KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Data analytics for algorithm design

Nguyen Thi Thanh Dang

Supervisors:
Prof. dr. P. De Causmaecker
Dr. T. Stützle
(ULB, Belgium)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Computer Science

April 2018

Data analytics for algorithm design

Nguyen Thi Thanh DANG

Examination committee:
Prof. dr. ir. J. Berlamont, chair
Prof. dr. P. De Causmaecker, supervisor
Dr. T. Stützle, supervisor
(ULB, Belgium)
Prof. dr. ir. G. Vanden Berghe
Prof. dr. ir. M. Denecker
Prof. dr. J.K. Hao
(University of Angers, France)
Dr. M. López-Ibánez
(University of Manchester, United Kingdom)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Computer Science

April 2018

© 2018 KU Leuven – Faculty of Engineering Science
Uitgegeven in eigen beheer, Nguyen Thi Thanh Dang, Etienne Sabbelaan 53 box 7659, 8500 Kortrijk (Belgium)

Alle rechten voorbehouden. Niets uit deze uitgave mag worden vermenigvuldigd en/of openbaar gemaakt worden door middel van druk, fotokopie, microfilm, elektronisch of op welke andere wijze ook zonder voorafgaande schriftelijke toestemming van de uitgever.

All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm, electronic or any other means without written permission from the publisher.

Acknowledgement

This PhD is a long, challenging, and great journey to me. During those five years, I have been able to learn many new things and gained invaluable experience from my supervisors, advisors, collaborators, colleagues, and friends. They have opened my mind and I am always thankful for that.

First of all, I am very grateful to my supervisor, Patrick De Causmaecker, and my co-supervisor, Thomas Stützle. This dissertation would not be available without their endless help and support. Their advice and contributions on all the researches conducted in this dissertation are highly important. It was a privilege for me to work with them.

I would like to thank Greet Vanden Berghe, Frits Spieksma, Marc Denecker, Manuel López-Ibáñez and Jin-Kao Hao for accepting to be in my advisory and examination committees, and for their valuable feedbacks which help to improve the quality of the dissertation. I thank Omer Van der Biest and Jean Belarmont for being the chair of my private and public defences.

I appreciate the great opportunities to collaborate with many researchers from different areas during this PhD. My thanks to Leslie Pérez Cáceres for our joint work on configuring irace. I thank Andrea Schaerf, Sara Ceschia, and Stefaan Haspeslagh for the work on the second international nurse rostering competition, Jeroen Cortijens, Benoit Depaire, and An Caris for the case study on analysing the importance of algorithm parameters. I also want to thank Túlio Toffolo for his help on studying their Verolog solver.

My time in a new research environment was eased by the welcome and friendliness of my CODeS's colleagues in Gent and Kortrijk, whom I would like to thank, especially Greet, Tony, Wim, Tulio, Eline, Farzaneh, Reshma, Pieter Smet, Pieter Leyman, and Morteza.

Besides professional and academic experience, the most amazing thing I have had during those years is the chance to make friends with people who come from

all over the world, and to learn about the beauty of cultural diversity. It was a pleasant time for me to be with them, and to have nice chats and enjoyable memories. My special thanks to San for being a long-time and lovely friend, Hans and Chathika for the interesting discussions on several topics, Caroline for the fun dance and music, Tommy, Paco, Chao, and Hoang for being my nice officemates, Ayman, Laleh, Hue and her husband for their awesome dishes, Gertrude for the fantastic teas, Aritri and Khushboo for teaching me about Indian cultures, Subhash and Martin for the excellent physic lectures, Bélen and Leslie for the amazing inspiration, and Sirima for always smiling.

Finally, I want to thank my family for their unconditional love and support. Cảm ơn bố mẹ, mẹ Tám, anh Trường chị Hòa, mợ Quynh, chị Bi, anh hai chị hai. Last but not least, lots of thanks to my husband, Phong Le. You are one of most wonderful things that ever happened to me.

Abstract

The development of algorithms solving computationally hard optimisation problems has a long history. Several important contributions can be traced back to the mid of the 20th century. Today, state-of-the-art optimisation problems are of high complexity, combining different components developed around a variety of conceptual approaches. The number of concepts as well as alternative implementations of those concepts makes algorithm design an intricate task and any automated support is welcome. The fast growth of computing power makes it possible to collect vast amounts of data on algorithm performance. The collected data can be processed by advanced data analytics resulting in a better understanding of algorithms as well as problems. This understanding supports further engineering efforts leading to even higher performing methods. The rise of 'data science' - a combination of machine learning, statistics and data analysis - has seen applications in optimisation. Automatic algorithm configuration, algorithm selection and hyper-heuristics with learning schemes, to name a few, are among the most prominent topics in this marriage of the two fields.

This dissertation centres around the theme "Data Science for Optimisation". We focus on two key aspects: automatic algorithm configuration and the analysis of algorithm parameters' impact on algorithm performance.

Algorithm configuration is the task of selecting the best design of an algorithm, given a parametrised algorithm with a number of design choices and a set of problem instances. Recently, a number of automatised tools have been proposed for algorithm configuration. In chapter 3, we improve the efficiency of algorithm configuration for a local-search based metaheuristic with multiple neighbourhoods. An algorithm parameter determines the probability that a specific neighbourhood is selected at each iteration. Our method recognises neighbourhoods behaving similarly in the course of the solving procedure on a given set of problem instances. The similarity is defined based on a number of pre-defined observables. The neighbourhoods are then grouped together using

clustering analysis. The number of parameters is reduced by assigning one single parameter per cluster.

In chapter 4, we consider improving the performance of irace - a state-of-the art automatic algorithm configurator - by meta-tuning. Meta-tuning uses an automatic configurator to optimise the parameters of a configurator on a set of algorithm configuration benchmarks. The benchmark algorithms are modelled by surrogates to make the procedure computationally feasible. Surrogate modelling uses regression to build a predictor for the algorithm performance on every problem instance.

During algorithm configuration, several algorithm configurations are evaluated and an algorithm performance dataset is being built. This dataset can provide insights into how different algorithm parameters influence algorithm performance. The idea is to build a predictor on the given performance dataset and then using it for analysing variable importance. In chapter 5, the combined usage of three techniques utilising this idea is demonstrated in three case studies. The three techniques are forward selection, fANOVA and ablation analysis. We illustrate how to interpret the analysis results in the three cases, and discuss the advantage of combining different analysis methods.

Beknopte samenvatting

De ontwikkeling val algoritmen om reken-technisch moeilijke optimalisatieproblemen te behandelen kent een lange geschiedenis. Belangrijke bijdragen werden reeds geleverd in het midden van de 20ste eeuw. Vandaag zijn hoog-performante optimalisatie-algoritmen vaak zelf van een even hoge complexiteit. Ze bestaan uit verschillende componenten die afgeleid werden volgens verschillende algoritmische benaderingen. De conceptuele diversiteit en de aanwezige alternatieven voor implementatie bemoeilijken het ontwerp van een algoritme. Dit is de bestaansreden van gereedschappen voor de ondersteuning van de ontwerper. Door de snelle toename in beschikbare computerkracht is het nu mogelijk om grote hoeveelheden data te vergaren over de performantie van algoritmen. Data analytische methodes kunnen dan gebruikt worden om het gedrag van de algoritmen zowel als de eigenschappen van de op te lossen problemen beter te begrijpen. Dit is dan weer nuttig bij het ontwerpen van algoritmen om tot top performantie te komen. De laatste decennia zijn de mogelijkheden van 'data science' - een combinatie van machine-leren, statistiek en data-analyse - sterk toegenomen en dit creëert mogelijkheden voor het gebruik van deze technieken in een optimalisatiecontext. Automatische algoritme configuratie, algoritme selectie, hyperheuristieken met ingebouwde leermechanismen... zijn slechts voorbeelden van zeer actuele onderwerpen.

Deze verhandeling concentreert zich op het onderwerp 'Data science voor optimalisatie'. Meer in het bijzonder concentreren we ons op twee aspecten bij het ontwerp van een algoritme: automatische configuratie en analyse van algoritme-parameters op de performantie.

Gegeven een parametrisch algoritme met verschillende ontwerpbeslissingen voor elk van zijn componenten en gegeven een distributie van probleeminstanties, moet algoritme-configuratie toelaten om het beste ontwerp voor het algoritme te bekomen. Recent werden een aantal gereedschappen voorgesteld die deze taak automatiseren. In hoofdstuk 3 verbeteren we de efficiëntie van automatische algoritme-configuratie voor een metaheuristiek die gebaseerd is op lokaal

zoeken met een verzameling nabuurschappen. Algoritme-parameters bepalen de waarschijnlijkheid om een bepaalde buurt te kiezen bij elke nieuwe iteratie. We stellen een methode voor om buurten te vinden die zich gelijkaardig gedragen. Deze worden dan gegroepeerd in clusters en krijgen één parameter. Hierdoor wordt het aantal parameters sterk gereduceerd.

In hoofdstuk 4 beschouwen we het verbeteren van de performantie van state-of-the-art automatische algoritme-configuratie software, met de naam irace. We gebruiken meta-tuning om de parameters van irace op een verzameling configuratiebenchmarks te optimaliseren. Dit gebeurt met behulp van een automatische configurator. Deze procedure is enkel mogelijk door het gebruik van surrogaat modellen voor het gedrag van de algoritmen. De surrogaten zijn regressie modellen die de performantie van een algoritme voorspellen op basis van de eigenschappen van een instantie.

De performantie-dataset die bekomen wordt nadat een algoritme geconfigureerd werd, kan gebruikt worden om inzicht te verkrijgen in de impact van algoritme-parameters op de performantie. We bouwen een voorspeller voor de gegeven performantie-dataset. Deze wordt dan gebruikt om het belang van de parameters te analyseren. In hoofdstuk 5 demonstreren we het gecombineerde gebruik van drie technieken in deze context: voorwaartse selectie, fANOVA en ablatie-analyse. We gebruiken drie gevalstudies. Deze laten ons toe om de resultaten van de analyse te interpreteren en de voordelen van de combinatie van analysetechnieken te onderzoeken.

Abbreviations

ACO Ant Colony Optimisation. 9, 10

EPM Empirical Performance Model. 40

MIP Mixed-Integer Programming. 42, 61

RMSE Root Mean Square Error. 62, 63

SAT Propositional Satisfiability. 12, 38, 42

SB-VRP Swap-Body Vehicle Routing Problem. 20, 21, 83

TSP Travelling Salesman Problem. 9, 10

VRP Vehicle Routing Problem. 21, 83

Contents

Acknowledgement i

Abstract iii

Abbreviations vii

Contents ix

List of Figures xiii

List of Tables xvii

1 Introduction 1

1.1 Context 1

1.2 Outline and main contributions 4

1.3 Complementary contributions 6

2 Background 9

2.1 Optimisation algorithms: design choices and parameters 9

2.2 Automatic algorithm configuration 11

2.3 Analysis of algorithm components and parameters 13

2.4 Conclusion 14

3 A method for characterising neighbourhood behaviour in local search 17

3.1 Introduction 18

3.1.1 Local search-based metaheuristics and neighbourhoods . 18

3.1.2 Motivation 19

3.2 A case study on a multi-neighbourhood local search-based metaheuristic 21

3.3 Neighbourhood characterisation 22

3.3.1 Step 1: collect simple and accumulable statistics 24

3.3.2 Step 2: identify solution quality regions 26

3.3.3 Step 3: characterise neighbourhood behaviour as feature vectors 27

3.4 Neighbourhood clustering 31

3.5 Experimental results 32

3.6 Conclusion 35

4 Configuring irace using surrogate configuration benchmarks 37

4.1 Introduction 38

4.1.1 Automatic algorithm configurators and their parameters 38

4.1.2 Surrogate modelling 39

4.1.3 Motivation 39

4.2 Parameters of irace 40

4.3 Surrogate benchmarks 41

4.3.1 Surrogate configuration benchmarks 42

4.3.2 Prediction accuracy analysis 43

4.3.3 Homogeneity analysis 44

4.3.4 Algorithm parameter importance analysis 45

4.4 Experimental results 48

4.4.1 Meta-tuning with a single benchmark 48

4.4.2 Meta-tuning with multiple benchmarks 49
4.5 Analysis on irace's parameters 51
4.5.1 Analysis of single parameters and pairwise interactions . 52
4.5.2 A higher-order interaction effect 54
4.5.3 Essential parameters 54
4.6 Results on real benchmarks 55
4.7 Conclusion 56

5 Analysis of algorithm components and parameters 59

5.1 Introduction 60
5.1.1 Analysis methods 60
5.1.2 Motivation 61
5.2 fANOVA 61
5.3 Forward selection 62
5.4 Ablation analysis with surrogates 63
5.5 Case study 1: Ant Colony Optimization algorithms for the Travelling Salesman Problem 64
5.5.1 fANOVA analysis 64
5.5.2 Ablation analysis 66
5.5.3 Summary 67
5.6 Case study 2: tuning irace on a CPLEX surrogate configuration benchmark 68
5.6.1 Ablation analysis 68
5.6.2 fANOVA analysis 71
5.6.3 Summary 74
5.7 Case study 3: a Large Neighbourhood Search for a Vehicle Routing Problem with Time Windows 75
5.7.1 fANOVA analysis 76

5.7.2 Forward selection 78
5.7.3 Summary 80
5.8 Conclusion 80

6 Conclusions **83**

Bibliography **87**

List of Figures

2.1 Input and output of an automatic algorithm configurator . . . 12

3.1 Visualization of $r_{improve}$, r_{worsen} and $r_{nothing}$ for four neighbourhoods on problem instance *new_without*. The x-axis represents solution quality (the larger the value, the better the corresponding solutions). The y-axis shows the values of the three characteristics. 25

3.2 Plots of *sum_nIters*, the sum of n_{iters} values of all neighbourhood on each solution quality interval for each of the six instances used in our case study. The x-axis represents the intervals. Although the original number of intervals is 1000, many of them at the end of the range are not reached during the collecting procedure and are eliminated from the x-axis. 29

3.3 Plots of *sum_nIters*, the sum of n_{iters} values of all neighbourhood on each solution quality interval for each instance in our case study and results of the partitioning with the number of regions $n = 5$. The vertical lines in each plot indicate the boundaries between the five resulting solution quality regions. 29

3.4 The box-plots show the test performance of the best parameter settings obtained from 18 runs of SMAC on three scenarios: original (no grouping of neighbourhoods), basic (manual neighbourhood groupings) and clustered (neighbourhood clustering using our characterisation method), whose the number of weight-value parameters are 42, 28 and 9 respectively. The horizontal line is the test performance of the default parameter configuration, whose weight-value parameters are set identically. The p-values are obtained from pairwise t-tests. 34

3.5 The box-plots show the test performance of two sets of algorithm configurations. The first one consists of the best configurations obtained from 18 runs of SMAC on three scenarios: original (no grouping of neighbourhoods), basic (manual neighbourhood groupings) and clustered (neighbourhood clustering using our characterisation method), whose number of weight-value parameters are 42, 28 and 9 respectively. The second one includes the Simple Random versions of the configurations from the first set. The horizontal line is the test performance of the default algorithm configuration. 35

4.1 Scatter plots of real performance values (x-axis) and the corresponding prediction values (y-axis) of 100 test algorithm configurations on the instance set of each benchmark. 44

4.2 Performance of irace using the default setting and the five elite settings obtained by meta-irace on each surrogate benchmark. The pairwise Wilcoxon signed rank test at confidence level 99% between irace default configuration and the best elite indicates statistically significant differences in all cases (p-value is given in each plot) in favor of the best performing elite settings. 51

4.3 Normalised performance of irace using the default setting and the five elite settings obtained by meta-irace configuring irace on the four CPLEX benchmarks and pairwise Wilcoxon signed rank test p-value (confidence level 99%) of the performance of irace using the default setting and the best performing elite setting. . 52

