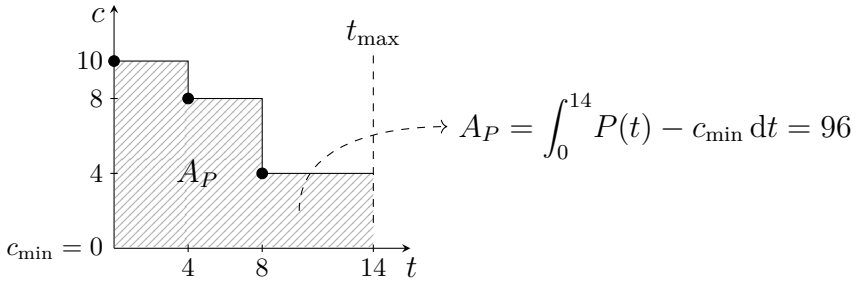


Figure 4.7 – Example of area calculation for a given performance profile. The area is calculated between a minimum cost value $c_{\min}$ and the curve given by the performance profile.

performance profile, with the calculated points, can be seen on the right side of Figure 4.6.

4.2.2 Area-based Envelope Generation

The area-based envelope is defined as the maximum area available for the execution. The area of a performance profile P is

$$A_P = \int_{t_s}^{t_f} P(t) - c_{\min}\,\mathrm{d}t, \tag{4.8}$$

where t_s and t_f are the start and final effort values, respectively, and $c_{\min}$ is a baseline cost. The start effort t_s should be the same for calculating the area of different performance profiles, to ensure a fair comparison between them. In our implementation, the target algorithms report the corresponding cost as soon as the first solution is obtained. Then, when evaluating k performance profiles, we define $t_s = \max_{i\in\{1,\ldots,k\}} t_s^i$, where t_s^i is the starting effort value of performance profile i. The final effort t_f is the cut-off effort ($t_{\max}$) for finished executions or the current effort of the execution in progress. Figure 4.7 illustrates a performance profile and the respective value of area.

The area depends on the baseline cost $c_{\min}$. If $c_{\min}$ is too low, we obtain a larger area, which makes capping less aggressive; if it is too high, we may have negative areas, and possibly a too aggressive capping. Ideally, we would like to set $c_{\min}$ to the optimal solution cost or a good lower bound. These are often unknown, thus we maintain for every instance the best found solution cost and set $c_{\min}$ to it.

Finally, we first compute the area of previous performance profiles on the current instance, and then aggregate the areas into the envelope, which is represented

here by the area budget $A_{\max}$. The current execution P will be capped as soon as $A_P > A_{\max}$. Similar to the profile-based methods, in the area-based envelope generation we also use elitist and adaptive aggregation strategies.

4.2.2.1 Elitist Area-based Envelope Generation

Given all performance profiles of the elite configurations on the current instance, the area-based elitist strategy calculates their area and uses functions AR and AC to aggregate replications and configurations, respectively. We use worst and best approaches for both AR and AC aggregation steps, replacing the pointwise behavior by the selection of the largest area

$$W(P_1, \ldots, P_k) = \max\{A_{P_1}, \ldots, A_{P_k}\}, \tag{4.9}$$

or the smallest area

$$B(P_1, \ldots, P_k) = \min\{A_{P_1}, \ldots, A_{P_k}\}. \tag{4.10}$$

4.2.2.2 Adaptive Area-based Envelope Generation

The area-based adaptive envelope generation is similar to the profile-based approach. We compute the area of all previous performance profiles of the current instance, and select the area which would cap a part of them, according to the aggressiveness parameter $a \in [0, 1]$. In this case, the performance profiles are sorted by their area values. Given a list of performance profiles $P_1, P_2, \ldots, P_k$ ordered such that $A_{P_i} \leq A_{P_{i+1}}$, the adaptive area-based envelope generation is given by

$$D(P_1, \ldots, P_k) = A_{P_{\lceil (1-a)k \rceil}}. \tag{4.11}$$

The aggressiveness is adjusted at the beginning of each iteration. It increases or decreases a according to the amount of capping reached in the last iteration, the aggressiveness goal parameter $a_{\mathrm{g}} \in [0, 1]$, and the tolerance ε.

As done in the profile-based approach, when the amount of capped executions in the previous iteration is out of the range $[a_{\mathrm{g}} - \varepsilon, a_{\mathrm{g}} + \varepsilon]$, we calculate the value of a that would cap exactly the number of executions needed to achieve the aggressiveness goal a_{g}. The only difference with respect to the profile-based approach is in the

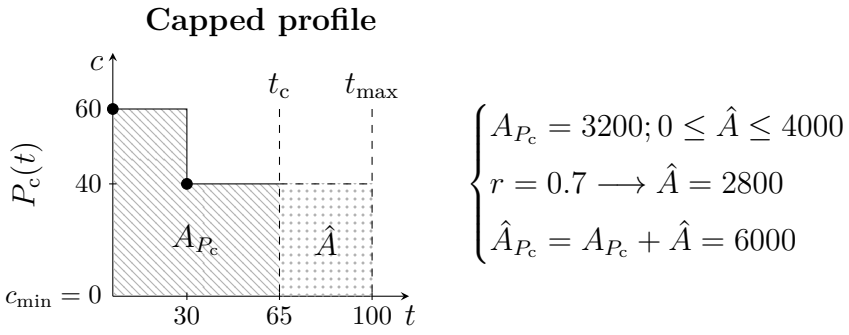


Figure 4.8 – Example of area prediction for a given performance profile. To estimate area $\hat{A}$ for the missing portion of the performance profile, we first compute a ratio $r = A''/A'$ of the real area A' and the upper bound area A'' of the performance profiles of non-capped previous executions.

estimation required when decreasing a. Here we need to estimate the area of the capped performance profile P_c by predicting its behavior after the capping point $(t_c, P_c(t_c))$. In the best case, the best possible solution would have been found just at t_c, such that $P_c(t_c) = c_{\min}$, and thus the remaining area would be zero. In the worst case, no solution better than $P_c(t_c)$ would have been found until $t_{\max}$, thus the upper bound of the unknown area would be $(P_c(t_c) - c_{\min}) \cdot (t_{\max} - t_c)$. Let $\hat{A}_{P_c} = A_{P_c} + \hat{A}$ be the (estimated) total area used by P_c if it had not been capped, where A_{P_c} is the known area until t_c and $\hat{A} \in [0, (P_c(t_c) - c_{\min}) \cdot (t_{\max} - t_c)]$ is the unknown area between t_c and $t_{\max}$, which must be estimated. Figure 4.8 shows an example where the execution was capped at effort $t_c = 65$ with $P(t_c) = 40$. Then, the real area of this execution until t_c is $A_{P_c} = 3200$ and the remaining area is $0 \leq \hat{A} \leq 4000$.

To estimate the value of $\hat{A}$, we use the performance profiles of non-capped previous executions on the current instance. For each performance profile P, we compute the real area A' from t_c to $t_{\max}$ and the upper bound $A'' = (P_c(t_c) - c_{\min}) \cdot (t_{\max} - t_c)$, and then calculate the ratio A''/A'. We use the median ratio r to estimate the remaining area of the capped performance profile P_c as $\hat{A} = r \cdot (P_c(t_c) - c_{\min}) \cdot (t_{\max} - t_c)$ and its predicted area $\hat{A}_{P_c} = A_{P_c} + \hat{A}$. In the example of Figure 4.8, by using $r = 0.7$, the remaining area is estimated as $\hat{A} = 2800$, giving a predicted total area of $\hat{A}_{P_c} = 6000$.

4.2.3 Summary of Proposed Methods

Table 4.1 summarizes the components of all the methods described in this section. A complete capping method consists of an envelope type (P or A), an

Table 4.1 – Summary of the proposed capping methods. We present the envelope types, main strategies and corresponding parameters.

Envelope	Strategy	Parameters
Profile (P)	Elitist (E)	AR$=\{W, B, M\}$, AC$=\{W, B\}$, $p \in [0, 1]$, $\alpha \in [1, \infty)$
	Adaptive (D)	$a_{\mathrm{g}} \in [0, 1]$, $\varepsilon \in [0, 1]$
Area (A)	Elitist (E)	AR$=\{W, B\}$, AC$=\{W, B\}$
	Adaptive (D)	$a_{\mathrm{g}} \in [0, 1]$, $\varepsilon \in [0, 1]$

envelope generation strategy (E or D), the corresponding aggregation functions (W, B or M, when applied), and parameter values (p and a_{g}, when applied).

The capping methods proposed here are mostly independent of the configurator, and can be applied whenever the following requirements are met. First, the target algorithm must periodically report the progress of the objective function. It must also report the effort if it is different from wall-clock time, e.g. the number of evaluations. Second, the elitist capping methods require that elite configurations are identified. Third, the adaptive capping methods require to indicate when the aggressiveness parameter needs to be updated. The latter two requirements can be satisfied by the configurator but may also be implemented in additional external components.

When integrated with irace, the capping methods identify the elites from the data already reported by irace, update the aggressiveness at the end of each race (Algorithm 1), and do not apply capping in the first race. This integration does not require any changes in irace except using the capping methods as a wrapper around the target algorithm.

4.3 Computational Experiments

In this section, we present our computational experiments to evaluate the capping methods. We detail the experimental setup, i.e. software versions, values for the parameters of the capping methods, configuration scenarios, and machine settings. Then, our first experiment evaluates how good the capping methods are in saving effort during the tuning process, as well as in finding good configurations (Section 4.3.2). We identified two conservative, robust methods and analyzed their

behavior in detail (Section 4.3.3). Finally, we asses the contributions of the capping methods when using the total execution time as budget for irace (Section 4.3.4).

4.3.1 Experimental Setup

We evaluated the proposed capping methods on six configuration scenarios: ACOTSP, HEACOL, TSBPP, HHBQP, LKH, and SCIP. They are all described in Chapter 3. In case of SCIP, some configurations produce infeasible solutions. We assign those configurations the worst possible cost value and do not use them to determine the performance envelopes in the capping methods. All heuristic algorithms were implemented in C or C++ and compiled with the GNU C/C++ compiler version 7.0.4 with maximum optimization. We used irace version 3.1 with its default parameter values. The capping methods have been implemented as a Python 3 script (tested with Python 3.6.8) that can be used together with irace and does not require any changes to irace. For the aggregation method using the exponential model (Eqs. 4.4 and 4.5), we used the penalty constant $\alpha = 10$. For the adaptive methods, we used a tolerance of the deviation from the aggressiveness goal $\varepsilon = 0.05$. When experiments are replicated with different random seeds, the same initial seed is used for all executions of irace in the same replication.

Most of the experiments were performed using a single core of a computer with an 8-core AMD FX-8150 processor running at 3.6 GHz and 32 GB main memory, under Ubuntu Linux. The experiments of the SCIP scenario were performed using a single core of a computer with a 12-core AMD Ryzen 9 3900X processor running at 3.8GHz and 32GB main memory, under Ubuntu Linux.

4.3.2 Evaluation of Capping Methods

The first experiment consists in the evaluation of all capping methods. For each method, we executed irace 20 times and computed the total effort used for the configuration process. Then, we executed the first ranked configuration of each irace execution on the set of test instances with 5 replications, and computed the average cost deviation from the best known solutions.

The results are shown in Table 4.2. In the method description (column

Table 4.2 – Average relative effort and average solution cost deviation for each capping method. The three best values of each column are shown in bold.

Capping method	ACOTSP		HEACOL		TSBPP		HHBQP		LKH		SCIP	
	r. e.	r. d.	r. e.	r. d.	r. e.	r. d.	r. e.	a. d.	r. e.	r. d.	r. e.	r. d.
No cap.	100.0	**0.33**	100.0	**4.14**	100.0	1.31	100.0	49.72	100.0	**0.04**	100.0	**0.04**
PEWW	40.3	0.37	38.7	4.22	87.4	1.25	55.7	65.16	37.8	**0.04**	69.0	0.05
PEBB	**22.3**	0.52	**25.2**	4.48	61.9	1.27	**25.1**	58.38	**24.3**	0.08	**21.7**	0.11
PEMW.1	74.5	**0.34**	77.9	**4.10**	95.1	**1.24**	82.6	72.44	60.2	**0.04**	89.9	**0.03**
PEMW.3	51.4	0.35	61.5	4.16	92.1	1.28	66.7	63.52	46.4	**0.04**	75.8	0.05
PEMW.5	30.2	0.44	**27.3**	4.48	**54.5**	1.35	40.0	72.16	35.4	0.05	54.4	0.07
PEMB.1	50.0	**0.32**	58.5	**4.11**	86.7	**1.22**	53.7	67.19	40.4	**0.04**	28.6	0.12
PEMB.3	**26.9**	0.46	31.5	4.23	74.6	1.30	37.2	55.75	**31.0**	0.05	**23.6**	0.12
PEMB.5	**21.8**	0.48	**23.5**	4.49	**44.6**	1.33	**25.8**	61.40	**24.0**	0.07	**21.6**	0.12
PD.2	65.0	0.38	62.4	4.28	92.6	**1.24**	78.4	**36.06**	66.2	**0.04**	65.3	0.15
PD.4	63.7	0.38	61.9	4.27	88.3	1.28	74.6	44.71	62.1	**0.04**	63.9	0.16
PD.6	55.7	0.40	56.5	4.30	80.7	1.29	65.7	48.87	53.6	0.05	58.6	0.16
PD.8	43.7	0.41	47.9	4.39	73.9	1.37	54.3	57.26	39.8	0.05	48.6	0.16
AEWW	73.2	0.35	72.8	4.18	87.6	1.28	82.7	46.97	52.5	**0.04**	94.0	**0.04**
AEBB	47.3	0.38	53.0	4.18	58.6	1.35	**34.1**	68.56	33.8	0.06	62.9	0.08
AD.2	82.8	0.35	84.6	4.16	85.6	1.30	85.2	**37.03**	78.7	**0.04**	83.4	**0.03**
AD.4	77.9	0.35	81.8	4.16	71.5	1.26	72.9	**37.48**	72.4	**0.04**	80.3	0.06
AD.6	69.7	0.35	74.0	**4.14**	58.5	1.34	58.2	37.71	59.5	**0.04**	71.1	0.08
AD.8	55.3	0.36	62.1	4.21	**52.6**	1.45	41.9	54.13	43.5	**0.04**	61.6	0.10

"Capping method"), the first letter indicates the envelope type (P for profile-based or A for area-based) of the capping method, followed by the strategy (E for elitist or D for adaptive). In the case of elitist methods, the next two letters represent the aggregation functions (B, W or M) used, respectively, for aggregating over multiple replications of a configuration (AR) and for aggregating over multiple configurations (AC). In the case of methods with a user-defined parameter (p or a_g), its value is given at the end of the description. For example, the method PEMW.1 represents the profile-based envelope, using the elitist strategy, model-based function M to aggregate replications, worst function W to aggregate configurations, and 0.1 for the parameter p required by the M function. Column "r. e." presents the average relative effort required when using each capping method in comparison to the effort required when configuring without capping. Column "r. d." presents the average relative deviation from the best known solutions, obtained by executing the best found

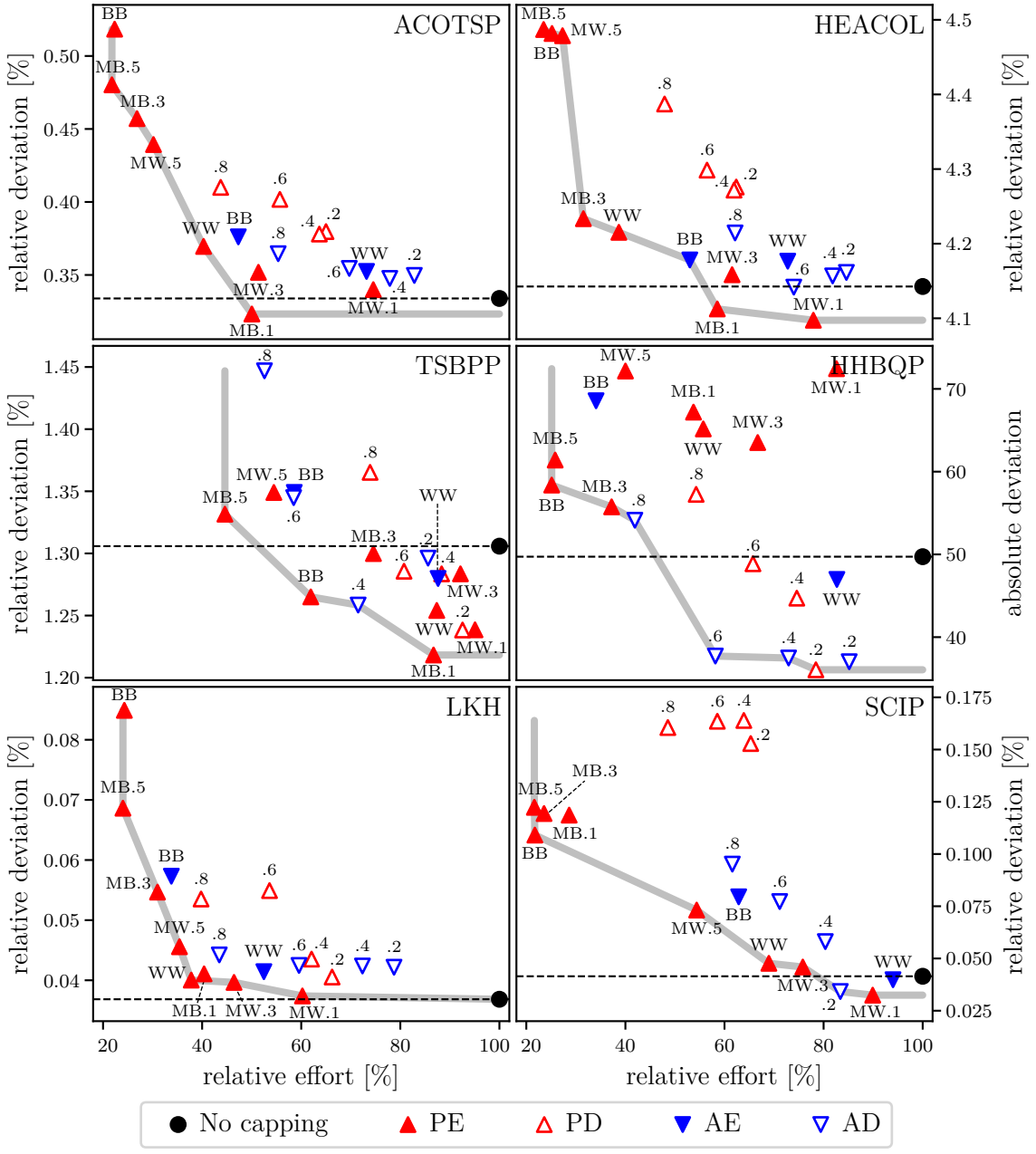


Figure 4.9 – Normalized configuration effort and quality of the resulting configurations for all capping methods.

configurations on the test instances. In the case of HHBQP, we report the average absolute deviation (column "a. d.") from the best known solutions, following the practice of the literature of UBQP (PALUBECKIS, 2006; GLOVER; LÜ; HAO, 2010; WANG et al., 2012). We say that a capping method presents better quality than another when its best found configurations have a smaller deviation. We highlighted the three best values of relative effort and deviation for each configuration scenario.

We observe that all capping methods reduce some amount of effort. The reduction ranges from about 5% (method PEMW.1 on TSBPP) to about 78%

(method PEMB.5 on ACOTSP and SCIP, and method PEBB on SCIP) of the effort required when configuring without capping. The resulting configurations present competitive quality in comparison to the one obtained without using capping. As expected, we can also observe that more aggressive methods (e.g. PEBB, PEMB.5 and PEMW.5) save more effort, but usually in exchange of producing worse configurations. This is the case when using best (B) instead of worst (W) aggregation functions, as well as using more aggressive (higher) values for the parameter p of the exponential model and the aggressiveness goal a_g.

We also measured the number of training instances used by each run of irace. For ACOTSP, HEACOL, TSBPP, LKH and SCIP, irace used an average of 35%, 71%, 61%, 37% and 49% of the available instances, respectively, with no more than 10% of variation over the different capping methods. Thus, irace never needed to perform more than one replication per training instance, and methods W and B have no effect when aggregating replications. The model-based method M has an effect even with a single replication, as explained in Section 4.2.1.1. For scenario HHBQP, irace does use all training instances and thus sometimes performs more than one replication per instance. In this case methods W and B for aggregating replications lead to different results.

Figure 4.9 presents the trade-off between the average relative effort and the average solution cost deviation of each capping method. It also presents the Pareto frontier defined by the non-dominated methods, considering relative effort and deviation as two distinct objectives of the capping methods that must be minimized. We can see that most of the Pareto-optimal methods are profile-based and elitist, except in the HHBQP scenario, where profile- and area-based adaptive methods are the majority in the Pareto frontier.

We also observe that all capping methods present competitive results in the quality of the configurations found. The difference of the observed relative deviations from the best known solutions in comparison to those obtained without capping is always less than 1% (and less than 100 of absolute deviation for HHBQP, whose objective values are in the order of magnitude of 10^6). Even the most aggressive methods (PEBB, PEMB.5, PEMW.5) produce acceptable configurations, while saving more than half of the total configuration effort. We would expect no capping would produce the best results, however, in some cases the tuning process with capping found a better final configuration (e.g. capping method PEMW.1 in scenarios HEACOL,

Table 4.3 – Results of the multiple comparison Dunn's test, adjusted with the Bonferroni method. Comparing the quality of the configurations produced using each capping method with those obtained using no capping. Symbol "ns" means no significant difference, while the asterisks denote the order of the p-values: "$*$" for $p \leq 0.01$, "$**$" for $p \leq 0.001$ and "$***$" for $p \leq 0.0001$.

Capping method	No capping		Capping method	No capping	
	ACOTSP	SCIP		ACOTSP	SCIP
PEWW	ns	ns	PD.4	ns	***
PEBB	***	ns	PD.6	**	***
PEMW.1	ns	ns	PD.8	***	**
PEMW.3	ns	ns	AEWW	ns	ns
PEMW.5	***	ns	AEBB	ns	ns
PEMB.1	ns	*	AD.2	ns	ns
PEMB.3	***	ns	AD.4	ns	ns
PEMB.5	***	*	AD.6	ns	ns
PD.2	ns	**	AD.8	ns	ns

TSBPP, and SCIP). Since capped runs are penalized by returning the current cost, this can lead irace to discard the corresponding configuration earlier and focus the sampling of the parameter space on better configurations.

We have performed a non-parametric Kruskal-Wallis test to check whether any capping method statistically dominates another in terms of the quality of the produced configurations (MONTGOMERY, 2012). The results indicate statistically significant differences in the ACOTSP, HEACOL, LKH and SCIP scenarios (p-values of 1.01×10^{-23}, 1.82×10^{-7}, 1.93×10^{-3} and 2.97×10^{-22}, respectively), while the null hypothesis could not be rejected in the TSBPP and HHBQP scenarios (p-values of 0.99 and 0.09, respectively). We have also performed a post-hoc analysis using the Dunn's multiple comparisons test (DUNN, 1964) to assess the pairwise differences of the capping methods. We used a significance level of 0.01, and the Bonferroni correction method (DUNN, 1961) to control the familywise error rate. The HEACOL and LKH scenarios presented statistical differences among distinct capping methods, but no difference was identified between any capping method and configuring with no capping. For ACOTSP and SCIP, however, some of the most aggressive methods presented statistically worse configurations than those obtained by configuring with no capping. Table 4.3 presents these results, and their statistical significance. We can see that the differences are statistically more significant for ACOTSP. For the area-based methods, even the more aggressive approaches (e.g. AEBB and AD.8)

Table 4.4 – Recommended capping methods. We present the average relative effort and the loss in the quality of the best found solutions for all configuration scenarios.

Category	Capping method	Rel. eff.	Quality loss					
			ACOTSP	HEACOL	TSBPP	HHBQP	LKH	SCIP
Conservative	PEMW.1	80.0	0.01	-0.04	-0.07	22.72	0.00	-0.01
	AD.4	76.1	0.02	0.02	-0.05	-12.24	0.00	0.02
Aggressive	PEMB.1	53.0	-0.01	-0.03	-0.09	17.47	0.00	0.08
	PEWW	54.8	0.04	0.08	-0.06	15.44	0.00	0.01

produce solutions which are statistically not worse than those found without capping.

4.3.3 Recommended Methods

Based on the results discussed above, we can identify specific recommendations presented in Table 4.4. First, we selected two conservative, robust capping methods (PEMW.1 and AD.4) that maintain the quality of the final configurations, but still save a reasonable effort. Then, we selected two aggressive methods (PEMB.1 and PEWW) that save more effort, but sometimes find worse configurations than tuning with no capping. Table 4.4 shows the average relative effort of those methods for all six configuration scenarios, as well as their quality loss on each configuration scenario. The quality loss is the difference between the average cost deviation obtained using the capping method and using no capping. We observe that all recommended methods reduce the tuning effort by at least 20%, but still produce solutions of acceptable quality. In some cases, the use of capping produced better configurations than tuning with no capping. We also note that the selected methods cover the different capping components, i.e. profile-based elitist and area-based adaptive methods. In this section, we analyze in detail the behavior of the recommended capping methods. We used the visualization tool acviz, presented in Chapter 6.

Figures 4.10 and 4.11 present the evolution of the configuration process of ACOTSP with no capping, when using the conservative methods PEMW.1 and AD.4 (Figure 4.10) or the aggressive methods PEWW and PEMB.1 (Figure 4.11). We selected one of the irace executions at random to produce this figure (the other executions present a similar behavior). We plot the quality over the executions. Since we used a budget of 2000 total executions in the ACOTSP scenario, the x axis ranges

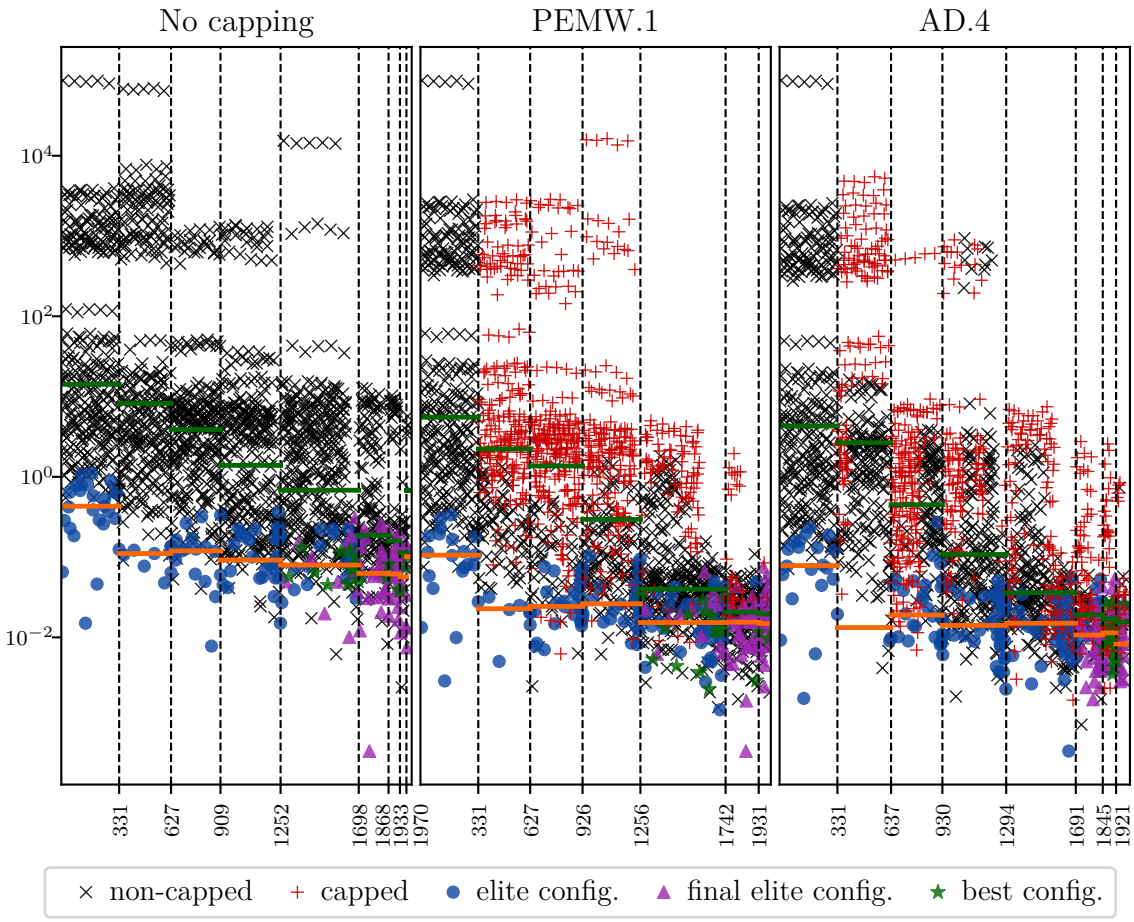


Figure 4.10 – Evolution of the automatic configuration of ACOTSP using the conservative capping methods.

from 1 to 2000. The quality is the relative deviation (on a logarithmic scale) from the best known solutions. A vertical dashed line marks the beginning of each irace iteration with the respective budget used so far. We also indicate if the execution was capped (+) or not (×), and the execution of elite configurations. A blue circle represent the execution of a configuration selected as elite in the corresponding iteration. A purple triangle represents the execution of a configuration which became elite in the last iteration, i.e. a final elite configuration. Executions of the best final configuration are represented by a green star. This is the configuration used to evaluate the quality of the irace run. In the case of capped executions (+ marker), we executed them again until the cut-off effort, i.e. without capping it, to obtain the quality of the complete execution. Therefore, the quality values of capped executions are those which would be obtained if they were not capped. Finally, the horizontal lines in each iteration represent the median relative deviations of all executions (green) and of the executions of elite configurations (orange) obtained in that iteration.

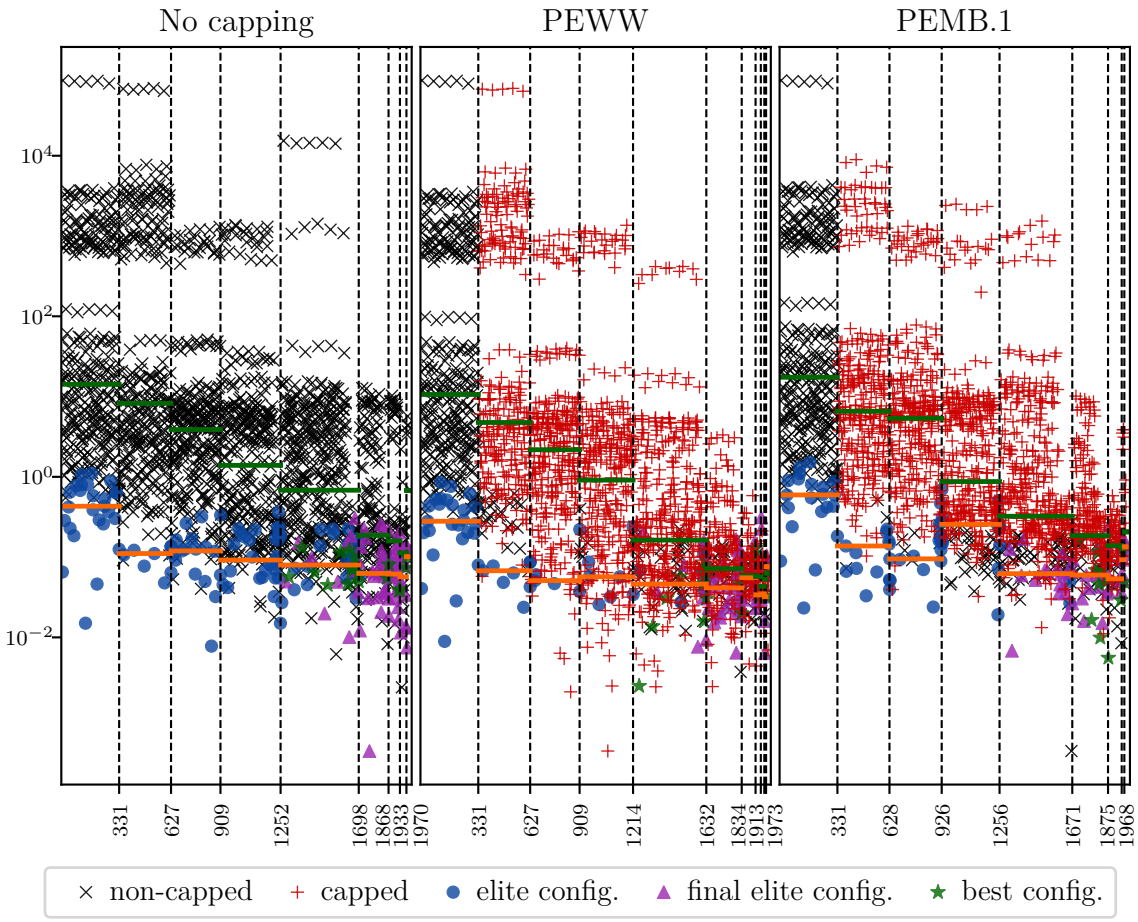


Figure 4.11 – Evolution of the automatic configuration of ACOTSP using the aggressive capping methods.

The behavior of how the observed execution quality changes over iterations is very similar for the different scenarios. We can see that the capping methods are effective in identifying poor performers and then stop executions that will not find the best solutions. Almost all executions capped by the analyzed methods turned out to be bad performers when executed until completion, as we observe in the final quality of capped executions in Figures 4.10 and 4.11. We can also analyze the aggressiveness of the recommended methods by looking at the amount of capped executions, which is clearly bigger in methods PEWW and PEMB.1. Besides that, we observe that the separation between capped and non-capped executions is higher in the conservative methods, indicating more tolerance in allowing configurations to run until completion. On the other hand, in some iterations the aggressive methods turn out to cap almost all executions of non-elite configurations. Finally, for users seeking a conservative method, we recommend to use PEMW.1, for those seeking an aggressive method, we recommend to use PEMB.1.

4.3.4 Time as Budget

A common scenario is using a time limit for the budget of the configuration process. We designed an experiment to evaluate the benefits of using capping in these conditions. We defined tight configuration time limits of 21000 s for ACOTSP, 3200 s for HEACOL, 700 s for TSBPP, 7000 s for HHBQP, 3500 s for LKH, and 21000 s for SCIP. This means around 350 non-capped executions for ACOTSP, HEACOL, HHBQP, LKH and SCIP, which had a budget of 2000 executions in previous experiments, and around 100 non-capped executions for TSBPP, which had a budget of 500 executions. These budget values make the configuration a challenging task.

When using the total execution time as budget, irace first estimates the time required by a single execution by evaluating a few random configurations. Based on this estimated time, it plans the iterations to be performed and the number of executions in each of them. Every time an iteration finishes, the time required by each previous execution is used to update the time estimate and re-plan the next iterations. When using capping, the time saved by early stopping poorly-performing executions becomes available to evaluate more configurations in the next iterations. The redistribution of the saved time is performed by irace considering the average time used so far in each execution. Thus, irace implicitly uses the amount of capping done to plan the next iterations. Thus, when using capping, we expect that the time saved is used to further explore the configuration space.

We evaluated all capping methods in the described scenario with 5 independent runs of irace. Table 4.5 shows the percentage of increase in the number of executions, generated configurations and evaluated instances, in comparison to configuring with no capping. We observe that the capping methods can help irace to make better use of the available budget, since poor performers are discarded early and the saved time can be used to further explore the configuration space. When using capping, irace samples more configurations and performs more executions during the tuning process. Besides that, it uses more instances to evaluate the quality of the configurations. For example, the increase in the number of configurations ranges from around 30% (less aggressive method AD.2) to around 2250% (more aggressive method PEMB.5). The corresponding increase in the number of total executions exceeds 1500% in the most aggressive methods.

Table 4.5 – Evaluating the capping methods with a configuration time budget. We show the percentage of increase in the number of executions, configurations generated, and instances evaluated for each capping method, in comparison with using no capping.

Capping method	Exe. [%]	Conf. [%]	Ins. [%]	Capping method	Exe. [%]	Conf. [%]	Ins. [%]
PEWW	193.2	203.2	19.0	PD.4	78.8	71.9	18.2
PEBB	1598.2	2023.5	63.8	PD.6	108.7	95.3	22.0
PEMW.1	49.2	50.7	1.0	PD.8	202.6	193.1	25.8
PEMW.3	106.2	102.5	17.6	AEWW	55.1	55.1	4.6
PEMW.5	264.7	298.6	29.1	AEBB	222.5	252.4	28.6
PEMB.1	258.6	305.4	25.4	AD.2	33.9	31.1	7.6
PEMB.3	627.4	753.2	45.3	AD.4	51.8	48.0	8.5
PEMB.5	1792.1	2256.6	66.7	AD.6	92.4	85.2	24.5
PD.2	61.1	56.4	12.8	AD.8	171.1	177.6	20.4

Table 4.6 presents the mean relative deviation (and the mean absolute deviation for HHBQP) from the best known solutions when configuring with each capping method and using the budget as a total configuration time. These values were determined by running 5 replications of the best configuration found in each irace run on the set of test instances. We highlight in bold the deviation values less than the one obtained by configuring with no capping. For most of the scenarios, the configurations found by using capping performed better than those obtained without capping. In HEACOL, TSBPP, HHBQP and LKH, the use of capping allowed irace to find configurations competitive to those obtained in the experiment with executions as budget (Table 4.2 and Figure 4.9), but using less computational effort.