4.4 Marginal plots of the two single effects *minNbSurvival* and *testType* on the performance dataset of configuring irace with multiple CPLEX surrogate benchmarks. The yellow area and the box-plots show the variance of the marginal prediction. . . 53

4.5 Test performance of irace using the default configuration and the best configuration obtained by meta-irace, measured on the real benchmarks. 56

4.6 Test performance of irace using the default configuration and the best on returned by meta-irace, measured on a number of real configuration benchmarks that the meta-tuning never saw. . . . 57

5.1 Marginal plot of parameter *localsearch* given by fANOVA analysis on the performance dataset obtained when applying irace on ACOTSP. Each box-plot shows the mean and the variance of the marginal prediction. 66

5.2 Ablation analysis (with surrogate) on the path from ACOTSP default configuration to the best configuration returned by irace. The y-axis shows the percentage of performance gain when switching the value of a parameter on the ablation path. 67

5.3 Three ablation paths from the irace default configuration to the best one returned by meta-irace on CPLEX-REG surrogate benchmark. The plots show the percentage of performance gain when the corresponding irace parameter is changed along the paths. The red line indicates the 5% performance gain threshold. Any variables with contribution below this line are marked as non-bold for visualisation purpose. 71

5.4 Three ablation paths from the best one returned by meta-irace on the CPLEX-REG surrogate benchmark to the default irace configuration. The plots show the percentage of performance loss when the corresponding irace parameter is changed along the paths. The red line indicates the 5% performance gain threshold. Any variables with performance loss below this line are marked as non-bold for visualisation purpose. 72

5.5 Marginal plot of the problem instance feature *customer_number* given by fANOVA analysis 77

5.6 Marginal plot of algorithm parameter *repair*, and pairwise interaction between algorithm parameter *repair* and problem instance feature *customer_number* given by fANOVA analysis with performance normalisation. 78

5.7 Prediction error (on validation set) of the model trained on a subset of variables chosen during one run of forward selection analysis and one run of backward elimination analysis. 79

List of Tables

3.1 Names and sizes of the 42 neighbourhoods used in our case study algorithm. In the left column, each neighbourhood is defined by its neighbourhood structure name and an instantiation of the size (each size of a neighbourhood is a single value, except for neighbourhood Each-sequence-cheapest-insertion, whose size is a pair of values). In the right column, the neighbourhoods without sizes are listed. The table also shows the manual grouping of neighbourhoods having similar size values within the same neighbourhood structure. Each group is represented by a line in the left column. For the neighbourhoods without sizes, each one is treated as a group itself. This grouping, which helps to reduce the number of weight-value parameters to 28, is learned from the setting provided by the algorithm developers. 23

4.1 Parameters of irace 42

4.2 Seven configuration benchmarks used in our study. 43

4.3 Mean values of Pearson's correlation coefficient (CC) across 10-fold cross validation. 44

4.4 Kendall W values on three different datasets. 46

4.5 fANOVA results on both the original and the predicted performance dataset of each benchmark. For the sake of brevity, only effects with percentage of contribution on performance variance larger than 5% are shown. 47

5.1 Eleven algorithm parameters of ACOTSP 64

5.2 The five best configurations returned by irace and the default configuration of ACOTSP 65

5.3 Nine parameters of irace 69

5.5 Eight algorithm parameters and five problem instance features for the case study with a Large Neighbourhood Search metaheuristic algorithm for solving a Vehicle Routing Problem with Time Windows [1]. 75

Chapter 1

Introduction

1.1 Context

Optimisation problems can be found everywhere in our daily life. For example, a postman wants to deliver all the mails he needs to distribute today and return to the office in the smallest amount of time (the Travelling Salesman Problem [2]), or a faculty staff has to make a weekly timetable to assign teachers to classes and time slots in a way such that the preferences of the teachers are satisfied as much as possible (the School Timetabling Problem [3]). The basic components of an optimisation problem include an objective function to be optimised, and a number of decision variables whose possible values are restricted by a set of constraints. In the School Timetabling Problem, the objective function to be minimised is the total number of violations of teachers' preferences in a timetable, the decision variables are discrete variables representing the choices of whether or not a teacher is assigned to a specific class and time slot, and the constraints are for defining the availability of teachers. The aim of solving an optimisation problem is to find a feasible instantiation of the decision variables, i.e., a solution that satisfies the constraints, so that the resulting objective function value is optimised.

Many optimisation problems belong to the NP-hard class [4], for which no deterministic polynomial-time algorithms have been found. Several algorithmic approaches for these problems have been proposed and studied for decades. Each one can offer its own advantages over the others in specific situations. Those approaches can be categorised into four groups: exact, approximation, heuristic and metaheuristic algorithms. *Exact methods* can guarantee the optimality of

the solutions found. The running time required by exact approaches might increase exponentially with instance size and make these methods impractical when problem size increases. *Approximation methods* offer a trade-off between optimality and algorithm performance by guaranteeing a certain bound on the gap to the optimal solution value. These algorithms can sometimes be quite complicated to implement due to their sophistication (and often the bounds are rather weak). *Heuristic methods* usually focus on finding a reasonably good solution within a small amount of time compared to the two former approaches. This property makes heuristics popular in practice despite their inability of proving the optimality of solutions. *Metaheuristics* bring the idea of heuristic algorithms to a higher level of abstraction. Metaheuristics are general purpose frameworks for developing heuristic algorithms; they provide a set of concepts, guidelines and strategies for heuristic design. Starting from 1980s, this fourth category of algorithms has continuously evolved. A vast number of metaheuristics have been proposed and successfully applied in several real-world applications [5].

Approaches belonging to the same category also offer a variety of advantages depending on particular cases. As an example, according to a survey in 2003 [6], exact methods at the time can solve the general Travelling Salesman Problem within a reasonable amount of running time for instances with only up to 2000 cities, but for some special shapes of distance matrix, there exist methods to practically find optimal solutions for instances with up to 13000 cities (recently this number has been increased to 85900 [7]). Another example is metaheuristics making use of the *cheapest insertion neighbourhood* for solving the same problem. These are instantiations of algorithms belonging to a group called *iterated greedy methods* [8] or *large neighbourhood search*. The idea is to iteratively search for a better solution by removing a number of cities in the current solution and sequentially reinserting each of them back into the cheapest position, i.e., the position that causes the least increase in the total travelling distance. The number of cities removed defines the size of the neighbourhood. Larger sizes make it more time-consuming, but can be more effective when problem instance sizes are large.

Given the fact that the effectiveness of an algorithm can change according to various factors such as different contexts, the hardness of the problem being tackled, the availability of many options on how to design an algorithm, and the stochasticity of algorithm performance, *algorithm design* is difficult and normally requires a lot of effort and expertise. Recently, thanks to the fast growth of computational resources, we have seen the emergence of data science, or *data analytics* techniques (machine learning, statistics, data analysis, etc) that make use of a large amount of algorithm performance data to support algorithm design. Developers can freely integrate several design choices into their

algorithm, each of which may be accompanied by different parameters to vary the way the algorithm works. The task of choosing the best-performing design and parameter setting given a specific problem instance distribution, namely *algorithm configuration*, can be done automatically using approaches proposed in *automatic algorithm configuration* [9] such as irace [10], ParamILS [11], SMAC [12] and GGA [13]. Another topic is *algorithm selection* [14], where different algorithms can be combined into a portfolio and be selected given a specific case. This allows utilising and combining the strength of all algorithms. One example is the SATzilla [15], a porfolio-based algorithm selector for SAT solvers. Systems that combine automatic algorithm configuration and algorithm selection are also proposed, such as Hydra-MIP [16] for Mixed Integer Programming solvers and Auto-WEKA [17] for machine learning classification algorithms. Components and search behaviour of algorithms can also be automatically chosen and adapted on-the-fly in *hyper-heuristics* with learning schemes [18]. Those examples of support, in a sense, encourage the creativity of algorithm developers and extend the opportunities to utilise various existing algorithmic approaches as well as to create novel methods.

This dissertation follows the same line of research, where data analytics is applied to support two key aspects during the algorithm design process: automatic algorithm configuration and the analysis of algorithm components and parameters' influence on algorithm performance.

In automatic algorithm configuration, the underlying algorithm that is configured is normally treated as a black-box, i.e., no knowledge of the algorithm itself is used. This allows automatic algorithm configuration tools to be general-purpose and widely applicable. The same tool can be applied to any algorithms. In the first contribution of this dissertation, presented in Chapter 3, we extend this by considering integrating knowledge from the configured algorithm into the configuration process in order to improve the configurator's efficiency. This contribution is for a particular type of local search-based metaheuristic algorithms. It serves as an example for the idea of "opening the box", i.e., adding extra information obtained from the algorithm being configured to the configuration procedure, which we believe can provide benefits towards automatic algorithm configuration.

Another potential for enhancing automatic algorithm configuration procedures is to improve the performance of the configurators themselves. The second contribution of this dissertation investigates this idea by considering finding better parameter settings for the current configurators through surrogate modelling. This contribution is described in Chapter 4.

The last contribution is on the analysis of algorithm parameters' influence on algorithm performance, which also plays an important role during algorithm

design. Several analysis methods have been proposed in the literature. In Chapter 5, we look into combining them to gain more insights than using each one separately.

1.2 Outline and main contributions

This section describes the outline of the remaining chapters and more details on the three main contribution of the dissertation.

- Chapter 2 presents a brief introduction to automatic algorithm configuration and the analysis of algorithm components and parameters.
- Chapter 3 focuses on a specific type of algorithms that has been successfully applied in many complex real-world optimisation problems: local search-based metaheuristics with multiple neighbourhoods. In particular, we consider a scheme where the probability of choosing a neighbourhood at each iteration of the algorithm is defined as a weight-value parameter. Any automatic algorithm configuration tool can be used to tune these weights. However, the possibly large number of neighbourhoods might deteriorate the efficiency of the configurator. One possible solution to reduce the number of algorithm parameters is to group similar neighbourhoods and assign a common weight value for all neighbourhoods within a group. In order to do that, we propose a method to characterise the behaviour of each neighbourhood on a given set of problem instances as a feature vector, and use a clustering technique to recognise similar neighbourhoods automatically. The novelty of the method is the ability of reflecting changes of neighbourhood behaviour according to the difficulty of different solution quality regions, without the requirement of any domain-specific knowledge. Since the method is problem independent, it can be applied in similar contexts to support the automatic algorithm configuration task, as well as to gain interesting insights into neighbourhoods behaviour. The content of this chapter is based on the following publication:

  Nguyen Thi Thanh Dang and Patrick De Causmaecker. Characterization of neighborhood behaviours in a multi-neighborhood local search algorithm. In *International Conference on Learning and Intelligent Optimization*, pages 234–239. Springer, 2016

  An R implementation of the proposed method is publicly available at `https://bitbucket.org/nguyendang/neighborhoodcharacterization`
- The support we proposed for the automatic algorithm configuration task in chapter 3 is on a particular type of algorithms. In chapter 4, we bring

it into a more general level. By looking at configurators themselves as optimisation algorithms with parameters that can impact their search behaviour, we consider the possibility of improving their performance on a range of algorithm configuration benchmarks. This work, for the first time, presents a systematic study on the parameters of the popular automatic algorithm configurator irace [10] through a process called *meta-tuning*, i.e., using a configurator to tune the parameters of a configurator. This meta-tuning is made computationally feasible using *surrogate modelling* techniques, where the real target algorithms are replaced by prediction models for cheap evaluations. Results of the study suggest the possibility of significantly improving irace performance. As a side product of the meta-tuning, some insights into the importance of irace's parameters and their influence on the configurator's performance are revealed through a number of post-analysis techniques. This study is a first step in the effort to improve the current state-of-the-art automatic algorithm configurators by automatically optimising their parameters, which offers benefit to the optimisation algorithm community due to the wide range of applications of those tools.

The content of the chapter is based on the following publication, which received the best paper award from the *Evolutionary Combinatorial Optimization and Metaheuristics (ECOM) track* of the Genetic and Evolutionary Computation Conference (GECCO) 2017.

Nguyen Dang, Leslie Pérez Cáceres, Patrick De Causmaecker, and Thomas Stützle. Configuring irace using surrogate configuration benchmarks. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 243–250. ACM, 2017

- In addition to the algorithm configuration task - the main focus of chapters 3 and 4 - another crucial aspect during the algorithm design process is the understanding of the influence of different algorithm components and parameters to algorithm performance. In chapter 5, we address this topic as a next step after the application of an algorithm configuration procedure. More specifically, after an algorithm is tuned either manually by algorithm developers or automatically using a configurator, a large amount of data on algorithm performance is usually available. This data, in turn, can be used to build surrogate models to predict the performance of the algorithm on the whole configuration space. It allows gaining insights into algorithm parameters, such as explaining why the tuning choose specific values for some parameters, or how the average performance would change when some parameters are modified. A number of analysis techniques exploiting this idea have been recently proposed, but have rarely been applied in metaheuristic literature. This chapter presents the applications

of three analysis techniques along this line, namely forward selection [21], fANOVA [22] and ablation analysis with surrogates [23], on a number of case studies, with an emphasis on metaheuristic algorithms. Through those examples, we illustrate the potentials of these techniques and discuss the complementary usage of them to support the cycle of algorithm design.

The content of the chapter is based on the following publication:

Nguyen Dang and Patrick De Causmaecker. Analysis of algorithm components and parameters: some case studies. 2018. Accepted at the International Conference on Learning and Intelligent Optimization (LION12)

- Finally, conclusions are given and future work is discussed in Chapter 6.

1.3 Complementary contributions

Besides the main contributions listed in the previous section, three additional works have been carried out during my PhD program: a study on the motivations for the development of a multi-objective automatic algorithm configurator, a solving approach for a multi-stage nurse rostering problem, and a proposal for the complementary usage of two analysis approaches on a case study for the Vehicle Routing Problem with Time Windows.

In the first work [25], we presented two algorithm configuration problems in which more than one performance measure should be considered at the same time. These motivate the development of multi-objective automatic algorithm configurators. The first problem is an algorithm configuration case study for a deterministic single machine scheduling algorithm [26] where there is a trade-off between the efficiency of the algorithm and the total number of problem instances solved to optimality. The second problem is the configuration problem for non-exact multi-objective optimization algorithms, for which multiple unary indicators [27] can be used to avoid bias towards some specific parts of the true Pareto front.

The second work is a collaboration with Sara Ceschia, Andrea Schaerf and Stefaan Haspeslagh. We proposed a multi-stage formulation of the nurse rostering problem and organised the second International Nurse Rostering Competition [28]. We developed two solvers as baselines for examining the computational hardness of the competition's testbeds. The first one is an exact method based on the problem formulation proposed in [29] and makes use of the MILP solver CPLEX [30]. The second one is a Simulated Annealing-based algorithm implemented using the EasyLocal++ framework (v. 3) [31]. We

also proposed a strategy [32] to deal with the "forecasting" problem induced by the multi-stage formulation. The proposed strategy was parametrised and the solvers were tuned by the automatic algorithm configurator irace [10]. As a post-analysis after the application of the configurator, we used the fANOVA toolbox[1] to understand how different components of the forecasting strategy influence the performance of the solvers.

The third work is a collaboration with Jeroen Corstjens, Benoît Depaire and An Caris. We considered the application of two algorithm parameter analysis methods, namely fANOVA [22] and multilevel regression analysis [1], on a case study with a Large Neighbourhood Search algorithm for solving a Vehicle Routing Problem with Time Windows. We discussed a complementary usage of the two methods that was able to provide more insights than when they were used separately.