Given the above results, we recommend using AEBB for scenarios where the configuration budget is defined relative to the time of the target algorithm. Although the percentage increase in configurations and executions achieved by AEBB is more modest than for other methods (Table 4.5), these additional executions lead to consistent improvements in quality for all scenarios evaluated here (Table 4.6). Thus, the AEBB capping method improves the quality of the automatic tuning of optimization algorithms in this type of scenario.

4.4 Discussion

We have proposed capping methods that speed up the automatic configuration of optimization algorithms. Previous methods (HUTTER et al., 2009b; PÉREZ

Table 4.6 – Average deviation of the resulting configurations with configuration time as budget. The values better than configuring with no capping are shown in bold and the three best values of each scenario are underlined.

Capping method	ACOTSP	HEACOL	TSBPP	HHBQP	LKH	SCIP
No capping	0.46	4.48	1.31	206.91	0.11	0.12
PEWW	0.51	**4.12**	1.31	**81.37**	**0.05**	**0.09**
PEBB	0.60	**4.24**	**1.24**	**117.81**	**0.06**	**0.11**
PEMW.1	**0.41**	**4.29**	1.31	**98.58**	**0.05**	**0.09**
PEMW.3	**0.41**	**4.20**	1.31	**78.77**	**0.06**	**0.05**
PEMW.5	0.56	**4.18**	1.31	**89.93**	**0.06**	**0.09**
PEMB.1	**0.40**	**4.21**	1.31	**80.94**	**0.07**	0.17
PEMB.3	0.54	**4.18**	**1.24**	**93.26**	**0.07**	0.13
PEMB.5	0.53	**4.40**	1.31	**62.43**	**0.10**	0.12
PD.2	**0.43**	**4.18**	1.38	**72.82**	**0.10**	0.23
PD.4	**0.41**	**4.26**	1.40	**165.60**	**0.05**	0.24
PD.6	**0.42**	**4.25**	1.31	**35.62**	**0.09**	0.24
PD.8	0.59	**4.30**	1.40	**39.64**	**0.07**	0.20
AEWW	**0.40**	**4.17**	**1.24**	**60.30**	**0.08**	0.13
AEBB	**0.40**	**4.20**	**1.24**	**33.21**	**0.05**	**0.10**
AD.2	**0.39**	**4.09**	**1.24**	**84.22**	**0.06**	0.13
AD.4	**0.38**	**4.28**	**1.24**	**65.77**	**0.10**	0.17
AD.6	**0.44**	**4.15**	**1.24**	**67.05**	**0.08**	0.18
AD.8	**0.42**	**4.34**	**1.24**	**83.33**	**0.06**	0.23

CÁCERES et al., 2017b) were designed for the configuration of decision algorithms and are not suitable for optimization scenarios. The methods described in this chapter use the previous executions to compute a performance envelope, which is used to evaluate new executions and cap those with unsatisfactory performance. The experimental results show the effectiveness of the capping methods to reduce the computational effort of the automatic algorithm configuration, while keeping the quality of the resulting configurations. We identified two conservative (PEMW.1 and AD.4) and two aggressive (PEMB.1 and PEWW) methods, which have been shown to be robust and present good trade-offs between the saved effort and the quality of the final configurations. Their average effort savings ranges from 20% to 45% of the configuration time with no capping, and the resulting configurations are comparable in terms of quality.

We also evaluated the proposed capping methods with the total execution

time as configuration budget. In this case, the capping methods helped to discard poorly-performing configurations and allow irace to use the saved time to better explore promising regions of the parameter space. We recommend AEBB for this type of scenario as it improves the results over no capping in all benchmarks. These results indicate that capping can also be helpful to scale the automatic configuration techniques to challenging scenarios (MASCIA; BIRATTARI; STÜTZLE, 2013; STYLES; HOOS, 2013).

The software implementing the capping methods presented in this chapter was named capopt and is available online (SOUZA; RITT; LÓPEZ-IBÁÑEZ, 2020)[1]. We also provide supplementary material for this chapter in Souza, Ritt and López-Ibáñez (2021)[2], containing the configuration scenarios, source code for the target algorithms and all experimental data. Appendix A gives more information about the artifacts produced from this research.

[1]The capopt package is available at https://github.com/souzamarcelo/capopt.

[2]The supplementary material for this chapter is available at https://github.com/souzamarcelo/supp-cor-capopt.

5 IMPROVED PARAMETER REGRESSION MODELS

> *The sciences do not try to explain, they hardly even try to interpret, they mainly make models. By a model is meant a mathematical construct which, with the addition of certain verbal interpretations, describes observed phenomena. The justification of such a mathematical construct is solely and precisely that it is expected to work.*
>
> — John von Neumann

A particular characteristic of the configuration approaches we study in this work, e.g. irace (LÓPEZ-IBÁÑEZ et al., 2016), SMAC (HUTTER; HOOS; LEYTON-BROWN, 2011) or GPS (PUSHAK; HOOS, 2020), is that they determine *fixed* configurations for the whole set of problem instances. In such configurations, parameters assume constant values when solving different instances, no matter the different features they present. It is known, however, that good parameter values may vary depending on instance features (see Muja and Lowe (2009), Smith-Miles et al. (2014), and El Yafrani and Ahiod (2018) for some examples), including problem-dependent features, e.g. statistics about clauses and variables in boolean satisfiability (ANSÓTEGUI et al., 2016), or problem-independent ones, e.g. the instance size (BÖTTCHER; DOERR; NEUMANN, 2010) or characteristics of the solution landscape (REEVES, 1999; MERZ, 2004; SCHIAVINOTTO; STÜTZLE, 2007; WATSON, 2010; MERSMANN et al., 2011; FRANZIN; STÜTZLE, 2020). From a machine learning perspective, Bengio, Lodi and Prouvost (2021) argue that traditional algorithm configuration approaches correspond to constant regression.

In this chapter, we explore additional regression models for algorithm configuration. Parameters are represented by models that define their values according to the instance size. Regarding instance features, in practice there are no features defined for most problems and determining a good feature set can be highly complex. For example, the instance features successfully used in the literature for different problem domains are result of much research effort (HOOS; HUTTER; LEYTON-BROWN, 2021; KERSCHKE et al., 2018). Besides defining and filtering unhelpful features (KROER; MALITSKY, 2011), practitioners need to compute them for the instances of interest, which is sometimes computationally expensive. Contrary to the goals of automating the algorithm configuration, this makes the process complex

and increases human effort. Therefore, we explore the instance size as the single feature, since it is problem-independent and available "for free", i.e. there is no costly computation involved. Regarding the parameter regression models, we first explore a simple yet effective linear model on the instance size, then extend it to a piecewise linear model based on multiple support points. We also apply a log-log transformation to the parameter and instance size spaces, and then apply the linear model to the transformed scenario. Both piecewise and log-log linear approaches can address cases when the relation between instance size and optimal parameter values is not linear.

The rest of this chapter is organized as follows. We review the literature and discuss the related work in Section 5.1. We present our proposed approach in Section 5.2, detailing the basic linear model and the techniques for dealing with nonlinear behavior. We also compare these models to each other on an artificial scenario to illustrate their modeling capabilities. We evaluate the proposed methods on four configuration scenarios and discuss the results in Section 5.3. We give some concluding remarks and directions for future research in Section 5.4.

5.1 Related Work

In many problems, a configuration with good performance on a given instance may perform poorly on others. Using instance features to characterize different types of instances and incorporating this information into the configuration process can considerably improve algorithm performance (HUTTER; HAMADI, 2005; ANSÓTEGUI et al., 2016). However, this task is not simple, since one needs to define how to map instance features to high-performing configurations, considering only executions on a limited set of training instances.

One of the most common approaches for instance-specific algorithm configuration rely on empirical performance models. These models predict the algorithm running time, using instance features and parameter values (i.e. algorithm configurations) as predictor variables. Once the model is trained and given a new instance described by its features, the model is used to search for high-performing configurations. Hutter et al. (2006, 2005) use linear regression models proposed in Leyton-Brown, Nudelman and Shoham (2002) and apply this approach to configure algorithms for boolean satisfiability. Given the nature of the empirical performance

models used in Hutter et al. (2006), which predicts the algorithm running time, this approach is not directly applicable to optimization scenarios, in which the algorithm is configured to minimize the cost of the best found solution. This approach was also used by Belkhir et al. (2016, 2017) to configure the covariance matrix adaptation-evolution strategy (CMA-ES) for black-box continuous optimization. They also minimize the algorithm running time, outperforming default and fixed configurations.

An alternative approach, named ISAC (MALITSKY; SELLMANN, 2010; KADIOGLU et al., 2010; MALITSKY, 2014), consists in first clustering the training instances according to their instance features. The algorithm is configured for each cluster, i.e. using only its instances and obtaining optimized fixed configurations for each of them. When solving a new instance, the configuration associated with the closest cluster is chosen. Kadioglu et al. (2010) use the g-means algorithm for clustering and the GGA configurator (ANSÓTEGUI; SELLMANN; TIERNEY, 2009). The proximity measure used in g-means and to associate instances to clusters is the Euclidean distance with normalized feature values. Liefooghe et al. (2017) explore the same ideas, using irace to configure a memetic algorithm for black-box optimization problems. The success of this approach depends on different factors, e.g. choosing a discriminative set of instance features and normalizing them adequately, as well as choosing appropriate clustering methods. While the traditional algorithm configuration can be seen as a constant regression, cluster-based approaches can be seen as piecewise constant nearest neighbors regression (BENGIO; LODI; PROUVOST, 2021).

More recently, El Yafrani et al. (2021) proposed MATE, a configurator that represents parameters as expressions of the instance features. They use genetic programming to evolve such expressions, which are then used to determine the parameter values to solve new instances. In contrast to the aforementioned approaches, in MATE the configuration process searches for a direct mapping from instance features to parameter values, i.e. without using empirical performance models or clustering approaches. El Yafrani and Ahiod (2018) use a similar approach to configure the number of trials of a simulated annealing algorithm for the traveling thief problem. They divide the instances in groups according to their size and determine the best parameter value for each group. The resulting values are then used in a linear interpolation, which determines the parameter value as a function of the instance size.

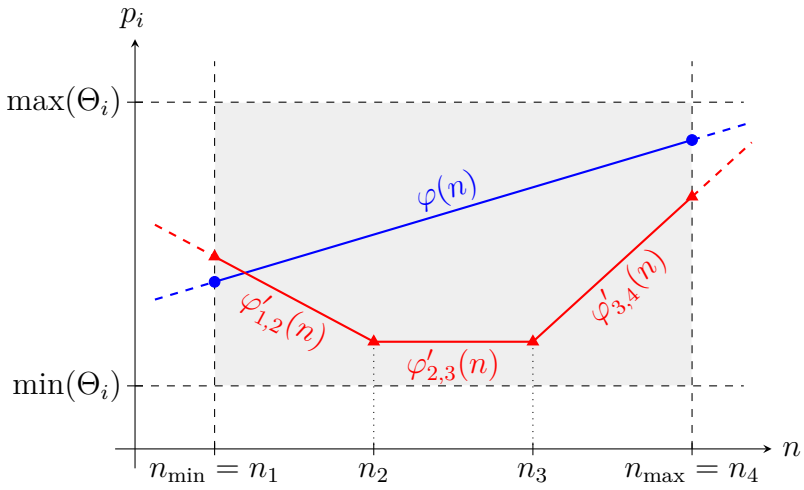


Figure 5.1 – Ideas behind linear (●) and piecewise linear (▲) regression models for algorithm configuration. The value of parameter p_i is set by the linear model φ or the piecewise linear model φ' (3 pieces in this example) for any instance size n.

Mascia, Birattari and Stützle (2013) also explore linear functions on the instance size to determine parameter values for an iterated local search and a tabu search. They propose an experimental protocol to configure such algorithms for solving large instances of the quadratic assignment problem, based on evaluations performed on small instances only. They use a strategy to select the cut-off running time of the target algorithm on the small instances, and determine a policy to set parameter values for larger instances by extrapolating from the configurations obtained for small instances.

5.2 Model-Based Parameters

The general idea of our proposal is to search for high-quality models for each numerical parameter, instead of exploring parameter values directly. Therefore, we transform the configuration space, replacing each parameter by a model. Then, a configurator calibrates such models to map the instance size to optimal parameter values.

5.2.1 Exploring Linear Models

Given a numerical parameter p_i with domain Θ_i, we define a model $\varphi : \mathcal{F} \to \Theta_i$ to determine its value based on a set of feature values $\mathcal{F}$. Since this work focuses on the instance size n as the single feature, $\mathcal{F}$ corresponds to the set of integer

numbers $\mathbb{Z}$. Our first approach assumes the optimal parameter response to be linear on the instance size, thus $\varphi(n) = an + b$, where a and b are the coefficients of the model. Figure 5.1 presents an example of such a linear model φ (blue segment with circle markers) on the instance size n. Note that this example highlights the area of interest defined by the allowed interval for parameter p_i, i.e. $[\min(\Theta_i), \max(\Theta_i)]$, and the possible values of instance size, defined by $n_{\min}$ and $n_{\max}$. Despite that, model φ can be used to determine parameter values for instance sizes out of the given interval, by preventing going outside the bounds of the parameter, i.e. $p_i = \min(\max(\varphi(n), \min(\Theta_i)), \max(\Theta_i))$.

Based on the above ideas, we can use a configurator to find a linear model that maps the instance size to optimal parameter values. To do this, we define a configuration space with two parameters, corresponding to the function values at $n_{\min}$ and $n_{\max}$ (i.e. $\varphi(n_{\min})$ and $\varphi(n_{\max})$), with domain Θ_i. From these two points we can derive the linear model to be used for determining the parameter value for any instance size. These two points are represented by the round markers associated with the linear model $\varphi(n)$ shown in Figure 5.1.

5.2.2 Addressing Nonlinear Behavior

Although being more representative than a simple constant model, the linear model presented above may not represent well scenarios with nonlinear relations between instance size and optimal parameter values. To address such cases, we explore a piecewise linear model based on multiple support points and defined by a linear interpolant (MONTGOMERY; PECK; VINING, 2021). Formally, we consider m support points, i.e. instance sizes $n_1 < n_2 < \cdots < n_m$, and interpolate each pair of sizes (n_i, n_{i+1}) with linear functions. Then, the parameter value for a given instance size $n_i \leq n \leq n_{i+1}$ is given by

$$\varphi'(n) = (1 - \alpha)\varphi'(n_i) + \alpha\varphi'(n_{i+1}),$$

with

$$\alpha = \alpha(n) = \frac{n - n_i}{n_{i+1} - n_i}.$$

Figure 5.1 presents an example of a piecewise linear model φ' (red segment with triangle markers) that uses four support points, thus defining three pieces and

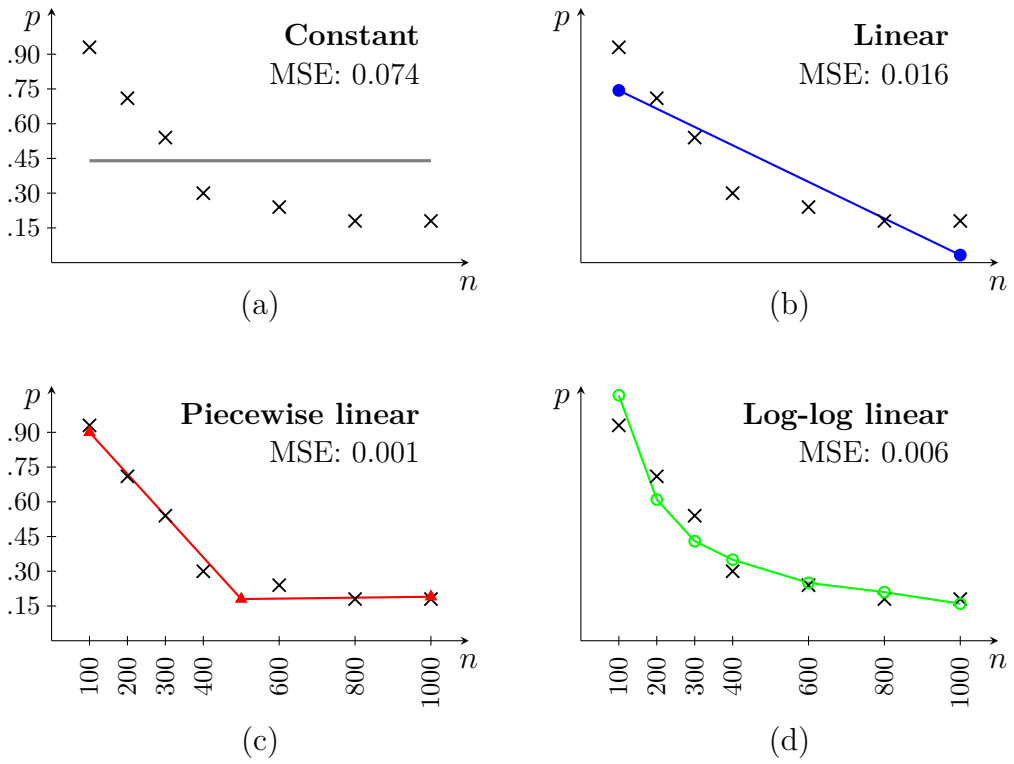


Figure 5.2 – Comparing all parameter regression models on a nonlinear artificial scenario. We show the optimal parameter response (i.e. the optimal values of parameter p for different instance sizes n), as well as optimal approximations and the corresponding mean squared errors (MSE) for constant (a), linear (b), piecewise linear (c) and log-log linear (d) models.

the corresponding linear functions. Note that although the values given by both linear and piecewise linear models at the extreme points ($n_{\min}$ and $n_{\max}$) are similar, the behavior between them is completely different and the only the latter is able to model such a nonlinearity. In this approach, the configurator sets the values at each support point, i.e. $\varphi'(n_i)$, for $i \in [m]$.

Finally, we propose an alternative approach to deal with nonlinear optimal parameter response. We apply log-transformations on the instance sizes and parameter values, in order to improve the linearity. The resulting log-log linear model is given by $\log(\varphi(n)) = a \log(n) + b$, and the configurator tunes φ in the transformed space. The parameter values are transformed back to the original interval only when used in the algorithm being configured.

5.2.3 Comparing Parameter Regression Models

Figure 5.2 presents a concrete example using an artificial scenario, upon which we analyze and compare the available models. In this example, there is a single

Table 5.1 – Algorithm configuration scenarios used to test the parameter regression models. Showing the values of budget, the range of instance sizes, parameter descriptions, types and domains.

Scenario	Budget	Instance size	Parameter	Domain
ILSBQP	$\{300, 1000, 2000\}$	$[250, 2500]$	Perturbation size (int)	$[1, 2500]$
BSFS	$\{300, 1000, 2000\}$	$[100, 10000]$	Randomness control (int)	$[50, 100]$
HHTA	$\{1000, 2000, 5000\}$	$[60, 395]$	Tabu tenure (int)	$[1, 395]$
			Elite set size (int)	$[1, 100]$
			Distance factor (real)	$[0.1, 0.5]$
TSCPP	$\{1000, 2000, 5000\}$	$[100, 500]$	Tabu tenure (int)	$[1, 100]$
			Perturbation size (int)	$[1, 100]$
			Candidates for pert. (int)	$[1, 50]$

parameter $p \in [0, 1]$, whose optimal values are presented by the cross markers for different instance sizes $n \in [100, 1000]$. For each model, we present an approximation that minimizes the mean squared error (MSE). As we can see in Figure 5.2a, the constant model implemented in most configurators does not represent well the optimal parameter response, thus obtaining a MSE of 0.074. The linear model (Figure 5.2b) improves considerably the representation (MSE of 0.016) in comparison to a constant value, but given the nonlinear behavior of the optimal parameter response, the linear model cannot approximate the optimal parameter values for some instance sizes (e.g. for $n = 400$).

We see a much better mapping when using the piecewise (Figure 5.2c) and log-log (Figure 5.2d) linear models, with MSEs of 0.001 and 0.006, respectively. In this particular example, the piecewise linear model is slightly better in terms of MSE. However, we must consider that this approach increases the number of parameters by a factor of 3 (considering 2 pieces), while the log-log linear model keeps the number of parameters fixed in 2.

5.3 Computational Experiments

In this section, we evaluate the proposed models on four configuration scenarios with different characteristics. After presenting the experimental setup and a summary of the configuration scenarios, we show the ability of our approaches in modeling

Table 5.2 – Average absolute deviations of constant and linear models on ILSBQP. Showing values for each instance size and the overall average results for different budgets. The best values for each size and the best overall are highlighted in bold.

Size	Constant			Linear		
	300	1000	2000	300	1000	2000
250	11.2	11.6	9.0	3.4	2.4	**0.0**
500	12.4	12.3	5.2	-11.0	-13.5	**-16.0**
1000	2708.0	2688.7	2664.0	2651.8	2637.6	**2637.0**
2500	2060.7	2282.2	2275.1	620.2	801.3	**533.0**
Avg.	1198.1	1248.7	1238.3	816.1	857.0	**788.5**

the relation between instance size and optimal parameter values, and discuss the resulting gains in algorithm performance for each scenario.

5.3.1 Experimental Setup

In our experiments we use the irace (LÓPEZ-IBÁÑEZ et al., 2016) configurator and test the parameter regression models on four configuration scenarios: ILSBQP, BSFS, HHTA and TSCPP (see Chapter 3 for more details). For ILSBQP and TSCPP, the instance size is defined by the number of variables and the amount of vertices, respectively. For BSFS, we define the instance size as the product between the number of jobs and machines. For HHTA, the instance size is the product between the number of desks and tests (adding an additional "nonexistent" test for assigning to unoccupied desks), which is also the resulting number of variables of the unconstrained binary quadratic program after the reduction from TA. Since the proposed models increase the configuration space (by up to four times the number of numerical parameters), the configuration task becomes more complex and larger configuration budgets might be necessary. Therefore, we test the proposed approaches under different budgets to assess their impact on the quality of the configurations produced. The budgets, instance sizes and the parameters of each configuration scenario are given in Table 5.1.

To evaluate the proposed approaches, we run five replications of irace for all configuration settings (configuration scenarios, budgets and models) using the set of training instances. The best configuration returned by irace in each replication is evaluated on the set of test instances with five more replications. To evaluate algorithm performance, we compute the average deviations from the best known

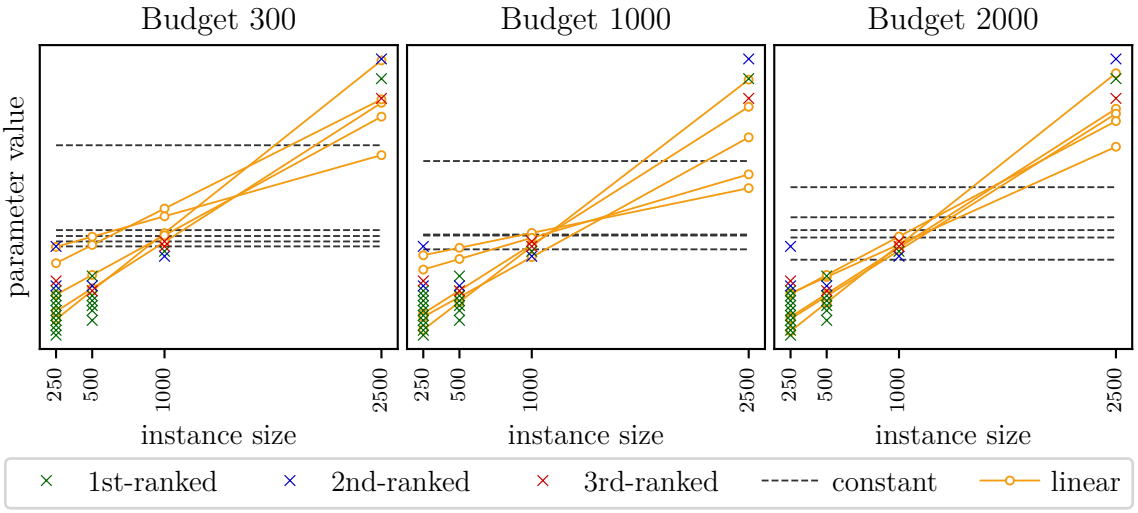


Figure 5.3 – Optimal parameter values and configurations produced using constant and linear models for ILSBQP. Showing the parameter values for each instance size and for different configuration budgets.

solutions.

Our experiments were ran on a computer with a 12-core AMD Ryzen 9 3900X processor running at 3.8GHz and 32GB main memory, under Ubuntu Linux, using only one core for each execution. The parameter regression models were written in Python (SOUZA; RITT, 2022d) and tested with Python 3.8.10, using irace version 3.3. The target algorithms are implemented either in C or C++ and compiled with the GNU C/C++ compiler version 9.3.0.

5.3.2 Configuring ILSBQP

Our first experiment applies the constant and linear models for configuring the ILSBQP scenario. The results are shown in Table 5.2. We report the average absolute deviations from the best known solutions for each configuration budget and each instance size, as well as the overall average deviations. The best results for each instance size and the best overall are shown in bold. We can see that the configurations obtained using the linear model perform better than those produced using the constant model. The absolute deviations are consistently better with the linear model, even with a low configuration budget. Moreover, even the constant configurations produced using a budget of 2000 executions are worse than the linear configurations produced with a budget of only 300 executions.

Additionally, we evaluated ILSBQP on all test instances using several values for the perturbation size, in order to visualize the relation between instance size

Table 5.3 – Average relative deviations of constant and linear models on BSFS. Showing values for each instance size and the overall average results for different budgets. The best values for each size and the best overall are highlighted in bold.

Size	Constant			Linear		
	300	1000	2000	300	1000	2000
100	1.34	1.43	1.31	**1.28**	1.23	1.31
200	2.77	2.85	2.74	**2.72**	2.73	2.74
250	0.30	0.30	0.28	**0.27**	**0.27**	0.28
400	2.17	2.21	2.16	**2.14**	**2.14**	2.15
500	2.50	2.50	2.50	**2.48**	**2.48**	**2.48**
1000	3.63	3.57	3.62	3.61	**3.56**	3.59
2000	**3.10**	3.06	**3.10**	3.11	**3.10**	**3.10**
4000	4.02	**4.00**	4.04	4.03	4.03	4.02
10000	2.00	2.00	2.01	1.98	**1.96**	1.97
Avg.	2.43	2.44	2.42	2.40	**2.39**	2.40

and optimal parameter values, and to compare it with the configurations produced in the previous experiment. Figure 5.3 shows the parameter values that perform best on ILSBQP. Green cross markers represent the first-ranked parameter values (i.e. sometimes different values achieve the same average result), while the blue and red ones represent the second- and third-ranked parameter values, respectively. We also show the constant configurations (gray dashed segments) and the linear ones (orange continuous segments) produced using budgets 300, 1000 and 2000.

We observe that the optimal values for the perturbation size increase as the instance size grows. The constant configurations cannot model this relation properly. On the other hand, we observe a good approximation of this relation when using the linear model, i.e. the values defined by the linear model for each instance size are very close to the observed optimal values, even using a budget of only 300 total executions. This ability to model the relation between instance size and optimal parameter values explains the observed performance gains obtained by configuring ILSBQP using the linear model (Table 5.2).

5.3.3 Configuring BSFS

We executed the experiments presented above on the BSFS scenario. Table 5.3 presents the average relative deviations from the best known solutions obtained using constant and linear models. Similar to the results on ILSBQP, we observe

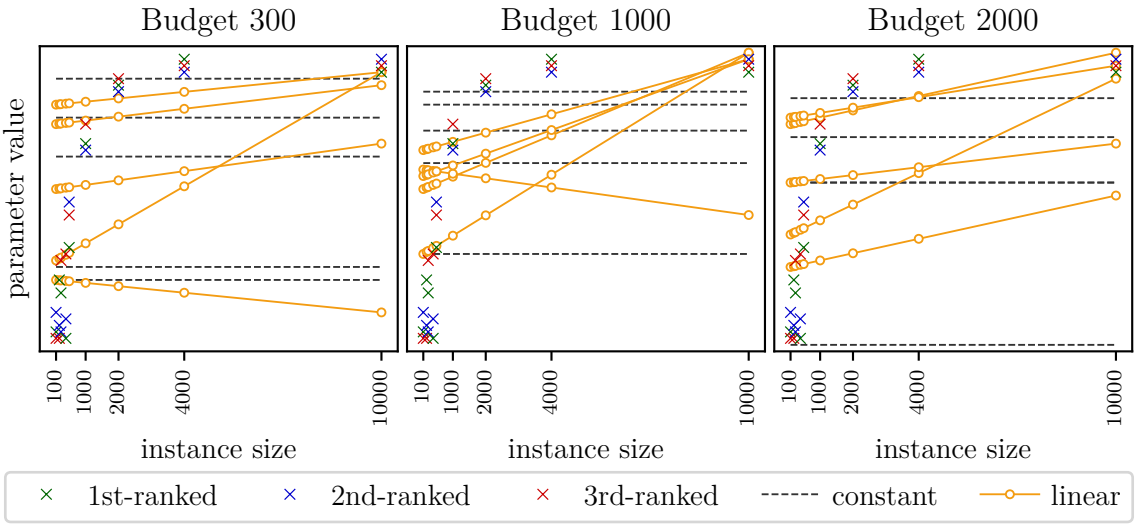


Figure 5.4 – Optimal parameter values and configurations produced using constant and linear models for BSFS. Showing the parameter values for each instance size and for different configuration budgets.

a better performance of bubble search under linear configurations. Although the differences between both models are weaker in comparison to the previous scenario, the experiments using the linear model are consistently better than using the constant model, even with a low budget. In fact, there is not much difference in increasing the budget for the linear model.

Figure 5.4 shows the relation between instance size and optimal parameter values, and also the configurations produced for the BSFS scenario. By looking at the best parameter values (cross markers), we note an interesting behavior. The response of the optimal parameter values is not linear in the instance size. Instead, there is a fast increase between sizes 100 and 2000, followed by a much slower increase until size 10000. Although better than the constant model, the linear approach cannot model well the observed behavior. Nevertheless, the increase in the parameter value as the instance size grows is captured by the linear model, with some exceptions in two particular configurations. This explains the better performance of the linear configurations. Due to the observed behavior, this scenario is further explored in Section 5.3.6, where we apply the piecewise and log-log linear models to address its nonlinearity.

5.3.4 Configuring HHTA

We made the same analyses for the HHTA scenario. The average relative deviations of constant and linear models are shown in Table 5.4. The performance of

Table 5.4 – Average relative deviations of constant and linear models on HHTA. Showing values for each instance size and the overall average results for different budgets. The best values for each size and the best overall are highlighted in bold.

Size	Constant			Linear		
	1000	2000	5000	1000	2000	5000
60	**0.00**	0.06	**0.00**	**0.00**	0.08	0.03
80	0.38	0.39	0.33	0.35	0.18	**0.15**
100	1.28	1.20	1.14	0.12	**-0.07**	-0.04
141	3.71	3.44	3.90	1.89	1.65	**1.28**
180	2.91	2.24	2.66	1.84	1.97	**1.76**
188	3.76	3.72	3.04	2.71	**2.24**	2.41
235	2.91	2.99	2.79	2.89	**2.24**	2.27
237	1.67	1.43	1.67	1.22	**1.14**	1.26
240	2.95	2.86	3.12	2.78	2.42	**2.33**
300	5.02	4.76	5.55	3.16	3.24	**3.05**
316	3.63	3.20	3.39	1.22	**0.98**	1.16
395	7.68	7.60	6.71	2.26	1.98	**1.97**
Avg.	2.99	2.82	2.86	1.70	1.50	**1.47**

HHTA is substantially improved using the configurations given by the linear model. The average relative deviations when using constant configurations are in the interval $[2.82, 2.99]$. These average deviations decrease to 1.70 using a budget of only 1000 executions. When using greater budgets, it decreases to less than 1.50. Similar to previous scenarios, the use of the linear model consistently improves algorithm performance in all test cases.

We also analyze the relation between instance size and optimal parameter values in the HHTA scenario. Figure 5.5 presents the resulting visualizations, where each plot corresponds to one of the three parameters of HHTA and the configurations are those obtained using budget 1000. To produce each visualization, we test several values for the corresponding parameter with the other parameters fixed at their default values. Therefore, the optimal parameter values we show (cross markers) are optimal only under these conditions (i.e. when setting the other parameters to their default values). If we change the values of the other parameters, the optimal values may be different from those of Figure 5.5, given potential parameter interactions. We need to take this into account when analyzing the constant and linear configurations produced by irace. Since no parameter is fixed at any value, the configuration space is different from the one explored in the basic analysis and optimal parameter values may be different.

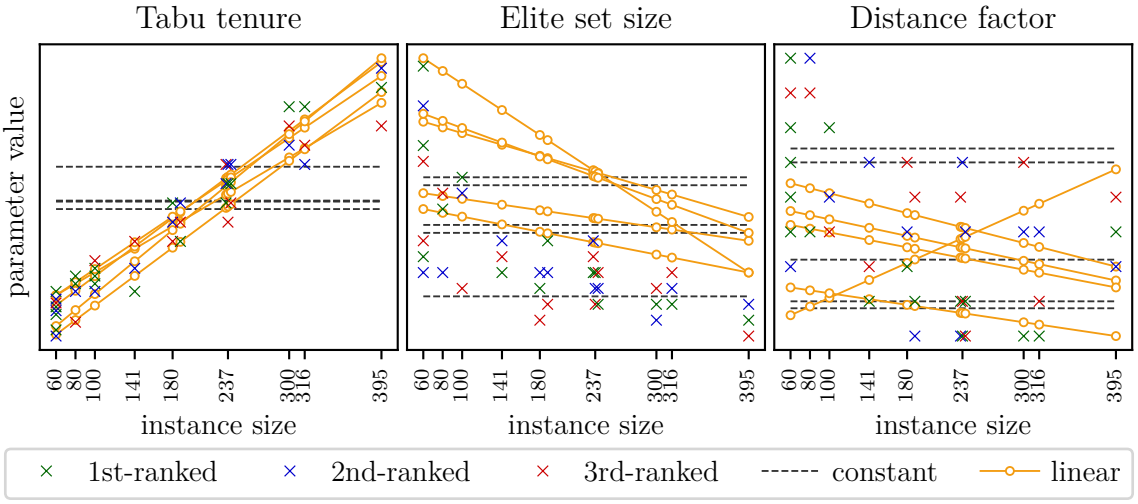


Figure 5.5 – Optimal parameter values and configurations produced using constant and linear models for HHTA. Showing the values for each parameter separately and for each instance size using a configuration budget of 1000 total executions.

Regarding the optimal parameter values of HHTA, we observe clear behaviors for the tabu tenure and elite set size parameters. As instance size grows, a larger tabu tenure performs better, while smaller elite sets are preferable. On the other hand, no clear relation between instance size and the distance factor is observed, although it seems to present some decrease of its optimal value with respect to the growth of instance size (e.g. observe the variation in the first-ranked green markers). We also note that the constant model finds configurations within the optimal ranges, but is not able to represent the relation with instance size. The linear model fits very well the behavior of the tabu tenure parameter, and also the decreasing behavior of the elite set size. With regard to the elite set size, however, the linear configurations present higher parameter values in comparison to the optimal ones. These differences can be an effect of the potential parameter interactions discussed above, i.e. better elite set sizes are found associated with different values for the other two parameters. For the distance scale parameter, the linear configurations lie within the range defined by the optimal parameter values and present decreasing behavior (except for one configuration).

5.3.5 Configuring TSCPP

Finally, the results on the TSCPP scenario are given in Table 5.5. Here, we note that all configurations solve the instances of size 100 easily, since the average

Table 5.5 – Average absolute deviations of constant and linear models on TSCPP. Showing values for each instance size and the overall average results for different budgets. The best values for each size and the best overall are highlighted in bold.

Size	Constant			Linear		
	1000	2000	5000	1000	2000	5000
100	0.32	**0.00**	0.16	**0.00**	**0.00**	**0.00**
200	29.48	64.94	36.54	5.90	**1.06**	5.68
300	82.80	**1.38**	22.80	91.76	122.64	42.10
400	164.60	169.64	223.72	137.12	149.90	**131.94**
500	778.84	956.16	791.92	594.88	**539.00**	567.56
Avg.	211.21	238.42	215.03	165.93	162.52	**149.46**

absolute deviations are always zero or very close to it. For instance size 300, we see that the constant configurations perform much better than the linear ones, especially those constant configurations obtained using a budget of 2000 total executions. For the other instance sizes, the linear configurations present better performance. The overall relative deviations also indicate the superiority of the linear model, with an average deviation of 149.46 (with budget 5000), better than the best average deviation of 211.21 obtained by the constant model.

Figure 5.6 shows the visualizations for the TSCPP scenario. To analyze one parameter, we also fix the other parameters at their default values. Thus, the same conditions regarding parameter interaction discussed above apply here. We note that instances with size 100 are solved easily, no matter the value for the three parameters. We also note that the optimal values for the tabu tenure parameter increase as instance size grows. This relation is mapped properly by the linear model, with an exception of one replication, which presents a decreasing behavior on instance size. On the other hand, there is no clear relation between instance size and the optimal values for population size and candidates for perturbation. In the latter, the linear configurations vary considerably and no tendency can be observed.

5.3.6 Nonlinear Models for BSFS

As discussed in Section 5.3.3, the relation between instance size and optimal parameter values for BSFS is nonlinear. As a consequence, the linear model cannot properly represent this relation. In this section, we revisit this scenario and test the

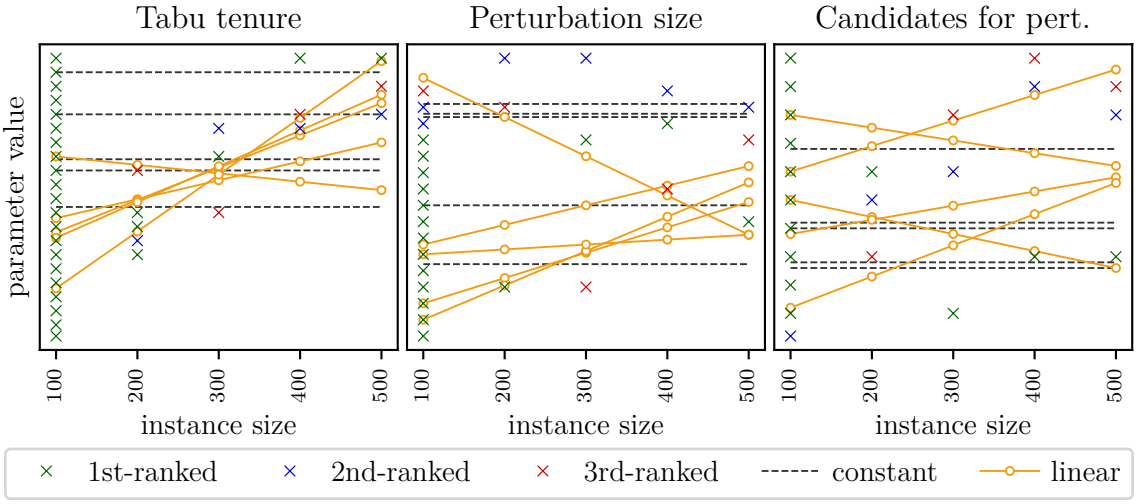


Figure 5.6 – Optimal parameter values and configurations produced using constant and linear models for TSCPP. Showing the values for each parameter separately and for each instance size using a configuration budget of 1000 total executions.

proposed approaches to deal with nonlinearity. In particular, we apply the piecewise linear model with 2 and 3 pieces (which use 3 and 4 support points, respectively), and the log-log linear model. We manually defined support points at 100, 2000 and 10000 for the piecewise linear model using 2 pieces, and at 100, 1500, 3000 and 10000 when using 3 pieces. In all cases, we use the same budget of 1000 total executions. The resulting average relative deviations are shown in Table 5.6. We also present the results of the piecewise linear model with a single piece, which reduces to the linear model (thus, the results for this column are the same of Table 5.3).

The approaches for addressing nonlinearity improve algorithm performance, with average deviations decreasing from 2.39 to 2.32. We note that the log-log linear model has better performance on small instances (up to 500), while the piecewise linear models are better on larger instances. Actually, the piecewise and log-log linear models are better than the linear model for all instance sizes, which points out the potential of nonlinear approaches to map this kind of relation between instance size and optimal parameter values.

Figure 5.7 shows the optimal parameter values for BSFS (the same presented in Figure 5.4) and the configurations obtained by the piecewise and log-log linear models. We observe that these approaches can model the optimal parameter response well. The values defined by these models are very close to the optimal values identified, which explains the associated performance gains. We also note that using 3 pieces improves the ability of the piecewise linear approach of modeling the optimal

Table 5.6 – Average relative deviations of the piecewise and log-log linear models on BSFS. Showing values for each instance size and the overall average results using a configuration budget of 1000 total executions. The best values for each size and the best overall are highlighted in bold.

Size	Piecewise linear			Log-log
	1	2	3	
100	1.23	1.14	1.09	**1.03**
200	2.73	2.65	2.61	**2.57**
250	0.27	0.26	0.25	**0.23**
400	2.14	2.11	2.07	**2.06**
500	2.48	**2.47**	**2.47**	**2.47**
1000	3.56	**3.54**	**3.54**	3.57
2000	3.10	**3.00**	3.01	3.07
4000	4.03	**3.91**	**3.91**	3.96
10000	1.96	**1.95**	**1.95**	**1.95**
Avg.	2.39	2.34	**2.32**	**2.32**

parameter response. For small instances (up to 500), the piecewise linear model with 2 pieces cannot model well the best parameter values. We also highlight the fact that the piecewise linear models find better configurations using the same budget as the linear model, even dealing with larger configuration spaces as a consequence of increasing the number of parameters.

Regarding the log-log linear model, we observe a good approximation of the optimal parameter response. For instance sizes 1000 to 4000, however, it gives values smaller than the optimal ones, which explains the worse performance on these instances in comparison to the piecewise linear models. For other instance sizes, the log-log linear model gives values very close to the optimal ones, which is in line with the good performance observed on these instances (Table 5.6; being also better than other models).

5.4 Discussion

In this chapter, we propose a new approach for instance-specific algorithm configuration considering the instance size as the single feature. Instead of configuring parameter values, we configure models that define these values according to the instance size. Specifically, we propose a basic linear model and let the configurator calibrate it to map instance sizes to parameter values. For scenarios in which this

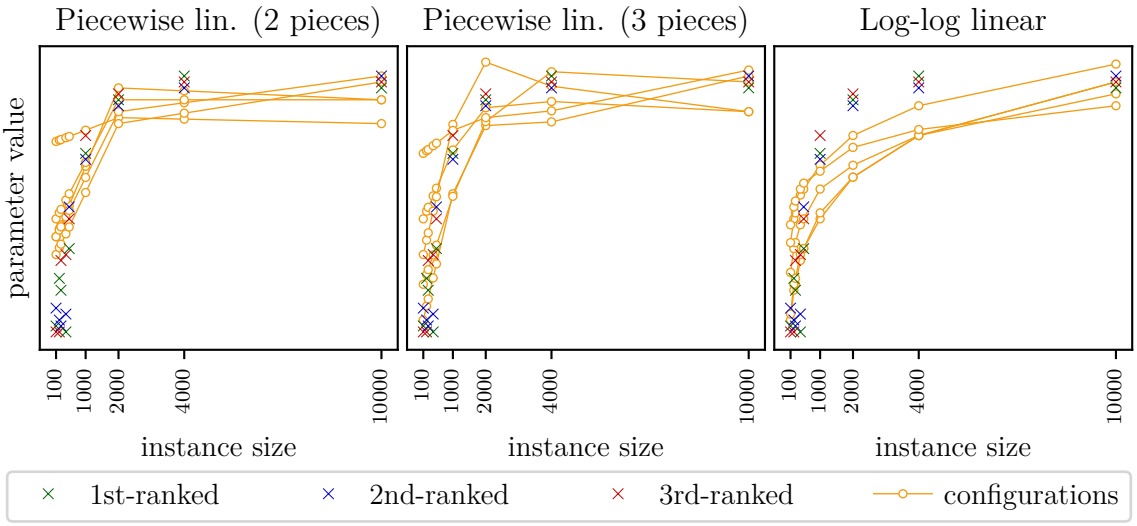


Figure 5.7 – Configurations for BSFS using the pointwise and log-log linear models. Showing the configurations produced using a budget of 1000 total executions. The piecewise linear models with 2 and 3 pieces use support points at $\{100, 2000, 10000\}$ and $\{100, 1500, 3000, 10000\}$, respectively.

relation is nonlinear, we propose a piecewise linear model, as well as a log-log linear model.

We tested these approaches on four configuration scenarios, showing that the configurations obtained by the proposed models are better than those obtained using the original configuration approach. In all cases, we observe performance gains by using the linear model, even with low configuration budgets. We also test the piecewise and log-log linear models on a scenario with a nonlinear optimal parameter response, and show that they are able to model such a nonlinearity. In this case, the configurations are better than the constant ones, and also better than those obtained using the basic linear model.

Although effective, the proposed models are simple. First, we consider a single, problem-independent, and maybe the most common instance feature: the instance size. Practitioners do not need to worry about feature definition and filtering, and no additional computational power must be spent in this regard. For scenarios where the instance size is unknown for some reason, one could use the size of the instance file as a surrogate value for the instance size. Concerning the proposed models, they are simple to understand and easy to visualize, i.e. their outcome is human readable and easily explainable. Besides that, these models can be computed and used efficiently. That said, we believe this kind of instance-specific approaches can improve algorithm configuration and reach good performance gains, as suggested by

the presented experimental results.

We provide a set of utility functions in implementing the proposed models in Souza and Ritt (2022d)[1]. We also provide supplementary material to this chapter in Souza and Ritt (2022e)[2], containing the configuration scenarios, source code for the corresponding algorithms, problem instances and further details about the experimental results. More information about the artifacts produced from this research are given in Appendix A.

[1]The implementations regarding the parameter regression models are available at https://github.com/souzamarcelo/regression-models-ac.

[2]The supplementary material for this chapter is available at https://github.com/souzamarcelo/supp-models-ac.

6 VISUAL ANALYSIS OF THE CONFIGURATION PROCESS

> *The improvement of understanding is for two ends: first, our own increase of knowledge; secondly, to enable us to deliver that knowledge to others.*
>
> — John Locke

As discussed previously, the use of configurators like irace allows users to adjust algorithms for obtaining high performance without the need of vast expert knowledge about the algorithm or the problem. The configuration process implemented in irace generates (usually significant volumes of) algorithm performance data that are used to guide the search for good configurations. The data produced by irace can be used to obtain insights about the configured algorithm and the configuration process itself.

Although irace can be used as a black-box method for configuring algorithms, in some cases it might be helpful to understand how the configurator works and analyze its execution, in order to obtain the best results from the configuration process and ensure the efficient use of the computational resources. This understanding is important when designing the configuration scenario, i.e. mainly the configuration space, training instances, and configuration budget. The configuration scenario can be setup inadequately, e.g. using too little or too much computational effort, which may lead to poor results or the waste of available computational resources. Using training instances that are not representative of typical problem instances may lead to overtuning (BIRATTARI, 2009) and poor results when using the algorithm in production. A detailed analysis of the configuration process helps to identify such cases and adjust the configuration scenario. Unfortunately, the analysis of the data generated by irace is not simple, since they must be processed and interpreted, and knowledge about the configurator is often necessary. In addition, there are no tools available to directly visualize the configuration process data and thus, promoting and simplifying the analysis of it.

In this chapter we present acviz, a visual tool to analyze runs of irace based on the graphical representation of the configuration process. The acviz tool provides two types of visualizations. The first shows the evolution of a single run of the configuration process performed by irace. The second visualizes the performance of the best found configurations on test instances and contrasts them with the

performance on the training instances used by irace. We first describe acviz and its functionalities (Section 6.1), then we present examples that show how it can be used to understand the configuration process, and how it can provide useful information to design better configuration scenarios (Section 6.2). Finally, we present a brief discussion on these ideas (Section 6.3).

6.1 The acviz Program

Given a log file produced by running irace, the acviz program provides visualizations of the configuration process. Figure 6.1 gives examples of the configuration of two different algorithms. A point (i, v) shows the performance v obtained in the ith evaluation in the configuration process. Note that each evaluation is associated to a unique configuration-instance pair (θ, π). The first example shows the configuration of an optimization algorithm. In this case, the performance value v is the relative deviation of the best solution found in each evaluation from an instance-based reference value. These reference values can be provided by the user when, for example, there are best known solutions for the instances or there is a current default configuration and its performance can be used as reference. When no reference value is provided, the best values found by irace are used. The plot also shows the beginning of each iteration by a vertical dashed line, with the number of evaluations (bottom) and the number of different instances (top) used until that iteration. This vertical line is presented in red for iterations in which a soft restart was applied. Finally, evaluations on different instances are indicated by different colors, and evaluations of elite configurations are represented using different markers (● for elite configurations of the current iteration, ◆ for configurations that were elite in the final iteration, and ★ for the best found configuration, i.e. the first ranked elite configuration of the final iteration).

The horizontal lines present the estimated performance of the elite (purple line) and non-elite (orange line) configurations in each iteration. The estimated performance is determined by the median of the results obtained by all configurations of the current iteration on all instances evaluated so far, considering evaluations in the current and previous iterations. Some of the non-elite configurations may not be evaluated on a subset of the instances, e.g. when the configuration is discarded in the middle of an iteration. For the calculation of the estimated performance, we replace

Table 6.1 – Arguments of acviz. Default options are shown in bold.

Argument	Options	Description
`--iracelog`	<log file>	The irace log file (.Rdata)
`--typeresult`	{*aval*, *adev*, ***rdev***}	Which values are presented
`--bkv`	<bkv file>	The file containing reference values
`--imputation`	{***elite***, *alive*}	Imputation strategy for missing values
`--scale`	{***log***, *lin*}	Scaling of the y-axis
`--noelites`	–	Disables different markers for evaluations of elite configurations
`--noinstances`	–	Disables coloring evaluations on different instances
`--pconfig`	[**0**, 1]	Identifies the configurations of the best evaluations
`--overtime`	–	Presents the configuration time on the x-axis
`--alpha`	[0, **1**]	The opacity of the points
`--timelimit`	[**0**, ∞]	Time limit used to evaluate decision algorithms
`--testing`	–	Presents the plot of the test phase
`--testcolors`	{***instance***, *overall*}	The scheme for the color map
`--exportdata`	–	Exports the data of the configuration process to a csv file
`--exportplot`	–	Exports the produced plot to pdf and png files
`--output`	<prefix>	The prefix name of the exported files
`--monitor`	–	Monitors the irace log file and updates the plot after each iteration

missing values by the worst result of the elite configurations, since the eliminated configuration is not better than the worst elite configuration (called *elite* imputation strategy). An alternative approach is to use the worst result of the configurations being evaluated in the current iteration (called *alive* imputation strategy).

Table 6.1 presents the input arguments of acviz. The command to produce the first visualization shown in Figure 6.1 is:

```
python3 acviz.py --iracelog irace.Rdata --bkv bkv.txt
```

which provides the irace log file to be used, in this case `irace.Rdata`, and the file containing the reference values used to compute the relative deviations (`bkv.txt`). Additional options control the elements of the visualization, like presenting the absolute performance values or the absolute deviations from the reference values (option `--typeresult`), or changing the imputation strategy. Users can also disable the coloring of instances and the markers of elite configurations, or tell acviz to show the ID of the configurations associated with the $p\%$ best performing evaluations of each iteration (option `--pconfig`). The opacity of the points can be changed and the default logarithmic scale of the y-axis can be disabled.

The second visualization in Figure 6.1 shows the configuration of a decision algorithm, where the performance of each evaluation is the running time used to solve the corresponding instance. In this case, the configuration budget is a time limit, then users can opt to plot the starting time of evaluations on the x-axis (option `--overtime`), making it possible to observe how the configuration time is distributed over the iterations, and identify evaluations that took a long time. To produce this visualization, we select to show absolute performance values in the y-axis and disable the logarithmic scale.

During the configuration process, evaluations that reach the running time limit without solving the instance are penalized by returning to irace the time limit multiplied by a penalization factor (PÉREZ CÁCERES et al., 2017a). If we inform the time limit to acviz, each evaluation with a result that exceeds this limit is presented in the upper border of the plot, indicating that these evaluations did not solve the instance (see those cases in the second visualization shown in Figure 6.1). The following command produces this visualization (observe that argument `--iracelog` can be omitted):

```
python3 acviz.py irace.Rdata --typeresult aval \
                             --scale lin \
                             --timelimit 10 \
                             --overtime
```

A second plot provided by acviz presents the results obtained by the best found configurations on the set of test instances (this requires the testing feature to be enabled when running irace). Figure 6.3 shows an example, presenting the results of the best elite configurations of each iteration and all elite configurations

of the last iteration. Each column in the plot is associated with a configuration. The acviz tool presents its ID and, in parenthesis, the iterations in which it was the first ranked elite configuration (e.g. 251 $(3, 4)$ means configuration 251 was the best ranked elite configuration in iterations 3 and 4). For the final iteration, we also present the rank of the corresponding configuration in the elite set in a subscript (e.g. 9_1 means that the configuration was ranked first in the 9th iteration). The instance name is black if the instance has been used during training and testing, and blue, if it has been used only for testing. The subplot on the left shows the mean relative deviations from the reference values that, as for the previous plot, can be provided using the `--bkv` option, or are determined by acviz based on the best values found during the execution of irace (in both training and test phases). The subplot on the right presents the ranking of each configuration on each instance, allowing us to compare the performance of different configurations across instances. The command to produce the visualization shown in Figure 6.3 indicates that acviz should present the plot of the test phase:

```
python3 acviz.py irace.Rdata --bkv bkv.txt --testing
```

In the visualization of the test phase, we can also use option `--typeresult` to present the mean absolute values or the mean absolute deviations from the reference values. The colors in the plot help to differentiate the performance obtained by the resulting configurations. In Figure 6.3, the color map is calculated according to the results obtained within each instance. Worst values for each instance are in red while the best values are in green. Alternatively, the color map can be defined according to the whole range of values in all instances, thus visualizing the overall performance obtained in the test phase.

When using the interactive presentation mode, acviz allows the user to control the visualization by moving the plot, zooming and controlling the margins of the figure. When positioning the cursor over a point, acviz shows a tooltip box with the corresponding evaluation number, the associated instance name and configuration ID. It is also possible to export the data and both of the plots. Finally, when option `--monitor` is enabled, acviz monitors the irace log file during the configuration process and updates the visualization after each iteration, allowing the user to analyze the evolution of the configuration process during its execution. All acviz options discussed above are summarized in Table 6.1.

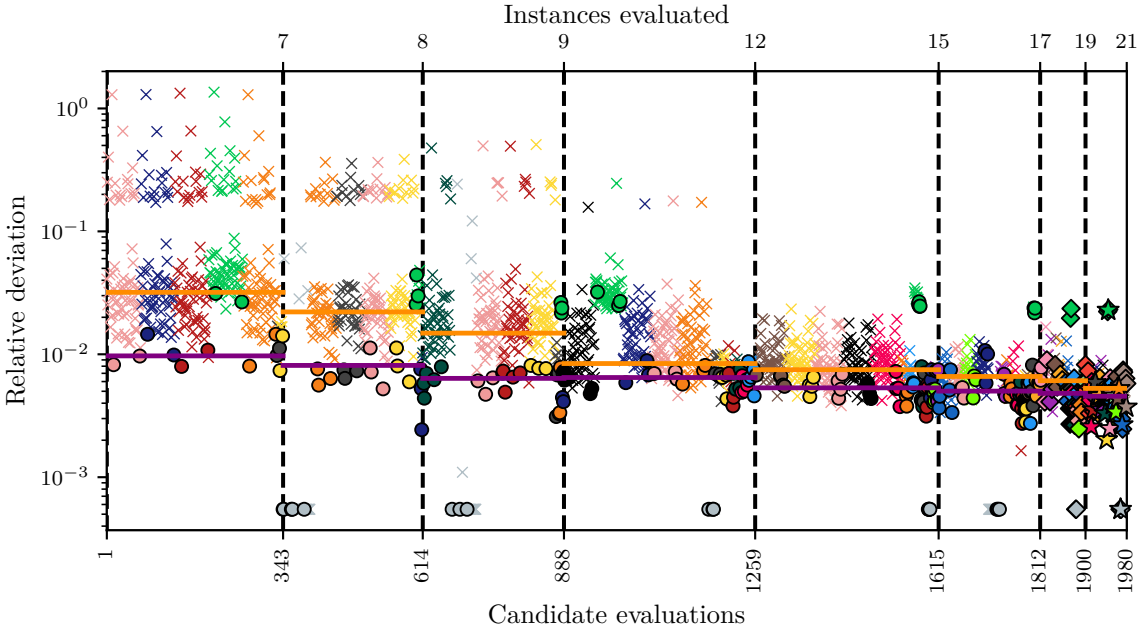
6.2 Analyzing the Configuration Process with acviz

In this section we present three exemplary case studies of configurations with flaws that can be easily identified when using acviz. Experiments were run on a GNU/Linux platform running on an 8-core AMD FX-8150 CPU 3.6 GHz and 32 GB memory. We used acviz 1.0, irace 3.1, ACOTSP 1.03, and SPEAR 1.2.1. The acviz program was written in Python 3 and requires R ($\geq$ 3.4) and the following libraries: numpy ($\geq$ 1.18), pandas ($\geq$ 1.0.3), matplotlib ($\geq$ 3.1), and rpy2 ($\geq$ 3.2).

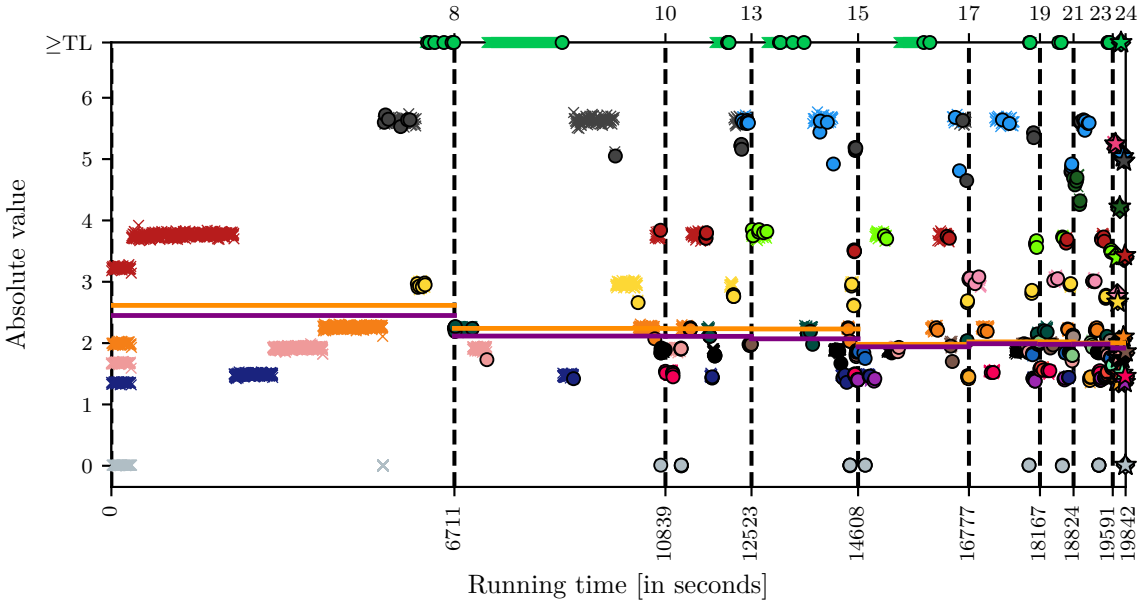
We use ACOTSP and SPEAR configuration scenarios (see Chapter 3 for details about them), which provide different characteristics and allow us testing all functions implemented in acviz. For ACOTSP, irace optimizes the cost of the best found solution after a running it for 20 seconds (instead of the 60 seconds defined in Chapter 3). On the other hand, irace minimizes SPEAR's solving time, where for evaluations in which the instance is not solved, a penalized performance value is returned, i.e. the PARX penalization approach (PÉREZ CÁCERES et al., 2017a). For the third case study, where we configure ACOTSP and evaluate the resulting configurations on a set of test instances with different structure, we additionally use ten TSP instances with uniformly random distance matrices, generated with *portmgen* from the 8th DIMACS Implementation Challenge (JOHNSON et al., 2001). Finally, we use the default settings of irace in all our experiments.

6.2.1 Case Study 1: Easy and Hard Instances

Here, we discuss two example scenarios and show how easy and hard instances can be identified. We configure ACOTSP with a budget of 2K evaluations, and SPEAR with a budget of 20K seconds. Figure 6.1 shows the visualizations produced by acviz. When configuring ACOTSP we observe the evolution of the configuration process, i.e. how the performance of the sampled configurations change over the iterations. At the beginning of the configuration, there is a subset of the configurations with bad performance on almost all evaluated instances (points in the upper part of the figure in the four first iterations). The number of such bad performers decreases over the iterations, while the estimated performance of elite and non-elite configurations (the median values given by the horizontal lines) becomes better. We can also see that the instance selection strategy implemented in irace iteratively



Scenario 1: ACOTSP with a budget of 2K evaluations.



Scenario 2: SPEAR with a budget of 20K seconds.

Figure 6.1 – Configuring ACOTSP and SPEAR with instances of different hardness.

increases the number of instances on which the configurations are evaluated.

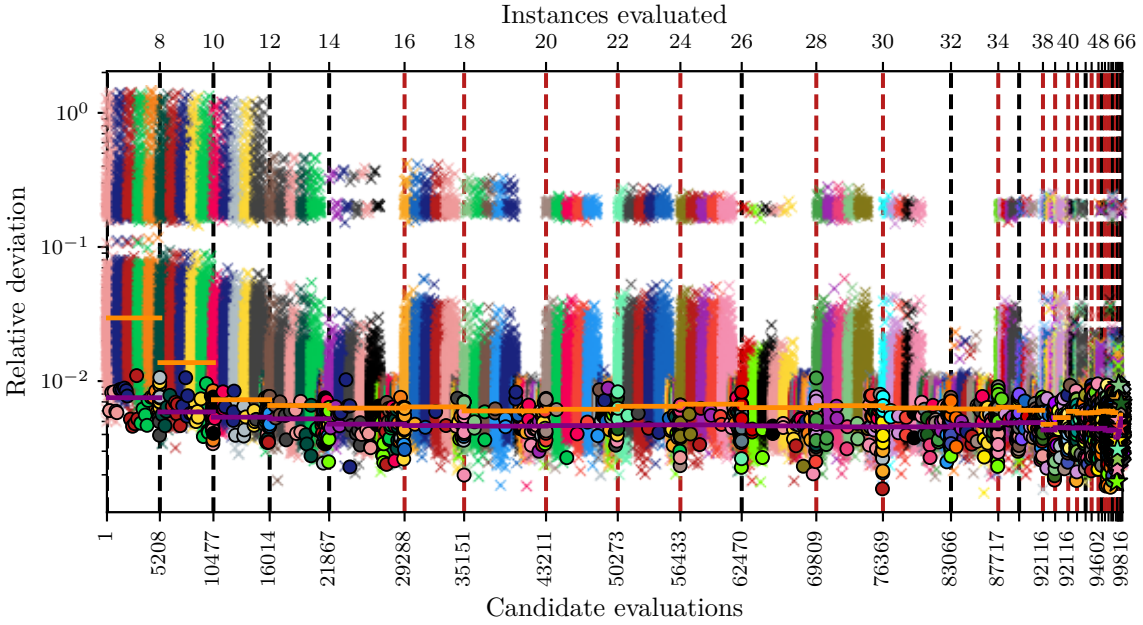
In the configuration of SPEAR, the performance of the configurations also improves over the iterations. Besides that, we see that different configurations have a similar performance on each instance. Note that the evaluations of different

configurations on a particular instance, represented by clusters of points of the same color, present a small variation of the running time. Nevertheless, we observe that elite configurations perform better than others, since they are often among the best in each cluster.

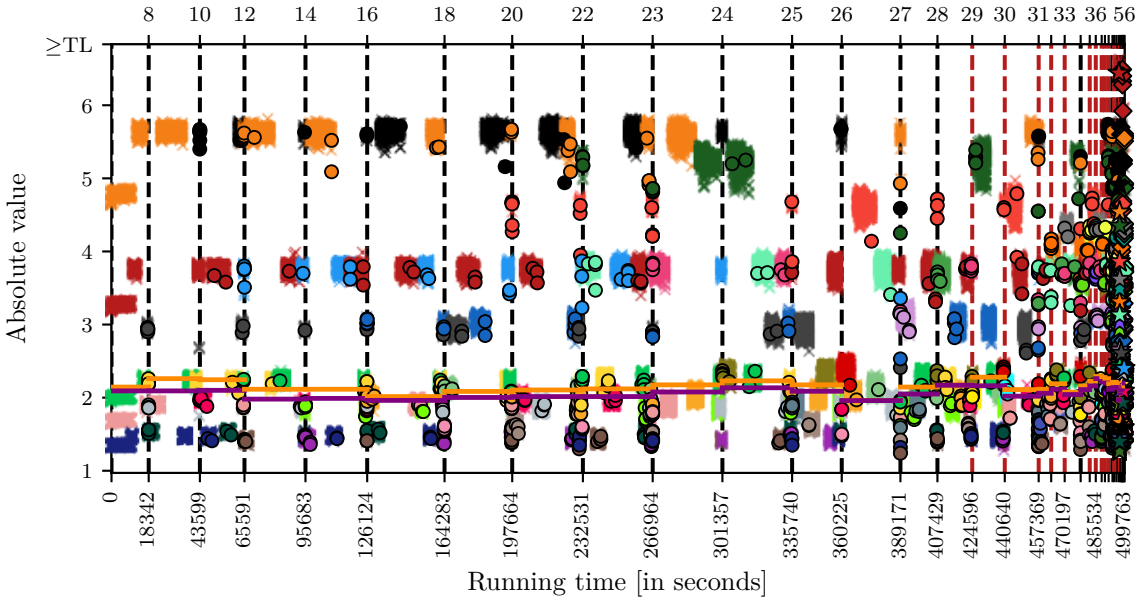
The visualizations shown in Figure 6.1 also provide some information about the configuration scenarios. We can see that both scenarios are quite homogeneous, i.e. a configuration with good performance on one instance often presents good performance on the others. For example, if we look at the elite configurations (● markers) of each iteration, we see that they present the best results for almost all instances. This contributes to irace easily identifying the best configurations in the racing phase, and consequently, using less evaluations than the budget available for the iterations. The saved budget is then used to perform more iterations than the five initially scheduled, as observed in the plot. Those additional iterations are increasingly shorter because they are consuming the remaining budget and fewer new configurations are sampled.

We included in both scenarios two additional instances: one that is easy to solve, shown in gray, and another that is hard to solve, shown in green. Figure 6.1 shows an interesting behavior of the configurations on those instances. In ACOTSP, we can see that almost all configurations perform very well on the easy instance. Besides that, there is no variation of different configurations on this instance. Therefore, the evaluations on this instance do not help to determine the quality of different configurations and decide which one is better. In the case of the hard instance, we observe that it helps to differentiate the quality of the configurations in the first iterations. However, as for the easy instance, from the fifth iteration on, it stops being useful for the configuration process.

In the configuration of SPEAR it is even more evident that the easy and hard instances do not contribute to the configuration process. We observe that all configurations immediately solve the easy instance, while no configuration solves the hard instance. In this case, we could exclude the easy instance from the configuration scenario, since it does not help to evaluate the configurations. We could also exclude the hard instance, or increase the time limit, trying to find configurations that can solve it. The acviz tool helps to identify such cases by visualizing and comparing how the configurations perform on the training instances and which ones are actually contributing to the configuration process.



Scenario 1: ACOTSP with a budget of 100K evaluations.



Scenario 2: SPEAR with a budget of 500K seconds.

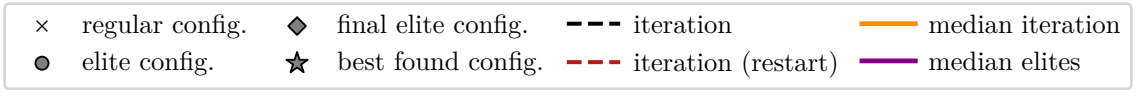


Figure 6.2 – Configuring ACOTSP and SPEAR with large budgets.

6.2.2 Case Study 2: Unnecessarily Large Budget

Choosing an adequate configuration budget can be difficult. A small budget may not be sufficient to find good configurations. A common practice is to use the

highest possible budget, according to time constraints and the available computational resources. However, even after running irace, it may not be clear if the chosen budget was appropriate. In this second experiment, we configure both ACOTSP and SPEAR with very large budgets of 100K evaluations and 500K seconds, respectively. Figure 6.2 shows the resulting visualizations. Since the budget is larger, irace samples more configurations and uses more instances to evaluate them.

In ACOTSP, the observed behavior is similar to the first case study. We can see a fast evolution of the configuration process in the first iterations, producing configurations with better performance compared to those obtained in the first case study. From the fifth iteration on, after approximately 20K evaluations, the quality of the sampled configurations stagnates. Note that the estimated performance of both elite and non-elite configurations (orange and purple horizontal lines) does not improve from that point until the end of the configuration process. We can also see that irace performs a soft restart (red dashed line) in almost all subsequent iterations, which indicates that the sampling models converged. The same behavior is observed in the configuration of SPEAR, where soft restarts are present after about 400K seconds.

In both scenarios, if we needed to repeat the process, we could decrease the budget to about 20K $\sim$ 30K evaluations (ACOTSP) or 300K $\sim$ 400K seconds (SPEAR), for example. Alternatively, if we have the time for a large budget, we could tell irace to sample more configurations at each iteration to increase diversification (parameter `nbConfigurations`). For heterogeneous scenarios, the additional budget could be better spent in increasing the number of instances evaluated before the first and between each elimination test (parameters `firstTest` and `eachTest` of irace, respectively).

6.2.3 Case Study 3: Unrepresentative Instances

A common mistake when configuring algorithms is to choose training instances that are not representative of the instances to be used in production. Suppose, for example, we use an algorithm that has been configured on a certain class of instances Π. Now, we want to solve additional instances of class Π'. In order to get the best performance, we may want to configure the algorithm again to reflect the new instance distribution. If the goal is to obtain a configuration that performs

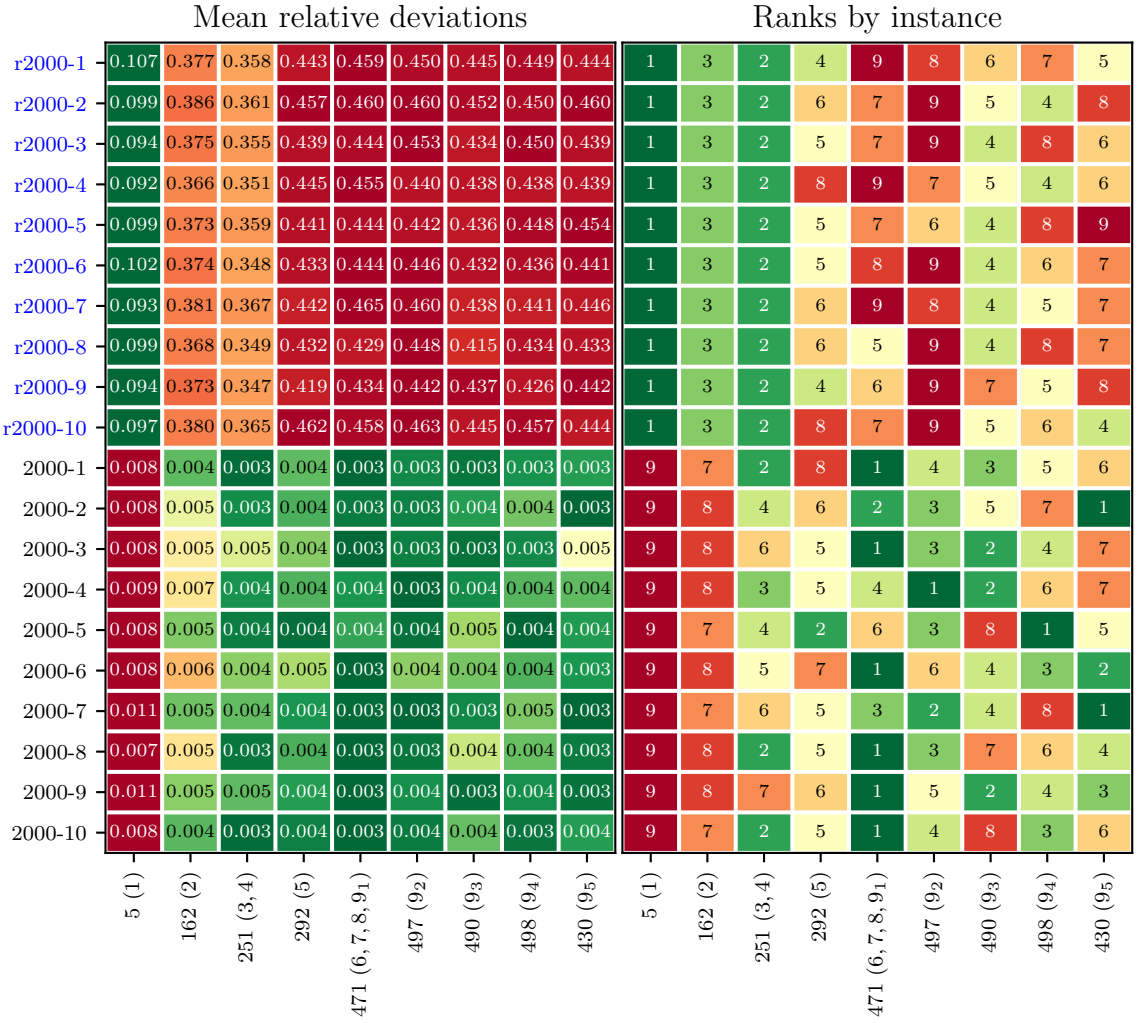


Figure 6.3 – Test results after configuring ACOTSP with unrepresentative training instances. Instances with random distances (starting with 'r' and in blue) were used only for test, while Euclidean instances (in black) were used for both training and test.

well for both instance classes, then starting the configuration process from the current configuration (tuned for Π) and training only on instances of Π' would be a methodological mistake. Note that the previous example is an extreme case of an unrepresentative training set. Nevertheless, a similar situation is obtained when the training set is composed of different instance classes and one or more classes are strongly underrepresented in the set.

In this experiment, we reproduce the above situation using ACOTSP to analyze how the configuration process behaves. In a first step, we select ten TSP instances with random distances and tune ACOTSP on them to obtain a set of initial configurations. Then, we select ten Euclidean TSP instances, and use them as training instances for an irace run with a budget of 3K evaluations. We provide the configurations from the first step as initial configurations. We use the testing options

of irace to evaluate the resulting configurations on all Euclidean instances (used as training set) and all random distance instances (not used as training set, thus we call it test set). Random distance and Euclidean instances define structurally different TSP instances and thus, ACOTSP configurations that exhibit high performance in one instance class are not expected to maintain such high performance in the other class. The testing results are shown in Figure 6.3. Since we evaluate the resulting configurations on both training and test instances, we have useful information about how they perform on both instance sets. The mean deviations give an overview of the results, allowing us to observe the evolution in the quality of configurations found during the configuration process. We can also observe how those configurations compare with each other by analyzing the obtained ranks.