[1] http://www.ml4aad.org/algorithm-analysis/fanova/

Chapter 2

Background

This chapter gives a brief overview of the two key aspects during algorithm design on which this dissertation focuses. We start with an informal description of three basic concepts - algorithm design choices, parameters and algorithm configuration - through an example. Next, automatic algorithm configuration, analysis of algorithm components and parameters, and relevant literature are described.

2.1 Optimisation algorithms: design choices and parameters

Given an optimisation problem to be solved, a typical approach for an algorithm designer is to choose a general solution method, such as an algorithm framework, and make necessary specifications in order to get a well-performing algorithm for their particular case. The specifications are usually *design choices*, i.e., generic components defined depending on the choices of algorithm designers and/or the problem being tackled, and *parameters* that control the search behaviour of the algorithm.

To illustrate these concepts, we consider the ACOTSP software package,[1] an implementation of various Ant Colony Optimisation (ACO) algorithms for solving the Travelling Salesman Problem (TSP). The generic solution approach is the Ant Colony Optimisation algorithm framework [33]. Ant Colony Optimisation is a metaheuristic inspired by the way ants in a colony exchange

[1] http://www.aco-metaheuristic.org/aco-code/

information to find the best path between a nest and a food source through indirect communication using pheromone trails. In ACO algorithms, ants are computational agents that incrementally construct a solution in a probabilistic way exploiting pheromone information. Once all ants have completed their solution construction, they leave pheromone trails at each part of their solution to guide future ants. Heuristic information about the problem being solved can also be integrated into the probabilistic construction decisions of ants to guide better the search.

One example of an ACO parameter is the pheromone evaporation rate, a real value in the range of [0,1] that controls the balance between exploitation and exploration of the search procedure. To avoid premature convergence, i.e., to keep the whole colony from converging too fast and end up in some local optimum, pheromone trails are made to "evaporate" over time as a way to forget previous experience and explore new regions of the search space. For large and difficult TSP problem instances with several local optima, high evaporation rates might lead to premature convergence, while for small and easy instances, low evaporation rates might make the algorithm take much more time than necessary to reach high quality solutions.

Starting from the first ACO algorithm - Ant System [34] - several ACO variants have been proposed. Following the same biological inspiration as the original version, the variants have a number of components modified or added, which can be seen as different design choices in the ACO framework [33]. One example is the *elitist* version, where all parts of the best solution ever found by the ants receive extra pheromone trails so that future ants will focus more on those parts. An algorithm designer can choose to use either the original Ant System or this elitist version by switching to the corresponding pheromone update procedure. If the elitist version is used, a new integer parameter defining the amount of extra pheromone has to be added.

The design choices can be represented as the ACO framework's *categorical* parameters, whose each value represents the choice of an ACO algorithm variant. The "child parameters" of each variant, such as the one required by the elitist version described above, will only be activated if its "parent parameter" (the categorical one) is set as "elitist".

Besides design choices purely related to an algorithm framework, the choices of *solution representation*, which is a factor on the problem side, can also be represented as design choices and may have a high impact on algorithm performance [35] [36] [37]. This factor can define which algorithmic design choices are available for certain parts of the algorithm as well [38].

The problem of finding the best design and parameter setting for an algorithm

was first formally defined in [39, 40]. Later on, it was proposed to be called *algorithm configuration problem* [11] as a generalisation of *parameter tuning*, where there are typically few and only numerical parameters to be tuned. The algorithm that has parameters to be configured is called *target algorithm*. The algorithm configuration problem consists in finding a *configuration* (an instantiation of the target algorithm parameter values) that optimises a predefined *performance measure*, such as mean of execution time to reach a certain level of solution quality, or mean of solution quality within a limit of execution time, over a given set of problem instances.

2.2 Automatic algorithm configuration

The process of configuring highly parametrised algorithms is often tedious and computationally expensive. It is basically an optimisation problem where the evaluation of each solution, i.e., configuration, can be very expensive if the given problem instance set is large. For the last fifteen years, several attempts have been made to ease this process and the idea of an automatic configuration of algorithms has been extensively studied [9] [41]. Approaches based on *design of experiments (DOE)* such as CALIBRA [42] and the one in [43] are limited to cases with a relatively small number of parameters due to the number of algorithm runs needed to be sampled. REVAC (Relevance Estimation and Value Calibration) [44] is a semi-automated parameter calibration method initially tested only in the context of evolutionary algorithms [45], a class of algorithms belonging to metaheuristics. This method also assumes a low-dimensional configuration space. When target algorithms only have numerical parameters, state-of-the-art black-box optimisation techniques such as CMA-ES [46] can be used to solve the algorithm configuration task automatically (see, e.g., [47]).

For general cases where parameters can be categorical, ordinal, integer, continuous or a mix of these types and the number of parameters can be relatively large (e.g., hundreds), general-purpose automatic algorithm configuration tools (also called *configurators*) such as *ParamILS* [11], *GGA* [13], *GGA++* [48], *SMAC* [12] and *irace* [10] have been proposed and successfully applied in several cases [9]. For example, ParamILS and SMAC have been shown to speed up the popular, commercial mixed-integer programming solver CPLEX by one or more orders of magnitude on several optimisation problems [49], or irace has been used to configure a multi-objective evolutionary algorithm framework and the resulting algorithms were shown to outperform the state-of-the-arts algorithms for widely used, classical benchmark sets in multi-objective continuous optimization [50].

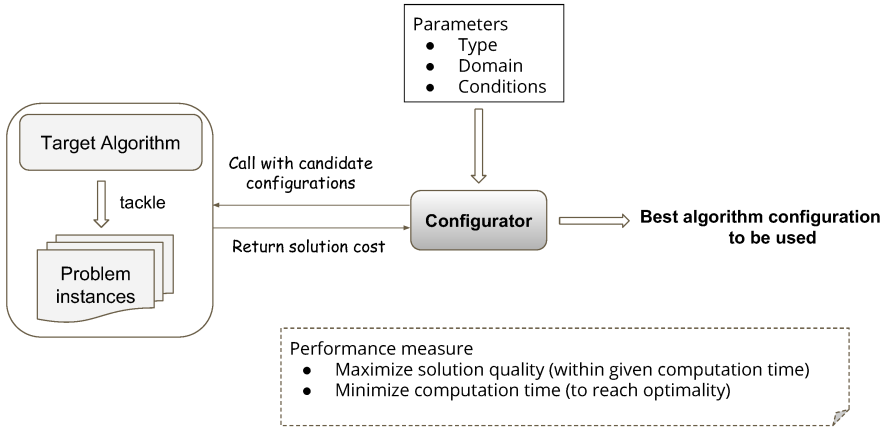


Figure 2.1: Input and output of an automatic algorithm configurator

Input and output of an automatic algorithm configurator are illustrated in Figure 2.1. The configurator receives a target algorithm, its parameter definition, a problem instance set and a performance measure. Parameter definition includes *type* (categorical, ordinal, integer or continuous) and *domain* (range of values) of each parameter. Some parameters can only be activated if their parent parameters are assigned some certain values. These can be expressed as conditions in the parameter definition. The configurator will evaluate candidate configurations by executing the target algorithm on different problem instances until a configuration budget, such as a maximum number of target algorithm runs, is exhausted. It then returns the best configuration found.

Automatic algorithm configuration has offered great benefits to algorithm design, as it releases algorithm developers from the typical dullness of the configuration task. They can now concentrate on collecting and designing potential elements for their algorithm, and let a configurator choose the best design and parameter setting for the particular input data under consideration. Unified algorithm frameworks with several design choices deliberately left open by the algorithm developers are now available, such as MOACO [51] and AutoMOEA [52] for multi-objective optimisation algorithms, or SATenstein [53] for Stochastic Local Search algorithms that solve Propositional Satisfiability (SAT) problems. From these frameworks, a high-performing algorithm instantiation given a specific use context can be generated by an automatic algorithm configurator. This

whole algorithm development process is called the *Programming by Optimization (PbO)* [54] paradigm. Various research topics within this line has been actively progressing and its great impacts on the way we look at how to design an optimisation algorithm can be foreseen.

2.3 Analysis of algorithm components and parameters

Algorithm designers are usually interested not only in finding a good configuration to be used, but also in understanding how different algorithm components and parameters influence algorithm performance. Insights into how an algorithm works may produce useful knowledge that can be transferred back into the algorithm development process in an iterative manner, leading to higher performing algorithms. The knowledge may also provide suggestions for better algorithm design in other application domains or similar contexts [55].

Unlike the algorithm configuration problem, where the aim is to find a good configuration to be used, the analysis of algorithm components and parameters does not have a unique output. Different analysis methods can provide results on different perspectives. They may also require some particular way of sampling algorithm performance data for analyses and make some assumptions on the performance data. It is up to the algorithm designers to choose which method to use at which stage of the algorithm development process, and how to interpret and make use of the analysis results. As an example, the analysis method described in [56] is based on the assumptions that algorithm performance is normally distributed and that it can be explained by additive linear models. Algorithm performance also needs to be specifically collected according to the corresponding experimental design.

Some approaches allow parameter tuning (or more generally, algorithm configuration) and analysis of algorithm parameters at the same time. Information on the importance of algorithm components and parameters is used to guide the tuning procedure. REVAC [44] uses the normalised Shannon entropy in information theory to estimate the relevance of all parameters to algorithm performance. This method assumes smoothness, low-dimensionality and low-modality of the configuration space. Another one is SPOT (Sequential Parameter Optimisation Toolbox) [57, 58]. This is an R toolbox that supports parameter tuning in both interactive and automated ways by using various surrogate models, optimisers and design of experiment approaches. Insights into how different algorithm parameters influence algorithm performance by

means of interactive contour plots are given by exploratory data analysis tools provided by the toolbox.

The scope of the analysis can also vary between different methods. Instead of analysing the whole algorithm configuration space, the ablation analysis proposed in [55] addresses the local region between two algorithm configurations, normally between a default configuration and the best configuration found by the algorithm configuration procedure. The method helps to recognise the contribution of each algorithm component or parameter on performance gain between two configurations under study.

Many analysis methods exploit the idea of learning prediction models from an algorithm performance dataset and then using them for the data analysis on the importance of variables. Among them, there is a group of methods that do not require a specific way of sampling algorithm performance data or particular experimental designs, is able to handle any types of algorithm parameters (including categorical ones), and have been shown to be applicable to cases with a large number of parameters. They include forward selection [21], fANOVA [22] and ablation analysis with surrogates [23]. Forward selection identifies a key subset of algorithm parameters. fANOVA is based on functional analysis of variance [59], where the overall algorithm performance variance is decomposed into components associated with parameter subsets. Ablation analysis with surrogate is a version of the ablation analysis method mentioned above, in which algorithm performance is given by a surrogate instead of running the original algorithm. Due to the general applicability of these methods, they can be used as a next step after the automatic algorithm configuration procedure to give more insights into the decisions of the configurator. More specifically, results of algorithm runs called by the configurator can be given to these analysis methods for building their prediction models. These methods are the main focus of chapter 5, where we illustrate and discuss their combined usage on a number of case studies.

2.4 Conclusion

In this chapter, we have presented a general review on the two main aspects treated in this dissertation: automatic algorithm configuration and analysis of algorithm parameters' influence on algorithm performance. Various works supporting these two aspects of algorithm design have been proposed. Their success in a wide range of applications throughout the last two decades indicates the huge potential of data analytics for algorithm design and many more

researches can be expected. In the next three chapters, our contributions on selected topics within these lines are presented.

Chapter 3

A method for characterising neighbourhood behaviour in local search

For solving complex optimisation problems, local search-based metaheuristic algorithms are among the most common choices. During the development of these algorithms, one of the most important components is the *neighbourhood*, which defines the set of solutions that can be reached from a current solution at each iteration of the algorithm. The exploitation of multiple neighbourhoods has become a popular trend in recent literature. In this chapter, we focus on a multi-neighbourhood local search-based metaheuristic framework where the probability of choosing a neighbourhood at each iteration is defined by a weight-value parameter. The main contribution of the chapter is to propose a general method for characterising the behaviour of neighbourhoods during the optimisation process and recognising similar neighbourhoods without any expert-knowledge. The method is used to *support the automatic algorithm configuration task of the metaheuristic*. The characterisation can reflect the changes of neighbourhood behaviour according to the *hardness of different solution quality regions* based on simple and accumulable statistics collected during any algorithm run, and its validity is illustrated in a case study where the weight-value parameters are tuned using the off-line automatic algorithm configuration tool SMAC [12]. It is shown that the method helps to statistically significantly improve the automatic algorithm configurator's efficiency.

The method proposed in this work does not depend on any specific problems

or local search algorithms, making it generally applicable in similar contexts. An R-implementation of the method is provided at `https://bitbucket.org/nguyendang/neighborhoodcharacterization`.

This chapter is organised as follows. Section 3.1 gives an introduction to the research context, including a brief explanation of the two main concepts used in this work and our motivation. The considered case study for illustrating our proposed method is described in Section 3.2. The characterisation method is presented in Section 3.3. Clustering analysis and its results on the case study are discussed in Section 3.4. Experimental results with the automatic algorithm configuration task on the case study algorithm, which confirms the validity of our method, are shown in Section 3.5. Finally, conclusions are given in Section 3.6.

3.1 Introduction

3.1.1 Local search-based metaheuristics and neighbourhoods

In this section, we introduce the main concepts and the background for the work considered in this chapter. We first describe the two key concepts: *local search-based metaheuristics* and *neighbourhoods*. Next, we emphasise the advantages of using multiple neighbourhoods in local search algorithms for solving complex optimisation problems.

Metaheuristics are often divided into three basic groups [60]: constructive, local search-based and population-based. *Constructive metaheuristics*, such as the Greedy Randomized Adaptive Search Procedure (GRASP), construct a solution by iteratively adding elements into a partial solution, while the other two metaheuristic groups work on complete solutions. In *local search-based metaheuristics*, such as Simulated Annealing [61] or Tabu Search [62], a single solution is considered at each iteration, and the algorithm iteratively makes changes in the current solution during its search for the best one. On the other hand, population-based metaheuristics, such as Genetic Algorithms [63], maintain a set of solutions called population, and iteratively generate new solutions by selecting, combining and updating solutions in the current population. Those three groups of metaheuristics can also be combined, leading to a new group called *hybrid* metaheuristics. An example is the Memetic Algorithm [64], which is basically a genetic algorithm that uses local search for local improvement or repair. Successful applications of metaheuristics belonging to all of those groups have been widely found in both academic and real-world cases [5].

In local search-based metaheuristics, the focus of this work, the set of possible changes made on a current solution at each iteration is defined by a *neighbourhood structure*, or in short, *neighbourhood*. In the literature, there has been a big trend of integrating multiple neighbourhoods into local search-based metaheuristic algorithms (see, e.g., [65], [66], [67]). As explained in [67], there are two advantages of using multiple neighbourhoods: first, they can "overcome the myopic behaviour of a single neighbourhood", due to the fact that a local optimum of a neighbourhood might not be a local optimum of another one; second, it provides "flexibility and adaptability to different problems", i.e., allowing the algorithm to easily adapt to different problem specification requirements. This idea can be extremely beneficial, especially when the tackled optimisation problem is very complex such as different variants of the Vehicle Routing problem [68]. The most representative metaheuristic that exploits multiple neighbourhoods is Variable Neighbourhood Search [69], where neighbourhoods are applied in a pre-defined sequence as soon as the current neighbourhood reaches a local optimum. However, this is not the only way to apply the idea of multiple neighbourhoods. There are various schemes existing in the literature to control the switching between different neighbourhoods at each iteration of the local search algorithm, from the simplest scheme where each neighbourhood is randomly chosen using a uniform distribution (Simple Random) to advanced techniques with learning approaches (see, e.g., *hyper-heuristics* [70]). In this work, we consider a multiple neighbourhood local search framework where each neighbourhood is proportionally chosen according to a pre-defined weight value.