When the training instances are not representative, the found configurations may specialize on the known training instances and present poor performance on unseen test instances. Such an overtuning can be observed in Figure 6.3. As the configuration progresses, the performance of the configurations is becoming better on the training instances. On the other hand, the performance on the test instances degrades over the iterations. Since we initialized irace with configurations known to perform well on the test instances, the best configuration in the first iteration still performs well on the test set, but the performance quickly degrades on subsequent iterations. To solve this problem, we need to include some random distance instances in the training set, and make sure that the relative frequency of each type of instance seen during training matches their relative frequency in the test set, or the frequency expected in unseen instances.

6.3 Discussion

As we show in the case studies, the visualizations provided by acviz contribute to improve the usability of irace, since they facilitate the understanding of the configuration process, allow users to analyze irace runs, and help evaluating the configuration scenario (e.g. the quality of the training instances, or the adequability of the configuration budget). Such understanding is important to avoid mistakes when defining the configuration scenarios, to get the best performance from the automatic configuration process and, consequently, to achieve better results in terms of the quality of final configurations.

The acviz program is available online in Souza et al. (2020a)[1]. The irace log files used in the experiments and other material necessary for reproducing the experimental results are available as supplementary material for this chapter (SOUZA et al., 2020b)[2]. More information about the artifacts produced from this research are given in Appendix A.

[1]The acviz program is available at https://github.com/souzamarcelo/acviz.

[2]The supplementary material for this chapter is available at https://doi.org/10.5281/zenodo.4028904.

Part III

Applications

7 AUTOBQP: A COMPONENT-WISE SOLVER TO BINARY OPTIMIZATION

Civilization advances by extending the number of important operations which we can perform without thinking of them.

— Alfred North Whitehead

In this chapter, we use automatic algorithm design methods as the core piece of AutoBQP, a component-wise heuristic solver for binary optimization problems. Given a problem description with the implementation of the objective function and a set of training instances, this solver uses automatic algorithm design to produce an algorithm with optimized performance on such instances. Specifically, we follow a bottom-up design approach by defining a flexible algorithm framework of heuristic components that can be selected and combined. Then we apply the irace configurator to explore the corresponding design space of components and parameter values, searching for the best algorithms for different problem domains. The heuristic components were extracted from state-of-the-art algorithms for the unconstrained binary quadratic programming (UBQP), including constructive heuristics, neighborhoods for local searches, perturbation strategies for iterated local searches, and solution recombination strategies. We focus on heuristic components for the UBQP since many problems can be reduced to it. Kochenberger et al. (2004, 2014) discuss general reduction techniques to binary quadratic optimization and present around thirty problems that can be reduced to UBQP. The resulting design space is expressed by a context-free grammar, whose design choices and parameter values are represented by categorical and numerical parameters, allowing us to use irace to automate the design process. The flexibility of this approach allows to produce different combinations of heuristic components, leading to hybrid and potentially high-performing algorithms, whose can be applied to a wide range of binary problems via UBQP formulations.

This chapter presents the structure of the proposed AutoBQP solver and discuss its inner techniques. Section 7.1 reviews the related literature, focusing on previous applications of automatic algorithm design for optimization problems, as well as discussing existing heuristic solvers for binary problems. Section 7.2 gives an overview of our base problem, the unconstrained binary quadratic programming. Section 7.3 presents our design space of components and its grammar-based representation,

Table 7.1 – Summary of grammar-based bottom-up approaches for automatic algorithm design. The proposed approach is presented in the last line.

Approach	Framework	Problems
Mascia et al. (2013)	Iterated greedy	Permutation flowshop
Mascia et al. (2014b)	Iterated greedy	Bin packing; Permutation flowshop
Mascia et al. (2014a)	ParadisEO (hybrid)	Permutation flowshop
Marmion et al. (2013)	ParadisEO (hybrid)	Permutation flowshop
López-Ibáñez, Marmion and Stützle (2017)	ParadisEO (hybrid)	Permutation flowshop; Traveling salesperson; UBQP
Brum and Ritt (2018a, 2018b)	FSSolver (hybrid)	Permutation flowshop
Pagnozzi and Stützle (2019, 2021)	EMILI (hybrid)	Permutation flowshop
Alfaro-Fernández et al. (2020)	EMILI (hybrid)	Hybrid flowshop
AutoBQP	UBQP heuristics	Binary problems

discussing each heuristic component in detail. Section 7.4 details our automatic design methodology, discussing the algorithm framework and the use of irace to explore its design space. Finally, Section 7.5 concludes the chapter and presents a general discussion of the contributions of this research.

7.1 Related Work

The bottom-up approach for automatic algorithm design we follow in this chapter has four main ingredients: (i) a flexible framework of algorithm components; (ii) a grammar describing the corresponding design space; (iii) a parametric representation of this grammar; and (iv) an algorithm configurator, in this case irace, to explore the design space and produce algorithms automatically. In Section 2.3.2 we reviewed the literature and presented different approaches that follow the same above ideas. Now, we discuss them in more detail, focusing on their algorithm frameworks and problem domains.

Table 7.1 summarizes the algorithm frameworks of each approach and the problem domains they were applied, including our approach (AutoBQP). Regarding the algorithm frameworks, Mascia et al. (2013, 2014b) define a simple design space of

iterated greedy components. Their approach was extended in Mascia et al. (2014a) by using ParadisEO (CAHON; MELAB; TALBI, 2004; DRÉO et al., 2021), an algorithm framework that contains several heuristic components and provides a much larger design space. The same framework was used in Marmion et al. (2013) and López-Ibáñez, Marmion and Stützle (2017), allowing them to produce hybrid algorithms by combining components from different heuristic approaches. Brum and Ritt (2018a, 2018b) extracted heuristic components from state-of-the-art algorithms for the permutation flowshop problem and build their own framework. The same was done in Pagnozzi and Stützle (2019, 2021) with the EMILI framework, which is also used in Alfaro-Fernández et al. (2020). We follow the same idea and build our framework with heuristic components extracted from the literature of UBQP. Regarding the problem domains, we observe that all previous works shown in Table 7.1 focus on solving permutation and hybrid flowshop problems. Mascia et al. (2014b) extend the application of these techniques to the bin packing problem, while López-Ibáñez, Marmion and Stützle (2017) additionally consider the traveling salesperson and UBQP problems. Besides solving the UBQP, López-Ibáñez, Marmion and Stützle (2017) present only a simple experimental analysis, do not compare to results from the literature, and do not consider any additional problem formulated as binary optimization. In summary, all these works are limited to one or just few problems, while AutoBQP is designed for a general class of binary problems, which gives it a broader applicability.

Other approaches in the literature aim at solving general classes of problems, e.g. binary problems. This is the case of LocalSolver[1] (BENOIST et al., 2011), a commercial heuristic solver for different problems, and BinarySS (GORTAZAR et al., 2010), a heuristic solver based on scatter search for binary problems. In addition, the algorithms for UBQP (e.g. Palubeckis (2006), Glover, Lü and Hao (2010) and Wang et al. (2012)) can also be applied to solve general binary problems. However, all these approaches implement fixed heuristic strategies, i.e. they are not based on automatic design techniques. In consequence, their adaptation to new problem domains other than those they were designed for is limited. In contrast, AutoBQP combines the automatic design techniques with a framework of components extracted from the literature of UBQP, allowing us to produce specialized heuristic algorithms for a general class of binary problems that can be reduced to UBQP.

[1]For more information about LocalSolver, see https://www.localsolver.com.

7.2 Unconstrained Binary Quadratic Programming

Given a symmetric matrix of coefficients $Q = (q_{ij}) \in \mathbb{R}^{n \times n}$, UBQP asks to

$$\begin{aligned} \textbf{maximize} \quad & x^t Q x \\ \textbf{subject to} \quad & x \in \{0,1\}^n. \end{aligned}$$

This model comprises a wide range of applications in combinatorial optimization, including problems on graphs (KOCHENBERGER et al., 2013; WANG; XU, 2013; KOCHENBERGER et al., 2015), boolean satisfiability (HANSEN; JAUMARD, 1990; KOCHENBERGER et al., 2005), set partitioning (LEWIS; KOCHENBERGER; ALIDAEE, 2008) and machine scheduling (ALIDAEE; KOCHENBERGER; AHMADIAN, 1994). Given a UBQP formulation, these problems can be solved by methods initially designed for UBQP. A concrete example is the decision version of the maximum clique (MC) problem, which can be reduced to binary quadratic optimization (PARDALOS; XUE, 1994; BOMZE et al., 1999). Given an undirected graph $G = (V, E)$ and a value k, is there a clique (i.e. a complete subgraph) of size k or more? With binary variables $x_v \in \{0,1\}$, $v \in V$, and weights $q_{uv} = 1$ for $\{u, v\} \in E$, and $q_{uv} = -\binom{n}{2}$ otherwise, we have a "yes"-instance of MC iff UBQP has a solution of value $\binom{k}{2}$ or more. This reduction can be done in polynomial time and since MC is NP-hard, UBQP is NP-hard too. UBQP remains hard if there is a unique solution (PARDALOS; JHA, 1992). As a consequence, current exact methods for UBQP can only solve small instances (for more details about the exact methods see Kochenberger et al. (2014)). For this reason, most research focuses on heuristic methods to solve medium and large instances in the range of 2500 to about 15000 variables.

Palubeckis (2006) proposes an iterated tabu search (ITS). Given a random initial solution, it iteratively applies a tabu search as an improvement step, followed by a perturbation step to escape from local optima. The author uses a constant tabu tenure, and the least-loss perturbation procedure. Least-loss perturbation ranks variables according to the loss when flipping their value. Then, it randomly flips variables from the b variables of least loss. The size of the perturbation is selected uniformly at random from the interval $[d_1, n/d_2]$, where n is the size of the instance, and d_1 and d_2 are input parameters. Glover, Lü and Hao (2010) propose an iterated

tabu search algorithm using an elite set as a diversification mechanism (D^2TS, which stands for *diversification-driven tabu search*). The elite set stores the best solutions found so far, which are used as initial solutions for a perturbation and a search step. They compute the tabu tenure according to $n/t_d + c \in [0, t_c]$, where c and t_c are input parameters. Their diversity-based perturbation method scores variables based on their flip frequency and the values they present in the elite solutions, and uses a parameter β as a weight factor for the frequency contribution. Then it selects variables to be randomly assigned to 0 or 1, with probability of selection proportional to the variable's score. Parameter λ defines the importance of the score in this step. The perturbation size is given by n/g, where g is a parameter.

Wang et al. (2012) propose a repeated elite recombination algorithm based on path relinking. The algorithm uses an elite set to store the best found solutions. Initially, it generates random solutions for the elite set and applies a tabu search procedure to improve them. Iteratively, each pair of solutions from the elite set is recombined by path relinking. The resulting solution is improved by a tabu search and replaces the worst solution of the elite set, if its quality is better. The path relinking step applies a local search to the first solution (also called starting solution), allowing only modifications that get it closer to the second solution. In other words, the modification of a variable i is only allowed if the new value is equal to the value of variable i in the second solution. The authors propose two path relinking strategies. The first one (PR1) uses a best improvement strategy for moving to the next neighbor, selecting the best neighbor (i.e. the solution with highest quality) when no improvement is possible. The second one (PR2) always selects a random neighbor. Moreover, PR1 and PR2 require that path relinking returns a solution with minimum and maximum distances from the endpoints as $d_{min} = \gamma \times H$ and $d_{max} = H - d_{min}$, where H is the Hamming distance between both solutions. If no such solution is found, the result is the starting solution.

7.3 The Design Space of Components

We extracted the heuristic components from the literature of UBQP to build our algorithm framework, specifically the state-of-the-art algorithms of Palubeckis (2006), Glover, Lü and Hao (2010) and Wang et al. (2012). We also include some additional heuristic components frequently found in literature, e.g. different local

```
1   ⟨start⟩          →  ⟨search⟩ | ⟨construction⟩ | ⟨recombination⟩
2   ⟨search⟩         →  ls (⟨improvement⟩) | nmls (⟨improvement⟩) | ⟨ts⟩
3                       | ils (⟨search⟩, ⟨pert⟩) | ilse (⟨search⟩, ⟨pert⟩)
4   ⟨improvement⟩    →  fi | fi-rr | bi | si | si-partial | si-partial-rr
5   ⟨ts⟩             →  sts | rts
6   ⟨pert⟩           →  random | least-loss (⟨step⟩) | diversity (⟨step⟩)
7   ⟨step⟩           →  uniform | gaussian | exponential | gammam
8   ⟨construction⟩   →  gra (⟨constructor⟩) | grasp (⟨constructor⟩, ⟨search⟩)
9   ⟨constructor⟩    →  zero | half
10  ⟨recombination⟩  →  rer (⟨improvement⟩, ⟨search⟩)
```

Figure 7.1 – The reduced grammar expressing the design space of components. We omit the input parameters of each component to highlight the algorithm components and the combination rules. The complete list of input parameters, their types, descriptions and ranges of values are presented later in Table 7.2.

search strategies and the well known GRASP metaheuristic (FEO; RESENDE, 1995). This section describes the design space of components and their input parameters.

Figure 7.1 shows the reduced grammar in Backus-Naur form, representing the design space we propose to solve problems based on UBQP. The presented grammar is reduced since we omit the input parameters of the algorithm components, in order to focus on its components and their possible combinations. Such input parameters will be detailed later in Table 7.2. The grammar consists of a set of rules, through which the heuristic algorithms can be instantiated. Each rule describes a decision, i.e. a component to be chosen. The start symbol of the grammar is the non-terminal ⟨start⟩ in line 1. Starting from ⟨start⟩, three main heuristic strategies can be chosen: heuristics based on local search (⟨search⟩), on solution construction (⟨construction⟩), or on recombination of candidates from a population of solutions (⟨recombination⟩). In the following we describe all three strategies.

7.3.1 Search Methods

The search-based heuristics (line 2) start with an initial solution and modify it, in order to explore the search space. The solutions obtained by all possible modifications form a solution neighborhood. Then, search methods apply some

strategy to select the next solution from the neighborhood (line 3). Given that in the UBQP the neighborhood of a solution is the set of solutions with one modified variable (a flip in a position of the vector x), we access the neighbors in the order of the variables. The first improvement (fi) strategy selects the first neighbor that improves the solution. We can apply a round-robin strategy (fi-rr), starting the exploration from the position where the previous one has finished. The best improvement (bi) strategy selects a neighbor that improves the solution most. The some improvement strategy (si) selects a random improving neighbor. A variant, called some improvement with partial exploration (si-partial) considers only the first $f\%$ of variables in the exploration for some improvement. If no improving neighbor is found, the rest of variables is explored. Alternatively, we can use a round robin strategy to start the exploration from the position where the previous one has finished (si-partial-rr).

The simplest search-based method is the local search (ls), which iteratively applies an improving modification until no better neighbor is found. In order to avoid local optimum, a common strategy is to select a random neighbor with probability p, and an improving neighbor with probability $1 - p$. This method is called non-monotone local search (nmls). Tabu search ($\langle$ts$\rangle$) was first proposed by Glover (1989) and consists of a local search that keeps a list of prohibited solutions (tabu list), in order to avoid the search coming back to previous visited solutions in a short-term period. When a solution is selected, the modified variable is stored in the tabu list for some number of iterations (tabu tenure). During this period, this variable cannot be changed. There are different strategies to define the tabu tenure, like using a constant, or a value according to the instance size. The simple tabu search (sts) always apply a best improvement strategy to select a neighbor. A randomized tabu search (rts) applies a random move with a probability p. Commonly used stopping criteria for tabu search are a maximum number of iterations or a maximum number of iterations in stagnation. For example, Palubeckis (2006) proposes the maximum number of iterations to be a factor multiplied by the instance size. This factor is set to 15000 if the instance has more than 5000 variables, 12000 if it has between 3000 and 5000 variables, and 10000 for instances up to 3000 variables.

Iterated local search (ils) was proposed by Lourenço, Martin and Stützle (2003) and iteratively applies a local search, followed by a perturbation step ($\langle$pert$\rangle$). The iterated local search can also be combined with an elitist strategy (ilse), following

the ideas of Glover, Lü and Hao (2010). An elite set stores the best solutions found so far, which are used as initial solutions for the perturbation and search steps. The ⟨pert⟩ rule defines the available perturbation methods. The first one sets all variables of the current solution to values chosen uniformly at random (random). The second strategy implements the least-loss approach of Palubeckis (2006), and the third one implements the diversity approach of Glover, Lü and Hao (2010). The grammar has also several strategies to define the size of the perturbation (⟨step⟩), i.e. the number of variables that will be flipped in least-loss or diversity perturbation methods. Given the instance size n, the gammam strategy implements the approach of Glover, Lü and Hao (2010). The uniform strategy implements the approach of Palubeckis (2006), which consists in selecting a size value according to an uniform distribution in the interval $[d_1, n/d_2]$. The gaussian and exponential strategies follow the same idea, but apply a Gaussian and exponential distributions, respectively.

Our grammar allows the combination of the iterated local search with any search and perturbation procedures. Therefore, we can instantiate state-of-the-art algorithms such as the one proposed by Palubeckis (2006), which consists of an iterated local search that applies a tabu search and the least-loss perturbation procedure with the uniform strategy to compute the perturbation size. We can also instantiate the diversification method of Glover, Lü and Hao (2010), selecting the iterated local search combined with the elite set, a tabu search and a diversity-based perturbation procedure with the gammam strategy to compute the perturbation size.

7.3.2 Construction Methods

The constructive methods are based on the heuristics of Merz and Freisleben (2002). They start with an empty solution and iteratively set values to its variables. The variable is chosen according to an α-greedy algorithm, which randomly selects one of the $\alpha\%$ best variables. The greedy randomized adaptive algorithm (gra) repeatedly constructs m solutions and picks the best one. Another approach, proposed by Feo and Resende (1995), is the greedy randomized adaptive search procedure (grasp), which applies a search procedure whenever a solution is constructed. A first strategy for the construction process for UBQP starts with all variables equal to zero, and then sets some of them to one (zero). An alternative strategy starts all variables at 0.5, and then sets the values to zero or one (half).

7.3.3 Recombination Methods

The recombination-based method implements the idea of evolving a population of solutions. The rer component implements the repeated elite recombination method of Wang et al. (2012). Any improvement strategy (⟨improvement⟩) can be selected for the path relinking step. If using best improvement and no improving neighbor is found, the exploration selects the best neighbor, according to Wang et al. (2012). When using the other strategies, if no improving neighbor is found, the exploration selects a random neighbor. Besides that, any search method can be selected for the search step (⟨search⟩).

7.3.4 Input Parameters

Most of the components presented above have input parameters whose values must be selected to ensure high performance. For example, the constructive approaches require the definition of the number of repetitions. We omit these parameters in Figure 7.1, but a complete derivation of the grammar not only gives an algorithm, but also its parameter values. Table 7.2 shows all parameters, their type, the related component and possible values. Categorical parameters (like the different strategies to compute the tabu tenure) have a set of limited candidate values. For numerical parameters, we define the correspondent interval of possible values. Parameters t, s, and i choose the strategies for tabu tenure, maximum iterations in stagnation and maximum iterations, respectively. Strategies t_1 to t_4 are presented in the table, and strategy t_5 sets the tabu tenure as the average degree of variables in the instance being solved, i.e. the average number of non-zero coefficients associated with each variable. Strategy i_2 implements Palubeckis' rule to determine the maximum number of iterations, while i_3 and s_3 allow iterations and stagnation ∞. The rest of parameters were explained above, when discussing the heuristic components.

7.4 Automatic Design Methodology

The proposed algorithm framework allows a flexible combination of its components into hybrid metaheuristics, many of which probably have never been explored

Table 7.2 – Input parameters of the algorithm components (n is the instance size). We show all parameters, the corresponding types and components, a brief description and the set or range of possible values.

Par.	Type	Component	Description	Values
t	cat	⟨ts⟩	Strategy for tabu tenure	$\{t_1, \ldots, t_5\}$
t_v	int	⟨ts⟩ (t_1)	Constant for tabu tenure	$[1, 50]$
t_p	int	⟨ts⟩ (t_2)	Tabu tenure is $(t_p \times n)/100$	$[10, 80]$
t_d	int	⟨ts⟩ (t_3, t_4)	Tabu tenure is n/t_d	$[1, 500]$
t_c	int	⟨ts⟩ (t_4)	Tabu tenure is $n/t_d + c \in [0, t_c]$	$[1, 100]$
s	cat	⟨ts⟩	Strategy for max. stagnation	$\{s_1, s_2, s_3\}$
s_v	int	⟨ts⟩ (s_1)	Constant for max. stagnation	$[500, 100\,000]$
s_m	int	⟨ts⟩ (s_2)	Maximum stagnation is $s_m \times n$	$[1, 100]$
i	cat	⟨ts⟩	Strategy for maximum iterations	$\{i_1, i_2, i_3\}$
i_v	int	⟨ts⟩ (i_1)	Constant for maximum iterations	$[1\,000, 50\,000]$
p	real	nmls; rts	Probability of a random move	$[0.0, 1.0]$
f	int	si-partial[-rr]	Size of the partial exploration	$[5, 50]$
d_1	int	⟨pert⟩	Minimum perturbation size	$[1, 100]$
d_2	int	⟨pert⟩	Maximum perturbation size is n/d_2	$[1, 100]$
g	int	gammam	Perturbation size is n/g	$[2, 100]$
b	int	least-loss	Candidate variables for perturbation	$[1, 20]$
β	real	diversity	Frequency contribution	$[0.1, 0.9]$
λ	real	diversity	Selection importance factor	$[1.0, 3.0]$
r	int	ilse	Elite set size for ilse	$[1, 30]$
e	int	rer	Elite set size for rer	$[1, 20]$
γ	real	rer	Distance scale	$[0.1, 0.5]$
α	real	⟨construction⟩	Greediness of the construction	$[0.0, 1.0]$
m	int	⟨construction⟩	Number of repetitions	$[10, 100]$

before. For example, we can derive a repeated elite recombination with an iterated local search, which can use an internal non-monotone search procedure. We can generate the algorithms found in the literature of UBQP, and also combine their components with other ones to improve performance. However, the manual exploration of such designs is impractical, given the large number of possible combinations and the time required to evaluate them. This section describes the automatic algorithm design methodology we use to explore this design space and produce hybrid and high-performing algorithms for different problem domains with almost no human intervention.

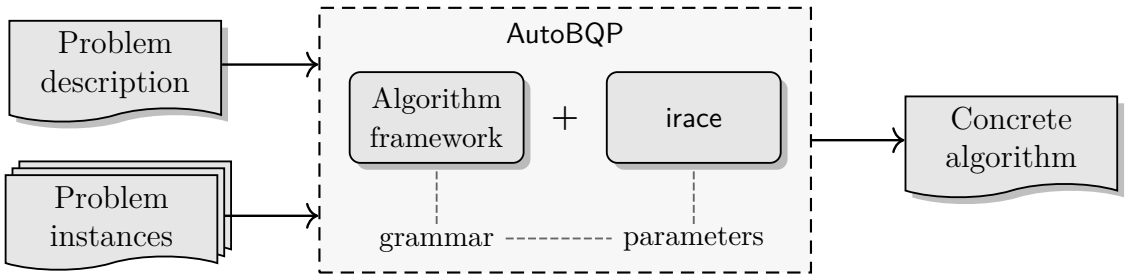


Figure 7.2 – General setup for the automatic algorithm design process. Given a problem description and a set of problem instances, the solver returns a concrete algorithm. Internally, the solver implements the algorithm framework of components and uses irace as automatic configurator. The design space is expressed by a context-free grammar, whose decisions are represented by parameters, allowing the use of irace to explore the design space and search for the best performing algorithms on the given instances.

The general idea behind AutoBQP is shown in Figure 7.2. We provide a problem description containing the objective function and a set of problem instances. AutoBQP automatically searches for high-performing algorithms and returns the best one. To do this, we combine the algorithm framework with irace to explore the design space by selecting and combining algorithm components and setting values for their input parameters. The resulting algorithm is a complete and configured heuristic method. For communicating the algorithm framework to irace, we follow the fully parametric representation of Mascia et al. (2014b), in which the decisions made in the grammar are defined by parameters to be tuned by the configurator. For example, the ⟨start⟩ non-terminal in line 1 of the grammar (Fig. 7.1) has three options: ⟨search⟩, ⟨construction⟩, and ⟨recombination⟩. Therefore, we define a corresponding categorical parameter with those three options. By defining a parameter for each non-terminal, we avoid problems of low locality, i.e. when a small change in the representation leads to big changes in the produced algorithm, and redundancy, i.e. when different representations lead to the same algorithm. These problems are common in token-based approaches (MASCIA et al., 2014b; ROTHLAUF; OETZEL, 2006; LOURENÇO; PEREIRA; COSTA, 2016).

We can see that our grammar is recursive in the rule of line 2 (⟨search⟩). However, unlimited depth is unhelpful and we limit the recursiveness by not allowing an iterated local search being the internal search procedure of another iterated local search. This is a characteristic of the fully parametric representation, because we need a finite number of parameters, and helps to reduce the size of the design space. In this case, we need two parameters for the ⟨search⟩ rule, because when selecting

one of the iterated local searches, we need to define the internal search. However, the second parameter of this rule does not have the option to select an iterated local search. Nevertheless, the proposed grammar is quite flexible and can generate a wide range of hybrid metaheuristics by combining its components. In fact, the grammar can generate 2396 different algorithms. This number increases substantially if we consider the possible values of the 3 categorical and 15 integer parameters (see Table 7.2), and is infinite considering the 5 real parameters.

Finally, some choices are conditional. The choice for rule ⟨start⟩ has to always have a value, because this decision is taken in all possible algorithms that the grammar can generate. This is not the case for the rest of rules. For example, rule ⟨ts⟩ is applied only if a tabu search is selected in rule ⟨search⟩. The same idea is applied to the input parameters, e.g., a value to parameter p is only set if a randomized heuristic was selected (nmls or rts). In other words, the structure of design choices reflects a relationship between them, defining the conditions to the need of each specific choice. We implemented these conditions, such that the configuration task needs to set values only for required choices.

7.5 Discussion

The main contributions of the research presented in this chapter are: (i) a flexible algorithm framework that implements several heuristic components, which are selected and combined to produce hybrid algorithms that handle a wide range of binary problems; (ii) the application of automatic algorithm design techniques to build a component-wise heuristic solver for binary problems; (iii) the proposed AutoBQP solver, which can be used without knowledge of the heuristic components, requiring only a problem description and a set of representative instances to produce an algorithm; (iv) we provide guidelines for researchers and practitioners on the automatic design of algorithms.

As stated before, Kochenberger et al. (2014) present several examples of problems modeled and solved via UBQP, and show that the methods based on UBQP are usually competitive in comparison to those specially designed for the original problems. In some cases, the solutions produced by the UBQP methods are better than those obtained by specialized methods, although the former do not exploit the original problem domain and structure. Said that, we expect AutoBQP

to exhibit competitive performance with state-of-the-art approaches for UBQP and problems reduced to it. We perform an experimental evaluation in Chapter 8. In addition, we made the AutoBQP solver available online in Souza and Ritt (2018a)[2]. Other artifacts produced from this research are given in Appendix A.

[2]The AutoBQP solver is available at https://github.com/souzamarcelo/autobqp.

8 AN EXPERIMENTAL EVALUATION OF AUTOBQP

> *It doesn't matter how beautiful your theory is, it doesn't matter how smart you are. If it doesn't agree with experiment, it's wrong.*
>
> — Richard Feynman

This chapter presents an experimental evaluation of AutoBQP, our component-wise heuristic solver described in Chapter 7. As previously discussed, a wide range of combinatorial optimization problems can be modeled in the form of an unconstrained binary quadratic program. Some of these problems occur in the form of a UBQP, while other must be reduced into it via one or more transformations, e.g. using penalization strategies to eliminate constraints. First, we evaluate AutoBQP on UBQP instances and on the maximum cut (MaxCut) problem, a direct UBQP application; then, we evaluate its performance on the linearly constrained test-assignment problem, a variant of graph coloring that can be reduced to UBQP via reformulation. We compare the performance of AutoBQP with state-of-the-art algorithms specialized in each problem.

Regarding the experimental setup, all algorithm components of AutoBQP have been implemented in C++ and were compiled using the GNU CCC compiler version 5.3.1 with maximum optimization. The experiments use irace version 2.4 and were conducted on a computer with an 8-core AMD FX-8150 processor running at 3.6 GHz and 32 GB main memory, under Ubuntu Linux, using only one core for each execution. In the rest of the chapter, we detail our computational experiments and results. Section 8.1 introduces the MaxCut problem and its reduction to UBQP. Section 8.2 presents the automatic design of heuristics for the UBQP and MaxCut problems. Section 8.3 introduces the test-assignment problem and its reduction to UBQP via reformulation, and discusses the solving methods from literature. Section 8.4 presents the automatic design of heuristics for the test-assignment problem. Finally, Section 8.5 gives some concluding remarks about the experimental evaluation of AutoBQP.

8.1 Maximum Cut on Graphs

The maximum cut on graphs (MaxCut) can be defined as follows. Given an undirected graph $G = (V, E)$, where $V = \{1, \ldots, n\}$ is the set of vertices, $E \subseteq [V]^2$ is the set of edges and each edge $\{i, j\} \in E$ has an associated weight w_{ij}, find a subset $V' \subseteq V$ such that the total weight of the edges $\{i, j\}, i \in V', j \notin V'$ (i.e. the weight of the cut $(V', \overline{V'})$) is maximum. Formally, the MaxCut problem asks to

$$\begin{aligned} \textbf{maximize} \quad & \sum_{i \in [n]} \sum_{j \in [n]} w_{ij} x_i (1 - x_j) \\ \textbf{subject to} \quad & x_i \in \{0, 1\}, \forall i \in [n]. \end{aligned}$$

An instance of the MaxCut problem can be naturally modeled as a unconstrained binary quadratic program (see Section 7.2) by setting

$$\begin{aligned} q_{ii} &= \sum_{j \in [n] \setminus \{i\}} -w_{ij}, && \forall i \in [n], \\ q_{ij} &= w_{ij}, && \forall i, j \in [n], i \neq j. \end{aligned}$$

Then, any algorithm for UBQP can be used to solve MaxCut instances without any reformulation.

The corresponding decision problem of cut on graphs (*given a graph G and an integer k, determine whether exists a cut of total weight at least k in G*) is NP-hard (KARP, 1972) and, since a solution can be verified in polynomial time, it belongs to NP and therefore is NP-complete too.

8.2 Automatic Design for UBQP and MaxCut

This section presents our experiments on automatically designing heuristic algorithms for both UBQP and MaxCut problems. We use two UBQP instance sets. The first is a set of 10 instances proposed by Beasley. They have 2500 variables and approximately 10% density, i.e. around 10% of the elements of matrix Q have non-zero values. The second is a set of 21 instances proposed by Palubeckis (PALUBECKIS, 2006). They have between 3000 and 7000 variables and densities from 50% to 100%. The coefficients of these instances are uniform random integers in the interval

Table 8.1 – Performance of the specialized algorithms on UBQP and MaxCut. We present the average absolute deviations obtained by the automatically produced algorithms and the state-of-the-art algorithms on each instance set over 20 replications.

Instances	ITS	D^{2}TS	PR1	PR2	HHPAL	HHMC
Beasley	2.0	**0.0**	1.34	5.84	**0.0**	34.0
Palubeckis	1109.6	2082.9	457.1	690.4	**186.8**	2424.3
MaxCut	-	-	5.6	4.7	25.0	**4.4**

$[-100, 100]$. For the MaxCut problem, we use the 54 instances of Helmberg and Rendl (2000), consisting in toroidal, planar and random graphs, which have between 800 and 3000 vertices, edge weights taking values from $\{-1, 1\}$, and densities less than 1%.

Our machine is approximately four times faster than the machine used by Palubeckis (2006), so we used 25% of the time limit he used. This results in a time limit of 150 s for the Beasley instances and 225 s, 450 s, 900 s, 1350 s, and 2250 s for the instances of Palubeckis with 3000, 4000, 5000, 6000, and 7000 variables, respectively. For the MaxCut instances, we used 66.6% (1200 s) of the time limit used by Wang et al. (2012), because our machine is no more than 33% faster than their machine.

8.2.1 Producing Specialized Algorithms for Each Problem

The density and coefficients of the Beasley and Palubeckis instances, are significantly different from those of the MaxCut instances. Therefore, we test in our first experiment an individual design process for each of these two instance groups. For each of the two experiments, we used samples of the instances for training with a budget of 2000 runs, and then validate the resulting algorithm using the complete set of instances. The first design process uses instances p3000-1, p4000-1, p5000-1, p6000-1, and p7000-1 from Palubeckis, and the second uses the MaxCut instances G1, G5, G10, G15, G20, G25, G30, G35, G40, G45, and G50. We call the algorithm produced with the Palubeckis training instances HHPAL (hybrid heuristic for the Palubeckis instances), and the one produced using the MaxCut training instances HHMC (hybrid heuristic for the MaxCut problem).

The results of running each algorithm on the test instances with 20 replications

can be seen in Table 8.1. We evaluate the automatically produced algorithms and the state-of-the-art algorithms ITS (PALUBECKIS, 2006), D^{2}TS (GLOVER; LÜ; HAO, 2010), PR1 and PR2 (WANG et al., 2012). We report the average absolute deviation obtained by each algorithm, i.e. the difference between the quality of the best solutions found and the quality of best known solutions. We can see that the automatic approach produces algorithms that outperform state-of-the-art algorithms. The algorithm produced using Palubeckis training instances, HHPAL, presents an average absolute deviation of 186.8 on the complete Palubeckis instance set, which is much better than the average of PR1. This algorithm also works well on Beasley's instances, since their structure is similar to Palubeckis' instances, but is worse that the state of the art on MaxCut instances, since they have a different structure. On the other hand, the algorithm produced using MaxCut training instances, HHMC, presents very good results for the MaxCut, improving slightly over PR2. However, the performance of HHMC is worse on the other instance sets, since it is specialized on MaxCut instances. Similar behavior can be observed for algorithms PR1 and PR2 of Wang et al. (2012). PR1 performs well on Beasley and Palubeckis instances, while PR2 performs well on MaxCut instances.

8.2.2 Producing a Single Algorithm for UBQP and MaxCut

In our second experiment, we aim at producing a single algorithm for solving both UBQP and MaxCut problems. We create an independent set of 20 training instances with a similar structure than those of the Beasley and Palubeckis instances. They have 2000 to 7000 variables and densities from 10% to 100%. The coefficients were selected uniformly at random from $[-100, 100]$. We analyze here the structure of the produced algorithm and its performance on all benchmark instances.

The design process using the random instance set produced a recombination-based algorithm, which internally applies an iterated tabu search. Based on the rules presented in the grammar of Figure 7.1, it consists of an algorithm based on the recombination strategy ($\langle$recombination$\rangle$), which runs the repeated elite recombination method (rer) with best improvement strategy (bi) for the path relinking and an iterated local search (ils). The latter applies a stochastic tabu search (sts) and the least loss perturbation method (least-loss plus gammam). We can see that the recombination strategy is used by Wang et al. (2012), the iterated tabu search is used

Algorithm 3: Hybrid heuristic for UBQP (HHBQP).

```
1  while stopping criterion not satisfied do
2      E ← create an elite set of size e
3      while E has any novel solution do
4          foreach (s, t) ∈ E × E | s ≠ t do
5              V ← variables v | s[v] ≠ t[v]
6              d_min ← γ × |V|
7              d_max ← |V| − d_min
8              d ← 0
9              s* ← s
10             while V ≠ ∅ do
11                 v ← select the best variable from V
12                 Flip s[v] and remove v from V
13                 d ← d + 1
14                 if s is better than s* then
15                     if d_min ≤ d ≤ d_max then
16                         s* ← s
17             s ← iteratedTabuSearch(s*)        ▷ According to Algorithm 4
18             if s is better than any solution of E then
19                 Replace the worst solution of E by s
20 return best solution of E
```

by Palubeckis (2006), and the strategies to define the tabu tenure, the termination criteria of the tabu search, and the perturbation size are used by Glover, Lü and Hao (2010). Therefore, our automatic approach combined components from all three algorithms from the literature. We call this algorithm HHBQP (hybrid heuristic for binary quadratic programming). This algorithm could also be classified as a steady-state memetic algorithm with a path-relinking-based recombination operator and no mutation. It maintains diversity in the pool of elite solutions by requiring the recombined solutions to have at least $\gamma|V|$ different variables.

The outline of HHBQP are shown in Algorithm 3. Repeatedly, the set of elite solutions is randomly created. Each pair of solutions is selected and then recombined using path relinking. The resulting solution is improved by an iterated tabu search, and then the elite set is updated. This process is repeated while better solutions are found and added to the elite set. The path relinking method operates on the pair of solutions s and t. It searches the space of variables that, if flipped, make s closer to t. Iteratively, this method selects the best variable and flips it in solution s. The new solution is accepted if it is better than the best solution found so far,

Algorithm 4: Iterated tabu search used in HHBQP.

```
1  while stopping criterion not satisfied do
2      p_size ← n/g
3      while p_size > 0 do
4          V ← create list with the b best variables to flip
5          v ← random variable from V
6          Flip s[v]
7          p_size ← p_size − 1
8      while maximum iterations and maximum stagnation not reached do
9          Select non-tabu variable v that leads to the best neighbor
10         Update tabu list
11         Flip s[v] and set v as tabu
12 return best solution found
```

and according to some minimum and maximum modification steps (d_{min} and d_{max}). The iterated tabu search procedure is given in Algorithm 4. It applies the least loss perturbation method, iteratively flipping a random variable from the b best variables to flip. This is repeated until p_{size} variables are flipped. After perturbing the solution, the perturbation's search step performs a tabu search. The best found solution is stored during the perturbation and search steps, and then returned at the final of the execution. The values for the input parameters of the above components, which are also set by the automatic design approach, are presented in Table 8.2.

The performance results of HHPAL and HHBQP on the instances of Palubeckis are presented in Table 8.3. It presents the best and average absolute deviations obtained by HHPAL, HHBQP and state-of-the-art approaches after 20 replications. Although the performance of HHBQP is slightly worse than the specialized HHPAL algorithm, we can see that HHBQP still improves the results in comparison to algorithm PR1 of Wang et al. (2012), presenting an average absolute deviation of 211.7. HHPAL and HHBQP also scale better and thus present more uniform results, with the average absolute deviation increasing less with the size of the instance.

Table 8.4 presents the performance results of HHMC and HHBQP on the

Table 8.2 – Values of the input parameters of HHBQP. These values were set by the automatic design approach.

Parameter	t	t_d	t_c	s	s_m	i	g	b	e	γ
Value	t_4	310	25	s_2	22	i_2	10	8	16	0.2405

Table 8.3 – Performance results on the instances of Palubeckis. We present the best and average absolute deviations obtained by the automatically produced algorithms and the state-of-the-art algorithms over 20 replications.

Instance	ITS		D^2TS		PR1		HHPAL		HHBQP	
	Best	Avg.	Best	Avg.	Best	Avg.	Best	Avg.	Best	Avg.
p3000-1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
p3000-2	0.0	97.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
p3000-3	0.0	344.0	0.0	0.0	0.0	36.0	0.0	138.6	0.0	107.5
p3000-4	0.0	154.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
p3000-5	0.0	501.0	0.0	0.0	0.0	90.0	0.0	129.6	0.0	49.1
p4000-1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
p4000-2	0.0	1285.0	0.0	0.0	0.0	71.0	0.0	249.8	0.0	323.1
p4000-3	0.0	471.0	0.0	0.0	0.0	0.0	0.0	21.6	0.0	2.7
p4000-4	0.0	438.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
p4000-5	0.0	572.0	0.0	0.0	0.0	491.0	0.0	12.0	0.0	0.0
p5000-1	700.0	971.0	325.0	656.0	0.0	612.0	325.0	465.8	0.0	386.6
p5000-2	0.0	1068.0	0.0	12533.0	0.0	620.0	0.0	313.9	0.0	338.7
p5000-3	0.0	1266.0	0.0	12876.0	0.0	995.0	0.0	56.7	0.0	76.5
p5000-4	934.0	1952.0	0.0	1962.0	0.0	1258.0	0.0	582.7	0.0	553.6
p5000-5	0.0	835.0	0.0	239.0	0.0	51.0	0.0	162.2	0.0	36.9
p6000-1	0.0	57.0	0.0	0.0	0.0	201.0	0.0	83.3	0.0	18.4
p6000-2	88.0	1709.0	0.0	1286.0	0.0	221.0	0.0	200.8	0.0	148.4
p6000-3	2729.0	3064.0	0.0	787.0	0.0	1744.0	0.0	366.6	0.0	671.9
p7000-1	340.0	1139.0	0.0	2138.0	0.0	935.0	0.0	514.7	0.0	903.4
p7000-2	1651.0	4301.0	104.0	8712.0	0.0	1942.0	0.0	589.9	8.0	828.0
p7000-3	0.0	3078.0	0.0	2551.0	0.0	332.0	0.0	34.6	0.0	0.0
Avg.	306.8	1109.6	20.4	2082.9	0.0	457.1	15.5	186.8	0.4	211.7

MaxCut instances, as well as of the state-of-the-art algorithms. Column "SS" shows the results for the scatter search proposed by Martí, Duarte and Laguna (2009), and column "CC" shows the results for the CirCut method proposed by Burer, Monteiro and Zhang (2002). We can see that HHMC improved the results of algorithm PR2 of Wang et al. (2012). Algorithm HHBQP has worse results for MaxCut instances compared to the other approaches. This is due to the fact that the random instances have the same structure of Palubeckis and Beasley instances, whose density and scale of coefficients are quite different from those of the MaxCut instances. Nevertheless, our results are comparable to the state-of-the-art approaches. Moreover, algorithms HHMC and HHBQP found new best known values for the MaxCut instances, which are presented in Table 8.5.

Table 8.4 – Performance results on the MaxCut instances. We present the average absolute deviations obtained by the automatically produced algorithms and the state-of-the-art algorithms over 20 replications.

Inst.	PR2	SS	CC	HHMC	HHBQP	Inst.	PR2	SS	CC	HHMC	HHBQP
G1	0.0	0.0	0.0	0.0	16.2	G28	7.1	11.0	36.0	8.3	10.8
G2	0.0	0.0	3.0	0.0	13.8	G29	13.1	16.0	29.0	14.8	21.6
G3	2.0	0.0	0.0	0.0	10.8	G30	7.2	9.0	27.0	6.5	14.2
G4	0.0	0.0	5.0	0.0	12.8	G31	6.5	18.0	21.0	5.3	12.2
G5	0.0	0.0	4.0	0.0	10.9	G32	5.4	12.0	20.0	8.7	34.6
G6	0.0	13.0	0.0	0.0	5.9	G33	5.9	20.0	22.0	8.7	29.8
G7	0.0	24.0	3.0	0.0	10.2	G34	5.8	20.0	16.0	8.0	31.7
G8	0.0	19.0	2.0	0.0	6.5	G35	13.2	16.0	14.0	15.2	41.5
G9	0.0	140.	6.0	0.0	9.6	G36	18.3	17.0	17.0	17.0	43.5
G10	0.2	7.0	6.0	0.0	10.2	G37	21.1	25.0	23.0	19.0	44.0
G11	0.0	2.0	4.0	0.0	7.6	G38	11.6	1.0	36.0	10.3	39.5
G12	0.0	4.0	4.0	0.0	10.0	G39	15.9	14.0	12.0	11.8	65.5
G13	0.0	4.0	8.0	0.0	14.0	G40	15.7	25.0	12.0	11.2	74.1
G14	1.4	4.0	6.0	1.4	8.1	G41	15.1	18.0	6.0	11.0	74.8
G15	0.7	1.0	1.0	0.1	14.2	G42	11.8	21.0	9.0	10.2	73.1
G16	0.6	7.0	7.0	0.1	13.6	G43	0.1	4.0	4.0	0.0	10.7
G17	0.6	4.0	10.0	0.3	12.7	G44	0.1	2.0	7.0	0.0	8.6
G18	0.0	4.0	14.0	0.0	17.7	G45	0.1	12.0	2.0	0.0	6.6
G19	0.0	3.0	18.0	0.0	19.1	G46	0.2	15.0	4.0	0.0	7.5
G20	0.0	0.0	0.0	0.0	25.9	G47	0.2	8.0	1.0	0.1	9.0
G21	0.0	1.0	0.0	0.0	27.5	G48	0.0	0.0	0.0	0.0	0.0
G22	4.5	13.0	13.0	9.8	20.6	G49	0.0	0.0	0.0	0.0	0.0
G23	10.4	25.0	25.0	9.9	12.0	G50	0.0	0.0	0.0	0.0	5.0
G24	11.7	34.0	23.0	15.1	15.9	G51	1.6	2.0	11.0	1.4	15.6
G25	10.8	19.0	13.0	9.8	14.1	G52	2.6	2.0	18.0	1.1	14.9
G26	13.7	32.0	12.0	11.2	14.3	G53	2.3	4.0	8.0	2.0	14.8
G27	10.1	19.0	31.0	8.8	20.8	G54	4.2	6.0	10.0	1.9	15.7
						Avg.	4.7	10.2	10.8	4.4	20.3

8.3 The Test-Assignment Problem

Given a set of desks in a classroom and a set of test variants, the test-assignment (TA) problem consists in assigning tests to desks, in order to minimize the likelihood of cheating. Each pair of desks has a known (physical) proximity, and each pair of test variants has a known similarity, and the likelihood of cheating is defined as the product of proximity and similarity. Therefore, desks that are close-by should receive less similar tests. If there are fewer students than desks in a classroom, one may additionally select a subset of free desks that remain without a test. The

Table 8.5 – New best known values found for the MaxCut instances. The best known solutions were found by the automatically produced algorithms.

Instance	G25	G26	G27	G28	G30	G31	G38
Previous value	13339	13326	3337	3296	3412	3306	7682
New value	13340	13327	3341	3298	3413	3310	7683

goal is to assign tests to desks minimizing the overall likelihood of cheating, defined as the sum of the likelihoods for each pair of desks.

The problem can be modeled as an undirected graph, whose vertices are the desks, and where pairs of desks (below a certain distance) are connected by an edge, which is weighted by the proximity of the incident desks. Each pair of test variants has an associated weight that defines their similarity. In this model it is easy to see that test-assignment generalizes the vertex coloring problem, where vertices represent the desks and each color represents a test variant. Indeed, for a given undirected graph $G = (V, E)$ we can set the proximity of all edges to 1, define k tests corresponding to k colors and set the similarity for identical tests to 1 and for different tests to 0. Then G admits a k-coloring if and only if there is an assignment of tests to desks of total likelihood 0. This implies that test-assignment is strongly NP-Hard, since vertex coloring is (GAREY; JOHNSON, 1977).

The test-assignment problem was introduced by Duives, Lodi and Malaguti (2013) to improve the assignment of tests at the Engineering Faculty of the University of Bologna. The authors also have shown NP-hardness of the problem by the reduction mentioned above. To the best of our knowledge, this currently is the only published paper on the test-assignment problem. Duives, Lodi and Malaguti (2013) formulate the problem as a non-convex binary quadratic program. Three different convex reformulations of that program are then solved with the commercial solver CPLEX: a standard reformulation, and two reformulations with stronger lower bounds obtained by solving an auxiliary semi-definite problem to partially and to optimality.

Duives, Lodi and Malaguti (2013) further propose a tabu search that explores the space of complete colorings in a neighborhood that selects a vertex and changes its color greedily to the one that reduces the total likelihood of cheating most. The initial solution is a random feasible coloring. After changing the color of a vertex it cannot be changed again during the tabu tenure, to avoid visiting the same

solution again. Different from a standard tabu search, the neighborhood is greedy and randomized: in each iteration a random vertex is chosen to change its color from a fixed percentage of the non-tabu vertices with the highest score, i.e. their contribution to the objective function. Unoccupied desks are taken into account by a greedy algorithm, that frees the desks with highest score. The tabu search shows a good performance in experiments with instances up to 122 desks.

Before showing the reformulation of the test-assignment problem into UBQP, let us first define it more formally. Let D be the set of desks, T the set of test variants, and γ_{tu} the similarity between tests $t, u \in T$. We assume that only a subset $E \subseteq \{\{d, e\} \mid d, e \in D\}$ of desks are close enough, and have a defined proximity p_{de}, $\{d, e\} \in E$. We further assume that $s \leq |D|$ students will take the exam, and therefore $F = |D| - s$ desks remain free. Let $T^+ = T \cup \{0\}$ be an extended set of tests, where test "0" represents a "nonexistent" test which will be assigned to free desks. Then the test-assignment problem can be formulated as follows (DUIVES; LODI; MALAGUTI, 2013):

$$\textbf{minimize} \quad \sum_{\{d,e\} \in E} p_{de} \sum_{t \in T} \sum_{u \in T} \gamma_{tu} x_{dt} x_{eu}, \tag{8.1}$$

$$\textbf{subject to} \quad \sum_{t \in T} x_{dt} = 1, \qquad d \in D, \tag{8.2}$$

$$\sum_{d \in D} x_{d0} = F, \tag{8.3}$$

$$x_{dt} \in \{0, 1\}, \qquad d \in D, t \in T. \tag{8.4}$$

In this model the binary variable $x_{dt} = 1$, if test $t \in T$ is assigned to desk $d \in D$, and $x_{dt} = 0$, otherwise. The model has a quadratic objective function (8.1), which multiplies the proximity of each pair of desks with the similarity between the assigned tests and represents the total likelihood of cheating that is to be minimized. Constraint (8.2) makes sure that exactly one test is assigned to each desk. Constraint (8.3) ensures that the "nonexistent" test is assigned to exactly F free desks.

Model (8.1 – 8.4) is a linearly-constrained binary quadratic program of the

form

$$\begin{aligned} \textbf{minimize} \quad & x^t Q x, \\ \textbf{subject to} \quad & Ax = b, \\ & x \in \{0,1\}^{nm}, \end{aligned}$$

by setting $Q = (q_{ij}) \in \mathbb{R}^{nm \times nm}$ with $q_{ij} = p_{de}\gamma_{tu}$, $n = |D|$, $m = |T|$, and a corresponding choice of $A \in \mathbb{R}^{(n+1)\times nm}$ and $b \in \mathbb{R}^{n+1}$.

Such a linearly-constrained binary quadratic program can be transformed to an equivalent unconstrained binary quadratic program by penalty methods (see e.g. Kochenberger et al. (2014)). We relax $Ax = b$ and penalize the deviation from the equality in the objective function. For a large enough penalty $P \in \mathbb{R}$ the optimal solutions of the original and the transformed program (if any) will be the same. The penalty is given by

$$\begin{aligned} P(Ax-b)^t(Ax-b) &= P(x^tA^tAx - x^tA^tb - b^tAx + b^tb) \\ &= Px^t(A^tA - 2\mathrm{diag}(A^tb))x + Pb^tb. \end{aligned}$$

Given this penalty, we can define a new coefficient matrix

$$\hat{Q} = Q + P(A^tA - 2\mathrm{diag}(A^tb)),$$

and finally rewrite the binary quadratic problem as

$$\begin{aligned} \textbf{minimize} \quad & x^t\hat{Q}x + Pb^tb, \\ \textbf{subject to} \quad & x \in \{0,1\}^{nm}. \end{aligned}$$

When optimizing, the constant term Pb^tb can be omitted. To apply this transformation to the test-assignment problem, we can choose $P = \sum_{i,j|q_{ij}>0} q_{ij}$. In this way, we make sure that the best solution which violates a constraint has a larger objective value than the worst feasible solution. This reduction allows us to solve the test-assignment problem using algorithms for the UBQP.

8.4 Automatic Design for the Test-Assignment Problem

This section presents our experiments on automatically designing heuristic algorithms for the test-assignment problem, via its reformulation to UBQP. We use the set of real-world instances proposed by Duives, Lodi and Malaguti (2013). It contains 36 instances based on four different classrooms ranging from 20 to 79 desks, of which between 0 and 20 were unoccupied, 50 to 250 proximity relations between neighboring desks, and two to four different tests per exam. Duives, Lodi and Malaguti (2013) also report results for instances with 122 desks, which are not available. We randomly selected 12 instances for the automatic design process and used a budget of 10000 executions. The results of Duives, Lodi and Malaguti (2013) have been obtained with a time limit of 100 s on a PC with a Pentium IV at 3.4 GHz and 2 GB main memory running Linux. For a fair comparison, we use a time limit of 40 s, to compensate for the relative performance of the two machines. We use this same time limit for the automatic design process and to evaluate the produced algorithms.

8.4.1 Producing Specialized Algorithms for the Test-Assignment Problem

We repeated the automatic design process two times, and AutoBQP produced two very similar hybrid heuristic algorithms HHTA_1 and HHTA_2, with slightly different strategies and parameter values. Algorithm 5 shows the algorithmic structure of both heuristics, which uses the same repeated elite recombination method of the HHBQP algorithm presented above. While there are novel solutions in the elite set, a recombination by path relinking followed by a search step is performed for each pair of elite solutions. The first difference is that HHBQP uses a best neighbor strategy in the path relinking, while HHTA_1 and HHTA_2 use a first improvement strategy. If no improving variable is found, a random variable which leads to the smallest increase of the objective function value is chosen. Besides that, HHTA_1 and HHTA_2 apply a tabu search to improve the resulting solutions from the path relinking step (instead of an iterated tabu search). The tabu search is applied in lines 17 to 20 of Algorithm 5 and uses a best improvement strategy to select non-tabu variables to flip.

The values for the input parameters of the components of algorithms HHTA_1

Algorithm 5: Hybrid heuristics for the test-assignment (HHTA_X).

```
1  while stopping criterion not satisfied do
2      E ← create an elite set of size e
3      while E has any novel solution do
4          foreach (s, t) ∈ E × E | s ≠ t do
5              V ← variables v | s[v] ≠ t[v]
6              d_min ← γ × |V|
7              d_max ← |V| − d_min
8              d ← 0
9              s* ← s
10             while V ≠ ∅ do
11                 v ← select the first improving variable with round-robin
12                 Flip s[v] and remove v from V
13                 d ← d + 1
14                 if s is better than s* then
15                     if d_min ≤ d ≤ d_max then
16                         s* ← s
17             while max. iterations/stagnation not reached do
18                 Select the best improving non-tabu variable v
19                 Update tabu list
20                 Flip s[v] and set v tabu
21             if s is better than any solution of E then
22                 Replace the worst solution of E by s
23     return best solution of E
```

and HHTA_2, which are also set by the automatic design approach, are presented in Table 8.6. Besides the parameter values, the only difference between the two algorithms lies in the choice of the improving variable during the recombination of two solutions in line 11 of Algorithm 5: HHTA_1 chooses the first improving variable in a round-robin manner, i.e. it starts from the variable chosen in the previous iteration, while HHTA_2 chooses the first improving variable in the input variable order, always starting from the first variable. These small differences suggest that the repeated recombination followed by a tabu search is a good strategy and AutoBQP can reliably identify it.

8.4.2 Evaluating the Algorithms on the Test-Assignment Instances

In this section we evaluate algorithms HHTA_1 and HHTA_2 on the complete set of instances and compare it to the tabu search proposed by Duives, Lodi and

Table 8.6 – Values of the input parameters of $HHTA_1$ and $HHTA_2$. These values were set by the automatic design approach.

Parameter	t	t_d	s	s_m	i	i_v	e	γ
Value $HHTA_1$	t_3	1	s_2	81	i_1	4204	20	0.32
Value $HHTA_2$	t_5	–	s_2	1	i_2	–	20	0.22

Malaguti (2013). The results are presented in Table 8.7. Each instance is identified by the total number of desks, the number of unoccupied desks, and the number of different tests in the exam. For each instance we report the average absolute deviation (a. d.) and the average relative deviation (r. d.) from the best known value b (defined as $v/b - 1$ for an objective function value v). We show results for the tabu search of Duives, Lodi and Malaguti (2013) (TS) and algorithms $HHTA_1$ and $HHTA_2$ over 20 replications with different seeds. The best relative deviations for each instance are highlighted in bold. Negative relative deviations indicate improvements over the current best known values. As mentioned above, the instances which have been used for the automatic design process are marked with an asterisk.

We can see that $HHTA_2$ leads to the best overall results, with an average relative deviation of 1.09%, followed by $HHTA_1$ with 1.33% and the tabu search with 2.90%. The newly found algorithms have a similar performance, and are significantly better than the existing tabu search, in particular on the large instances where the previous best solutions can be improved. The quality of the new best solutions found are presented in Table 8.8. In these large instances, $HHTA_1$ and $HHTA_2$ show a complementary performance, suggesting that different strategies for a small or a large number of unoccupied desks may be helpful. Both heuristics are consistently better than the tabu search on the majority of the instances, with $HHTA_1$ finding a worse solution in only 6 and $HHTA_2$ in only 3 cases. In two instances TS found the best solutions.

The few instances where the tabu search has a slight advantage over $HHTA_1$ or $HHTA_2$ have 20 unoccupied desks (with the exception of the instance with 47 desks, 10 of them unoccupied, and 4 tests). This can be explained by the problem specific representation used by Duives, Lodi and Malaguti (2013) which ignores unoccupied desks, and greedily removes exams from desks on evaluation, which simplifies the search space and the algorithm. Since we reduce the test-assignment to the UBQP, we do not use problem-specific components. The fact that we penalize violated

Table 8.7 – Detailed performance results on the test-assignment instances. We present the average absolute and relative deviations obtained by the automatically produced algorithms and the state-of-the-art algorithm over 20 replications. The best relative deviations for each instance are highlighted in bold.

Instance				TS		$HHTA_1$		$HHTA_2$	
Desks									
Total	Empty	Tests	Best known	a. d.	r. d. [%]	a. d.	r. d. [%]	a. d.	r. d. [%]
20	0	2	20.90	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**
20	5	2 *	7.95	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**
20	10	2	1.85	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**
20	0	3	15.15	0.20	1.32	0.00	**0.00**	0.00	**0.00**
20	5	3	5.58	0.01	0.18	0.00	**0.00**	0.00	0.01
20	10	3 *	1.22	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**
20	0	4	11.95	0.15	1.26	0.00	**0.00**	0.00	**0.00**
20	5	4	3.98	0.07	1.76	0.00	0.13	0.00	**0.00**
20	10	4 *	0.93	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**
47	0	2	72.60	1.10	1.52	0.00	**0.00**	0.00	**0.00**
47	10	2	35.45	0.15	0.42	0.00	**0.00**	0.00	**0.00**
47	20	2 *	12.65	0.15	**1.19**	0.55	4.35	0.15	**1.19**
47	0	3	53.72	1.73	3.22	0.00	**0.00**	0.00	**0.00**
47	10	3 *	24.84	0.45	1.81	0.23	0.94	0.20	**0.80**
47	20	3 *	8.52	0.16	1.88	0.43	5.05	0.10	**1.17**
47	0	4	43.88	0.72	1.64	-0.05	**-0.11**	0.20	0.46
47	10	4	19.05	0.15	0.79	0.20	1.07	0.06	**0.32**
47	20	4	6.38	0.02	**0.31**	0.26	4.14	0.47	7.30
60	0	2 *	74.05	2.25	3.04	0.00	**0.00**	0.05	**0.07**
60	10	2	43.00	1.70	3.95	0.20	**0.47**	0.20	**0.47**
60	20	2 *	20.35	1.85	9.09	0.55	**2.70**	0.55	**2.70**
60	0	3 *	54.03	1.11	2.05	0.00	**0.00**	0.00	**0.00**
60	10	3	29.95	1.36	4.54	0.84	2.79	0.56	**1.88**
60	20	3	14.11	0.82	5.81	0.56	**3.97**	0.98	6.93
60	0	4	42.93	1.52	3.54	0.67	**1.56**	0.79	1.84
60	10	4	23.18	1.15	4.96	0.05	**0.22**	0.25	1.08
60	20	4	10.70	0.53	**4.95**	1.12	10.46	0.82	7.62
79	0	2 *	109.20	2.43	2.23	0.00	**0.00**	0.10	0.09
79	10	2	71.35	0.73	1.02	0.56	0.78	0.30	**0.42**
79	20	2	43.33	0.77	1.78	0.77	1.78	0.60	**1.38**
79	0	3	80.83	3.37	4.17	-0.71	**-0.88**	-0.68	-0.84
79	10	3 *	50.26	2.24	4.46	0.53	1.05	-0.26	**-0.51**
79	20	3	29.87	1.07	3.58	0.81	2.71	-0.04	**-0.13**
79	0	4	64.40	2.93	4.55	-0.19	**-0.29**	-0.04	-0.06
79	10	4	38.65	1.78	4.61	-0.22	**-0.56**	-0.07	-0.17
79	20	4 *	21.94	1.19	5.42	1.25	5.71	1.14	**5.19**
Averages			32.46	0.94	2.90	0.23	1.33	0.18	1.09

Table 8.8 – New best known values found for the test-assignment instances. The best known solutions were found by the automatically produced algorithms.

Instance			Solution values	
Desks				
Total	Empty	Tests	Previous value	New value
Algorithm $HHTA_1$				
47	10	3	24.84	24.64
47	0	4	43.88	43.83
47	10	4	19.05	19.03
79	0	3	80.83	80.12
79	10	3	50.26	50.01
79	0	4	64.40	64.10
79	10	4	38.65	38.00
Algorithm $HHTA_2$				
47	10	4	19.05	18.93
79	0	3	80.83	80.03
79	10	3	50.26	49.71
79	20	3	29.87	29.83
79	0	4	64.40	64.33
79	10	4	38.65	38.15

constraints in the objective function generates a more irregular search space, with new local optima. For example, to go from a feasible solution to another, a constraint must be violated and, consequently, the quality of the intermediate solutions will be worse. The algorithm must handle this new search landscape.

Finally, we have evaluated the effective contribution of AutoBQP in finding good heuristic algorithms. To this end, we generated ten random hybrid heuristics from the grammar and evaluated them on the test-assignment instances under the same conditions. We found that the random heuristics have an average relative deviation of 46.5% with a standard deviation of 31.1%. This shows that the search space of algorithms contains heuristics of strongly varying quality and that the automatic algorithm design approach is effective in finding heuristics that perform well.

8.5 Discussion

The experimental results described in this chapter show the potential of the proposed AutoBQP solver in automatically designing heuristic algorithms for binary

problems. First, AutoBQP replaces the manual, laborious and often biased search for good heuristic algorithms by an automatic algorithm design approach. It requires only a problem description, a set of training instances and the configuration setup, producing heuristic algorithms automatically from components. Second, AutoBQP handles not only one problem, but a class of binary optimization problems. This is due to the algorithm components based on UBQP, which can be used to solve a wide range of problems via reformulation. Lastly, we show the flexibility of the proposed approach, which combines components from different approaches and produces hybrid and high-performing heuristic algorithms.

In addition, this research also contributes to different problem domains. AutoBQP produced new state-of-the-art algorithms for the UBQP, MaxCut and test-assignment problems, which even found new better solutions for several MaxCut and test-assignment instances. We made HHPAL, HHMC, HHBQP and HHTA$_X$ algorithms available online (see Souza and Ritt (2018f, 2018e, 2018d, 2018g))[1]. Appendix A gives more information about the artifacts produced from this research.

[1]In the following, we give links to the repositories with implementations of the algorithms produced in this chapter: HHPAL (https://github.com/souzamarcelo/hhpal); HHMC (https://github.com/souzamarcelo/hhmc); HHBQP (https://github.com/souzamarcelo/hhbqp); and HHTA (https://github.com/souzamarcelo/hhta).

Part IV

Conclusions

9 CONCLUDING REMARKS

> *We shall not cease from exploration, and the end of all our exploring will be to arrive where we started and know the place for the first time.*
>
> — T. S. Eliot

This thesis presents a comprehensive study on automatic algorithm configuration. After formalizing the algorithm configuration problem and reviewing the corresponding literature and solving approaches, we proposed a set of methods to increase efficiency, improve quality, and better understand the automatic configuration of algorithms. Besides that, we applied such methods to automatically design heuristic algorithms for binary optimization. In this chapter we discuss the main results and contributions of our research. Then, we present some open challenges in automatic algorithm configuration and give potential avenues for future research.

9.1 Configuring Optimization Algorithms Faster

We introduced a set of capping methods for the automatic configuration of optimization algorithms (Chapter 4). They explore the performance observed in previous executions to determine a performance envelope, which is used to identify unpromising executions and terminate them early. We proposed different approaches to represent the performance of an execution, e.g. using the observed performance profile or the area under such profile. We defined two general strategies for selecting previous executions: an elitist approach that consider only the best configurations found so far; and an adaptive approach that consider all previous executions and determine the performance envelope based on a given aggressiveness parameter. For the elitist methods, we proposed functions to aggregate the performance of previous executions into the envelope, e.g. worst, best and model-based aggregation functions.

We presented an extensive experimental evaluation of the proposed capping methods on six configuration scenarios with diverse characteristics. We observed that they consistently reduce the configuration effort, with reductions ranging from

about 5% to 78%, without loss of quality. The capping methods were also evaluated in scenarios with a budget defined as a total configuration time. In such scenarios, the time saved by terminating early unpromising executions is used to further explore the configuration space. The capping methods proved to be useful under these conditions, finding better configurations in comparison to using no capping at all. In conclusion, we believe the proposed capping methods contribute to the configuration of optimization algorithms, since they: (i) speed up the configuration process; (ii) allow scaling it to larger configuration spaces; and (iii) help finding better configurations in scenarios with a total configuration time budget.

9.2 Producing Better Configurations

We proposed new and improved parameter regression models for algorithm configuration (Chapter 5). Each parameter is associated with a model that sets its value according to the size of the instance being solved. Instead of searching for good parameter values, the configurator calibrates the associated model. By considering the instance size as the single problem-independent feature, we made the proposed models simple and easily applicable to any problem domain, without the need of feature engineering or the often computationally expensive feature computation. Specifically, we proposed a simple yet effective linear model to map the instance size to optimal parameter values. For cases when this relation is nonlinear, we proposed piecewise and log-log linear models.

We evaluated the parameter regression models on four configuration scenarios. Both linear and nonlinear approaches were able to approximate well the relation between instance size and optimal parameter values, even using low budgets with only hundreds of allowed executions. As a consequence, we observed good performance gains in comparison to the traditional configuration approach using constant parameter values. In summary, the proposed regression models improve the quality of the algorithm configuration without increasing complexity much. Although the transformation of the configuration space increases the number of parameters, we observed no effective impact in the quality of the configurations produced, since they were better than constant configurations even using low budgets.

9.3 Understanding the Configuration Process

We provided a visual tool to analyze the configuration process using irace (Chapter 6). This tool processes and interprets the configuration data, producing visualizations that ease the analysis and understanding of the configuration process. They show each algorithm evaluation with the associated instance and configuration, as well as median performances of regular and elite configurations at each iteration, allowing to visualize the evolution of the configuration process. We also introduced visualizations for the performance of the produced configurations on the set of test instances.

We presented a set of case studies showing how the visualization tool can be used to analyze irace runs and identify flaws in the configuration scenario. In particular, we identified instances that could be removed from the training set, unnecessarily large budgets that could be decreased, and an overtuning effect caused by using an unrepresentative set of training instances. We showed that a proper analysis and understanding of the configuration process may help the design and improvement of configuration scenarios, potentially leading to better results. Finally, the proposed visualization tool may help researchers working on improving irace, since it allows to analyze the behavior and compare irace runs under different conditions.

9.4 Solving Binary Optimization Problems Automatically

We presented a component-wise solver for binary optimization problems based on automatic algorithm design techniques (Chapter 7). This solver implements an algorithm framework of heuristic components, which allows instantiating (potentially hybrid) algorithms by selecting and combining such components. The solver represents the design space of components using a grammar, which is mapped to a set of parameters. Then, it uses automatic configuration methods to explore this design space and search for the best algorithm for a given problem. Since we used heuristic components for the unconstrained binary quadratic programming (UBQP), we can solve many optimization problems that can be reduced to it.

We applied the proposed solver to automatically produce heuristic algorithms for UBQP, maximum cut, and the test-assignment problem (Chapter 8). The resulting algorithms presented better performance in comparison to state-of-the-art approaches,

and found new best solutions for several problem instances. Both algorithm framework and the proposed component-wise solver are useful for researchers and practitioners developing new heuristic solutions for binary optimization. The solver, in particular, proved to be effective in producing competitive algorithms for new problems without the need of specialized knowledge of the underlying heuristic components or the automatic configuration techniques.

9.5 Further Contributions

In addition to the discussion above, this research gives side contributions to the automatic design and configuration of algorithms. We demonstrated the practical relevance of these approaches, since we evaluated them on several configuration scenarios and showed substantial improvements in algorithm performance. We also provided guidelines on applying automatic design methods to produce heuristic algorithms from components, which can be used to apply similar approaches to other problem domains. We produced and made available a set of high-quality algorithms for different problems using the proposed component-wise solver. Finally, we provided a comprehensive set of (mostly new) algorithm configuration scenarios that can be useful either to adapting the underlying target algorithms to new problem domains, or testing and comparing different algorithm configuration techniques. All artifacts arising from this research are publicly available, including source code for all methods, tools and algorithms, configuration scenarios and supplementary material for each chapter. Appendix A gives all information about such artifacts.

9.6 Extending this Research

In this section, we present the directions for future work suggested at the end of each chapter, discussing how the methods proposed in this thesis can be extended and improved.

Additional applications of capping methods. Although the optimization scenarios were the primary focus of the proposed capping methods, a promising continuation of this research involves their application for the automatic configuration of algorithms in other domains. In fact, the proposed capping methods are suitable to

any scenario in which the progress of the algorithm execution can be monitored. One interesting possibility is the configuration of decision algorithms, in which we try to minimize their running time. If we can measure how close a decision algorithm is to its termination (e.g. the number of satisfied clauses in boolean satisfiability), we can build a performance profile and, therefore, apply the proposed optimization-based capping. We could also combine both decision- and optimization-based capping methods in configuring decision algorithms, discarding configurations before exhausting the cut-off time whenever the observed performance profile is unsatisfactory. Finally, another potential application is to configure computer simulation models (e.g. Ferrer, López-Ibáñez and Alba (2019), Cintrano et al. (2021)). In this domain, evaluations are computationally expensive, and capping methods can help to either reduce the configuration time or improve its quality when using time budgets.

Extending parameter regression models. In this research, we proposed parameter regression models that proved to be useful, improving the automatic configuration of algorithms. There is still plenty of room for improvement. A promising avenue is exploring more sophisticated regression approaches, like (piecewise) polynomial or Gaussian Process (RASMUSSEN, 2004) models, which can represent more complex relations between instance size and optimal parameter values. Besides that, the proposed approaches are suitable only for numerical parameters, thus an important improvement would be including regression models for binary, ordinal and categorical parameters. To accomplish this, classification methods must be explored and a promising candidate is using logistic regression models (DREISEITL; OHNO-MACHADO, 2002; HASTIE; TIBSHIRANI; FRIEDMAN, 2009). Another interesting extension of these methods is a mixed two-phase configuration approach. In a first phase, parameters are configured using constant models to identify good overall values. After that, the non-constant regression models are explored and initial candidates are generated by applying perturbations to the constant configurations already found. We believe this kind of two-phase approach can be useful, since it takes advantage of a simpler first step to identify good configurations for the whole set of instances, before increasing the number of parameters to explore more complex regression models to take into account the instance size. Therefore, good configurations are produced even for scenarios in which non-constant regression models are not beneficial (e.g. when the instance size has little influence on optimal parameter values).

Improving the visual analyses. Additional visualizations can be helpful to analyze further details about the configuration process. For example, the comparison between different irace runs can be simplified by showing side-by-side visualizations of them, highlighting their main differences. The set of initial configurations or the instances used for evaluating them change the trajectory of the configuration space exploration. This may lead to substantial variation in the quality of final configurations. Visualizing the particularities of different irace runs may help to understand and explain the observed performance of the produced configurations. Besides that, such comparative visualizations may help to analyze modified versions of irace (i.e. for those involved in the development of techniques for algorithm configuration). Finally, it would be interesting to visualize the trajectory of the configuration space exploration, i.e. how the parameter values or the associated sampling models change over the iterations.

Extending the component-wise solver. The experimental evaluation of the proposed component-wise solver demonstrated its capabilities in automatically designing competitive heuristic algorithms for different problems. The main direction to extend the solver is developing and including new components to the algorithm framework, leading to new possibilities in terms of algorithm designs and potentially producing better-performing algorithms. Besides, it can be integrated to existing heuristic frameworks that allow representing and managing solutions using binary strings, e.g. ParadisEO (CAHON; MELAB; TALBI, 2004; DRÉO et al., 2021) or DEAP (FORTIN et al., 2012). Such a task is relatively simple, since the grammar-based methodology we use is quite flexible to accommodate new components. However, increasing the size of the design space may require larger configuration budgets to explore it adequately. Finally, the potential of the component-wise solver can be further explored by applying it to produce algorithms for different problem domains.

9.7 Open Challenges

Beyond the ideas discussed above, there is much work to be done related to improving and spreading the use of automatic algorithm configuration. There are a plenty of exciting research opportunities and open problems to be explored, whose solutions could substantially increase the capabilities of existing configuration

methods, or even originate completely new ones.

The first challenge we highlight here is *selecting high-quality training instances* to evaluate configurations. As discussed in Chapter 6, too easy and too hard instances may not contribute to the configuration process, since different configurations perform the same to solve them, and thus cannot be effectively compared. Moreover, an instance may be too hard at the beginning of the configuration process, when the configurations did not evolve yet, but become easier later (and even too easy). Thus, we need strategies to smartly select instances that maximize the discrimination power at each point of the configuration process, i.e. instances that help to compare different configurations and identify the best one. A possible way to accomplish that is using a form of curriculum learning (BENGIO et al., 2009), an approach commonly used to select instances for training machine learning models.

A second challenge is developing strategies for *dealing with stagnation* in the context of algorithm configuration. A seen in Chapter 6, visualizing the configuration process after completion helps to identify stagnation. Detecting it automatically while configuring (and as soon as possible) is still a challenge. Once identified, the configurator must either restart the configuration process, or apply strategies to increase the search diversification. We believe that approaches from the literature of evolutionary computation can be explored for both detecting stagnation and balancing the trade-off between exploration and exploitation. By dealing with stagnation, we avoid the need of manual restarts (e.g. Pagnozzi and Stützle (2019)), or many parallel runs of the configuration process (called *standard protocol* in some papers, e.g. Styles, Hoos and Müller (2012), Hutter, Hoos and Leyton-Brown (2012), Pushak and Hoos (2020)).

Additional challenges related to automatic algorithm configuration involve the development of strategies to: (i) incorporate information about the target algorithm in the configuration process (e.g. by means of features) to guide the search for good configurations; (ii) improve the explainability of the outcome; (iii) scale the automatic methods to large configuration scenarios; and (iv) use the performance data generated by the configuration process to analyze the target algorithm and obtain insights to improve it. We hope that the research described in this thesis and the topics discussed here encourage other researchers to further develop automatic algorithm configuration methods.