3.1.2 Motivation

In 2013, Donald Ratliff formulated his "10 Rules for Supply Chain & Logistics Optimization" [71]. In short, he pleads for quantified and measurable objectives, faithful and essential models, taking variability explicitly into account, rely on accurate, timely and comprehensive data fully automated and connected, supporting execution management and control. Algorithms must exploit the individual problem structure and people with expertise in models, data and optimisation must be involved. Processes must support optimisation and continuously improve and there must be a provable return on investment taking into account costs of technology, people and operations.

Although stated from a specific application domain, the rules obviously have a general significance and they rightly identify a number of shortcomings or room for improvement. Concerning optimisation algorithms, exploitation of individual problem structure has been a target in optimisation research for at least 70 years, be it that as important has been the control of computational

complexity. In metaheuristics based on local search [72], neighbourhoods can often be suggested by domain experts involved in the application area that needs optimisation. While such people often acquire a general understanding of the algorithmic methodology, they are not always optimisation experts. Experience with metaheuristic development shows that performance often strongly relies on smart implementation decisions. Methods that can support the understanding of neighbourhood behaviour are then beneficial for the development of those algorithms. One example is the work in [73], where a hyper-heuristic framework is used as an analysis tool for understanding behaviour of underlying low-level heuristics.

Another observation recently made is that algorithm performance crucially depends on the application environment. Efficiency of an algorithm in a specific environment must be measured against relevant data representative of this environment. When looking at the neighbourhood-based metaheuristics, the functionality of the neighbourhoods and their appropriateness depends on data. This observation has led to the discipline of *automatic parameter tuning*, or more generally *automatic algorithm configuration*, where methods have been developed to effectively adapt the algorithm to the distribution of problem instances it is expected to encounter. General-purpose automatic algorithm configuration tools such as SMAC [12] and irace [10] have been successfully applied in several studies [9].

In this chapter, we consider a multi-neighbourhood scheme where each neighbourhood is assigned a fixed weight value, representing the probability that the corresponding neighbourhood is chosen at each iteration of the local search. These weight values are considered as the algorithm parameters and can be tuned automatically using any automatic algorithm configuration tool. However, if the number of neighbourhoods is large, as illustrated in our case study, the large number of parameters might deteriorate the algorithm configurator efficiency. A potential solution is to cluster the neighbourhoods into groups and assign a weight-value parameter to each group. It can help to reduce the parameter space so that the available configuration budget is used more efficiently. The key question raised while doing such a clustering is how to characterise the behaviour of each neighbourhood over a given set of problem instances and represent them as a feature vector. This motivates our proposal of a characterisation method based on simple statistics that can be easily collected throughout a small number of algorithm runs before the expensive tuning experiment is started. We demonstrate the validity of our method on a local search-based metaheuristics with 42 neighbourhoods for solving the Swap-Body Vehicle Routing Problem (SB-VRP) [74]. The automatic algorithm configurator used in this case study is SMAC [12].

3.2 A case study on a multi-neighbourhood local search-based metaheuristic

The algorithm [75] considered in our case study is the winner of *the First VeRoLog Solver Challenge* [74]. This is an Iterated Local Search algorithm [76] which includes a local search component for finding a good solution and a perturbation component for escaping from local optima. The Late Acceptance Hill Climbing algorithm [77, 78] is used as the local search component. The algorithm tackles the Swap-Body Vehicle Routing Problem, a real world-based generalization of the classical Vehicle Routing Problem (VRP) proposed by the VeRoLog challenge. The problem allows the capacity and size of a vehicle to be increased or decreased during a route via the addition or the removal of a swap body (an additional compartment for carrying goods) at a swap location (a place where the vehicle can park or pick up a swap body).

Several neighbourhood structures are integrated into the algorithm, ranging from general ones taken from the VRP literature (e.g., two-opt by [79]) to ones specifically designed for the Swap-Body features (e.g., changing swap locations). Some neighbourhood structures can be parametrised by their sizes. For example, in cheapest insertion neighbourhood structure, the neighbourhood size is the number of customers that are removed from the current solution and then being sequentially re-inserted back into the positions that cause the least increase in solution cost. A *neighbourhood* is an instantiation of a neighbourhood structure, i.e., a specific value is assigned to the size parameter if the neighbourhood structure is parameterised. A neighbourhood structure with a size-value instantiation, or a non-parameterized neighbourhood structure is called a *neighbourhood*.

Table 3.1 lists the 42 neighbourhoods used in the algorithm. We left out their detailed description, due to the fact that the neighbourhood behaviour characterisation method that we propose works at an abstract level, i.e., no specific knowledge about the neighbourhoods themselves is required. For details about the algorithm and the neighbourhoods, the readers are referred to [75].

At each iteration of the local search, a neighbourhood is selected randomly with a probability proportional to a pre-defined weight value. These weights in combination with two integer parameters for the Late Acceptance Hill Climbing algorithm, namely the size of the late acceptance list (*list_size*) and the maximum number of consecutive iterations without improvement (*max_nonimproving_iterations*), result in 44 algorithm parameters to be configured.

The number of weight-value parameters can be manually reduced using

some intuition. Some neighbourhoods having similar sizes within the same neighbourhood structure can be grouped together and assigned one weight-value. The probability of being chosen is then equally distributed among all neighbourhoods in the group. For example, the cheapest insertion neighbourhood with the sizes of 1, 2, 3, 4, 5 can be treated as one group, since the change expected when they are applied to a solution is generally small compared to the ones with larger sizes (e.g., 10 or 15). Each line in Table 3.1 represents a list of sizes that are grouped together (28 groups in total). This manual clustering was learnt from the setting provided by the target algorithm developers and is expected to make the automatic algorithm configuration more efficient compared to the 42-weight-value version.

We use the six large problem instances provided before the final phase of the challenge. Each run of the algorithm on an instance takes 600 seconds. Note that the algorithm version used in this work is not exactly the same as the one described in [75], as our aim is not to compete with other solvers, but to use this case study as a proof of concept for the validity of our proposed neighbourhood characterisation and clustering method. In particular, the one presented in [75] is a muli-threaded algorithm that uses a learning automaton to update the weights, while the version considered here, which was actually the original algorithm that entered the VeRoLog challenge, uses static weight values. We also simplify the algorithm by removing the multi-threaded property since each thread was run completely independently with the others.

3.3 Neighbourhood characterisation

In this section, we describe our neighbourhood characterisation method and illustrate it in the case study explained in the previous section. The main aim of the characterisation is to represent the behaviour of each neighbourhood on the given set of problem instances as a feature vector using information collected from a small number of algorithm runs. Given a neighbourhood N_k and a problem instance I, the following six characteristics, whose different forms or variations have been widely used in the hyper-heuristic community (see, e.g., [73, 80]), are used:

- The probabilities that N_k improves, worsens and does nothing on a solution of I, denoted by $r_{improve}$, r_{worsen} and $r_{nothing}$, respectively, where

$$r_{improve} + r_{worsen} + r_{nothing} = 1$$

- The magnitudes of improvement and worsening, denoted by $a_{improve}$ and a_{worsen}, respectively

Neighbourhoods with size		Neighbourhoods without size
Name	Size	Name
Cheapest-insertion	1; 2; 3; 4; 5	Swap
	10; 15	Intra-route-two-opt
	20; 25	Inter-route-two-opt
	35	Merge-route
	50	Remove-route
Ejection-chain	3; 4; 5	Convert-to-route
	10	Add-sub-route
	15	Split-to-sub-routes
	35	Convert-to-sub-route
Remove-chain	1; 2; 3; 4	Remove-sub-route
	5; 6	Remove-sub-route-with-cheapest-insertion
	7; 8	Change-swap-location
Each-sequence-cheapest-insertion	(2,5)	
	(5,2)	
	(4,4)	
Ruin-recreate	2; 3	

Table 3.1: Names and sizes of the 42 neighbourhoods used in our case study algorithm. In the left column, each neighbourhood is defined by its neighbourhood structure name and an instantiation of the size (each size of a neighbourhood is a single value, except for neighbourhood Each-sequence-cheapest-insertion, whose size is a pair of values). In the right column, the neighbourhoods without sizes are listed. The table also shows the manual grouping of neighbourhoods having similar size values within the same neighbourhood structure. Each group is represented by a line in the left column. For the neighbourhoods without sizes, each one is treated as a group itself. This grouping, which helps to reduce the number of weight-value parameters to 28, is learned from the setting provided by the algorithm developers.

- N_k's running time (this characteristic is only used indirectly, i.e., for tie-breaking, as will be explained in detail later)

The values of those characteristics normally vary among different regions of the solution space. For example, the probability that a neighbourhood improves a given solution (i.e., $r_{improve}$) is probably low if the quality of the solution is already good, and vice versa. This variance will be captured by the characterisation method to represent the behaviour of each neighbourhood on a problem instance. Before going to the details of how the method works, we first illustrate the changes of those characteristics across different solution quality regions by two examples in Figure 3.1, in which the values of the first three characteristics $r_{improve}$, r_{worsen} and $r_{nothing}$ for four neighbourhoods on a problem instance are plotted. In this figure, solutions (x-axis) are sorted

according to their quality (i.e., their objective function values): the larger the value in the x-axis, the better the quality of the associated solutions. In the first example (Figure 3.1a and 3.1b), we can see that when solution quality is low, i.e., the local search is in easy-to-improve regions of the solution quality space, both neighbourhoods Merge-route and Remove-route have very high probabilities of improving the given solutions. However, when the two neighbourhoods reach the good solution quality regions, their behaviour become very different. While Merge-route's probability of worsening solutions starts reaching one, Remove-route tends to preserve the quality of the given solutions. The second example (Figure 3.1c and 3.1d) shows that even neighbourhoods belonging to the same neighbourhood type can behave differently among regions. The small Cheapest-insertion neighbourhood (with size 2) has much smaller probabilities of worsening the given solutions in the hard-to-improve region compared to the large Cheapest-insertion neighbourhood (with size 25).

In order to quantify those variances, our characterisation method first collects some raw information about each neighbourhood to divide the whole range of solution quality values into different regions. After that, the characteristic values on each region are extracted. In this way, the characterisation is expected to reflect the changes of neighbourhood behaviour according to different *hardness* levels of solution quality space. This definition of hardness based on only solution quality values might be a bit of oversimplification compared to other approaches such as fitness landscape analysis [81], as other factors related to the structure of the solution space might have important roles in indicating the hardness of a solution. Nevertheless, the advantage of such a simplification is that it does not require any problem-specific knowledge.

The overall procedure of the characterisation is as follows. We first pick some algorithm configurations and run them multiple times on the given problem instances. During those runs, a number of simple and accumulable statistics are collected for each pair of neighbourhoods and problem instances. The whole range of solution quality values for each problem instance is then automatically partitioned into different regions. Afterwards, the characteristics of each neighbourhood are calculated by aggregating all the collected statistic values on every solution quality region.

3.3.1 Step 1: collect simple and accumulable statistics

We define a number of raw statistics that can be easily collected when an algorithm configuration (i.e., an instantiation of the algorithm's parameters, in our case, this is a parameter setting for the 42 weight values and the two additional integer parameters for the local search) is run on a problem instance.

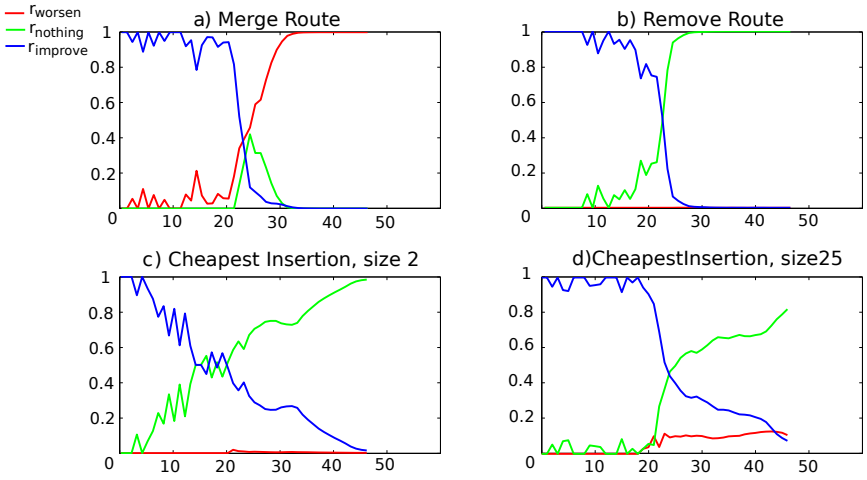


Figure 3.1: Visualization of $r_{improve}$, r_{worsen} and $r_{nothing}$ for four neighbourhoods on problem instance *new_without*. The x-axis represents solution quality (the larger the value, the better the corresponding solutions). The y-axis shows the values of the three characteristics.

These statistics are accumulable among different algorithm runs. We assume that an upper bound and a lower bound of the optimal solution quality for each problem instance are available. These bounds do not need to be tight, i.e., they can be the worst and the best solution quality values we have known so far. The characterisation of the neighbourhoods will be done on the range between these two bounds, so if the bounds are not accurate, the missing ranges are just simply ignored.

In our case study, the upper bound for each instance is obtained from a solution in which one route is assigned to each customer. This is also the initial solution of the algorithm. The lower bound for each instance, which is provided by the authors of the algorithm, is the best solution obtained from running their multi-threaded version in six hours.

We first divide the range between the upper bound and the lower bound into a large number of small intervals. Because higher-quality solutions in general are harder to improve, we let the size of the intervals decrease exponentially by a factor $\alpha < 1$. The size of the first interval is then calculated using Equation 3.1, in which L is the size of the whole range and n is the total number of intervals.

$$s_0 = L \times \frac{(1-\alpha)}{1-\alpha^n} \tag{3.1}$$

Ideally we would want s_0 to be independent of n, i.e., n should be very large. In this study, we set $n = 1000$, which is a sufficiently large number with which the studied algorithm, if being run within ten minutes in our experiment, never reaches the last two hundred intervals. α is set as 0.99.

Now during each algorithm run, every time a neighbourhood N_k is applied on a solution whose quality value belongs to an interval I_j, the following values are collected for the pair of (N_k, I_j):

- n_{iters}: the number of times N_k is applied I_j,
- n_I, n_{SN}, n_W: the numbers of times N_k improves, does nothing, or worsens solutions belonging to I_j, respectively
- s_I, s_W: the sums of the amount (i.e., the changes in the objective function values) of the improvement and the worsening, respectively
- s_{time}: the sum of N_k's running time.

Those values can be accumulated among different algorithm runs. The more runs there are, the better the estimated values of the characteristics. In our case study, we collect them by running two algorithm configurations (one with all weight values set as equal, and one with slightly different weight values; in both configurations, the local search's integer parameters are set as default values provided by the algorithm's authors) on all instances, with 10 runs per instance, so the total number of algorithm runs is 120. We use longer running time, 900 seconds instead of 600 seconds per run (this value was chosen arbitrarily), so that the collected information may have more chance to cover the hard parts of the solution quality space.

3.3.2 Step 2: identify solution quality regions

We partition the whole range of solution quality values into a pre-defined number of regions, each of which represents a certain level of the hardness of the solution quality space. In the previous step, we counted several characteristics according to our interval definition, which in fact results in an approximate logarithmic scale for the objective function range. The partition in this step is done by grouping the intervals together based on *sum_nIters*, the sum of all neighbourhoods' n_{iters} values on each interval. This value represents the visiting

frequency of each interval during the optimisation process. The grouping is based on the observation that the algorithm normally spends most of the time in intervals close to local optima or at plateaus. As an example, Figure 3.2 shows plots of *sum_nIters* on each instance in our case study. Note that because the lower bounds on solution quality are not reached during our collecting algorithm runs, intervals with zero *sum_nIters* at the end of the x-axis are removed. In this Figure, the high peak in every plot represents the intervals where the algorithm stays most of the time. We thus conjecture that local optima or plateaus should lie there. We can interpret the solution quality regions with low *sum_nIters* values before the peak as easy-to-reach and easy-to-escape, regions around that peak as easy-to-reach and hard-to-escape, and regions after that peak as hard-to-reach. The smaller peaks in the plots of the two instances `new_with` and `large_with` should indicate a second local optimum or plateau.