REFERENCES

ACHTERBERG, T. SCIP: Solving constraint integer programs. **Mathematical Programming Computation**, v. 1, n. 1, p. 1–41, jul. 2009.

ADENSO-DÍAZ, B.; LAGUNA, M. Fine-tuning of algorithms using fractional experimental design and local search. **Operations Research**, v. 54, n. 1, p. 99–114, 2006.

ALETI, A.; MOSER, I. A systematic literature review of adaptive parameter control methods for evolutionary algorithms. **ACM Computing Surveys**, v. 49, n. 56, p. 1–35, oct. 2016.

ALFARO-FERNÁNDEZ, P.; RUIZ, R.; PAGNOZZI, F.; STÜTZLE, T. Automatic algorithm design for hybrid flowshop scheduling problems. **European Journal of Operational Research**, v. 282, n. 3, p. 835–845, 2020.

ALIDAEE, B.; KOCHENBERGER, G. A.; AHMADIAN, A. 0-1 quadratic programming approach for optimum solutions of two scheduling problems. **International Journal of Systems Science**, Taylor & Francis, v. 25, n. 2, p. 401–408, 1994.

ANSEL, J.; KAMIL, S.; VEERAMACHANENI, K.; RAGAN-KELLEY, J.; BOSBOOM, J.; O'REILLY, U. M.; AMARASINGHE, S. OpenTuner: An extensible framework for program autotuning. In: **Proceedings of the 23rd International Conference on Parallel Architectures and Compilation**. New York, NY: ACM Press, 2014. p. 303–315.

ANSÓTEGUI, C.; GABÀS, J.; MALITSKY, Y.; SELLMANN, M. MaxSAT by improved instance-specific algorithm configuration. **Artificial Intelligence**, v. 235, p. 26–39, 2016. ISSN 0004-3702.

ANSÓTEGUI, C.; MALITSKY, Y.; SAMULOWITZ, H.; SELLMANN, M.; TIERNEY, K. Model-based genetic algorithms for algorithm configuration. In: YANG, Q.; WOOLDRIDGE, M. (Ed.). **Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI-15)**. Menlo Park, CA: IJCAI/AAAI Press, 2015. p. 733–739.

ANSÓTEGUI, C.; MALITSKY, Y.; SELLMANN, M. MaxSAT by improved instance-specific algorithm configuration. In: STRACUZZI, D. et al. (Ed.). **Proceedings of AAAI 2014 – Twenty-Eighth National Conference on Artificial Intelligence**. Menlo Park, CA: AAAI Press/MIT Press, 2014. p. 2594–2600.

ANSÓTEGUI, C.; SELLMANN, M.; TIERNEY, K. A gender-based genetic algorithm for the automatic configuration of algorithms. In: GENT, I. P. (Ed.). **Principles and Practice of Constraint Programming, CP 2009**. Heidelberg, Germany: Springer, 2009, (Lecture Notes in Computer Science, v. 5732). p. 142–157.

AUDET, C.; DANG, C.-K.; ORBAN, D. Optimization of algorithms with OPAL. **Mathematical Programming Computation**, v. 6, n. 3, p. 233–254, 2014.

AUDET, C.; ORBAN, D. Finding optimal algorithmic parameters using derivative-free optimization. **SIAM Journal on Optimization**, v. 17, n. 3, p. 642–664, 2006.

AYDIN, D.; YAVUZ, G.; STÜTZLE, T. ABC-X: A generalized, automatically configurable artificial bee colony framework. **Swarm Intelligence**, v. 11, n. 1, p. 1–38, 2017.

BABIĆ, D.; HUTTER, F. **Spear Theorem Prover**. 2008. SAT'08: Proceedings of the SAT 2008 Race.

BÄCK, T. Self-adaptation in genetic algorithms. In: VARELA, F. J.; BOURGINE, P. (Ed.). **Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life**. Cambridge, MA: MIT Press, 1992. p. 263–271.

BALAPRAKASH, P.; BIRATTARI, M.; STÜTZLE, T. Improvement strategies for the F-Race algorithm: Sampling design and iterative refinement. In: BARTZ-BEIELSTEIN, T.; BLESA, M. J.; BLUM, C.; NAUJOKS, B.; ROLI, A.; RUDOLPH, G.; SAMPELS, M. (Ed.). **Hybrid Metaheuristics**. Heidelberg, Germany: Springer, 2007, (Lecture Notes in Computer Science, v. 4771). p. 108–122.

BARTZ-BEIELSTEIN, T.; LASARCZYK, C.; PREUSS, M. Sequential parameter optimization. In: **Proceedings of the 2005 Congress on Evolutionary Computation (CEC 2005)**. Piscataway, NJ: IEEE Press, 2005. p. 773–780.

BARTZ-BEIELSTEIN, T.; LASARCZYK, C.; PREUSS, M. The sequential parameter optimization toolbox. In: BARTZ-BEIELSTEIN, T.; CHIARANDINI, M.; PAQUETE, L.; PREUSS, M. (Ed.). **Experimental Methods for the Analysis of Optimization Algorithms**. Berlin, Germany: Springer, 2010. p. 337–360.

BARTZ-BEIELSTEIN, T.; PREUSS, M. Considerations of budget allocation for sequential parameter optimization (SPO). In: PAQUETE, L.; CHIARANDINI, M.; BASSO, D. (Ed.). **Empirical Methods for the Analysis of Algorithms, Workshop EMAA 2006, Proceedings**. University of Southern Denmark, 2006. p. 35–40. Available from Internet: https://imada.sdu.dk/~marco/EMAA/Proceedings.html.

BARTZ-BEIELSTEIN, T.; ZIEGENHIRT, J.; KONEN, W.; FLASCH, O.; KOCH, P.; ZAEFFERER, M. **SPOT: Sequential Parameter Optimization**. 2011. R package. Available from Internet: http://cran.r-project.org/package=SPOT.

BEASLEY, J. E. **Heuristic Algorithms for the Unconstrained Binary Quadratic Programming Problem**. London, England: The Management School, Imperial College, 1998. Available from Internet: http://people.brunel.ac.uk/~mastjjb/jeb/bqp.pdf.

BELKHIR, N.; DRÉO, J.; SAVÉANT, P.; SCHOENAUER, M. Feature based algorithm configuration: A case study with differential evolution. In: HANDL, J.; HART, E.; LEWIS, P. R.; LÓPEZ-IBÁÑEZ, M.; OCHOA, G.; PAECHTER, B. (Ed.). **Parallel Problem Solving from Nature – PPSN XIV**. Cham, Switzerland: Springer, 2016. (Lecture Notes in Computer Science, v. 9921), p. 156–166. ISBN 978-3-319-45823-6.

BELKHIR, N.; DRÉO, J.; SAVÉANT, P.; SCHOENAUER, M. Per instance algorithm configuration of CMA-ES with limited budget. In: BOSMAN, P. A. N. (Ed.). **Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017**. New York, NY: ACM Press, 2017. p. 681–688.

BENAVIDES, A. J.; RITT, M. Iterated local search heuristics for minimizing total completion time in permutation and non-permutation flow shops. In: BRAFMAN, R. I.; DOMSHLAK, C.; HASLUM, P.; ZILBERSTEIN, S. (Ed.). **Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015**. Menlo Park, CA: AAAI Press, 2015. p. 34–41.

BENGIO, Y.; LODI, A.; PROUVOST, A. Machine learning for combinatorial optimization: A methodological tour d'horizon. **European Journal of Operational Research**, v. 290, n. 2, p. 405–421, 2021. ISSN 0377-2217.

BENGIO, Y.; LOURADOUR, J.; COLLOBERT, R.; WESTON, J. Curriculum learning. In: **Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009**. New York, NY: ACM Press, 2009. p. 41–48. ISBN 9781605585161.

BENOIST, T.; ESTELLON, B.; GARDI, F.; MEGEL, R.; NOUIOUA, K. LocalSolver 1.x: A black-box local-search solver for 0-1 programming. **4OR – A Quarterly Journal of Operations Research**, Springer, v. 9, n. 3, p. 299–316, 2011.

BERGSTRA, J. S.; BARDENET, R.; BENGIO, Y.; KÉGL, B. Algorithms for hyper-parameter optimization. In: SHAWE-TAYLOR, J.; ZEMEL, R. S.; BARTLETT, P. L.; PEREIRA, F. B.; WEINBERGER, K. Q. (Ed.). **Advances in Neural Information Processing Systems (NIPS 24)**. Red Hook, NY: Curran Associates, 2011. p. 2546–2554. Available from Internet: http://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf.

BERKEY, J. O.; WANG, P. Y. Two-dimensional finite bin-packing algorithms. **Journal of the Operational Research Society**, v. 38, n. 5, p. 423–429, 1987.

BEZERRA, L. C. T.; LÓPEZ-IBÁÑEZ, M.; STÜTZLE, T. Automatic design of evolutionary algorithms for multi-objective combinatorial optimization. In: BARTZ-BEIELSTEIN, T.; BRANKE, J.; FILIPIČ, B.; SMITH, J. (Ed.). **Parallel Problem Solving from Nature – PPSN XIII**. Cham, Switzerland: Springer, 2014, (Lecture Notes in Computer Science, v. 8672). p. 508–517.

BEZERRA, L. C. T.; LÓPEZ-IBÁÑEZ, M.; STÜTZLE, T. Deconstructing multi-objective evolutionary algorithms: An iterative analysis on the permutation flowshop. In: PARDALOS, P. M.; RESENDE, M. G. C.; VOGIATZIS, C.; WALTEROS, J. L. (Ed.). **Learning and Intelligent Optimization, 8th International Conference, LION 8**. Cham, Switzerland: Springer, 2014, (Lecture Notes in Computer Science, v. 8426). p. 57–172.

BEZERRA, L. C. T.; LÓPEZ-IBÁÑEZ, M.; STÜTZLE, T. Automatic component-wise design of multi-objective evolutionary algorithms. **IEEE Transactions on Evolutionary Computation**, v. 20, n. 3, p. 403–417, 2016.

BEZERRA, L. C. T.; LÓPEZ-IBÁÑEZ, M.; STÜTZLE, T. Automatic configuration of multi-objective optimizers and multi-objective configuration. In: BARTZ-BEIELSTEIN, T.; FILIPIČ, B.; KOROŠEC, P.; TALBI, E.-G. (Ed.). **High-Performance Simulation-Based Optimization**. Cham, Switzerland: Springer International Publishing, 2020. p. 69–92.

BEZERRA, L. C. T.; LÓPEZ-IBÁÑEZ, M.; STÜTZLE, T. Automatically designing state-of-the-art multi- and many-objective evolutionary algorithms. **Evolutionary Computation**, v. 28, n. 2, p. 195–226, 2020.

BIERE, A.; HEULE, M.; VAN MAAREN, H.; WALSH, T. (Ed.). **Handbook of Satisfiability**. Amsterdam, The Netherlands: IOS Press, 2021. (Frontiers in Artificial Intelligence and Applications, v. 336). ISBN 978-1-64368-160-3.

BIRATTARI, M. **Tuning Metaheuristics: A Machine Learning Perspective**. Berlin, Heidelberg: Springer, 2009. (Studies in Computational Intelligence, v. 197).

BIRATTARI, M.; STÜTZLE, T.; PAQUETE, L.; VARRENTRAPP, K. A racing algorithm for configuring metaheuristics. In: LANGDON, W. B. et al. (Ed.). **Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2002**. San Francisco, CA: Morgan Kaufmann Publishers, 2002. p. 11–18.

BIRATTARI, M.; YUAN, Z.; BALAPRAKASH, P.; STÜTZLE, T. F-Race and iterated F-Race: An overview. In: BARTZ-BEIELSTEIN, T.; CHIARANDINI, M.; PAQUETE, L.; PREUSS, M. (Ed.). **Experimental Methods for the Analysis of Optimization Algorithms**. Berlin, Germany: Springer, 2010. p. 311–336.

BISCHL, B.; LANG, M.; KOTTHOFF, L.; SCHIFFNER, J.; RICHTER, J.; STUDERUS, E.; CASALICCHIO, G.; JONES, Z. M. mlr: Machine learning in R. **Journal of Machine Learning Research**, v. 17, n. 170, p. 1–5, 2016.

BLUM, C.; CALVO, B.; BLESA, M. J. FrogCOL and FrogMIS: New decentralized algorithms for finding large independent sets in graphs. **Swarm Intelligence**, v. 9, n. 2-3, p. 205–227, 2015.

BOMZE, I. M.; BUDINICH, M.; PARDALOS, P. M.; PELILLO, M. The maximum clique problem. In: DU, D.-Z.; PARDALOS, P. M. (Ed.). **Handbook of Combinatorial Optimization: Supplement Volume A**. Boston, MA: Springer US, 1999. p. 1–74.

BONDY, J. A.; MURTY, U. S. R. **Graph Theory with Applications**. London, UK: The Macmillan Press, 1976.

BÖTTCHER, S.; DOERR, B.; NEUMANN, F. Optimal fixed and adaptive mutation rates for the LeadingOnes problem. In: SCHAEFER, R.; COTTA, C.; KOŁODZIEJ, J.; RUDOLPH, G. (Ed.). **Parallel Problem Solving from Nature – PPSN XI**. Heidelberg, Germany: Springer, 2010. (Lecture Notes in Computer Science), p. 1–10. ISBN 978-3-642-15844-5.

BRAMEIER, M. F.; BANZHAF, W. **Linear Genetic Programming**. New York, NY: Springer-Verlag, 2007. (Genetic and Evolutionary Computation).

BRANKE, J.; ELOMARI, J. Simultaneous tuning of metaheuristic parameters for various computing budgets. In: KRASNOGOR, N.; LANZI, P. L. (Ed.). **Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2011**. New York, NY: ACM Press, 2011. p. 263–264.

BREIMAN, L. Random forests. **Machine Learning**, v. 45, n. 1, p. 5–32, 2001.

BRENDEL, M.; SCHOENAUER, M. Instance-based parameter tuning for evolutionary AI planning. In: KRASNOGOR, N.; LANZI, P. L. (Ed.). **Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2011 (Companion)**. New York, NY: ACM Press, 2011. p. 591–598.

BRUM, A.; RITT, M. Automatic algorithm configuration for the permutation flow shop scheduling problem minimizing total completion time. In: LIEFOOGHE, A.; LÓPEZ-IBÁÑEZ, M. (Ed.). **Proceedings of EvoCOP 2018 – 18th European Conference on Evolutionary Computation in Combinatorial Optimization**. Heidelberg, Germany: Springer, 2018. (Lecture Notes in Computer Science, v. 10782), p. 85–100.

BRUM, A.; RITT, M. Automatic design of heuristics for minimizing the makespan in permutation flow shops. In: **Proceedings of the 2018 Congress on Evolutionary Computation (CEC 2018)**. Piscataway, NJ: IEEE Press, 2018. p. 1–8.

BRUSCO, M. J.; KÖHN, H.-F. Clustering qualitative data based on binary equivalence relations: Neighborhood search heuristics for the clique partitioning problem. **Psychometrika**, Springer, v. 74, n. 4, p. 685–703, 2009.

BURER, S.; MONTEIRO, R. D. C.; ZHANG, Y. Rank-two relaxation heuristics for max-cut and other binary quadratic programs. **SIAM Journal on Optimization**, SIAM, v. 12, n. 2, p. 503–521, 2002.

BURKE, E. K.; GENDREAU, M.; HYDE, M. R.; KENDALL, G.; OCHOA, G.; ÖZCAN, E.; QU, R. Hyper-heuristics: A survey of the state of the art. **Journal of the Operational Research Society**, v. 64, n. 12, p. 1695–1724, 2013.

BURKE, E. K.; HYDE, M. R.; KENDALL, G. Grammatical evolution of local search heuristics. **IEEE Transactions on Evolutionary Computation**, v. 16, n. 7, p. 406–417, 2012.

BURKE, E. K.; HYDE, M. R.; KENDALL, G.; OCHOA, G.; ÖZCAN, E.; WOODWARD, J. R. A classification of hyper-heuristic approaches: Revisited. In: GENDREAU, M.; POTVIN, J.-Y. (Ed.). **Handbook of Metaheuristics**. New York, NY: Springer International Publishing, 2019, (International Series in Operations Research & Management Science, v. 272). chp. 14, p. 453–477.

CAHON, S.; MELAB, N.; TALBI, E.-G. ParadisEO: A framework for the reusable design of parallel and distributed metaheuristics. **Journal of Heuristics**, v. 10, n. 3, p. 357–380, 2004.

CHARON, I.; HUDRY, O. Noising methods for a clique partitioning problem. **Discrete Applied Mathematics**, v. 154, n. 5, p. 754–769, 2006. ISSN 0166-218X.

CHAU, D. P.; THONNAT, M.; BRÉMOND, F.; CORVÉE, E. Online parameter tuning for object tracking algorithms. **Image and Vision Computing**, v. 32, n. 4, p. 287–302, 2014. ISSN 0262-8856.

CINTRANO, C.; FERRER, J.; LÓPEZ-IBÁÑEZ, M.; ALBA, E. Hybridization of racing methods with evolutionary operators for simulation optimization of traffic lights programs. In: ZARGES, C.; VEREL, S. (Ed.). **Proceedings of EvoCOP 2021 – 21st European Conference on Evolutionary Computation in Combinatorial Optimization**. Cham, Switzerland: Springer, 2021. (Lecture Notes in Computer Science, v. 12692), p. 17–33.

CONOVER, W. J. **Practical Nonparametric Statistics**. 3rd. ed. New York, NY: John Wiley & Sons, 1999.

COY, S. P.; GOLDEN, B. L.; RUNGER, G. C.; WASIL, E. A. Using experimental design to find effective parameter settings for heuristics. **Journal of Heuristics**, v. 7, n. 1, p. 77–97, 2001.

DANG, N.; PÉREZ CÁCERES, L.; DE CAUSMAECKER, P.; STÜTZLE, T. Configuring irace using surrogate configuration benchmarks. In: BOSMAN, P. A. N. (Ed.). **Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017**. New York, NY: ACM Press, 2017. p. 243–250.

DANIEL, C.; TAYLOR, J.; NOWOZIN, S. Learning step size controllers for robust neural network training. In: SCHUURMANS, D.; WELLMAN, M. P. (Ed.). **Proceedings of AAAI 2016 – Thirtieth National Conference on Artificial Intelligence**. Menlo Park, CA: AAAI Press/MIT Press, 2016. p. 1519–1525.

DE VRIES, S.; VOHRA, R. V. Combinatorial auctions: A survey. **INFORMS Journal on Computing**, INFORMS, v. 15, n. 3, p. 284–309, 2003.

DELORME, M.; IORI, M.; MARTELLO, S. Bin packing and cutting stock problems: Mathematical models and exact algorithms. **European Journal of Operational Research**, Elsevier, v. 255, n. 1, p. 1–20, 2016.

DELORME, M.; IORI, M.; MARTELLO, S. BPPLIB: A library for bin packing and cutting stock problems. **Optimization Letters**, v. 12, n. 2, p. 235–250, 2018.

DI GASPERO, L.; SCHAERF, A. Easysyn++: A tool for automatic synthesis of stochastic local search algorithms. In: STÜTZLE, T.; BIRATTARI, M.; HOOS, H. H. (Ed.). **Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics. SLS 2007**. Heidelberg, Germany: Springer, 2007, (Lecture Notes in Computer Science, v. 4638). p. 177–181.

DORIGO, M.; STÜTZLE, T. **Ant Colony Optimization**. Cambridge, MA: MIT Press, 2004.

DRAKE, J. H.; KHEIRI, A.; ÖZCAN, E.; BURKE, E. K. Recent advances in selection hyper-heuristics. **European Journal of Operational Research**, v. 285, n. 2, p. 405–428, 2020. ISSN 0377-2217.

DREISEITL, S.; OHNO-MACHADO, L. Logistic regression and artificial neural network classification models: A methodology review. **Journal of Biomedical Informatics**, v. 35, n. 5, p. 352–359, 2002. ISSN 1532-0464.

DRÉO, J.; LIEFOOGHE, A.; VEREL, S.; SCHOENAUER, M.; MERELO, J. J.; QUEMY, A.; BOUVIER, B.; GMYS, J. Paradiseo: From a modular framework for evolutionary computation to the automated design of metaheuristics. In: CHICANO, F. (Ed.). **Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2021**. New York, NY: ACM Press, 2021. p. 1522–1530. ISBN 9781450383516.

DUBOIS-LACOSTE, J.; LÓPEZ-IBÁÑEZ, M.; STÜTZLE, T. Automatic configuration of state-of-the-art multi-objective optimizers using the TP+PLS framework. In: KRASNOGOR, N.; LANZI, P. L. (Ed.). **Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2011**. New York, NY: ACM Press, 2011. p. 2019–2026.

DUIVES, J.; LODI, A.; MALAGUTI, E. Test-assignment: A quadratic coloring problem. **Journal of Heuristics**, Springer, v. 19, n. 4, p. 549–564, 2013.

DUNN, O. J. Multiple comparisons among means. **Journal of the American Statistical Association**, Taylor & Francis Group, v. 56, n. 293, p. 52–64, 1961.

DUNN, O. J. Multiple comparisons using rank sums. **Technometrics**, Taylor & Francis Group, v. 6, n. 3, p. 241–252, 1964.

EIBEN, A. E.; HINTERDING, R.; MICHALEWICZ, Z. Parameter control in evolutionary algorithms. **IEEE Transactions on Evolutionary Computation**, v. 3, n. 2, p. 124–141, 1999.

EL YAFRANI, M.; AHIOD, B. Efficiently solving the traveling thief problem using hill climbing and simulated annealing. **Information Sciences**, v. 432, p. 231–244, 2018. ISSN 0020-0255.

EL YAFRANI, M.; SCOCZYNSKI, M.; SUNG, I.; WAGNER, M.; DOERR, C.; NIELSEN, P. MATE: A model-based algorithm tuning engine. In: ZARGES, C.; VEREL, S. (Ed.). **Proceedings of EvoCOP 2021 – 21st European Conference on Evolutionary Computation in Combinatorial Optimization**. Cham, Switzerland: Springer, 2021. (Lecture Notes in Computer Science, v. 12692), p. 51–67.

EMMONS, H.; VAIRAKTARAKIS, G. **Flow Shop Scheduling: Theoretical Results, Algorithms, and Applications**. Boston, MA: Springer, 2013. (International Series in Operations Research & Management Science, 1). ISBN 978-1-4614-5151-8.

FALKNER, S.; LINDAUER, M. T.; HUTTER, F. SpySMAC: Automated configuration and performance analysis of SAT solvers. In: HEULE, M.; WEAVER, S. (Ed.). **Theory and Applications of Satisfiability Testing – SAT 2015**. Cham, Switzerland: Springer, 2015, (Lecture Notes in Computer Science, v. 9340). p. 215–222.

FAWCETT, C.; HOOS, H. H. Analysing differences between algorithm configurations through ablation. **Journal of Heuristics**, v. 22, n. 4, p. 431–458, 2016.

FEO, T. A.; RESENDE, M. G. C. Greedy randomized adaptive search procedures. **Journal of Global Optimization**, v. 6, n. 2, p. 109–113, 1995.

FERRER, J.; LÓPEZ-IBÁÑEZ, M.; ALBA, E. Reliable simulation-optimization of traffic lights in a real-world city. **Applied Soft Computing**, v. 78, p. 697–711, 2019.

FORTIN, F.-A.; DE RAINVILLE, F.-M.; GARDNER, M.-A.; PARIZEAU, M.; GAGNÉ, C. DEAP: Evolutionary algorithms made easy. **Journal of Machine Learning Research**, v. 13, p. 2171–2175, jul. 2012.

FRANZIN, A.; STÜTZLE, T. Revisiting simulated annealing: A component-based analysis. **Computers & Operations Research**, v. 104, p. 191–206, 2019.

FRANZIN, A.; STÜTZLE, T. Towards transferring algorithm configurations across problems. In: VLASTELICA, M. et al. (Ed.). **Learning Meets Combinatorial Algorithms at NeurIPS2020**. OpenReview, 2020. Available from Internet: https://openreview.net/group?id=NeurIPS.cc/2020/Workshop/LMCA.

FRANZIN, A.; STÜTZLE, T. **A Landscape-based Analysis of Fixed Temperature and Simulated Annealing**. 2021. Available from Internet: http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2021-005.pdf.

GALINIER, P.; HAO, J.-K. Hybrid evolutionary algorithms for graph coloring. **Journal of Combinatorial Optimization**, Springer, v. 3, n. 4, p. 379–397, 1999.

GAREY, M. R.; JOHNSON, D. S. Two-processor scheduling with start-times and deadlines. **SIAM Journal on Computing**, v. 6, n. 3, p. 416–426, sep. 1977.

GEBSER, M.; KAMINSKI, R.; KAUFMANN, B.; SCHAUB, T.; SCHNEIDER, M. T.; ZILLER, S. A portfolio solver for answer set programming: Preliminary report. In: CALABAR, P.; SON, T. C. (Ed.). **Logic Programming and Nonmonotonic Reasoning**. Heidelberg, Germany: Springer, 2013, (Lecture Notes in Artificial Intelligence, v. 8148). p. 352–357.

GENT, I. P.; HOOS, H. H.; PROSSER, P.; WALSH, T. Morphing: Combining structure and randomness. In: HENDLER, J.; SUBRAMANIAN, D. (Ed.). **Proceedings of AAAI 1999 – Sixteenth National Conference on Artificial Intelligence**. Menlo Park, CA: AAAI Press/MIT Press, 1999. p. 654–660.

GLOVER, F. Tabu search. **ORSA Journal on Computing**, INFORMS, v. 1, n. 3, p. 190–206, 1989.

GLOVER, F.; LÜ, Z.; HAO, J.-K. Diversification-driven tabu search for unconstrained binary quadratic problems. **4OR: A Quarterly Journal of Operations Research**, v. 8, n. 3, p. 239–253, 2010.

GORTAZAR, F.; DUARTE, A.; LAGUNA, M.; MARTÍ, R. Black box scatter search for general classes of binary optimization problems. **Computers & Operations Research**, Elsevier, v. 37, n. 11, p. 1977–1986, 2010.

GRATCH, J.; CHIEN, S. Adaptive problem-solving for large-scale scheduling problems: A case study. **Journal of Artificial Intelligence Research**, v. 4, p. 365–396, 1996.

GRATCH, J.; DEJONG, G. COMPOSER: A probabilistic solution to the utility problem in speed-up learning. In: ROSENBLOOM, P.; SZOLOVITS, P. (Ed.). **Proceedings of AAAI 1992 – Tenth National Conference on Artificial Intelligence**. Menlo Park, CA: AAAI Press/MIT Press, 1992. p. 235–240.

GREFENSTETTE, J. J. Optimization of control parameters for genetic algorithms. **IEEE Transactions on Systems, Man, and Cybernetics**, v. 16, n. 1, p. 122–128, 1986.

GRÖTSCHEL, M.; WAKABAYASHI, Y. A cutting plane algorithm for a clustering problem. **Mathematical Programming**, Springer, v. 45, n. 1, p. 59–96, 1989.

GRÖTSCHEL, M.; WAKABAYASHI, Y. Facets of the clique partitioning polytope. **Mathematical Programming**, Springer, v. 47, n. 1, p. 367–387, 1990.

HANSEN, N. The CMA evolution strategy: A comparing review. In: LOZANO, J. A.; LARRAÑAGA, P.; INZA, I.; BENGOETXEA, E. (Ed.). **Towards a New Evolutionary Computation: Advances on Estimation of Distribution Algorithms**. 1st. ed. Heidelberg, Germany: Springer-Verlag, 2006, (Studies in Fuzziness and Soft Computing, v. 192). p. 75–102.

HANSEN, N.; OSTERMEIER, A. Completely derandomized self-adaptation in evolution strategies. **Evolutionary Computation**, v. 9, n. 2, p. 159–195, 2001.

HANSEN, P.; JAUMARD, B. Algorithms for the maximum satisfiability problem. **Computing**, v. 44, p. 279–303, 1990.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. **The Elements of Statistical Learning: Data Mining, Inference, and Prediction**. 2nd. ed. New York, NY: Springer, 2009. (Springer Series in Statistics). ISSN 0172-7397.

HAWKINS, D. M. The problem of overfitting. **Journal of Chemical Information and Computer Sciences**, ACS Publications, v. 44, n. 1, p. 1–12, 2004.

HELMBERG, C.; RENDL, F. A spectral bundle method for semidefinite programming. **SIAM Journal on Optimization**, SIAM, v. 10, n. 3, p. 673–696, 2000.

HELSGAUN, K. An effective implementation of the Lin-Kernighan traveling salesman heuristic. **European Journal of Operational Research**, v. 126, p. 106–130, 2000.

HELSGAUN, K. General *k*-opt submoves for the Lin-Kernighan TSP heuristic. **Mathematical Programming Computation**, v. 1, n. 2–3, p. 119–163, 2009.

HELSGAUN, K. Efficient recombination in the Lin-Kernighan-Helsgaun traveling salesman heuristic. In: AUGER, A.; FONSECA, C. M.; LOURENÇO, N.; MACHADO, P.; PAQUETE, L.; WHITLEY, D. (Ed.). **Parallel Problem Solving from Nature – PPSN XV**. Cham, Switzerland: Springer, 2018, (Lecture Notes in Computer Science, v. 11101). p. 95–107.

HELSGAUN, K. **Source Code of the Lin-Kernighan-Helsgaun Traveling Salesman Heuristic**. 2018. Available from Internet: http://webhotel4.ruc.dk/~keld/research/LKH.

HOOS, H. H. Automated algorithm configuration and parameter tuning. In: HAMADI, Y.; MONFROY, E.; SAUBION, F. (Ed.). **Autonomous Search**. Berlin, Germany: Springer, 2012. p. 37–71.

HOOS, H. H. Programming by optimization. **Communications of the ACM**, v. 55, n. 2, p. 70–80, feb. 2012.

HOOS, H. H.; HUTTER, F.; LEYTON-BROWN, K. Automated configuration and selection of SAT solvers. In: BIERE, A.; HEULE, M.; VAN MAAREN, H.; WALSH, T. (Ed.). **Handbook of Satisfiability**. Amsterdam, The Netherlands: IOS Press, 2021, (Frontiers in Artificial Intelligence and Applications, v. 336). p. 481–507. ISBN 978-1-64368-160-3.

HOOS, H. H.; STÜTZLE, T. **Stochastic Local Search – Foundations and Applications**. San Francisco, CA: Morgan Kaufmann Publishers, 2005.

HUTTER, F. **SAT Benchmarks Used in Automated Algorithm Configuration**. 2007. Available from Internet: http://www.cs.ubc.ca/labs/beta/Projects/AAC/SAT-benchmarks.html.

HUTTER, F.; BABIĆ, D.; HOOS, H. H.; HU, A. J. Boosting verification by automatic tuning of decision procedures. In: BAUMGARTNER, J.; SHEERAN, M. (Ed.). **FMCAD'07: Proceedings of the 7th International Conference Formal Methods in Computer Aided Design**. Washington, DC, USA: IEEE Computer Society, 2007. p. 27–34.

HUTTER, F.; HAMADI, Y. Parameter adjustment based on performance prediction: Towards an instance-aware problem solver. n. Technical Report MSR-TR-2005-125, 2005.

HUTTER, F.; HAMADI, Y.; HOOS, H. H.; LEYTON-BROWN, K. Performance prediction and automated tuning of randomized and parametric algorithms. In: BENHAMOU, F. (Ed.). **Principles and Practice of Constraint Programming, CP 2006**. Heidelberg, Germany: Springer, 2006. (Lecture Notes in Computer Science, v. 4204), p. 213–228.

HUTTER, F.; HOOS, H. H.; LEYTON-BROWN, K. Automated configuration of mixed integer programming solvers. In: LODI, A.; MILANO, M.; TOTH, P. (Ed.). **Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 7th International Conference, CPAIOR 2010**. Heidelberg, Germany: Springer, 2010, (Lecture Notes in Computer Science, v. 6140). p. 186–202.

HUTTER, F.; HOOS, H. H.; LEYTON-BROWN, K. Sequential model-based optimization for general algorithm configuration. In: COELLO COELLO, C. A. (Ed.). **Learning and Intelligent Optimization, 5th International Conference, LION 5**. Cham, Switzerland: Springer, 2011, (Lecture Notes in Computer Science, v. 6683). p. 507–523.

HUTTER, F.; HOOS, H. H.; LEYTON-BROWN, K. Parallel algorithm configuration. In: HAMADI, Y.; SCHOENAUER, M. (Ed.). **Learning and Intelligent Optimization, 6th International Conference, LION 6**. Heidelberg, Germany: Springer, 2012, (Lecture Notes in Computer Science, v. 7219). p. 55–70.

HUTTER, F.; HOOS, H. H.; LEYTON-BROWN, K. An efficient approach for assessing hyperparameter importance. In: XING, E. P.; JEBARA, T. (Ed.). **Proceedings of the 31st International Conference on Machine Learning, ICML 2014**. JMLR.org, 2014. v. 32, p. 754–762. Available from Internet: http://jmlr.org/proceedings/papers/v32/hutter14.html.

HUTTER, F.; HOOS, H. H.; LEYTON-BROWN, K.; MURPHY, K. P. An experimental investigation of model-based parameter optimisation: SPO and beyond. In: ROTHLAUF, F. (Ed.). **Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2009**. New York, NY: ACM Press, 2009. p. 271–278.

HUTTER, F.; HOOS, H. H.; LEYTON-BROWN, K.; STÜTZLE, T. ParamILS: An automatic algorithm configuration framework. **Journal of Artificial Intelligence Research**, v. 36, p. 267–306, oct. 2009.

HUTTER, F.; HOOS, H. H.; LEYTON-BROWN, K.; MURPHY, K. P. Time-bounded sequential parameter optimization. In: BLUM, C.; BATTITI, R. (Ed.). **Learning and Intelligent Optimization, 4th International Conference, LION 4**. Heidelberg, Germany: Springer, 2010, (Lecture Notes in Computer Science, v. 6073). p. 281–298.

HUTTER, F.; KOTTHOFF, L.; VANSCHOREN, J. **Automated Machine Learning: Methods, Systems, Challenges**. Cham, Switzerland: Springer Nature, 2019. (The Springer Series on Challenges in Machine Learning).

HUTTER, F.; LINDAUER, M. T.; BALINT, A.; BAYLESS, S.; HOOS, H. H.; LEYTON-BROWN, K. The configurable SAT solver challenge (CSSC). **Artificial Intelligence**, v. 243, p. 1–25, 2017.

HUTTER, F.; LÓPEZ-IBÁÑEZ, M.; FAWCETT, C.; LINDAUER, M. T.; HOOS, H. H.; LEYTON-BROWN, K.; STÜTZLE, T. AClib: A benchmark library for algorithm configuration. In: PARDALOS, P. M.; RESENDE, M. G. C.; VOGIATZIS, C.; WALTEROS, J. L. (Ed.). **Learning and Intelligent Optimization, 8th International Conference, LION 8**. Cham, Switzerland: Springer, 2014, (Lecture Notes in Computer Science, v. 8426). p. 36–40.

HYDE, M. R.; BURKE, E. K.; KENDALL, G. Automated code generation by local search. **Journal of the Operational Research Society**, Springer, v. 64, n. 12, p. 1725–1741, 2013.

JOHNSON, D. S.; MCGEOCH, L. A.; REGO, C.; GLOVER, F. **8th DIMACS Implementation Challenge: The Traveling Salesman Problem**. 2001. Available from Internet: http://dimacs.rutgers.edu/archive/Challenges/TSP.

JONES, D. R.; SCHONLAU, M.; WELCH, W. J. Efficient global optimization of expensive black-box functions. **Journal of Global Optimization**, v. 13, n. 4, p. 455–492, 1998.

KADIOGLU, S.; MALITSKY, Y.; SELLMANN, M.; TIERNEY, K. ISAC: Instance-specific algorithm configuration. In: COELHO, H.; STUDER, R.; WOOLDRIDGE, M. (Ed.). **Proceedings of the 19th European Conference on Artificial Intelligence**. Amsterdam, The Netherlands: IOS Press, 2010. p. 751–756.

KARAFOTIAS, G.; HOOGENDOORN, M.; EIBEN, A. E. Parameter control in evolutionary algorithms: Trends and challenges. **IEEE Transactions on Evolutionary Computation**, v. 19, n. 2, p. 167–187, 2015.

KARAPETYAN, D.; PARKES, A. J.; STÜTZLE, T. Algorithm configuration: Learning policies for the quick termination of poor performers. In: BATTITI, R.; BRUNATO, M.; KOTSIREAS, I.; PARDALOS, P. M. (Ed.). **Learning and Intelligent Optimization, 12th International Conference, LION 12**. Cham, Switzerland: Springer, 2018, (Lecture Notes in Computer Science, v. 11353). p. 220–224.

KARP, R. M. Reducibility among combinatorial problems. In: MILLER, R. E.; THATCHER, J. W.; BOHLINGER, J. D. (Ed.). **Complexity of Computer Computations**. Boston, MA: Springer, 1972. p. 85–103. ISBN 978-1-4684-2001-2.

KERSCHKE, P.; HOOS, H. H.; NEUMANN, F.; TRAUTMANN, H. Automated algorithm selection: Survey and perspectives. **Evolutionary Computation**, v. 27, n. 1, p. 3–45, mar. 2019. ISSN 1063-6560.

KERSCHKE, P.; KOTTHOFF, L.; BOSSEK, J.; HOOS, H. H.; TRAUTMANN, H. Leveraging TSP solver complementarity through machine learning. **Evolutionary Computation**, MIT Press, Cambridge, MA, v. 26, n. 4, p. 597–620, 2018.

KHUDABUKHSH, A. R.; XU, L.; HOOS, H. H.; LEYTON-BROWN, K. SATenstein: Automatically building local search SAT solvers from components. In: BOUTILIER, C. (Ed.). **Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09)**. Menlo Park, CA: IJCAI/AAAI Press, 2009. p. 517–524.

KHUDABUKHSH, A. R.; XU, L.; HOOS, H. H.; LEYTON-BROWN, K. SATenstein: Automatically building local search SAT solvers from components. **Artificial Intelligence**, v. 232, p. 20–42, 2016.

KIEFER, J. Sequential minimax search for a maximum. **Proceedings of the American Mathematical Society**, JSTOR, v. 4, n. 3, p. 502–506, 1953.

KOCHENBERGER, G. A.; GLOVER, F.; ALIDAEE, B.; REGO, C. A unified modeling and solution framework for combinatorial optimization problems. **OR Spektrum**, v. 26, n. 2, p. 237–250, 2004.

KOCHENBERGER, G. A.; GLOVER, F.; ALIDAEE, B.; LEWIS, K. Using the unconstrained quadratic program to model and solve Max 2-SAT problems. **International Journal of Operational Research**, Inderscience Publishers, v. 1, n. 1-2, p. 89–100, 2005.

KOCHENBERGER, G. A.; HAO, J.-K.; GLOVER, F.; LEWIS, R. M. R.; LÜ, Z.; WANG, H.; WANG, Y. The unconstrained binary quadratic programming problem: A survey. **Journal of Combinatorial Optimization**, v. 28, n. 1, p. 58–81, 2014.

KOCHENBERGER, G. A.; HAO, J.-K.; LÜ, Z.; WANG, H.; GLOVER, F. Solving large scale max cut problems via tabu search. **Journal of Heuristics**, Springer, v. 19, n. 4, p. 565–571, 2013.

KOCHENBERGER, G. A.; LEWIS, M.; GLOVER, F.; WANG, H. Exact solutions to generalized vertex covering problems: A comparison of two models. **Optimization Letters**, Springer, v. 9, n. 7, p. 1331–1339, 2015.

KOZA, J. R. **Genetic Programming: On the Programming of Computers by Means of Natural Selection**. Cambridge, MA: MIT Press, 1992.

KROER, C.; MALITSKY, Y. Feature filtering for instance-specific algorithm configuration. In: KHOSHGOFTAAR, T. M.; ZHU, X. (Ed.). **2011 IEEE 23rd International Conference on Tools with Artificial Intelligence, ICTAI 2011**. Piscataway, NJ: IEEE Press, 2011. p. 849–855.

LESH, N.; MITZENMACHER, M. BubbleSearch: A simple heuristic for improving priority-based greedy algorithms. **Information Processing Letters**, v. 97, n. 4, p. 161–169, 2006. ISSN 0020-0190.

LEWIS, M.; KOCHENBERGER, G. A.; ALIDAEE, B. A new modeling and solution approach for the set-partitioning problem. **Computers & Operations Research**, Elsevier, v. 35, n. 3, p. 807–813, 2008.

LEWIS, R. M. R. **A Guide to Graph Colouring: Algorithms and Applications**. Cham, Switzerland: Springer, 2016.

LEWIS, R. M. R. **Suite of Graph Colouring Algorithms – Supplementary Material to the Book "A Guide to Graph Colouring: Algorithms and Applications"**. 2016. Available from Internet: http://rhydlewis.eu/resources/gCol.zip.

LEYTON-BROWN, K.; NUDELMAN, E.; ANDREW, G.; MCFADDEN, J.; SHOHAM, Y. A portfolio approach to algorithm selection. In: **Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)**. San Francisco, CA: Morgan Kaufmann Publishers, 2003. v. 3, p. 1542–1543.

LEYTON-BROWN, K.; NUDELMAN, E.; SHOHAM, Y. Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In: VAN HENTENRYCK, P. (Ed.). **Principles and Practice of Constraint Programming, CP 2002**. Heidelberg, Germany: Springer, 2002. (Lecture Notes in Computer Science, v. 2470), p. 556–572.

LEYTON-BROWN, K.; PEARSON, M.; SHOHAM, Y. Towards a universal test suite for combinatorial auction algorithms. In: JHINGRAN, A. et al. (Ed.). **ACM Conference on Electronic Commerce (EC-00)**. New York, NY: ACM Press, 2000. p. 66–76.

LIAO, T.; MOLINA, D.; STÜTZLE, T. Performance evaluation of automatically tuned continuous optimizers on different benchmark sets. **Applied Soft Computing**, v. 27, p. 490–503, 2015.

LIAO, T.; MONTES DE OCA, M. A.; STÜTZLE, T. Computational results for an automatically tuned CMA-ES with increasing population size on the CEC'05 benchmark set. **Soft Computing**, v. 17, n. 6, p. 1031–1046, 2013.

LIAO, T.; STÜTZLE, T.; MONTES DE OCA, M. A.; DORIGO, M. A unified ant colony optimization algorithm for continuous optimization. **European Journal of Operational Research**, v. 234, n. 3, p. 597–609, 2014.

LIEFOOGHE, A.; DERBEL, B.; VEREL, S.; AGUIRRE, H.; TANAKA, K. Towards landscape-aware automatic algorithm configuration: Preliminary experiments on neutral and rugged landscapes. In: HU, A. J.; LÓPEZ-IBÁÑEZ, M. (Ed.). **Proceedings of EvoCOP 2017 – 17th European Conference on Evolutionary Computation in Combinatorial Optimization**. Cham, Switzerland: Springer, 2017. (Lecture Notes in Computer Science, v. 10197), p. 215–232.

LIN, S.; KERNIGHAN, B. W. An effective heuristic algorithm for the traveling salesman problem. **Operations Research**, v. 21, n. 2, p. 498–516, 1973.

LINDAUER, M. T.; HOOS, H. H.; HUTTER, F.; SCHAUB, T. AutoFolio: An automatically configured algorithm selector. **Journal of Artificial Intelligence Research**, v. 53, p. 745–778, 2015.

LODI, A.; MARTELLO, S.; VIGO, D. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. **INFORMS Journal on Computing**, INFORMS, v. 11, n. 4, p. 345–357, 1999.

LODI, A.; MARTELLO, S.; VIGO, D. TSpack: A unified tabu search code for multi-dimensional bin packing problems. **Annals of Operations Research**, Springer, v. 131, n. 1-4, p. 203–213, 2004.

LODI, A.; MARTELLO, S.; VIGO, D. **Two- and Three-Dimensional Bin Packing – Source Code of TSpack**. 2004. Available from Internet: http://or.dei.unibo.it/research_pages/ORcodes/TSpack.html.

LÓPEZ-IBÁÑEZ, M.; BLUM, C.; OHLMANN, J. W.; THOMAS, B. W. The travelling salesman problem with time windows: Adapting algorithms from travel-time to makespan optimization. **Applied Soft Computing**, v. 13, n. 9, p. 3806–3815, 2013.

LÓPEZ-IBÁÑEZ, M.; DUBOIS-LACOSTE, J.; PÉREZ CÁCERES, L.; STÜTZLE, T.; BIRATTARI, M. The irace package: Iterated racing for automatic algorithm configuration. **Operations Research Perspectives**, v. 3, p. 43–58, 2016.

LÓPEZ-IBÁÑEZ, M.; DUBOIS-LACOSTE, J.; STÜTZLE, T.; BIRATTARI, M. **The irace Package: Iterated Race for Automatic Algorithm Configuration**. 2011. Available from Internet: https://iridia.ulb.ac.be/IridiaTrSeries/rev/IridiaTr2011-004r001.pdf.

LÓPEZ-IBÁÑEZ, M.; MARMION, M.-E.; STÜTZLE, T. **Automatic Design of Hybrid Metaheuristics from Algorithmic Components**. 2017. Available from Internet: http://iridia.ulb.ac.be/IridiaTrSeries/link/IridiaTr2017-012.pdf.

LÓPEZ-IBÁÑEZ, M.; STÜTZLE, T. Automatic configuration of multi-objective ACO algorithms. In: DORIGO, M. et al. (Ed.). **Ant Colony Optimization and Swarm Intelligence, 7th International Conference, ANTS 2010**. Heidelberg, Germany: Springer, 2010, (Lecture Notes in Computer Science, v. 6234). p. 95–106.

LÓPEZ-IBÁÑEZ, M.; STÜTZLE, T. The automatic design of multi-objective ant colony optimization algorithms. **IEEE Transactions on Evolutionary Computation**, v. 16, n. 6, p. 861–875, 2012.

LÓPEZ-IBÁÑEZ, M.; STÜTZLE, T. Automatically improving the anytime behaviour of optimisation algorithms. **European Journal of Operational Research**, v. 235, n. 3, p. 569–582, 2014.

LÓPEZ-IBÁÑEZ, M.; STÜTZLE, T.; DORIGO, M. Ant colony optimization: A component-wise overview. In: MARTÍ, R.; PARDALOS, P. M.; RESENDE, M. G. C. (Ed.). **Handbook of Heuristics**. New York, NY: Springer International Publishing, 2018. p. 371–407. ISBN 978-3-319-07125-1.

LOURENÇO, H. R.; MARTIN, O. C.; STÜTZLE, T. Iterated local search. Springer, v. 57, p. 321–354, 2003.

LOURENÇO, N.; PEREIRA, F. B.; COSTA, E. Unveiling the properties of structured grammatical evolution. **Genetic Programming and Evolvable Machines**, Springer, v. 17, n. 3, p. 251–289, 2016.

MALITSKY, Y. **Instance-Specific Algorithm Configuration**. Cham, Switzerland: Springer, 2014. 15–24 p. ISBN 978-3-319-11230-5.

MALITSKY, Y.; MEHTA, D.; O'SULLIVAN, B. Evolving instance specific algorithm configuration. In: HELMERT, M.; RÖGER, G. (Ed.). **Proceedings of the Sixth International Symposium on Combinatorial Search**. Menlo Park, CA: AAAI Press, 2013. p. 132–140.

MALITSKY, Y.; MEHTA, D.; O'SULLIVAN, B.; SIMONIS, H. Tuning parameters of large neighborhood search for the machine reassignment problem. In: GOMES, C.; SELLMANN, M. (Ed.). **Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, CPAIOR 2013**. Heidelberg, Germany: Springer, 2013, (Lecture Notes in Computer Science, v. 7874). p. 176–192.

MALITSKY, Y.; SELLMANN, M. Stochastic offline programming. **International Journal on Artificial Intelligence Tools**, Singapore, v. 19, n. 4, p. 351–371, 2010.

MARMION, M.-E.; MASCIA, F.; LÓPEZ-IBÁÑEZ, M.; STÜTZLE, T. Automatic design of hybrid stochastic local search algorithms. In: BLESA, M. J.; BLUM, C.; FESTA, P.; ROLI, A.; SAMPELS, M. (Ed.). **Hybrid Metaheuristics**. Heidelberg, Germany: Springer, 2013, (Lecture Notes in Computer Science, v. 7919). p. 144–158. ISBN 978-3-642-38515-5.

MARTELLO, S.; VIGO, D. Exact solution of the two-dimensional finite bin packing problem. **Management Science**, INFORMS, v. 44, n. 3, p. 388–399, 1998.

MARTÍ, R.; DUARTE, A.; LAGUNA, M. Advanced scatter search for the max-cut problem. **INFORMS Journal on Computing**, INFORMS, v. 21, n. 1, p. 26–38, 2009.

MASCIA, F.; BIRATTARI, M.; STÜTZLE, T. Tuning algorithms for tackling large instances: An experimental protocol. In: PARDALOS, P. M.; NICOSIA, G. (Ed.). **Learning and Intelligent Optimization, 7th International Conference, LION 7**. Cham, Switzerland: Springer, 2013, (Lecture Notes in Computer Science, v. 7997). p. 410–422.

MASCIA, F.; LÓPEZ-IBÁÑEZ, M.; DUBOIS-LACOSTE, J.; STÜTZLE, T. From grammars to parameters: Automatic iterated greedy design for the permutation flow-shop problem with weighted tardiness. In: PARDALOS, P. M.; NICOSIA, G. (Ed.). **Learning and Intelligent Optimization, 7th International Conference, LION 7**. Cham, Switzerland: Springer, 2013, (Lecture Notes in Computer Science, v. 7997). p. 321–334.

MASCIA, F.; LÓPEZ-IBÁÑEZ, M.; DUBOIS-LACOSTE, J.; MARMION, M.-E.; STÜTZLE, T. Algorithm comparison by automatically configurable stochastic local search frameworks: A case study using flow-shop scheduling problems. In: BLESA, M. J.; BLUM, C.; VOSS, S. (Ed.). **Hybrid Metaheuristics**. Heidelberg, Germany: Springer, 2014, (Lecture Notes in Computer Science, v. 8457). p. 30–44. ISBN 978-3-319-07643-0.

MASCIA, F.; LÓPEZ-IBÁÑEZ, M.; DUBOIS-LACOSTE, J.; STÜTZLE, T. Grammar-based generation of stochastic local search heuristics through automatic algorithm configuration tools. **Computers & Operations Research**, v. 51, p. 190–199, 2014.

MASSEN, F.; LÓPEZ-IBÁÑEZ, M.; STÜTZLE, T.; DEVILLE, Y. Experimental analysis of pheromone-based heuristic column generation using irace. In: BLESA, M. J.; BLUM, C.; FESTA, P.; ROLI, A.; SAMPELS, M. (Ed.). **Hybrid Metaheuristics**. Heidelberg, Germany: Springer, 2013, (Lecture Notes in Computer Science, v. 7919). p. 92–106. ISBN 978-3-642-38515-5.

MERSMANN, O.; BISCHL, B.; TRAUTMANN, H.; PREUSS, M.; WEIHS, C.; RUDOLPH, G. Exploratory landscape analysis. In: KRASNOGOR, N.; LANZI, P. L. (Ed.). **Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2011**. New York, NY: ACM Press, 2011. p. 829–836.

MERZ, P. Advanced fitness landscape analysis and the performance of memetic algorithms. **Evolutionary Computation**, v. 12, n. 3, p. 303–325, sep. 2004. ISSN 1063-6560.

MERZ, P.; FREISLEBEN, B. Greedy and local search heuristics for unconstrained binary quadratic programming. **Journal of Heuristics**, v. 8, n. 2, p. 197–213, 2002.

MINTON, S. An analytic learning system for specializing heuristics. In: BAJCSY, R. (Ed.). **Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)**. San Francisco, CA: Morgan Kaufmann Publishers, 1993. v. 93, p. 922–929.

MIRANDA, P.; SILVA, R. M.; PRUDÊNCIO, R. B. Fine-tuning of support vector machine parameters using racing algorithms. In: **European Symposium on Artificial Neural Networks, ESSAN**. Louvain, Belgium: Université Catholique de Louvain, 2014. p. 325–330.

MOCKUS, J. **Bayesian Approach to Global Optimization: Theory and Applications**. 1st. ed. Dordrecht, The Netherlands: Springer, 1989. (Mathematics and Its Applications).

MONTES DE OCA, M. A.; AYDIN, D.; STÜTZLE, T. An incremental particle swarm for large-scale continuous optimization problems: An example of tuning-in-the-loop (re)design of optimization algorithms. **Soft Computing**, v. 15, n. 11, p. 2233–2255, 2011.

MONTGOMERY, D. C. **Design and Analysis of Experiments**. 8th. ed. New York, NY: John Wiley & Sons, 2012.

MONTGOMERY, D. C.; PECK, E. A.; VINING, G. G. **Introduction to Linear Regression Analysis**. 6th. ed. New York, NY: John Wiley & Sons, 2021. (Wiley Series in Probability and Statistics). ISBN 978-1-119-57872-7.

MÜHLENTHALER, M. **Fairness in Academic Course Timetabling**. Cham, Switzerland: Springer International Publishing, 2015.

MUJA, M.; LOWE, D. G. Fast approximate nearest neighbors with automatic algorithm configuration. In: RANCHORDAS, A.; ARAÚJO, H. (Ed.). **Proceedings of the Fourth International Conference on Computer Vision Theory and Applications, VISAPP, Lisbon, Portugal, February 5-8, 2009**. Lisbon, Portugal: INSTICC Press, 2009. v. 1, p. 331–340.

NANNEN, V.; EIBEN, A. E. A method for parameter calibration and relevance estimation in evolutionary algorithms. In: CATTOLICO, M. et al. (Ed.). **Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2006**. New York, NY: ACM Press, 2006. p. 183–190.

NANNEN, V.; EIBEN, A. E. Relevance estimation and value calibration of evolutionary algorithm parameters. In: VELOSO, M. M. (Ed.). **Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)**. Menlo Park, CA: IJCAI/AAAI Press, 2007. p. 975–980.

NELDER, J. A.; MEAD, R. A simplex method for function minimization. **The Computer Journal**, Oxford University Press, v. 7, n. 4, p. 308–313, 1965.

OLSSON, R.; LØKKETANGEN, A. Using automatic programming to generate state-of-the-art algorithms for random 3-SAT. **Journal of Heuristics**, v. 19, n. 5, p. 819–844, 2013.

O'MAHONY, E.; HEBRARD, E.; HOLLAND, A.; NUGENT, C.; O'SULLIVAN, B. Using case-based reasoning in an algorithm portfolio for constraint solving. In: **Proceedings of the 19th Irish Conference on Artificial Intelligence and Cognitive Science, AICS 2008**. Cork, Ireland: University College Cork, Ireland, 2008. p. 210–216.

O'NEILL, M.; RYAN, C. Grammatical evolution. **IEEE Transactions on Evolutionary Computation**, v. 5, n. 4, p. 349–358, 2001.

PAGNOZZI, F.; STÜTZLE, T. Automatic design of hybrid stochastic local search algorithms for permutation flowshop problems. **European Journal of Operational Research**, v. 276, p. 409–421, 2019.

PAGNOZZI, F.; STÜTZLE, T. Automatic design of hybrid stochastic local search algorithms for permutation flowshop problems with additional constraints. **Operations Research Perspectives**, v. 8, 2021.

PALUBECKIS, G. Iterated tabu search for the unconstrained binary quadratic optimization problem. **Informatica**, Vilnius University Institute of Data Science and Digital Technologies, v. 17, n. 2, p. 279–296, 2006.

PARDALOS, P. M.; JHA, S. Complexity of uniqueness and local search in quadratic 0–1 programming. **Operations Research Letters**, v. 11, n. 2, p. 119–123, 1992.

PARDALOS, P. M.; XUE, J. The maximum clique problem. **Journal of Global Optimization**, Springer, v. 4, n. 3, p. 301–328, 1994.

PARK, M.-W.; KIM, Y.-D. A systematic procedure for setting parameters in simulated annealing algorithms. **Computers & Operations Research**, v. 25, n. 3, p. 207–217, 1998.

PARSONS, R.; JOHNSON, M. A case study in experimental design applied to genetic algorithms with applications to DNA sequence assembly. **American Journal of Mathematical and Management Sciences**, Taylor & Francis, v. 17, n. 3-4, p. 369–396, 1997.

PEACE, G. S. **Taguchi Methods: A Hands-On Approach**. Boston, MA: Addison-Wesley, 1993.

PELIKAN, M.; GOLDBERG, D. E.; LOBO, F. G. A survey of optimization by building and using probabilistic models. **Computational Optimization and Applications**, Springer, v. 21, n. 1, p. 5–20, 2002.

PELLEGRINI, P.; BIRATTARI, M. Implementation effort and performance. In: STÜTZLE, T.; BIRATTARI, M.; HOOS, H. H. (Ed.). **Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics. SLS 2007**. Heidelberg, Germany: Springer, 2007, (Lecture Notes in Computer Science, v. 4638). p. 31–45.

PÉREZ CÁCERES, L.; LÓPEZ-IBÁÑEZ, M.; HOOS, H. H.; STÜTZLE, T. An experimental study of adaptive capping in irace. In: BATTITI, R.; KVASOV, D. E.;

SERGEYEV, Y. D. (Ed.). **Learning and Intelligent Optimization, 11th International Conference, LION 11**. Cham, Switzerland: Springer, 2017, (Lecture Notes in Computer Science, v. 10556). p. 235–250.

PÉREZ CÁCERES, L.; LÓPEZ-IBÁÑEZ, M.; HOOS, H. H.; STÜTZLE, T. **An Experimental Study of Adaptive Capping in irace: Supplementary Material**. 2017. Available from Internet: http://iridia.ulb.ac.be/supp/IridiaSupp2016-007.

PÉREZ CÁCERES, L.; LÓPEZ-IBÁÑEZ, M.; STÜTZLE, T. An analysis of parameters of irace. In: BLUM, C.; OCHOA, G. (Ed.). **Proceedings of EvoCOP 2014 – 14th European Conference on Evolutionary Computation in Combinatorial Optimization**. Heidelberg, Germany: Springer, 2014, (Lecture Notes in Computer Science, v. 8600). p. 37–48.

PÉREZ CÁCERES, L.; LÓPEZ-IBÁÑEZ, M.; STÜTZLE, T. Ant colony optimization on a limited budget of evaluations. **Swarm Intelligence**, v. 9, n. 2-3, p. 103–124, 2015.

PÉREZ CÁCERES, L.; PAGNOZZI, F.; FRANZIN, A.; STÜTZLE, T. Automatic configuration of GCC using irace. In: LUTTON, E.; LEGRAND, P.; PARREND, P.; MONMARCHÉ, N.; SCHOENAUER, M. (Ed.). **Artificial Evolution: 13th International Conference, Evolution Artificielle, EA, 2017**. Heidelberg, Germany: Springer, 2017, (Lecture Notes in Computer Science, v. 10764). p. 202–216.

PÉREZ CÁCERES, L.; SOUZA, M. de; LÓPEZ-IBÁÑEZ, M.; RITT, M. Evaluation of random forest importance measures for algorithm configuration. In: BISCHL, B.; LINDAUER, M. T. (Ed.). **Configuration and Selection of Algorithms Workshop (COSEAL 2021)**. COSEAL, 2021. Available from Internet: https://www.coseal.net/coseal-workshop-2021.

PÉREZ CÁCERES, L.; STÜTZLE, T. Exploring variable neighborhood search for automatic algorithm configuration. **Electronic Notes in Discrete Mathematics**, v. 58, p. 167–174, 2017.

POLI, R.; KOZA, J. R. Genetic programming. In: BURKE, E. K.; KENDALL, G. (Ed.). **Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques**. 2. ed. New York, NY: Springer, 2014. p. 143–185.

POLI, R.; LANGDON, W. B.; MCPHEE, N. F. **A Field Guide to Genetic Programming**. United Kingdom: Lulu Enterprises, UK Ltd, 2008. ISBN 978-1-4092-0073-4.

POWELL, M. **The BOBYQA Algorithm for Bound Constrained Optimization without Derivatives**. 2009. Available from Internet: https://www.damtp.cam.ac.uk/user/na/NA_papers/NA2009_06.pdf.

PUSHAK, Y.; HOOS, H. H. Algorithm configuration landscapes: More benign than expected? In: AUGER, A.; FONSECA, C. M.; LOURENÇO, N.; MACHADO, P.; PAQUETE, L.; WHITLEY, D. (Ed.). **Parallel Problem Solving from Nature – PPSN XV**. Cham, Switzerland: Springer, 2018. (Lecture Notes in Computer Science, v. 11101), p. 271–283.

PUSHAK, Y.; HOOS, H. H. Golden parameter search: Exploiting structure to quickly configure parameters in parallel. In: COELLO COELLO, C. A. (Ed.). **Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2020**. New York, NY: ACM Press, 2020. p. 245–253.

RASMUSSEN, C. E. Gaussian processes in machine learning. In: BOUSQUET, O.; VON LUXBURG, U.; RÄTSCH, G. (Ed.). **Advanced Lectures on Machine Learning**. Berlin, Heidelberg: Springer, 2004, (Lecture Notes in Computer Science, v. 3176). p. 63–71. ISBN 978-3-540-28650-9.

REEVES, C. Landscapes, operators and heuristic search. **Annals of Operations Research**, v. 86, p. 473–490, 1999.

RICE, J. R. The algorithm selection problem. **Advances in Computers**, v. 15, p. 65–118, 1976.

RIDGE, E.; KUDENKO, D. Sequential experiment designs for screening and tuning parameters of stochastic heuristics. In: PAQUETE, L.; CHIARANDINI, M.; BASSO, D. (Ed.). **Empirical Methods for the Analysis of Algorithms, Workshop EMAA 2006, Proceedings**. University of Southern Denmark, 2006. p. 27–34. Available from Internet: https://imada.sdu.dk/~marco/EMAA/Proceedings.html.

RIDGE, E.; KUDENKO, D. Tuning the performance of the MMAS heuristic. In: STÜTZLE, T.; BIRATTARI, M.; HOOS, H. H. (Ed.). **Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics. SLS 2007**. Heidelberg, Germany: Springer, 2007, (Lecture Notes in Computer Science, v. 4638). p. 46–60.

RIFF, M.-C.; MONTERO, E. A new algorithm for reducing metaheuristic design effort. In: **Proceedings of the 2013 Congress on Evolutionary Computation (CEC 2013)**. Piscataway, NJ: IEEE Press, 2013. p. 3283–3290.

ROTHLAUF, F.; OETZEL, M. On the locality of grammatical evolution. In: COLLET, P.; TOMASSINI, M.; EBNER, M.; GUSTAFSON, S.; EKÁRT, A. (Ed.). **Proceedings of the 9th European Conference on Genetic Programming, EuroGP 2006**. Heidelberg, Germany: Springer, 2006. (Lecture Notes in Computer Science, v. 3905), p. 320–330.

RYAN, C.; COLLINS, J.; NEILL, M. O. Grammatical evolution: Evolving programs for an arbitrary language. In: BANZHAF, W.; POLI, R.; SCHOENAUER, M.; FOGARTY, T. C. (Ed.). **Proceedings of the 1st European Conference on Genetic Programming, EuroGP 1998**. Heidelberg, Germany: Springer, 1998. (Lecture Notes in Computer Science, v. 1391), p. 83–96.

RYAN, C.; O'NEILL, M.; COLLINS, J. J. (Ed.). **Handbook of Grammatical Evolution**. Cham, Switzerland: Springer International Publishing, 2018. ISBN 978-3-319-78717-6.

SCHIAVINOTTO, T.; STÜTZLE, T. A review of metrics on permutations for search landscape analysis. **Computers & Operations Research**, v. 34, n. 10, p. 3143–3153, 2007. ISSN 0305-0548.

SCHONLAU, M.; WELCH, W. J.; JONES, D. R. Global versus local search in constrained optimization of computer models. **Lecture Notes-Monograph Series**, Institute of Mathematical Statistics, v. 34, p. 11–25, 1998.

SHAHRIARI, B.; SWERSKY, K.; WANG, Z.; ADAMS, R. P.; FREITAS, N. Taking the human out of the loop: A review of bayesian optimization. **Proceedings of the IEEE**, v. 104, n. 1, p. 148–175, 2016.

SMIT, S. K.; EIBEN, A. E. Comparing parameter tuning methods for evolutionary algorithms. In: **Proceedings of the 2009 Congress on Evolutionary Computation (CEC 2009)**. Piscataway, NJ: IEEE Press, 2009. p. 399–406.

SMIT, S. K.; EIBEN, A. E. Multi-problem parameter tuning using BONESA. In: HAO, J.-K.; LEGRAND, P.; COLLET, P.; MONMARCHÉ, N.; LUTTON, E.; SCHOENAUER, M. (Ed.). **Artificial Evolution: 10th International Conference, Evolution Artificielle, EA, 2011**. Heidelberg, Germany: Springer, 2011. (Lecture Notes in Computer Science, v. 7401), p. 222–233.

SMITH, J. Modelling gas with self-adaptive mutation rates. Morgan Kaufmann Publishers, San Francisco, CA, p. 599–606, 2001.

SMITH-MILES, K.; BAATAR, D.; WREFORD, B.; LEWIS, R. M. R. Towards objective measures of algorithm performance across instance space. **Computers & Operations Research**, v. 45, p. 12–24, 2014. ISSN 0305-0548.

SOUZA, M. de. On the automatic configuration of algorithms. In: CORTÉS, U.; ALONSO, J. M. (Ed.). **Proceedings of the 1st Doctoral Consortium at the European Conference on Artificial Intelligence (DC-ECAI 2020)**. Santiago de Compostela, Spain: Universidade de Santiago de Compostela, 2020.

SOUZA, M. de. Automatic design of heuristic algorithms for binary optimization problems. In: ZHOU, Z.-H. (Ed.). **Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI-21)**. Freiburg, Germany: IJCAI, 2021. p. 4881–4882.

SOUZA, M. de. How to speed-up the automated configuration of optimization algorithms. In: **Proceedings of the Fourteenth International Symposium on Combinatorial Search, SoCS 2021**. Menlo Park, CA: AAAI Press, 2021. v. 12, n. 1, p. 216–218.

SOUZA, M. de; RITT, M. A study of the automatic design of heuristics for binary quadratic programming. In: OCHI, L. S.; MARTINS, S. (Ed.). **Proceedings of the XLVIII Brazilian Symposium on Operational Research – SBPO 2016**. Rio de Janeiro, RJ, Brazil: SOBRAPO, 2016. p. 1673–1684.

SOUZA, M. de; RITT, M. **AutoBQP: A Component-Wise Solver to Binary Optimization – Source Code**. 2018. Available from Internet: https://github.com/souzamarcelo/AutoBQP.

SOUZA, M. de; RITT, M. Automatic grammar-based design of heuristic algorithms for unconstrained binary quadratic programming. In: LIEFOOGHE, A.; LÓPEZ-IBÁÑEZ, M. (Ed.). **Proceedings of EvoCOP 2018 – 18th European**

Conference on Evolutionary Computation in Combinatorial Optimization. Heidelberg, Germany: Springer, 2018, (Lecture Notes in Computer Science, v. 10782). p. 67–84.

SOUZA, M. de; RITT, M. An automatically designed recombination heuristic for the test-assignment problem. In: **Proceedings of the 2018 Congress on Evolutionary Computation (CEC 2018)**. Piscataway, NJ: IEEE Press, 2018. p. 2411–2418.

SOUZA, M. de; RITT, M. **HHBQP: Hybrid Heuristic for Unconstrained Binary Quadratic Programming – Source Code**. 2018. Available from Internet: https://github.com/souzamarcelo/hhbqp.

SOUZA, M. de; RITT, M. **HHMC: Hybrid Heuristic for the Maximum Cut on Graphs – Source Code**. 2018. Available from Internet: https://github.com/souzamarcelo/hhmc.

SOUZA, M. de; RITT, M. **HHPAL: Hybrid Heuristic for Unconstrained Binary Quadratic Programming Trained on PalubeckisÍnstances – Source Code**. 2018. Available from Internet: https://github.com/souzamarcelo/hhpal.

SOUZA, M. de; RITT, M. **HHTA: Hybrid Heuristic for the Test-Assignment Problem – Source Code**. 2018. Available from Internet: https://github.com/souzamarcelo/hhta.

SOUZA, M. de; RITT, M. Capping strategies based on performance envelopes for the automatic design of meta-heuristics. In: DURÁN, G.; CARRIZOSA, E.; PLÀ-ARAGONÉS, L. M. (Ed.). **XXIII Latin-American Summer School in Operational Research (ELAVIO 2019)**. Lleida, Spain: ALIO/IFORS, 2019.

SOUZA, M. de; RITT, M. Capping strategies based on performance envelopes for the automatic design of meta-heuristics. In: HOOS, H. H.; LINDAUER, M. T. (Ed.). **Configuration and Selection of Algorithms Workshop (COSEAL 2019)**. COSEAL, 2019. Available from Internet: https://www.coseal.net/coseal-workshop-2019.

SOUZA, M. de; RITT, M. **Algorithm Configuration Scenarios**. 2022. Available from Internet: https://github.com/souzamarcelo/ac-scenarios.

SOUZA, M. de; RITT, M. **ILSBQP: A Simple Iterated Local Search for the Unconstrained Binary Quadratic Programming – Source Code**. 2022. Available from Internet: https://github.com/souzamarcelo/ilsbqp.

SOUZA, M. de; RITT, M. **Improved Regression Models for Algorithm Configuration**. 2022. Article under review, title might change.

SOUZA, M. de; RITT, M. **Improved Regression Models for Algorithm Configuration – Source Code**. 2022. Available from Internet: https://github.com/souzamarcelo/regression-models-ac.

SOUZA, M. de; RITT, M. **Improved Regression Models for Algorithm Configuration – Supplementary Material**. 2022. Available from Internet: https://github.com/souzamarcelo/supp-models-ac.

SOUZA, M. de; RITT, M.; LÓPEZ-IBÁÑEZ, M. **CAPOPT: Capping Methods for the Automatic Configuration of Optimization Algorithms – Source Code**. 2020. Available from Internet: https://github.com/souzamarcelo/capopt.

SOUZA, M. de; RITT, M.; LÓPEZ-IBÁÑEZ, M. **Capping Methods for the Automatic Configuration of Optimization Algorithms – Supplementary Material**. 2021. Available from Internet: https://github.com/souzamarcelo/supp-cor-capopt.

SOUZA, M. de; RITT, M.; LÓPEZ-IBÁÑEZ, M. Capping methods for the automatic configuration of optimization algorithms. **Computers & Operations Research**, v. 139, p. 1–15, 2022. ISSN 0305-0548.

SOUZA, M. de; RITT, M.; LÓPEZ-IBÁÑEZ, M.; PÉREZ CÁCERES, L. **ACVIZ: A Tool for the Visual Analysis of the Configuration of Algorithms with irace – Source Code**. 2020. Available from Internet: https://github.com/souzamarcelo/acviz.

SOUZA, M. de; RITT, M.; LÓPEZ-IBÁÑEZ, M.; PÉREZ CÁCERES, L. **ACVIZ: Algorithm Configuration Visualizations for irace – Supplementary Material**. Zenodo, 2020. Available from Internet: https://doi.org/10.5281/zenodo.4028906.

SOUZA, M. de; RITT, M.; LÓPEZ-IBÁÑEZ, M.; PÉREZ CÁCERES, L. ACVIZ: A tool for the visual analysis of the configuration of algorithms with irace. **Operations Research Perspectives**, v. 8, p. 1–9, 2021. ISSN 2214-7160.

SPECK, D.; BIEDENKAPP, A.; HUTTER, F.; MATTMÜLLER, R.; LINDAUER, M. T. Learning heuristic selection with dynamic algorithm configuration. In: BIUNDO, S. et al. (Ed.). **Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling, ICAPS 2021**. Palo Alto, CA: AAAI Press, 2021. v. 31, p. 597–605.

STÜTZLE, T. **ACOTSP: A Software Package of Various Ant Colony Optimization Algorithms Applied to the Symmetric Traveling Salesman Problem**. 2002. Available from Internet: http://www.aco-metaheuristic.org/aco-code.

STÜTZLE, T. Some thoughts on engineering stochastic local search algorithms. In: VIANA, A. et al. (Ed.). **Proceedings of the EU/MEeting 2009. Debating the Future: New Areas of Application and Innovative Approaches**. European Chapter on Metaheuristics, 2009. p. 47–52. Available from Internet: https://www.dcc.fc.up.pt/eume2009/pdf/proceedings.pdf.

STÜTZLE, T.; LÓPEZ-IBÁÑEZ, M. Automated design of metaheuristic algorithms. In: GENDREAU, M.; POTVIN, J.-Y. (Ed.). **Handbook of Metaheuristics**. New York, NY: Springer International Publishing, 2019, (International Series in Operations Research & Management Science, v. 272). p. 541–579.

STÜTZLE, T.; LÓPEZ-IBÁÑEZ, M.; PELLEGRINI, P.; MAUR, M.; MONTES DE OCA, M. A.; BIRATTARI, M.; DORIGO, M. Parameter adaptation in ant colony optimization. In: HAMADI, Y.; MONFROY, E.; SAUBION, F. (Ed.). **Autonomous Search**. Berlin, Germany: Springer, 2012. p. 191–215.

STYLES, J.; HOOS, H. H. Ordered racing protocols for automatically configuring algorithms for scaling performance. In: BLUM, C.; ALBA, E. (Ed.). **Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2013**. New York, NY: ACM Press, 2013. p. 551–558. ISBN 978-1-4503-1963-8.

STYLES, J.; HOOS, H. H.; MÜLLER, M. Automatically configuring algorithms for scaling performance. In: HAMADI, Y.; SCHOENAUER, M. (Ed.). **Learning and Intelligent Optimization, 6th International Conference, LION 6**. Heidelberg, Germany: Springer, 2012. (Lecture Notes in Computer Science, v. 7219), p. 205–219.

TAILLARD, É. Benchmarks for basic scheduling problems. **European Journal of Operational Research**, v. 64, n. 2, p. 278–285, 1993. ISSN 0377-2217.

TAVARES, J.; PEREIRA, F. B. Automatic design of ant algorithms with grammatical evolution. In: MORAGLIO, A.; SILVA, S.; KRAWIEC, K.; MACHADO, P.; COTTA, C. (Ed.). **Proceedings of the 15th European Conference on Genetic Programming, EuroGP 2012**. Heidelberg, Germany: Springer, 2012, (Lecture Notes in Computer Science, v. 7244). p. 206–217.

TERASHIMA-MARÍN, H.; ROSS, P.; VALENZUELA-RENDÓN, M. Evolution of constraint satisfaction strategies in examination timetabling. In: BANZHAF, W.; DAIDA, J. M.; EIBEN, A. E.; GARZON, M. H.; HONAVAR, V.; JAKIELA, M. J.; SMITH, R. E. (Ed.). **Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 1999**. San Francisco, CA: Morgan Kaufmann Publishers, 1999. p. 635–642.