The grouping of the intervals into regions should reflect the interpretation described above. The idea is simply to roughly equalize the visited frequency among regions, while putting more emphasis towards the hard parts of the solution quality space, i.e., around and after the first peak. A pseudo-code of the grouping algorithm is given in Algorithm 1. Given a desired number of regions n, the grouping procedure is as follows. We first eliminate all the intervals at the end of the solution quality range which the collecting algorithm runs never reach. Next, the average number of intervals per region avg is calculated. Starting from the beginning of the solution quality range, consecutive intervals are combined into regions each of whose sizes (i.e., sum of n_{iters}) does not exceed avg, except the case where the size of an interval itself is not smaller than avg (then the interval will be a region). If the final total number of regions formed is larger than n, the first and the second regions are merged, followed by the combination of the third and the fourth, and so on. This is repeated until the total number of regions is equal to n.

Figure 3.3 shows results of the partitioning algorithm on the six problem instances in our case study with the number of regions $n = 5$, where we can see that the partitioning is finer on the parts around and after the peaks.

3.3.3 Step 3: characterise neighbourhood behaviour as feature vectors

In this step, neighbourhood characteristics are extracted by aggregating the collected raw statistics of all intervals on each region. The first three characteristics, the probability that a neighbourhood improves ($r_{improve}$), worsens (r_{worsen}) or does nothing ($r_{nothing}$) on a solution, can be easily calculated as the mean values (per iteration) of n_I, n_W and n_{SN}, respectively,

Algorithm 1 Group all solution quality intervals in the previous step into n regions

Require:

n: the number of regions we want to partition the whole solution quality range into,

$n_{intervals}$: the number of intervals after removing empty ending intervals,

A: an array of $n_{intervals}$ elements, each of which is the *sum_nIters* value of the corresponding interval.

Ensure:

E: an array of n elements, each of which contains the index of the last interval belonging to each region.

$r \leftarrow 0.05$, $avg \leftarrow \lfloor \sum_{i=1}^{n_{intervals}} A[i]/n \rfloor$
while *true* **do**
  $E \leftarrow \emptyset, i \leftarrow 1, l \leftarrow avg * (1 + r)$
  while $i \leq n_{interval}$ **do**
    if $A[i] \geq l$ **then**
      $E \leftarrow E \cup \{i\}$, $i \leftarrow i + 1$
    else
      Let k be the largest value such that $\sum_{j=i}^{k} A[j] \leq l$
      $E \leftarrow E \cup \{k\}$
      $i \leftarrow k + 1$
    end if
  end while
  if E contains less than n elements **then**
    $r \leftarrow r - 0.01$
  else
    if E contains more than n elements **then**
      Starting from the first region in E, combine every pair of regions into one (repeat from the first new region if necessary) until only n elements are left.
    end if
    break
  end if
end while
Return E

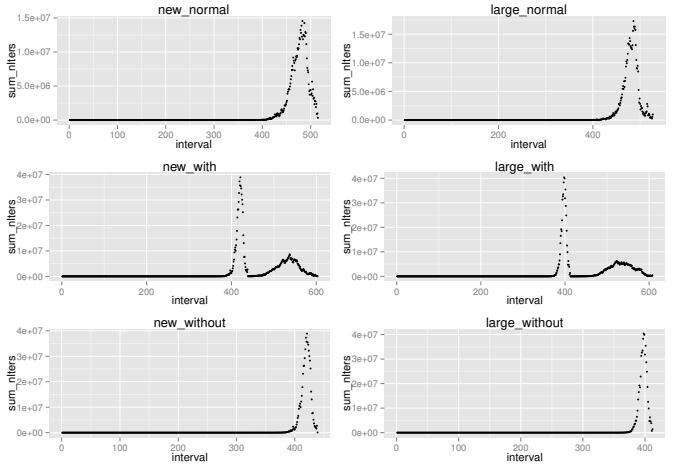


Figure 3.2: Plots of *sum_nIters*, the sum of n_{iters} values of all neighbourhood on each solution quality interval for each of the six instances used in our case study. The x-axis represents the intervals. Although the original number of intervals is 1000, many of them at the end of the range are not reached during the collecting procedure and are eliminated from the x-axis.

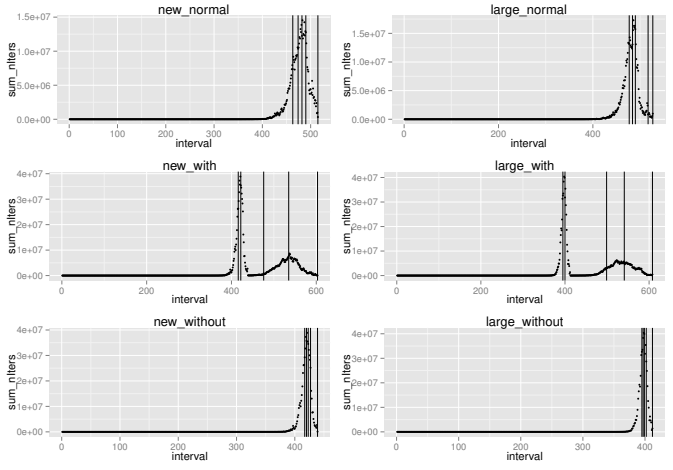


Figure 3.3: Plots of *sum_nIters*, the sum of n_{iters} values of all neighbourhood on each solution quality interval for each instance in our case study and results of the partitioning with the number of regions $n = 5$. The vertical lines in each plot indicate the boundaries between the five resulting solution quality regions.

of all intervals belonging to the same region (for example, $r_{improve}$ of a neighbourhood N_k on a region is calculated by dividing the sum of all n_I values of N_k by the sum of *sum_nIters* on all intervals of the region).

For the other two characteristics, the magnitudes of improvement ($a_{improve}$) and worsening (a_{worsen}), the aggregation is more complicated. We cannot sum the amount of improvement s_I or worsening s_W over different intervals and get the average due to the fact that their values are incomparable among different intervals. For example, we cannot say that an amount of improvement of 10 in the two intervals $[33762, 33621)$ and $[33621, 33482)$ is equal since the hardness of solutions belonging to those intervals is probably not the same. Therefore, we first calculate their average value (iteration-based) on each interval. Next, these values are converted into ranks, which are then used for the aggregation into each region. Because ties may occur, e.g., some neighbourhoods might never make any improvement in the hard solution quality regions, the mean values of neighbourhoods' running time (s_{time}) in the corresponding interval are used for tie-breaking. Finally, since the intervals are fine-grained, the resulting ranking list for each neighbourhood is possibly:

- noisy: for some intervals, the number of times some neighbourhoods are applied (i.e., n_{iters}) might be very small; it means we have little information about these neighbourhoods on these intervals, so that it might be inaccurate to use the corresponding rank values to estimate $a_{improve}$ and a_{worsen},
- partial: n_{iters} of some neighbourhoods might be even zero on some intervals, i.e., we do not have information of those neighbourhoods in those intervals (missing entries).

Therefore, we aggregate the ranking lists of all neigborhoods on each region using the R package *RobustRankAggreg* [82] - a robust ranking aggregation method specially designed for similar situations in bioinformatics. The method takes into account the possibly noisy and partial properties of its input.

In our case study, after the characterisation, we obtain for each neighbourhood a feature vector composed of 150 components, which is the combination of 5 characteristics (described at the beginning of section 3.3, except the running time characteristic, which was used for tie-breaking), 5 regions and 6 problem instances. Now any clustering technique can be applied to identify similar neighbourhoods.

Note that the characteristic extraction based on solution quality regions, as presented in this section, instead of the interval themselves is crucial for reasonable clustering results. As the intervals are fine-grained, especially at the

high-solution-quality parts, the number of applications of each neighbourhood can be very small or even be equal to zero on a considerable number of intervals. If the observables are calculated directly on each interval, we get issues similar to the ones with the ranking lists desribed above, i.e., the obtained values are likely inaccurate or even missing. Moreover, the aggregation based on regions also helps to avoid biased information during the clustering. As seen in the example in Figure 3.3, the lengths of the easy and hard regions among various instances are very different. If the clustering is based on the raw data (i.e., interval-based), the resulting clusters might be dominated by the information of the easy regions, which contain many intervals. Moreover, the lengths of these easy and hard regions can also vary among different instances, which potentially leads to another bias.

3.4 Neighbourhood clustering

We need a small preprocessing step before applying a clustering method to the extracted feature vectors. The first three characteristics, $r_{improve}$, r_{worsen} and $r_{nothing}$, sum up to one. As a result, their corresponding vector components belong to a special class called *compositional data*. As explained in [83]: "sample space for compositional vectors is radically different from the real Euclidean space associated with unconstrained data", multivariate statistical methods designed for unconstrained data cannot be applied directly. To convert them back to the Euclidean space, we apply the *isometric log-ratio transformation* proposed in [84]. After the transformation, the three characteristics are reduced to two, therefore, each feature vector is now 120-dimensional.

Now we can start doing cluster analysis on the given neighbourhoods. In our case study, since the number of dimensions (120) is larger than the number of individuals (42 neighbourhoods), the clustering method *High-Dimensional Data Clustering* [85]), which is implemented in the R package *HDclassif* [86], is used. This method has two desirable properties: the ability of dealing with high dimensional and low sample size data, and the optimal number of clusters automatically decided based on the *Bayesian Information Criterion* [87]. Eventually, the 42 neighbourhoods are grouped into nine clusters:

1. Ejection-chain 3, 4, 5; Remove-chain 1, 2, 3, 6, 7, 8; Remove-sub-route-with-cheapest-insertion;
2. Swap; Inter-route-two-opt
3. Cheapest-insertion 10, 15, 20, 25, 35, 50; Each-sequence-cheapest-insertion (2,5), (4,4), (5,2); Remove-chain 4

4. Cheapest-insertion 1, 2, 3, 4, 5
5. Change-swap-location; Merge-route
6. Add-sub-route; Convert-to-sub-route
7. Ejection-chain 10, 15, 35; Remove-chain 5; Intra-route-two-opt
8. Ruin-recreate 2, 3
9. Convert-to-route; Remove-sub-route; Remove-route; Split-to-sub-route

The resulting clusters reflect information that can be intuitively obtained by manually looking at the neighbourhood behaviour plots or their source code implementation. For example, the two neighbourhoods Merge-route and Remove-route behave quite differently in the second-half region as shown in Figure 3.1, and they are indeed clustered into two different groups. Another example is that by taking a look into the neighbourhood implementation, we know Add-sub-route and Convert-to-sub-route have an extreme behaviour when compared to the others: they will add an additional cost into the current solution and worsen it most of the time (it can also be seen in the plots of their r_{worsen} values). And the cluster analysis does recognize this extremeness, as the two neighbourhoods are grouped into a separated cluster.

Finally, the clustering also introduces some similarity between neighbourhood behaviour that is not intuitive from the neighbourhood structure point of view, such as the grouping of Ejection-chain 10, 15, 35 and Intra-route-two-opt. Since the clustering is based on lots of information (the number of dimensions is much larger than the number of individuals), those non-intuitive grouping, will always be hard to explain. However, the intuitive grouping described above gives us some idea about the validity of the clustering, so that we expect the non-intuitive ones to be feasible, i.e., they do reflect the neighbourhoods behaviour on the specific instance set in the same sense as the intuitive groupings do.

3.5 Experimental results

To verify the validity of our neighbourhood characterisation and clustering method, we compare results when applying the automatic algorithm configuration tool SMAC [12] to the considered algorithm in three scenarios:

- `original`: all neighbourhoods are treated independently, resulting in a number of 42 weight-value parameters for the algorithm.

- `basic`: neighbourhoods having similar sizes within the same neighbourhood structure are manually grouped together as presented in Table 3.1, i.e., the number weight-values parameters is reduced to 28.
- `clustered`: each neighbourhood cluster generated by our method is assigned a common weight, leading to 9 weight-value parameters in total.

The automatic algorithm configuration task consists of tuning the weight-value parameters, which are continuous parameters in the range of $[0, 1]$, together with two integer parameters of the Late Acceptance Hill Climbing step (the size of the late acceptance list *list_size* and the maximum number of consecutive iterations without improvement *max_nonimproving_iterations*).

If the characterisation method correctly represents the neighbourhood behaviour on the given problem instances, the clustering results should help to improve the algorithm configurator's efficiency. To test this hypothesis, eighteen runs of SMAC are carried out on each scenario. A budget of 2000 algorithm runs (i.e., 13.9 CPU days) is given to each SMAC run. To reduce the wall-time required, the shared-model mode provided by SMAC is used with ten cores, i.e., each SMAC run with a budget of 2000 algorithm runs is actually ten dependent SMAC runs, whose budget is 200 algorithm runs each, and the algorithm configuration with the best training performance among these runs is taken as the final one. In this way, each SMAC run can be finished within 1.39 days (wall-time).

Performance of each algorithm run on each instance is measured using the percentage optimality gap calculated by equation 3.2

$$optimalityGap = 100 * (solutionCost - bestCost)/bestCost \tag{3.2}$$

where *bestCost* is the best solution quality we know so far for the corresponding instance. This is provided by the algorithm developers, and was obtained by running the multi-threaded version of the algorithm on each instance for a long running time limit of six hours.

At the end of each SMAC run, the final best algorithm configuration is taken and its *test performance* is measured as the mean of the optimality gap over all problem instances throughout 30 runs (5 runs/instance). The test performance values obtained from all eighteen SMAC runs on each scenario are plotted in Figure 3.4. The *p-values* given by applying the *pairwise Student's t-test* on each pair of scenarios are also reported. As expected, the manual grouping generally gives better performance than the *original* setting. However, the difference is not statistically significant. The clustering results, on the other hand, offer

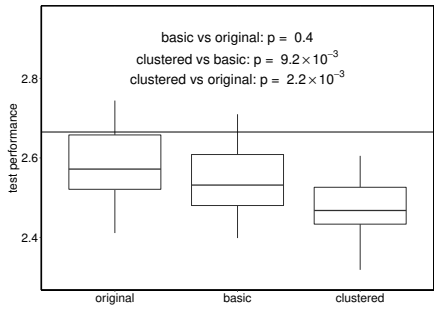


Figure 3.4: The box-plots show the test performance of the best parameter settings obtained from 18 runs of SMAC on three scenarios: original (no grouping of neighbourhoods), basic (manual neighbourhood groupings) and clustered (neighbourhood clustering using our characterisation method), whose the number of weight-value parameters are 42, 28 and 9 respectively. The horizontal line is the test performance of the default parameter configuration, whose weight-value parameters are set identically. The p-values are obtained from pairwise t-tests.

statistically significant improvement over both *original* and *basic* scenarios with a confidence level of 99%. This confirms the advantage of using the clustering and the validity of our characterisation method.