TOLSON, B. A.; SHOEMAKER, C. A. Dynamically dimensioned search algorithm for computationally efficient watershed model calibration. **Water Resources Research**, Wiley Online Library, v. 43, n. 1, 2007.

TRICK, M. A. **Graph Coloring Instances**. 2018. Available from Internet: https://mat.gsia.cmu.edu/COLOR/instances.html.

TRIOLA, M. F. **Essentials of Statistics**. 6th. ed. Boston, MA: Pearson Addison Wesley, 2019.

VALLATI, M.; FAWCETT, C.; GEREVINI, A. E.; HOOS, H. H.; SAETTI, A. Automatic generation of efficient domain-optimized planners from generic parametrized planners. In: HELMERT, M.; RÖGER, G. (Ed.). **Proceedings of the Sixth International Symposium on Combinatorial Search**. Menlo Park, CA: AAAI Press, 2013.

VILLALÓN, C. L. C.; DORIGO, M.; STÜTZLE, T. PSO-X: A component-based framework for the automatic design of particle swarm optimization algorithms. **IEEE Transactions on Evolutionary Computation**, IEEE Press, p. 1–1, 2021.

WANG, F.; XU, Z. Metaheuristics for robust graph coloring. **Journal of Heuristics**, Springer, v. 19, n. 4, p. 529–548, 2013.

WANG, Y.; LÜ, Z.; GLOVER, F.; HAO, J.-K. Path relinking for unconstrained binary quadratic programming. **European Journal of Operational Research**, Elsevier, v. 223, n. 3, p. 595–604, 2012.

WATSON, J.-P. An introduction to fitness landscape analysis and cost models for local search. In: GENDREAU, M.; POTVIN, J.-Y. (Ed.). **Handbook of Metaheuristics**. 2. ed. Boston, MA: Springer, 2010, (International Series in Operations Research & Management Science, v. 146). p. 599–623.

WIEGELE, A. **Biq Mac Library – A Collection of Max-Cut and Quadratic 0-1 Programming Instances of Medium Size**. Klagenfurt, Austria, 2007. Available from Internet: http://biqmac.aau.at/biqmaclib.pdf.

WIEGELE, A. **Biq Mac Library – Binary Quadratic and Max Cut Library**. 2007. Available from Internet: http://biqmac.aau.at/biqmaclib.html.

WOLPERT, D. H.; MACREADY, W. G. No free lunch theorems for optimization. **IEEE Transactions on Evolutionary Computation**, v. 1, n. 1, p. 67–82, 1997.

XU, L.; HOOS, H. H.; LEYTON-BROWN, K. Hydra: Automatically configuring algorithms for portfolio-based selection. In: FOX, M.; POOLE, D. (Ed.). **Proceedings of AAAI 2010 – Twenty-Fourth National Conference on Artificial Intelligence**. Menlo Park, CA: AAAI Press/MIT Press, 2010. p. 210–216.

XU, L.; HUTTER, F.; HOOS, H. H.; LEYTON-BROWN, K. SATzilla: Portfolio-based algorithm selection for SAT. **Journal of Artificial Intelligence Research**, v. 32, p. 565–606, jun. 2008.

YARIMCAM, A.; ASTA, S.; ÖZCAN, E.; PARKES, A. J. Heuristic generation via parameter tuning for online bin packing. In: ANGELOV, P. et al. (Ed.). **Evolving and Autonomous Learning Systems (EALS), 2014 IEEE Symposium on**. New York, NY: IEEE Press, 2014. p. 102–108.

YUAN, Z.; MONTES DE OCA, M. A.; STÜTZLE, T.; BIRATTARI, M. Continuous optimization algorithms for tuning real and integer algorithm parameters of swarm intelligence algorithms. **Swarm Intelligence**, v. 6, n. 1, p. 49–75, 2012.

ZHOU, Z.-H.; HAO, J.-K.; GOËFFON, A. A three-phased local search approach for the clique partitioning problem. **Journal of Combinatorial Optimization**, Springer, v. 32, p. 169–491, 2016.

ZHOU, Z.-H.; HAO, J.-K.; GOËFFON, A. **A Three-Phased Local Search Approach for the Clique Partitioning Problem – Supplementary Material**. 2016. Available from Internet: https://leria-info.univ-angers.fr/~jinkao.hao/cpp.html.

ZILBERSTEIN, S. Using anytime algorithms in intelligent systems. **AI Magazine**, v. 17, n. 3, p. 73–83, 1996.

ZUBARAN, T.; RITT, M. A simple, adaptive bubble search for improving heuristic solutions of the permutation flow shop scheduling problem. In: FERREIRA, L.; ALMEIDA, M. R. (Ed.). **Proceedings of the XLV Brazilian Symposium on Operational Research – SBPO 2013**. Rio de Janeiro, RJ, Brazil: SOBRAPO, 2013. p. 1847–1856.

Part V

Appendices

APPENDIX A — ARTIFACTS OF THIS RESEARCH

In this appendix, we present the artifacts arising from our research. We enumerate the scientific publications, talks and presentations in workshops, doctoral consortia and Ph.D. schools, software implementations of the proposed methods, and supplementary data for most of our experiments.

A.1 Contributions to Literature

The work herein described has appeared in a number of articles listed below. The results presented in Chapter 4 regarding the capping methods for optimization scenarios were published in *Computers & Operations Research* (1). A research article about the proposed regression models for algorithm configuration, described in Chapter 5, is under review and should be published soon (2). The visualization tool described in Chapter 6 was published in *Operations Research Perspectives* (3). The component-wise solver to binary optimization, described and evaluated in Chapters 7 and 8, respectively, originated the following publications in selected conferences: an initial experimental study on applying automatic design of algorithms for the unconstrained binary quadratic programming (4); a detailed description of the methodology used in the solver and its experimental evaluation on the UBQP and maximum cut problems (5); and the evaluation of the solver on the test-assignment problem (6).[1]

1. Marcelo de Souza, Marcus Ritt and Manuel López-Ibáñez. **Capping methods for the automatic configuration of optimization algorithms**. Computers & Operations Research, v. 139, p. 1–15, 2022.
   [Souza, Ritt and López-Ibáñez (2022); DOI: 10.1016/j.cor.2021.105615]

2. Marcelo de Souza and Marcus Ritt. **Improved regression models for algorithm configuration**. *Article under review, title might change*, 2022.
   [Souza and Ritt (2022c)]

[1] As presented in the publications list, the capping methods for optimization scenarios and the visualization tool were developed in collaboration with Dr. Manuel López-Ibáñez. Dr. Leslie Pérez Cáceres also collaborated in the development of the visualization tool.

3. Marcelo de Souza, Marcus Ritt, Manuel López-Ibáñez and Leslie Pérez Cáceres. **ACVIZ: A tool for the visual analysis of the configuration of algorithms with irace**. Operations Research Perspectives, v. 8, p. 1–9, 2021.
[Souza et al. (2021); DOI: 10.1016/j.orp.2021.100186]

4. Marcelo de Souza and Marcus Ritt. **A study of the automatic design of heuristics for binary quadratic programming**. XLVIII Brazilian Symposium on Operational Research (SBPO), p. 1673–1684, 2016.
[Souza and Ritt (2016)]

5. Marcelo de Souza and Marcus Ritt. **Automatic grammar-based design of heuristic algorithms for unconstrained binary quadratic programming**. 18th European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP), p. 67–84, 2018.
[Souza and Ritt (2018b); DOI: 10.1007/978-3-319-77449-7_5]

6. Marcelo de Souza and Marcus Ritt. **An automatically designed recombination heuristic for the test-assignment problem**. 2018 IEEE Congress on Evolutionary Computation (CEC), p. 2411–2418, 2018.
[Souza and Ritt (2018c); DOI: 10.1109/CEC.2018.8477801]

A.2 Talks and Presentations

In addition, this research was selected for presentation in several scientific events, like workshops, doctoral consortia and Ph.D. schools. The corresponding papers appear in the conferences proceedings and are enumerated below.

1. Marcelo de Souza and Marcus Ritt. **Capping strategies based on performance envelopes for the automatic design of meta-heuristics**. XXIII Latin-American Summer School in Operational Research (ELAVIO), 2019.
[Souza and Ritt (2019a)]

2. Marcelo de Souza and Marcus Ritt. **Capping strategies based on performance envelopes for the automatic design of meta-heuristics**. Configuration and Selection of Algorithms Workshop (COSEAL), 2019.
[Souza and Ritt (2019b)]

3. Marcelo de Souza. **On the automatic configuration of algorithms**. 1st Doctoral Consortium at the European Conference on Artificial Intelligence (DC-ECAI), p. 15–16, 2020.
   [Souza (2020)]

4. Marcelo de Souza. **How to speed-up the automated configuration of optimization algorithms**. Doctoral Consortium of the 14th Annual Symposium on Combinatorial Search (SoCS), AAAI, 2021.
   [Souza (2021b)]

5. Marcelo de Souza. **Automatic design of heuristic algorithms for binary optimization problems**. Doctoral Consortium of the 30th International Joint Conference on Artificial Intelligence (IJCAI), 2021.
   [Souza (2021a); DOI: 10.24963/ijcai.2021/672]

6. Leslie Pérez Cáceres, Marcelo de Souza, Manuel López-Ibáñez and Marcus Ritt. **Evaluation of random forest importance measures for algorithm configuration**. Configuration and Selection of Algorithms Workshop (COSEAL), 2021.[2]
   [Pérez Cáceres et al. (2021)]

A.3 Software

We provide open-source implementations for all methods and algorithms proposed in this work, together with instructions and examples of use. The capping methods of Chapter 4 are implemented in the capopt program, which can be used within irace to speed-up the configuration process (1). A set of utility functions to use the regression models proposed in Chapter 5 is provided in (2). The ILSBQP algorithm we use as a test case for those models is available in (3). The acviz visualization tool described in Chapter 6 is available in (4). The component-wise solver described and evaluated in Chapters 7 and 8 was dubbed AutoBQP and is available in (5). It can be used either to produce algorithms automatically for a given binary problem, or to reproduce the algorithms presented in the experimental

[2]This publication presents a preliminary study on using random forest models to estimate parameter importance in algorithm configuration scenarios. Since it is an ongoing research, it was left out of this thesis.

evaluation of Chapter 8. To ease access and reproducibility, we also make the implementations of these algorithms available separately: HHPAL (6), HHMC (7), HHBQP (8) and HHTA (9).

1. Marcelo de Souza, Marcus Ritt and Manuel López-Ibáñez. **CAPOPT: Capping methods for the automatic configuration of optimization algorithms – source code**, 2020.
   [Souza, Ritt and López-Ibáñez (2020)]
   [URL: https://github.com/souzamarcelo/capopt]

2. Marcelo de Souza and Marcus Ritt. **Improved regression models for algorithm configuration – source code**, 2022.
   [Souza and Ritt (2022d)]
   [URL: https://github.com/souzamarcelo/regression-models-ac]

3. Marcelo de Souza and Marcus Ritt. **ILSBQP: A simple iterated local search for unconstrained binary quadratic programming – source code**, 2022.
   [Souza and Ritt (2022b); URL: https://github.com/souzamarcelo/ilsbqp]

4. Marcelo de Souza, Marcus Ritt, Manuel López-Ibáñez and Leslie Pérez Cáceres. **ACVIZ: A tool for the visual analysis of the configuration of algorithms with irace – source code**, 2020.
   [Souza et al. (2020a); URL: https://github.com/souzamarcelo/acviz]

5. Marcelo de Souza and Marcus Ritt. **AutoBQP: A component-wise solver to binary optimization – source code**, 2018.
   [Souza and Ritt (2018a); URL: https://github.com/souzamarcelo/autobqp]

6. Marcelo de Souza and Marcus Ritt. **HHPAL: Hybrid heuristic for unconstrained binary quadratic programming trained on Palubeckis' instances – source code**, 2018.
   [Souza and Ritt (2018f); URL: https://github.com/souzamarcelo/hhpal]

7. Marcelo de Souza and Marcus Ritt. **HHMC: Hybrid heuristic for the maximum cut on graphs – source code**, 2018.
   [Souza and Ritt (2018e); URL: https://github.com/souzamarcelo/hhmc]

8. Marcelo de Souza and Marcus Ritt. **HHBQP: Hybrid heuristic for unconstrained binary quadratic programming – source code**, 2018.
   [Souza and Ritt (2018d); URL: https://github.com/souzamarcelo/hhbqp]

9. Marcelo de Souza and Marcus Ritt. **HHTA: Hybrid heuristics for the test-assignment problem – source code**, 2018.
   [Souza and Ritt (2018g); URL: https://github.com/souzamarcelo/hhta]

A.4 Supplementary Material

Finally, we provide supplementary material for Chapters 4, 5 and 6 (items 1, 2 and 3 below, respectively). We include the source code of the tested algorithms, all data, instructions and scripts to reproduce the experiments, and additional details for the results. We also provide all files for the configuration scenarios we used throughout this thesis (4), including algorithm implementations, execution scripts, problem instances, parameter descriptions and the configurator setup.

1. Marcelo de Souza, Marcus Ritt and Manuel López-Ibáñez. **Capping methods for the automatic configuration of optimization algorithms – supplementary material**, 2021.
   [Souza, Ritt and López-Ibáñez (2021)]
   [URL: https://github.com/souzamarcelo/supp-cor-capopt]

2. Marcelo de Souza and Marcus Ritt. **Improved regression models for algorithm configuration – supplementary material**, 2022.
   [Souza and Ritt (2022e); URL: https://github.com/souzamarcelo/supp-models-ac]

3. Marcelo de Souza, Marcus Ritt, Manuel López-Ibáñez and Leslie Pérez Cáceres. **ACVIZ: Algorithm configuration visualizations for irace – supplementary material**, Zenodo, 2020.
   [Souza et al. (2020b); DOI: 10.5281/zenodo.4028904]

4. Marcelo de Souza and Marcus Ritt. **Algorithm configuration scenarios**, 2022.
   [Souza and Ritt (2022a); URL: https://github.com/souzamarcelo/ac-scenarios]

APPENDIX B — RESUMO EXPANDIDO

CONFIGURAÇÃO AUTOMÁTICA DE ALGORITMOS: MÉTODOS E APLICAÇÕES

Muitos algoritmos possuem parâmetros de entrada, que permitem aos usuários adaptar seu comportamento a diferentes domínios de problema. O conjunto de valores para os parâmetros de um algoritmo, chamado de *configuração*, está diretamente relacionado ao seu desempenho. Neste contexto, dado um conjunto de instâncias de treinamento, a *configuração de algoritmos* busca pelas melhores configurações, i.e. aquelas que otimizam o desempenho do algoritmo nessas instâncias. Esse processo envolve testar várias configurações em diferentes instâncias, tornando-o custoso e exigindo um esforço considerável do pesquisador.

Diante desse contexto, diversas abordagens foram propostas nos últimos anos com o objetivo de automatizar o processo de configuração de algoritmos. Dentre elas se destacam configuradores baseados em busca heurística (HUTTER et al., 2009b; ANSÓTEGUI; SELLMANN; TIERNEY, 2009), otimização baseada em modelos (HUTTER; HOOS; LEYTON-BROWN, 2011) e algoritmos de *racing* (LÓPEZ-IBÁÑEZ et al., 2016). Além de reduzir o esforço humano envolvido no processo de configuração, esses métodos minimizam potenciais vieses de abordagens manuais, facilitam a adaptação de algoritmos a diferentes problemas, e permitem avaliar e comparar algoritmos de forma mais adequada. Além disso, esses métodos se mostraram úteis na melhoria do desempenho de algoritmos em diferentes domínios, incluindo *solvers* matemáticos (HUTTER; HOOS; LEYTON-BROWN, 2010; LÓPEZ-IBÁÑEZ; STÜTZLE, 2014), compiladores (PÉREZ CÁCERES et al., 2017), algoritmos de decisão (HOOS; HUTTER; LEYTON-BROWN, 2021) e algoritmos de otimização (YARIMCAM et al., 2014; PÉREZ CÁCERES; LÓPEZ-IBÁÑEZ; STÜTZLE, 2015).

Este trabalho apresenta um estudo abrangente sobre configuração automática de algoritmos, cujas contribuições se baseiam em quatro objetivos: (i) aumentar a eficiência da configuração automática de algoritmos de otimização; (ii) melhorar a qualidade das configurações produzidas; (iii) facilitar a análise e o entendimento do processo de configuração; e (iv) aplicar os métodos de configuração automática

na construção de *solvers* heurísticos para classes de problemas. As seções a seguir apresentam os métodos e ferramentas propostos neste trabalho e os principais resultados alcançados.

B.1 Métodos de Poda para Cenários de Otimização

Ao configurar algoritmos visando a minimização do seu tempo de execução, a maioria dos configuradores da literatura implementam métodos de poda, os quais visam evitar o gasto de tempo na avaliação de configurações não promissoras (HUTTER et al., 2009b; HUTTER; HOOS; LEYTON-BROWN, 2011; PÉREZ CÁCERES et al., 2017a; PUSHAK; HOOS, 2020). Esses métodos determinam um limite de tempo de execução com base em execuções anteriores da melhor configuração encontrada até o momento, e então terminam antecipadamente execuções que excedam esse limite. Essas abordagens não são adequadas para a configuração de algoritmos de otimização, uma vez que nesse caso o objetivo consiste em minimizar o custo da melhor solução encontrada após executar o algoritmo com um critério de terminação predeterminado. Os configuradores atuais, portanto, não implementam nenhum método de poda para cenários de otimização, o que torna o processo de configuração desses algoritmos mais custoso e em alguns casos inviável.

Este trabalho preenche essa lacuna, propondo um conjunto de métodos de poda capaz de reduzir o tempo de configuração de algoritmos de otimização. A Figura B.1 apresenta a ideia geral desses métodos. Cada execução é representada pelo seu perfil de desempenho, i.e. o custo da melhor solução encontrada ao longo do tempo de execução (ou ao longo de qualquer outra medida de esforço computacional, como o número de iterações). Antes de iniciar uma nova execução, os desempenhos observados em execuções anteriores são agregados em um envelope de desempenho, o qual é usado para avaliar a qualidade da nova execução. Caso ela não satisfaça o desempenho mínimo definido pelo envelope, a execução é interrompida antecipadamente. São propostas duas abordagens gerais para a construção dos envelopes de desempenho. As abordagens baseadas em perfil (ilustradas na parte superior direita da Figura B.1) definem esse envelope como um perfil de desempenho agregado. Já as abordagens baseadas em área (ilustradas na parte inferior direita da Figura B.1) consideram a área usada pelos perfis de desempenho das execuções anteriores, agregando-as em um valor máximo de área. Se em algum momento a nova execução violar as condições

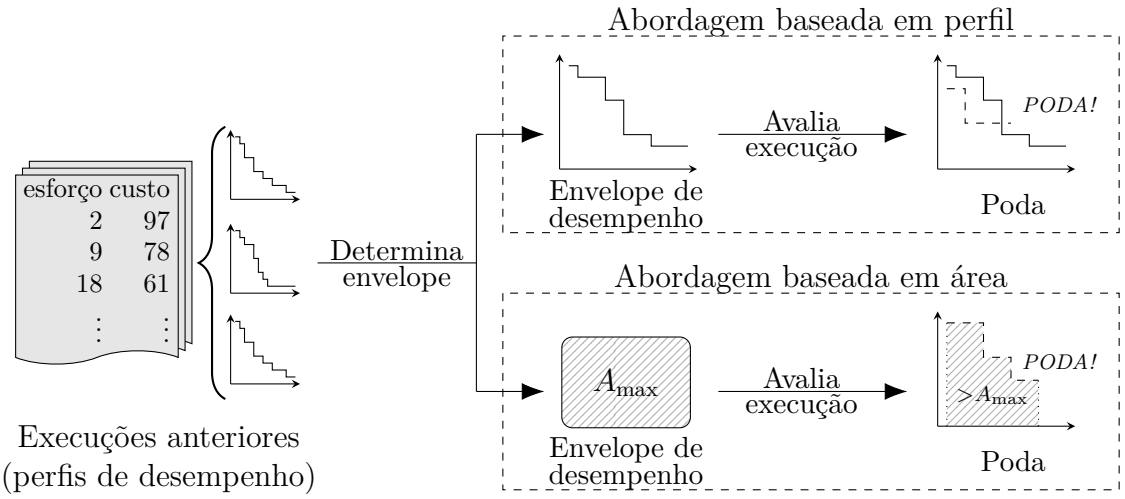


Figura B.1 – Visão geral dos métodos de poda.

definidas pelo envelope, i.e. apresentar soluções piores que o esperado ou uma área maior que o limite por ele definido, a execução é interrompida.

Foram propostas diferentes funções de agregação, tanto para perfis de desempenho quanto para os valores correspondentes de área, dando origem a 18 métodos de poda distintos. Esses métodos foram avaliados em seis cenários de otimização, comparando-os com a configuração sem uso de poda. Ao usar métodos de poda, observou-se uma redução do esforço de configuração entre 5% e 78%, ao mesmo tempo que mantida a qualidade das configurações produzidas. Também foi avaliada a contribuição dos métodos de poda em cenários onde o esforço de configuração é definido em termos do tempo máximo de execução. Nestes casos, o tempo economizado ao podar execuções não promissoras é usado para uma exploração mais abrangente do espaço de configurações. Como consequência, o uso de métodos de poda permitiu encontrar configurações de maior qualidade em comparação àquelas obtidas nas mesmas condições sem o uso de poda. O Capítulo 4 apresenta em detalhes os métodos de poda propostos, bem como sua avaliação experimental completa.

B.2 Modelos Melhorados para Regressão de Parâmetros

Uma característica dos métodos de configuração estudados neste trabalho é que eles determinam configurações *fixas* para todo o conjunto de instâncias de treinamento. Ou seja, os parâmetros assumem valores constantes na solução de diferentes instâncias, não importa as variadas características que elas apresentem. É conhecido, no

entanto, que bons valores de parâmetros podem variar em função de características das instâncias (MUJA; LOWE, 2009; SMITH-MILES et al., 2014; EL YAFRANI; AHIOD, 2018), incluindo características específicas do problema, e.g. estatísticas sobre cláusulas e variáveis em problemas de satisfatibilidade booleana (ANSÓTEGUI et al., 2016), ou independentes de problema, e.g. tamanho da instância (BÖTTCHER; DOERR; NEUMANN, 2010) ou propriedades da paisagem de solução (REEVES, 1999; MERZ, 2004; SCHIAVINOTTO; STÜTZLE, 2007; WATSON, 2010; MERSMANN et al., 2011; FRANZIN; STÜTZLE, 2020). Sob uma perspectiva de aprendizagem de máquina, abordagens de configuração de algoritmos tradicionais correspondem a uma regressão constante no espaço de parâmetros (BENGIO; LODI; PROUVOST, 2021).

Este trabalho explora modelos adicionais de regressão para a configuração de algoritmos. Os parâmetros são representados por modelos que definem seus valores em função do tamanho da instância. Essa característica de instância é independente de problema e obtida sem a necessidade de processamento computacional custoso, o que torna o método simples e pronto para ser usado na configuração de algoritmos para qualquer domínio de problema. Inicialmente, o trabalho explora modelos lineares que, apesar de simples, são capazes de modelar a relação entre tamanho de instância e valores ótimos de parâmetros para vários cenários de configuração. Nos casos em que essa relação é não linear, este trabalho propõe um modelo linear em trechos (ou modelo *piecewise linear*) baseado em múltiplos pontos de apoio. Além disso, o trabalho propõe um modelo *log-log linear* que aplica uma transformação logarítmica aos espaços de parâmetros e tamanhos de instância, e então aplica o modelo linear básico ao cenário transformado. Ambos modelos *piecewise* e *log-log linear* são capazes de representar relações não lineares entre o tamanho da instância e valores ótimos de parâmetros.

A Figura B.2 apresenta um cenário artificial com a resposta do valor ótimo de um parâmetro p em função de diferentes tamanhos de instância n. Para cada modelo, é apresentada uma aproximação que minimiza o erro quadrático médio (EQM). É possível verificar que o modelo constante (a) não representa de maneira adequada a relação entre tamanho da instância e valor ótimo do parâmetro, obtendo um EQM de 0.074. A representação é melhorada ao usar o modelo linear (b, com um EQM de 0.016) porém, dada a não linearidade dessa relação, os modelos *piecewise* (c) e *log-log linear* (d) apresentam uma melhor aproximação (EQMs de 0.001 e 0.006,

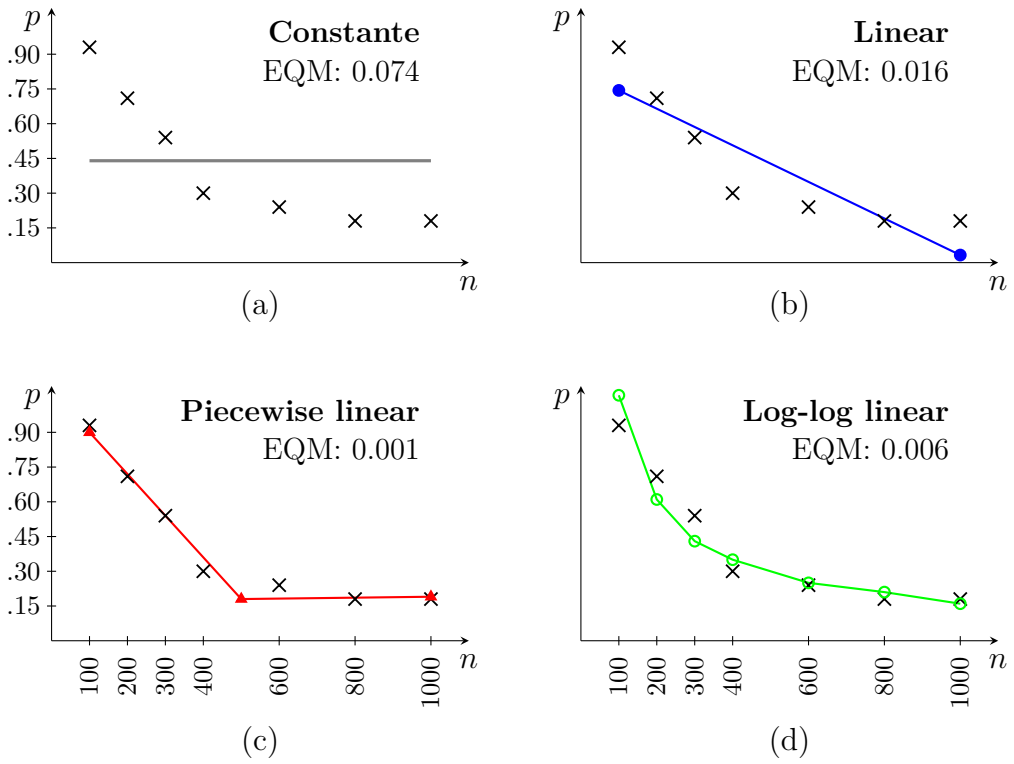


Figura B.2 – Comparação dos modelos de regressão em um cenário artificial não-linear.

respectivamente).

Esses modelos foram avaliados em quatro cenários reais. Para cada cenário, foi verificada a relação entre o tamanho da instância e os melhores valores dos parâmetros. Em seguida, os algoritmos foram configurados usando a abordagem tradicional (constante) e os modelos de regressão propostos. Finalmente, a qualidade das configurações produzidas foi avaliada em instâncias de teste. Em geral, o modelo linear foi capaz de aproximar de maneira satisfatória a relação entre tamanho da instância e valor ótimo dos parâmetros. Em um dos cenários, onde essa relação é não linear, os modelos *piecewise* e *log-log linear* foram capazes de melhorar a aproximação. Como consequência, os algoritmos sob configurações não constantes mostraram desempenho superior, em comparação ao uso de configurações constantes. O Capítulo 5 descreve os modelos propostos e apresenta sua avaliação experimental completa.

B.3 Análise Visual do Processo de Configuração

Diante da facilidade de uso de métodos de configuração automática de algoritmos, é possível e até mesmo comum sua aplicação como métodos de caixa-preta,

i.e. sem a necessidade de uma análise e entendimento aprofundados do seu funcionamento. Por outro lado, é importante analisar as execuções do configurador usado e entender seu funcionamento, de modo a obter os melhores resultados do processo de configuração. Os dados gerados durante a execução do configurador fornecem informações úteis sobre o desempenho do algoritmo sob diferentes configurações e sobre as decisões tomadas durante o processo de configuração. No entanto, analisar esses dados (usualmente encontrados em grandes quantidades) é uma tarefa desafiadora, além de exigir conhecimento sobre o funcionamento interno do configurador, e.g. como ele seleciona configurações para serem avaliadas. Diante disso, este trabalho propõe uma ferramenta que processa, interpreta e fornece visualizações do processo de configuração de algoritmos, com o objetivo de simplificar sua análise e entendimento.

A ferramenta desenvolvida, chamada acviz, fornece duas visualizações principais com base em execuções do configurador irace. A primeira delas apresenta o processo de configuração e sua evolução, mostrando cada avaliação do algoritmo, junto da configuração e instância associadas e o resultado correspondente. Como o irace implementa um método de configuração iterativo, o acviz mostra ainda o desempenho mediano das configurações de cada iteração, bem como das melhores configurações encontradas até o momento, o que permite identificar a evolução na qualidade das configurações ao longo do tempo. A segunda visualização mostra o desempenho das melhores configurações encontradas, tanto do processo completo como de cada iteração, em um conjunto de instâncias de teste. O principal benefício dessa visualização é verificar o poder de generalização dessas configurações para novas instâncias e o desempenho esperado do algoritmo em produção.

O uso e benefícios da ferramenta acviz foram avaliados em três estudos de caso envolvendo a configuração de algoritmos de decisão e otimização. O primeiro estudo de caso considera cenários onde o conjunto de instâncias de treinamento inclui uma instâncias muito fácil de ser resolvida e outra muito difícil. As visualizações produzidas permitem identificar essas características das duas instâncias e perceber que ambas não contribuem para a avaliação e comparação de configurações, visto que o desempenho de diferentes configurações são igualmente bons na instância fácil e igualmente ruins na instância difícil. O segundo estudo de caso considera cenários onde o esforço computacional empregado no processo de configuração (número máximo de avaliações ou tempo limite) é demasiado alto. As visualizações mostram

a convergência do processo de configuração muito antes do seu término. Novas configurações produzidas são muito similares àquelas já avaliadas, com a consequente estagnação de desempenho. Finalmente, o terceiro estudo de caso considera cenários com *overtuning*, i.e. quando as configurações produzidas se especializam nas instâncias de treinamento e, dada sua baixa representatividade, o desempenho observado em novas instâncias é pobre. As visualizações dos resultados das melhores configurações de cada iteração nos conjuntos de instâncias de treinamento e teste permitem identificar tal efeito, pois mostram a melhoria do desempenho nas instâncias de treinamento ao longo do processo de configuração, ao passo que o desempenho nas instâncias de teste piora simultaneamente.

Conforme apresentado nos estudos de caso propostos, a ferramenta acviz permite identificar problemas nos cenários de configuração que não são facilmente detectados de outra forma. Com isso, o projetista pode corrigir tais falhas, como selecionar instâncias adequadas ou reduzir o esforço de configuração, e melhorar a qualidade das configurações produzidas. O Capítulo 6 apresenta em detalhes a ferramenta acviz e algumas funcionalidades adicionais, e.g. as diferentes medidas de desempenho e opções de visualização implementadas, bem como a discussão detalhada dos estudos de caso propostos.

B.4 Solver Baseado em Componentes para Otimização Binária

Um uso mais avançado de métodos de configuração consiste em aplicá-los para automatizar o processo completo do projeto de algoritmos. Em vez de focar na configuração dos parâmetros de um algoritmo predeterminado, define-se um conjunto de componentes algorítmicos que podem ser selecionados, combinados e calibrados para a produção de distintos algoritmos. Um configurador é então aplicado para automatizar a exploração do espaço de componentes e buscar pelos melhores algoritmos para um dado domínio de problema. Essa abordagem é chamada de *projeto automático de algoritmos* e tem o potencial de produzir algoritmos híbridos e competitivos com o estado-da-arte, a exemplo de *solvers* para satisfatibilidade booleana (KHUDABUKHSH et al., 2016), buscas locais para problemas de sequenciamento de produção (MARMION et al., 2013; PAGNOZZI; STÜTZLE, 2019), e algoritmos evolutivos para otimização multiobjetivo (BEZERRA; LÓPEZ-IBÁÑEZ; STÜTZLE, 2014a; BEZERRA; LÓPEZ-IBÁÑEZ; STÜTZLE, 2020b). Apesar da variada aplicabilidade e dos resultados positivos reportados na literatura, os algo-

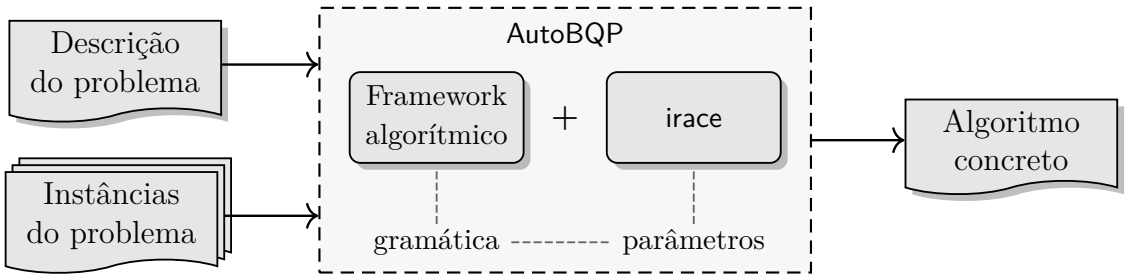


Figura B.3 – Visão geral do projeto automático de algoritmos com AutoBQP.

ritmos produzidos usando as técnicas supracitadas são geralmente especializados em um problema específico. Em contrapartida, este trabalho apresenta um *solver* chamado AutoBQP, que aplica os conceitos de projeto automático de algoritmos para resolver problemas de otimização binários, i.e. a classe geral de problemas de otimização cujas soluções são representadas por um vetor de *bits*.

A Figura B.3 apresenta a estrutura e funcionamento do *solver* AutoBQP. Dada uma descrição do problema contendo sua função objetivo e um conjunto de instâncias de treinamento, o *solver* busca automaticamente por algoritmos de alto desempenho. Para isso, o AutoBQP combina um *framework*, que implementa os componentes algorítmicos e permite sua seleção e combinação, com o configurador irace, responsável por buscar as melhores combinações de componentes e valores para seus parâmetros de entrada. As regras para combinação de componentes e produção de algoritmos completos são representadas por uma gramática, cujas decisões são mapeadas para parâmetros (MASCIA et al., 2014b), o que permite a comunicação do *framework* com o irace. O *framework* algorítmico foi construído com base em componentes heurísticos para programação quadrática binária irrestrita (ou UBQP, do inglês *unconstrained binary quadratic programming*). Como muitos problemas podem ser reduzidos e resolvidos via UBQP (KOCHENBERGER et al., 2004; KOCHENBERGER et al., 2014), o AutoBQP é capaz de resolver toda essa classe de problemas binários.

O *solver* AutoBQP foi usado para produzir algoritmos heurísticos para três problemas distintos: o próprio UBQP; o problema de corte máximo em grafos (MaxCut), que pode ser representado diretamente como um modelo de programação quadrática binária irrestrita; e a alocação de testes, uma variante da coloração de grafos que pode ser reduzida ao UBQP via reformulação. Os algoritmos produzidos se mostraram competitivos, apresentando desempenho melhor em comparação com abordagens do estado-da-arte. Além disso, os algoritmos produzidos encontraram

novas e melhores soluções para diversas instâncias do MaxCut e da alocação de testes. Esses resultados mostram o potencial do projeto automático de algoritmos, que permite construir soluções de alta qualidade com pouco esforço humano e sem a necessidade de conhecimento especializado nas técnicas de configuração automática e nos problemas explorados. O Capítulo 7 apresenta o *solver* AutoBQP, detalhando seus componentes algorítmicos e as técnicas exploradas. Já o Capítulo 8 apresenta a avaliação experimental do *solver* e discute os resultados obtidos.

B.5 Discussão

Conforme discutido nas seções anteriores, os métodos propostos neste trabalho apresentam contribuições relevantes para o projeto e configuração automáticos de algoritmos. Eles permitem reduzir o tempo de configuração e melhorar a qualidade dos resultados, enquanto as visualizações produzidas facilitam a análise e entendimento do processo de configuração. Além disso, o *solver* desenvolvido permite projetar algoritmos de maneira automática para uma ampla classe de problemas de otimização.

Este trabalho também apresenta uma série de contribuições indiretas. As aplicações apresentadas ao longo do trabalho demonstram a relevância prática e a eficácia do projeto e configuração automáticos de algoritmos, bem como apresentam um conjunto de diretrizes a pesquisadores e projetistas quanto ao uso desses métodos. Com o *solver* desenvolvido, foram produzidos algoritmos para diferentes problemas e com melhor desempenho em comparação com abordagens da literatura. Finalmente, o trabalho fornece um conjunto amplo de cenários de configuração, cujos algoritmos podem ser adaptados para diferentes domínios de problema. Esses cenários também são úteis no desenvolvimento, avaliação e comparação de técnicas de configuração de algoritmos.

As implementações de todos os métodos e ferramentas propostos neste trabalho estão disponíveis para uso, bem como as implementações dos algoritmos produzidos. Também estão disponíveis materiais suplementares para cada capítulo, contendo os algoritmos correspondentes, definição dos cenários de configuração, instâncias de treinamento e teste e detalhes adicionais dos resultados experimentais. Todos os cenários de configuração usados nos experimentos foram também disponibilizados separadamente para facilitar o acesso. O Apêndice A lista todos os artefatos produzidos a partir dessa pesquisa, incluindo publicações científicas e apresentações em eventos.