In the hyper-heuristic community, in particular the Selection Hyper-heuristic class [88] where the aim is to manage a set of low-level heuristics during the search by selecting one of them at each iteration, the *Simple Random (SR)* heuristic selection mechanism is often used as a baseline [18]. In our setting, SR is equivalent to the algorithm configuration that has identical weights for all neighbourhoods. We compare SR with the resulting configurations obtained from the automatic algorithm configuration: for each scenario, the 18 best configurations are taken and the neighbourhood weights inside them are set to identical. Their test performance values are shown in Figure 3.5 as `original with identical weights`, `basic with identical weights`, and `clustered with identical weights`. The horizontal line represents the test performance of the algorithm configuration in which all neighbourhood weights are identical and `nbConsecutiveIterations` and `listSize` are set to values suggested by the algorithm developers. This configuration is also used as the default configuration for all SMAC runs mentioned above. We can see that the SR versions produce worse test performance in all scenarios. A pairwise *t-test* is conducted for each scenario and the corresponding p-value is shown in Figure 3.5. The p-value of the t-test between *clustered with identical weight* and *clustered*

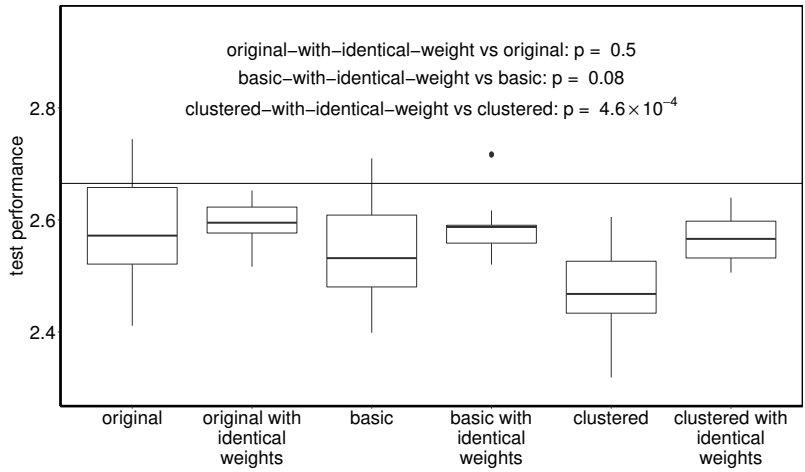


Figure 3.5: The box-plots show the test performance of two sets of algorithm configurations. The first one consists of the best configurations obtained from 18 runs of SMAC on three scenarios: original (no grouping of neighbourhoods), basic (manual neighbourhood groupings) and clustered (neighbourhood clustering using our characterisation method), whose number of weight-value parameters are 42, 28 and 9 respectively. The second one includes the Simple Random versions of the configurations from the first set. The horizontal line is the test performance of the default algorithm configuration.

indicates that the neighbourhood weight-values have influence on the algorithm performance, as tuning them can provide statistically significant performance improvement. The tests also reflect the hardness of tuning those weights, as the algorithm configuration results on `original` and `basic` fail to show statistically significant improvement over their corresponding identical-weight versions, and emphasize the advantage of `clustered` over `original` and `basic`.

3.6 Conclusion

In this chapter, we have proposed a systematic method to characterise neighbourhood behaviour in a multi-neighbourhood local search framework, where the probability of choosing a neighbourhood at each iteration is decided in an off-line manner. The characterisation is based on the probabilities that a neighbourhood will improve, worsen or do nothing on a solution, on the magnitudes of its improvement and worsening, and on its running time. We have

observed that these characteristics change according to the hardness of different regions in solution quality space. As a result, we design our method such that it can detect these regions based on a number of simple and accumulative statistics, and represent neighbourhood behaviour on them as feature vectors. Cluster analysis is then applied to form groups of similar neighbourhoods. A parameter tuning experiment with the automatic algorithm configuration tool SMAC [12] shows that using these clusters gives statistically significant improvement on the performances of the obtained algorithm configurations over the non-clustered versions, which supports the hypothesis that our characterisation method is able to correctly reflect neighbourhood behaviour on the given instance set.

Since the information used in this method does not depend on any specific problem, the characterisation and clustering procedure can be generally applied in similar contexts to support the automatic algorithm configuration task. An R implementation of the method is publicly available at `https://bitbucket.org/nguyendang/neighborhoodcharacterization`, where given the raw statistics collected by users (section 3.3.1), the feature extraction and the clustering are done automatically.

In future work, as the current case study has a limited number of problem instances, problems with larger instance sets, where the clustering method may have to deal with very high-dimensional data, should be investigated. Moreover, the role of neighbourhood analysis as part of a broader algorithm design methodology could also be studied. The analysis could provide information to the designer about where to look for the next improvement, hence providing assistance to support application domain experts in realising high performing heuristic algorithms.

In this chapter, the automatic parameter tuning task of the metaheuristic scheme being supported by the method we proposed is a special case of the *automatic algorithm configuration* problem. The main difference is that while the parameters of the current case have integer and continuous domains, in the general problem, they can also be categorical or ordinal, and conditions on the activation of parameters are allowed. In the next chapter, we will move to the general one, and study the usage of data analytics to support the automatic algorithm configuration task by looking at the configurators themselves as optimisation parametrised algorithms.

Chapter 4

Configuring irace using surrogate configuration benchmarks

Automatic algorithm configurators themselves also have parameters that may influence their search behaviour and make them malleable to different kinds of configuration tasks. Current default settings are defined manually based on experience of configurator developers. Studying the impact of these parameters or configuring them is very expensive as it requires many executions of these tools on algorithm configuration tasks, each of which often takes several hours or days of computation. In this work, we address this problem using a meta-tuning process based on surrogate benchmarks that are much faster to evaluate. This chapter studies the feasibility of this process on irace, one of the most popular configurators in the literature. Experimental results indicate the feasibility of this process and a clear potential for the improvement of irace over its default configuration. This is the first systematic effort towards improving irace performance by configuring its parameters automatically on several configuration tasks with different characteristics. Since automatic algorithm configurators have a wide range of applications and provide a great support during the algorithm design process, making these tools work more efficiently in general will offer benefits to the algorithm community.

This chapter is organised as follows. An introduction to the context of this work, including the difficulty in studying configurators' parameters, the surrogate modelling technique and our motivation, are given in Section 4.1. irace and its

parameters are explained in Section 4.2. The surrogate configuration benchmarks used in this study and three different analyses on their consistency with the real benchmarks are described in Section 4.3. Experimental results of the meta-tuning on irace parameters using surrogate configuration benchmarks are shown in Section 4.4. As a side product of the meta-tuning, analyses on the importance of different irace parameters on the configurator performance over the studied surrogate benchmarks are conducted in Section 4.5. To verify the performance of the tuned irace parameter settings, experiments on a number of real benchmarks are carried out in Section 4.6. Finally, conclusions and future work are given in Section 4.6.

4.1 Introduction

4.1.1 Automatic algorithm configurators and their parameters

Automatic algorithm configurators are handy and useful for configuring algorithms, and their effectiveness in improving the performance of optimisation algorithms have been shown through several applications [9]. However, these configurators themselves also have a number of parameters that must be set. Examples of these parameters are the initial population size of the genetic algorithm-based configurator GGA [13], or the perturbation strength in ParamILS [11]. Users can use the default parameter values recommended by the configurator developers, or modify them according to the characteristics of their configuration scenarios, e.g. their configuration budget and the homogeneity of their instance set. Given these parameters, the obvious question is how to set them to optimise the performance of the configurator being used, that is, make it as high performing as possible when tackling configuration tasks. A straightforward idea is to first define a set of algorithm configuration benchmarks, each of which consists of a parameter space of a target algorithm, a set of training instances of the optimisation problem under concern, and a configuration budget, then to configure the parameters of the considered configurator for those benchmarks by means of an algorithm configurator, a process we call *meta-tuning*. However, this approach is normally extremely computationally-expensive, if feasible at all. For example, we consider the popular configurator irace, which has 9 parameters, and a configuration benchmark namely SPEAR-IBM [89], which involves the tree search solver SPEAR [90] for solving a set of SAT problem instances taken from the IBM Formal Verification Benchmarks Library [91]. One configuration process for this benchmark is often associated with about two days of CPU time for the execution of the solver [89]. Consequently, configuring irace's parameters on it with a meta-tuning budget of 2000 irace

evaluations would take about 11 CPU years on a single core. This amount of CPU time is not really feasible, even with the help of parallel processors.

Due to the computationally expensive computational requirement of such kind of study, there are only a few works with restricted conditions in the literature that have investigated the impact of automatic algorithm configurator's parameters. In [92], a limited number of irace's parameter settings were studied using three configuration benchmarks. In [11], the configurator ParamILS was used to configure itself with a fraction of the usual configuration budgets used. The common conclusion from those studies is that the current default parameter settings of those configurators are already sufficiently robust, as no significantly better configurations have been found.

4.1.2 Surrogate modelling

The idea of surrogate modelling is to build a regression model to predict performance of an algorithm configuration on each problem instance. This prediction model, called *empirical hardness model* [93] or *empirical performance model* [94], is then used as a surrogate for the algorithm. Surrogate modelling approaches have been used in a variety of applications where the evaluation is very costly. In continuous optimisation, the EGO algorithm [95] is a well-known example. More recent is the SPOT package [58], which has also been used for the configuration of algorithm parameters. Surrogate models are also used in several applications of algorithm selection, automatic algorithm configuration and algorithm parameter analysis. In algorithm selection, empirical hardness models are used as the core of the well-known SAT algorithm selector SATzilla [15]. For analysing the importance of algorithm parameters, they are used to identify a subset of key algorithm parameters through forward selection [21], to build data for the functional analysis of variance toolbox *fANOVA* [22]; or as a surrogate for an efficient ablation analysis [23]. In the context of automatic algorithm configuration, empirical performance models are used to guide the search of the model-based algorithm configurators *SMAC* [12] and *GGA++* [48]. Finally, the usage of surrogate modelling in [96], which inspires our work in this chapter, is an alternative for real benchmarks to study hyper-parameter optimisers in machine learning.

4.1.3 Motivation

Given the expensive and potentially unrealistic nature of doing meta-tuning on algorithm configuration benchmarks, an alternative approach is to replace the real benchmarks by surrogates that ideally exhibit properties similar to the real

ones, while providing inexpensive evaluations. In this chapter, we study the feasibility of this process on the state-of-the-art automatic algorithm configurator irace. We perform the meta-tuning of irace using the Empirical Performance Model (EPM) proposed in [94] as surrogate configuration benchmarks. These EPMs, which are obtained by training a regression model on real performance data, constitute the surrogate model that is used to predict the performance of a target algorithm configuration on a problem instance. In this way, executing the target algorithm is avoided, thus saving a significant amount of computation time. With this approach, an execution of irace on the SPEAR-IBM benchmark only takes about 5 CPU minutes, and the meta-tuning using a budget of 2000 evaluations can be finished within about 7 CPU days on a single core. This strong saving of computation time makes the idea of meta-tuning attractive and computationally feasible. Of course, collecting the necessary algorithm performance datasets for building the regression models can be expensive, but this only needs to be done once and the surrogates can be used for any meta-tuning experiments afterwards. Moreover, as a side product of the meta-tuning process, we obtain a large performance dataset of different irace's configurations on the surrogate configuration benchmarks under study. This dataset can then be used to analyse the importance of irace's parameters and their influence on irace's performance.

4.2 Parameters of irace

In this section, we briefly explain irace and its parameters. For a detailed description, the readers are referred to the original paper of the configurator [10] and irace's user guide [97].

irace is an automatic algorithm configurator, which supports the task of configuring (highly) parametrised optimisation algorithms. The target algorithm, i.e., the one that needs to be configured, can be either deterministic or stochastic and its parameters can be categorical, ordinal, integer, continuous or a mix of these. irace exploits a machine learning technique called *racing*. Given a fixed configuration budget, instead of running every algorithm configuration on every problem instance and taking the one with the best performance measure value, all configurations are first tested on a limited number of instances. Then a statistical test (e.g., Friedman-test, multiple t-tests with/without corrections) is applied to eliminate, given a specific confidence level, significantly worse configurations. The remaining "surviving" algorithm configurations are run on a larger set of problem instances before the statistical test is applied again to discard poor performing configurations. This procedure (*race*) continues until the budget given to the race is reached, or until there are only one or a few

configurations left. At each iteration of irace, a race with Friedman test as the default choice is done. The surviving configurations at the end of each race are used to update a sampling model, from which new configurations for the next race (i.e., the next iteration of irace) are generated.

Table 4.1 lists the parameters of irace, their types, domains and default values. The original domains of the numerical parameters can be larger than the ones specified in Table 4.1, e.g., the domain of parameter *confidence* can be [0, 1], however, in this work, we set them in reasonable ranges, which will be used for the meta-tuning experiments in section 4.4. The first parameter, *nbIterations*, defines the number of iterations (i.e., the number of races) during the configuration process. Its default value depends on the number of parameters of the target algorithm (denoted as n^{params}). The configuration budget assigned to each iteration is decided based on the second parameter, *mu*. The next parameter, *firstTest*, is the number of instances all configurations are evaluated on before the statistical test is applied the first time in a race. Parameter *minNbSurvival* defines the minimum number of configurations required to continue the execution of a race. An iteration is terminated if either *minNbSurvival* is reached or its corresponding budget is exhausted. This means that an iteration might not use up all the configuration budget assigned to it. If it is the case, there will be some configuration budget left after the last iteration is finished. irace will then decide to start a new iteration if the remaining budget is sufficient or stop the configuration procedure otherwise. The categorical parameter *testType* is the statistical test used for elimination (Friedman test or Student t-test with/without corrections). The confidence level of the test is set by the parameter *confidence*. The sampling model for generating new configurations for each race is updated at the end of each iteration based on the surviving configurations of the previous race. Sometimes this might lead to premature convergence. Therefore, a *soft restart* can be enabled by setting the parameter *softRestart* as *true*. The last parameter, *elitistInstances*, sets the number of unseen instances initially evaluated in the race and it is active only when the *elitist* strategy is enabled (setting the parameter *elitistInstances* as *true*). This strategy is available from version 2.0 of irace and is used to ensure that the best configurations obtained are also the ones that have been evaluated on the largest number of training instances.

4.3 Surrogate benchmarks

In this section, we describe the surrogate configuration benchmarks used. These benchmarks are only useful if they can represent real configuration scenarios, i.e., they should show consistency with the original ones to a certain extent.

Parameter	Type	Domain	Default value
nbIterations	int	[1, 50]	$2 + log_2 n^{params}$
mu	int	[1, 20]	5
firstTest	int	[4, 20]	5
minNbSurvival	int	[1, 20]	$2 + log_2 n^{params}$
testType	categorical	{F-test, t-test, t-test-holm, t-test-bonferroni}	F-test
confidence	real	[0.5, 0.99]	0.95
softRestart	categorical	{true, false}	true
elitist	categorical	{true, false}	true
elitistInstances	int	[1, 10]	1

Table 4.1: Parameters of irace

Therefore, before using the surrogate benchmarks for the meta-tuning, we study their consistency with the real benchmarks. First, we check the prediction accuracy of the regression models used inside the surrogates. Second, we examine the homogeneity of the instance sets regarding the configurations. Finally, we compare the list of important algorithm parameters in both surrogate and real benchmarks using the toolbox *fANOVA*) [22].

4.3.1 Surrogate configuration benchmarks

It has been shown that the running time of a number of algorithms can be well predicted using a regression model based on a random forest [94]. The algorithm performance datasets used to train their prediction models and their model's source code are publicly available at `http://www.cs.ubc.ca/labs/beta/Projects/EPMs/`. We use these models as surrogate configuration benchmarks. They include four Mixed-Integer Programming (MIP) benchmarks, namely CPLEX-BIGMIX, CPLEX-REG, CPLEX-CORLAT and CPLEX-RCW, and two SAT benchmarks, namely SPEAR-IBM and SPEAR-SWV, with the MIP solver CPLEX [30] and the SAT solver SPEAR [90] as the target algorithms, respectively. In [94], all continuous algorithm parameters were discretised before applying the prediction models. We want to study the configuration performance of irace on all possible situations, which means that all types of target algorithm's parameters should be considered in the algorithm configuration benchmarks. Therefore, the discretisation is removed in our experiments, i.e., the original continuous domains of the discretised parameters are used instead.

Due to the large amount of memory required for saving the models, we limit the number of instances to 1000. Also, we add a new configuration benchmark called ACOTSP. It configures ACOTSP [98], a software package that implements various Ant Colony Optimization algorithm variants for solving the Travelling

Algorithm	Parameters (int/real/cat/conditional)	Instance sets	#Instances	Metric
CPLEX	76 parameters (6/4/66/4)	BIGMIX	1000	Running time
		REG	1000	
		CORLAT	1000	
		RCW	1000	
SPEAR	26 parameters (4/12/10/9)	IBM	765	Running time
		SWV	604	
ACOTSP	11 parameters (4/4/3/5)	TSP	50	Solution quality

Table 4.2: Seven configuration benchmarks used in our study.

Salesman Problem. The cost metric is the solution quality obtained after running the algorithm for 20 seconds. The seven surrogate configuration benchmarks used in this study are listed in Table 4.2. All configuration benchmarks in our study use the mean of the cost metric values over the problem instance set as the performance measure. The prediction models described in [94] are applied on log-transformed performance values since it is well-known that this transformation gives better prediction quality on running time data [15]. However, for the configuration benchmark on solution quality (ACOTSP), we use the original (un-transformed) performance values. All models are built using performance datasets of 1000 random configurations on all problem instances. The hyper-parameters of the random forest are kept as in [94].

4.3.2 Prediction accuracy analysis

The prediction accuracy measures the concordance between the real performance of configurations on the instances and their predicted values. High prediction accuracy implies high consistency between the real and the surrogate benchmarks. It has been shown that the prediction quality of the random forest model for running time performance data is generally the best compared to other regression approaches [94]. Since we make some changes in these benchmarks (continuous parameters, number of instances) and add a new one (ACOTSP), we re-examine their prediction accuracy using *Pearson's correlation coefficient (*CC*)* together with scatter plots. The mean values of *CC* across 10-fold cross validation are shown in Table 4.3. The closer the value to one, the stronger the linear correlation between the predicted and the real performance values. The scatter plots of the test set in one of the 10 folds are given in Figure 4.1. The prediction quality in all CPLEX and SPEAR benchmarks is generally in line with results reported in [94]. For the new ACOTSP benchmark, the high value of the *CC* and the scatter plot indicate a good prediction accuracy, this is not surprising since for this specific benchmark we know that most of the variability in the

Benchmark	CC	Benchmark	CC	Benchmark	CC
ACOTSP	0.999	CPLEX-BIGMIX	0.852	SPEAR-SWV	0.9
CPLEX-CORLAT	0.875	CPLEX-RCW	0.723	SPEAR-IBM	0.946
CPLEX-REG	0.701				

Table 4.3: Mean values of Pearson's correlation coefficient (CC) across 10-fold cross validation.

algorithm performance can be captured by the scaling of objective function values by instance size.

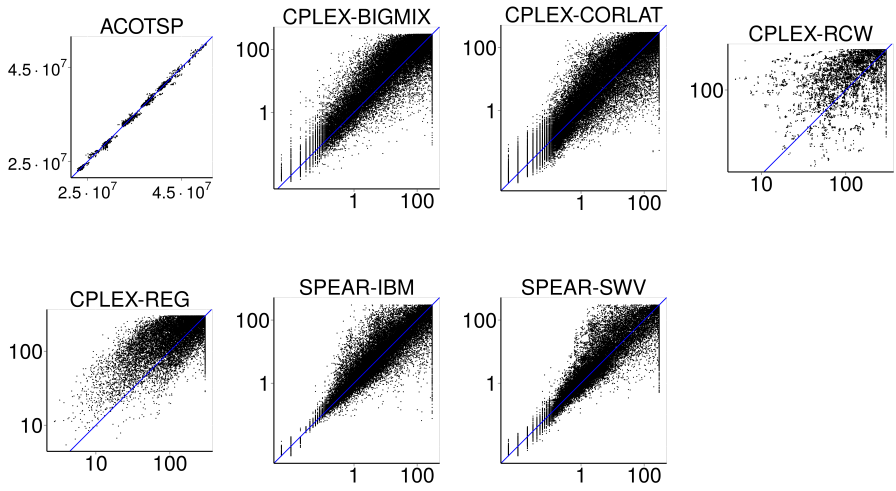


Figure 4.1: Scatter plots of real performance values (x-axis) and the corresponding prediction values (y-axis) of 100 test algorithm configurations on the instance set of each benchmark.

4.3.3 Homogeneity analysis

The homogeneity of a problem instance set measures the similarity of the hardness of the instances w.r.t. different configurations. The homogeneity of the configuration scenario might have a strong influence on the design of a configurator. If a scenario is homogeneous, i.e., algorithm configurations that

work well on a subset of the instance set have a high chance of doing well on the other instances, the configurator does not need to evaluate algorithm configurations on many problem instances, in this way the configuration budget can be reserved for exploring the algorithm configuration space. On the other hand, if a scenario is heterogeneous, the configurator needs to evaluate as many instances as possible to make sure that the final best configurations work generally well on the whole instance set. We measure the homogeneity of the studied configuration benchmarks using the *Kendall concordance coefficient (W)*, as done in [92]. The closer W is to one, the stronger the agreement; $W = 1$ indicates perfect homogeneity. We calculate W on three datasets:

- m_{real} and $m_{prediction}$: W values of the real and predicted performance of the 1000 random configurations used to train the surrogate model.
- $mean_{1000}$ and std_{1000}: mean and standard deviation of W on 10 sets of the predicted performance of 1000 randomly generated configurations.

All these values are reported in Table 4.4. The $mean_{1000}$ and std_{1000} values might give a better overview on the homogeneity of the surrogate benchmarks, given that they are not based on the configurations that were used to train the models. The benchmark ACOTSP is very homogeneous as its m_{real} value is nearly one, while the other ones are more heterogeneous, especially CPLEX-BIGMIX and SPEAR-SWV. All surrogate benchmarks are more homogeneous than their original ones, as $m_{prediction} > m_{real}$ in all cases. Except for ACOTSP, we have also that $mean_{1000} > m_{prediction}$. This indicates that the surrogate landscapes are somehow flattened with respect to the original search space, this is expected due to the trade-off of using prediction models instead of real target algorithm evaluations to obtain the performance values. The $mean_{1000}$ value can be used as a general indication for the whole surrogate benchmark search space since the std_{1000} values are relatively small. Additionally, the relative ordering of homogeneity between the original benchmarks is reproduced by the surrogates, which indicates consistency between the surrogate and the real benchmarks.

4.3.4 Algorithm parameter importance analysis

Given a performance dataset of a configurable algorithm on a set of problem instances, it is possible to analyse which algorithm parameters impact the most on algorithm performance [22]. Since this information can partially represent the landscape of the corresponding algorithm configuration space, we do the analyses on the surrogate and the real performance data and compare them. Similar analyses would indicate consistency between the real and surrogate

Benchmark	m_{real}	$m_{prediction}$	$mean_{1000}$	std_{1000}
ACOTSP	0.9689	0.9724	0.9665	0.0029
CPLEX-BIGMIX	0.1323	0.1796	0.221	0.0046
CPLEX-CORLAT	0.3233	0.4192	0.4406	0.0112
CPLEX-RCW	0.28	0.2936	0.368	0.0097
CPLEX-REG	0.4295	0.5077	0.7029	0.0144
SPEAR-IBM	0.2414	0.267	0.3118	0.0067
SPEAR-SWV	0.176	0.2034	0.2468	0.0057

Table 4.4: Kendall W values on three different datasets.

benchmarks. The toolbox *fANOVA (functional Analysis of Variance)* [22] is used for this task. For each benchmark, we apply fANOVA to the original performance dataset of 1000 random configurations, and their corresponding predicted performance dataset. Due to the large amount of memory required with fANOVA on large datasets, we randomly sample 200 instances if the benchmark has more than that number, so that the maximum size of fANOVA's input performance dataset is 200000 data points. Table 4.5 shows results of fANOVA on both original and prediction cases on all benchmarks. Since the full lists are long, we only show effects with a percentage of contribution on the overall performance variance above 5%. This threshold is chosen arbitrarily. We can see that the list of the important main and pairwise interaction effects are the same in both cases. The percentage values are not exactly the same, but the amount of difference is relatively small. We repeat this analysis five times with different sets of 200 instances randomly chosen, and still observe the same degree of similarity. Therefore, we can conclude that the surrogate and the real benchmarks are consistent according to the algorithm parameter importance aspect.

Original performance dataset	Predicted performance dataset
CPLEX-BIGMIX	CPLEX-BIGMIX
Sum of fractions for main effects 39.75%	Sum of fractions for main effects 42.77%
Sum of fractions for pairwise interaction effects 18.76%	Sum of fractions for pairwise interaction effects 17.31%
14.72% due to main effect: mip_strategy_subalgorithm	17.98% due to main effect: mip_strategy_subalgorithm
9.73% due to main effect: mip_strategy_variableselect	9.47% due to main effect: mip_strategy_variableselect
6.56% due to main effect: mip_limits_cutsfactor	7.61% due to main effect: mip_limits_cutsfactor
5.57% due to main effect: preprocessing_reduce	
CPLEX-REG	CPLEX-REG
Sum of fractions for main effects 32.73%	Sum of fractions for main effects 34.17%
Sum of fractions for pairwise interaction effects 21.70%	Sum of fractions for pairwise interaction effects 22.30%
19.12% due to main effect: mip_strategy_subalgorithm	19.61% due to main effect: mip_strategy_subalgorithm
8.10% due to main effect: mip_strategy_variableselect	8.55% due to main effect: mip_strategy_variableselect
5.07% due to interaction: mip_strategy_variableselect x mip_strategy_subalgorithm	5.72% due to interaction: mip_strategy_variableselect x mip_strategy_subalgorithm
CPLEX-RCW	CPLEX-RCW
Sum of fractions for main effects 45.53%	Sum of fractions for main effects 44.96%
Sum of fractions for pairwise interaction effects 21.56%	Sum of fractions for pairwise interaction effects 21.54%
38.23% due to main effect: preprocessing_reduce	37.87% due to main effect: preprocessing_reduce
8.74% due to interaction: preprocessing_reduce x mip_strategy_subalgorithm	8.46% due to interaction: preprocessing_reduce x mip_strategy_subalgorithm
CPLEX-CORLAT	CPLEX-CORLAT
Sum of fractions for main effects 27.68%	Sum of fractions for main effects 30.21%
Sum of fractions for pairwise interaction effects 21.03%	Sum of fractions for pairwise interaction effects 20.55%
9.45% due to main effect: mip_strategy_variableselect	10.36% due to main effect: mip_strategy_variableselect
7.08% due to main effect: mip_strategy_subalgorithm	7.85% due to main effect: mip_strategy_subalgorithm
5.30% due to main effect: mip_cuts_covers	5.54% due to main effect: mip_cuts_covers
SPEAR-IBM	SPEAR-IBM
Sum of fractions for main effects 77.32%	Sum of fractions for main effects 78.32%
Sum of fractions for pairwise interaction effects 8.49%	Sum of fractions for pairwise interaction effects 8.23%
75.53% due to main effect: sp-var-dec-heur	76.73% due to main effect: sp-var-dec-heur
SPEAR-SWV	SPEAR-SWV
Sum of fractions for main effects 59.95%	Sum of fractions for main effects 61.88%
Sum of fractions for pairwise interaction effects 17.91%	Sum of fractions for pairwise interaction effects 17.14%
50.59% due to main effect: sp-var-dec-heur	52.37% due to main effect: sp-var-dec-heur
7.72% due to main effect: sp-phase-dec-heur	7.94% due to main effect: sp-phase-dec-heur
6.54% due to interaction: sp-var-dec-heur x sp-phase-dec-heur	6.65% due to interaction: sp-var-dec-heur x sp-phase-dec-heur
ACOTSP	ACOTSP
Sum of fractions for main effects 75.77%	Sum of fractions for main effects 79.23%
Sum of fractions for pairwise interaction effects 13.24%	Sum of fractions for pairwise interaction effects 9.96%
69.11% due to main effect: localsearch	74.70% due to main effect: localsearch

Table 4.5: fANOVA results on both the original and the predicted performance dataset of each benchmark. For the sake of brevity, only effects with percentage of contribution on performance variance larger than 5% are shown.

4.4 Experimental results

The analyses performed in Section 4.3 showed an acceptable level of consistency between the original and the surrogate benchmarks. We therefore now use irace to configure the parameters of irace on the configuration scenarios defined by these surrogate benchmarks. The highest level irace (the one that is used to tune the execution of irace as a configurator for target algorithms) we call from now on meta-irace. Meta-irace uses irace's default setting in all the following experiments to configure irace. The training instances for meta-irace correspond to random seeds that are used to execute target-irace. The execution of target-irace, given a specific irace parameter setting δ, is used to determine how well this configuration δ of target-irace is capable of configuring the actual target algorithm (here CPLEX, SPEAR, or ACOTSP). Instead of executing the target algorithm on one of its instances, we use the predictions from the surrogate benchmark as the predicted performance. In turn, the performance of a candidate configuration of irace (defined by δ) in meta-irace is then evaluated as the mean performance of the final configuration obtained by target-irace (using configuration δ) on the training set of the actual target algorithm (as estimated by the surrogate benchmark). Table 4.1 gives the domain of the parameters configured by meta-irace. The default settings of *nbIterations* and *minNbSurvival* depend on the number of parameters of the target algorithm configured by irace; their default values are 8, 6 and 5 for the CPLEX, SPEAR and ACOTSP benchmarks respectively. In our experiments, we set the default statistical test of irace to the Student t-test instead of the Friedman-test due to the fact that the former one is more compatible with the performance measure to be optimised in the surrogate benchmarks. When comparing different irace parameter settings, the performance of a parameter setting (default or tuned) is evaluated as the performance of the best configurations found by 50 executions of irace using those parameter settings. We set the configuration budget as 10 000 evaluations for all CPLEX and SPEAR benchmarks, and 5 000 evaluations for ACOTSP. These budgets are "reasonable" based on the knowledge we have of these benchmarks. However, as these configuration budgets could effect the best irace settings, the results of this study might not apply for different budgets.

4.4.1 Meta-tuning with a single benchmark

We first perform the configuration of irace on each surrogate benchmark scenario. The aim of these experiments is to determine, individually for each benchmark, if it is possible to find a better parameter setting for irace than the current default one. Meta-irace had a budget of 5000 target-irace executions. Each meta-irace execution returns the five best target-irace parameter settings found

(*elites*). Figure 4.2 compares the performance of irace using the elite and default settings. The best elite setting significantly improves the default setting on all surrogate benchmarks, but the amount of improvement varies from benchmark to benchmark. A list of the best settings for all benchmarks and the corresponding default setting is shown in Table 4.6. Examining these settings more carefully, a number of common characteristics are shared among all benchmarks: compared to the default settings, the values of parameters *minNbSurvival* and *confidence* are generally lower, while *nbIterations* values are higher, in addition, the elitist strategy is now always disabled.

These settings suggest that a more intense search is beneficial for the configuration of these benchmarks. A higher number of iterations and a low confidence statistical test allows the probabilistic sampling model of irace to converge to a small area of the parameter search space. Moreover, a low number of surviving configurations (*minNbSurvival*) enables that only the very best configurations continue guiding the search. The elitist strategy (activated by default) is not selected by the configuration process; we believe the reason for this is that non-elitist strategy is more flexible, allowing to adapt strongly the search behaviour of irace through the parameters settings.

4.4.2 Meta-tuning with multiple benchmarks

In this section, we study the possibility of finding an irace parameter setting that obtains an improved perfomance simultaneously for multiple benchmarks. We choose to perform the configuration of irace using the surrogate benchmarks that set CPLEX as the target algorithm. We select only these benchmarks to keep constant the number of parameters of the target algorithm, since by default, the setting of the two irace parameters *nbIterations* and *minNbSurvival* vary according to this value. The budget for the meta-tuning is increased to 10000 target-irace evaluations. The final performance is evaluated, as previously described, by 50 executions of irace on each benchmark. Figure 4.3 shows the normalised test performance values of irace using the default setting and the five elite irace settings obtained by meta-irace for all four CPLEX benchmarks. Again, the improvement of the best elite setting over irace with the default setting is statistically significant, as the corresponding Wilcoxon signed rank test's p-value is 5.5×10^{-25}.

As shown in Table 4.6, we get the same observations on the difference between the best and the default settings of irace as in the single-benchmark case, i.e., the

	nbIterations	*minNbSurvival*	*confidence*	*elitist*	*elitistInstances*	*testType*	*mu*	*firstTest*	*softRestart*
ACOTSP	15	3	0.87	false		t-test	11	9	false
SPEAR-IBM	26	4	0.53	false		t-test-bonferroni	5	20	true
SPEAR-SWV	32	1	0.79	false		t-test-bonferroni	20	17	false
CPLEX-BIGMIX	33	2	0.74	false		t-test	9	8	true
CPLEX-CORLAT	14	5	0.81	false		t-test	7	5	false
CPLEX-RCW	35	1	0.82	false		t-test	3	5	false
CPLEX-REG	35	2	0.52	false		t-test-holm	4	6	true
CPLEX-all	37	1	0.65	false		t-test	5	5	true
default-ACO-TSP	5	5	0.95	true	1	t-test	5	5	true
default-SPEAR	6	6	0.95	true	1	t-test	5	5	true
default-CPLEX	8	8	0.95	true	1	t-test	5	5	true

Table 4.6: Best irace's parameter settings given by meta-irace and the corresponding default setting of irace for each target algorithm. Results of both single-benchmark irace configurations and the multiple-benchmark irace configuration are shown.

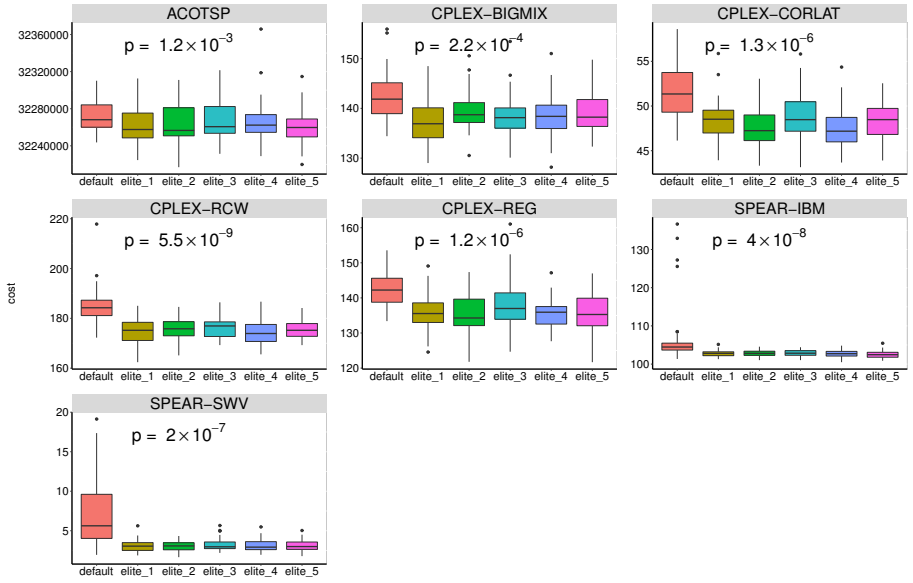


Figure 4.2: Performance of irace using the default setting and the five elite settings obtained by meta-irace on each surrogate benchmark. The pairwise Wilcoxon signed rank test at confidence level 99% between irace default configuration and the best elite indicates statistically significant differences in all cases (p-value is given in each plot) in favor of the best performing elite settings.

elite settings have higher values for *nbIterations*, lower values for *minNbSurvival* and *confidence*, and the elitist strategy is disabled.

4.5 Analysis on irace's parameters

As shown before, the meta-tuning of irace produced significant improvements over its default settings on all surrogate benchmarks. Here, we study the contribution of irace parameters to those improvements, addressing the following questions: (i) Which parameters should be changed to improve performance? (ii) What characteristics do the best parameter settings have? Do they match the ones observed in the settings that improved the default performance on the surrogate scenarios in Section 4.4? (iii) Which parameters do not significantly impact

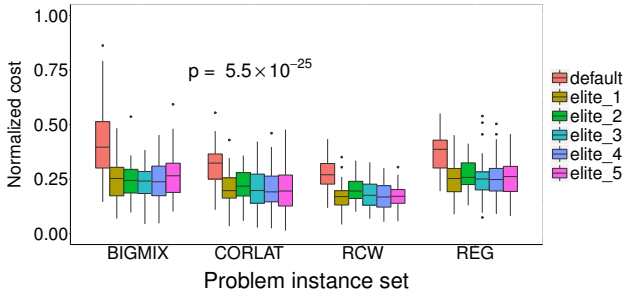


Figure 4.3: Normalised performance of irace using the default setting and the five elite settings obtained by meta-irace configuring irace on the four CPLEX benchmarks and pairwise Wilcoxon signed rank test p-value (confidence level 99%) of the performance of irace using the default setting and the best performing elite setting.

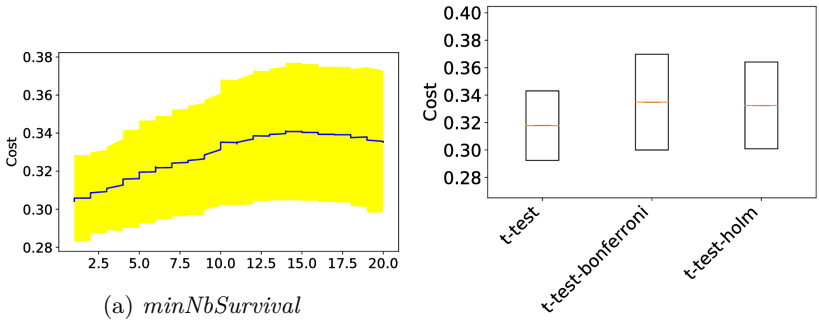
performance? (iv) Do parameters have the same contribution to performance for all benchmarks? Are the best parameter settings scenario-dependent?

We study the contribution of single parameters and pairwise parameter interactions to the performance of irace using the fANOVA toolbox [22]. We also show the possibility of an important higher-order interaction which is not given by the fANOVA analysis.

4.5.1 Analysis of single parameters and pairwise interactions

Each irace configuration experiment produces a dataset of the evaluated performance of different irace parameter settings. The *toolbox fANOVA* [22] can then be applied on that dataset to analyse the importance of each single parameter (i.e., single effect) and each parameter pair (i.e., pairwise interaction effect). For each effect, fANOVA provides two outputs: a value representing the importance of the effect, and a marginal plot that shows the "quality" of the values (with a degree of uncertainty) of all parameters belonging to the effect. For the analysis we use the fANOVA option that focuses only on the area of the configuration space where the average performance is not worse than the one of the default setting. As an example, we give a part of the output of fANOVA analysis on configuring irace with multiple CPLEX benchmarks:

- Sum of fractions for main effects 24.59%



(a) *minNbSurvival*

(b) *testType*

Figure 4.4: Marginal plots of the two single effects *minNbSurvival* and *testType* on the performance dataset of configuring irace with multiple CPLEX surrogate benchmarks. The yellow area and the box-plots show the variance of the marginal prediction.

- Sum of fractions for pairwise interaction effects 28.15%
- 10.35% due to main effect: *minNbSurvival*
- 3.87% due to main effect: *testType*
- 3.57% due to interaction: *nbIterations* x *minNbSurvival*

The single effect of parameter *minNbSurvival* explains the highest percentage of the variation of the performance (10.35%) and the good values for *minNbSurvival* in the marginal plot (Figure 4.4a) are not close to the default value (8), so we can conclude that *minNbSurvival*'s default value should be adjusted to obtain better performance. On the other hand, the best choice for parameter *testType* in its marginal plot (Figure 4.4b) is the same as its default (t-test).

We apply fANOVA on all irace configuration experiments. The main observations from this analysis are the following. (i) The total percentage of variance explained by single and pairwise effects is in the range of 52% to 66%, which indicates that there can be room for other important effects of higher-order interaction. (ii) For the irace configuration experiments with the cost metric of running time, the two parameters *minNbSurvival* and *nbIterations* are most often among the top contributing effects and the marginal plots confirm the observations in Section 4.4: *minNbSurvival* should be set lower than its default value, but *nbIterations* higher than the default. For the irace configuration

experiment with the cost metric of solution quality (ACOTSP), only *nbIterations* seems to play an important role for improving irace settings. (iii) The observation obtained in Section 4.4 for parameter *confidence* is supported by the marginal plots in all experiments; however, the percentage values of the relevant effects are not always on the top list, which indicates that this parameter might not be as important as *minNbSurvival* and *nbIterations*. (iv) The degree of contribution of irace parameters on the improvement over the default settings varies among different experiments, especially for the parameters that are not *nbIterations* and *minNbSurvival*.

4.5.2 A higher-order interaction effect

The analysis performed by fANOVA does not explain the consistent selection of the non-elitist strategy reported in Section 4.4. This could evidence that the importance of this parameter lies in some higher-order interaction effect that the toolbox fANOVA doesn't analyse. We examine this hypothesis by running the following experiment on each benchmark: we take the best settings found by configuring irace and enable the elitist strategy in them (with *elitistInstances* set as 1), then we compare the test performance of the original and the modified best settings to see whether enabling the elitist strategy causes significant deterioration on the performance or not. Results show that the hypothesis is confirmed on all experiments except SPEAR-SWV and ACOTSP:

- For all CPLEX experiments (including the multiple-benchmark) and for SPEAR-IBM, there is statistically significant worsening (with a confidences level of 95%) on all elite configurations.
- For SPEAR-SWV and ACOTSP, there is worsening on the elite configurations, but it is not always statistically significant.

4.5.3 Essential parameters

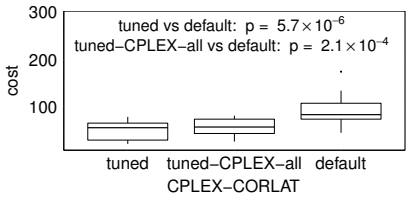
In this part, we show that the list of essential parameters actually varies among different benchmarks. We identify this list by sequentially setting a parameter in the elite configurations as its default value as long as no statistically significant worsening (with a confidence level of 95%) is observed on any elite configuration. This procedure is called ablation [55]. Below are the lists found for four of the benchmarks, which imply that the contribution of irace's parameters can have some dependency with the specific configuration scenario to some certain degree:

- ACOTSP: *nbIterations*, *minNbSurvival* and *elitist*
- CPLEX-REG: *nbIterations*, *minNbSurvival* and *elitist*
- CPLEX-BIGMIX: *nbIterations*, *minNbSurvival*, *elitist* and *confidence*
- SPEAR-SWV: *nbIterations*, *minNbSurvival*, *elitist* and *firstTest*

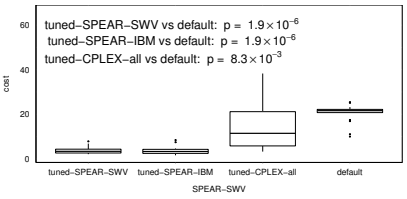
4.6 Results on real benchmarks

We now compare the performance of the default and the best irace settings from Section 4.4 on available, real configuration benchmarks. We first examine the performance of the best irace settings obtained by meta-irace by testing them on the benchmarks they were configured on. For all irace executions, we use the same experimental settings (irace configuration budget and cut-off time) as in the irace configuration with surrogate models. Figure 4.5 shows the real test performance of 20 irace executions for each setting on the three benchmarks CPLEX-CORLAT, SPEAR-SWV and ACOTSP. In addition, the best setting from irace configuration with multiple CPLEX benchmarks, denoted as *tuned-CPLEX-all*, is tested on CPLEX-CORLAT and SPEAR-SWV. The p-value given by the pairwise Wilcoxon signed rank tests between the default and the best configurations in each plot confirms our results with the surrogates on CPLEX-CORLAT and SPEAR-SWV. However, the result for ACOTSP shows that the tuned irace configuration performs statistically significantly worse than irace's default setting. Hence, despite the indication given in Section 4.4 that the surrogate benchmark could be representative for the real benchmark, there is apparently some discrepancy between surrogates and real benchmarks that led to this result.

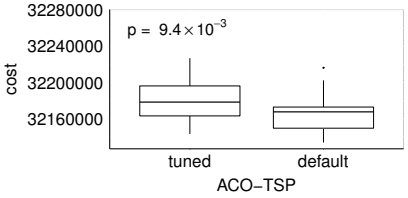
We also did a number of tests in which the default and the best irace settings are run on real benchmarks that were not used to build the surrogates studied in the previous sections to see whether or not the meta-tuning results can be generalized. In particular, we tested the tuned configurations of irace obtained from ACOTSP on an Iterated Local Search algorithm (43 parameters) and solution quality as the cost metric, and CPLEX with multiple benchmarks on Regions100 [11], Regions200 [11] and the SAT-solver Lingeling [89] with 137 parameters and with running time as the cost metric. Results are shown in Figure 4.6. In most of these benchmarks we observe rather similar behaviour between the tuned and the default irace settings (no significant difference) with the exception of the Regions100 benchmark, where tuned irace settings improve statistically significantly over the default settings.



(a) CPLEX-CORLAT



(b) SPEAR-SWV



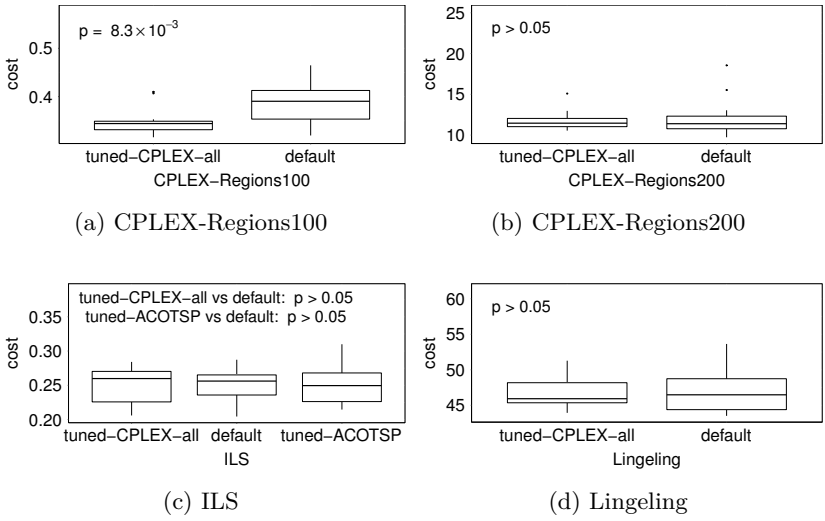
(c) ACOTSP

Figure 4.5: Test performance of irace using the default configuration and the best configuration obtained by meta-irace, measured on the real benchmarks.

In general, the results on the real configuration benchmarks indicate that the meta-tuning of irace on the surrogate benchmarks has helped to some extent, but not as much as the results on the surrogate benchmark alone would have suggested. Hence, the surrogate appears not to be a complete replacement of the real configuration benchmarks, which may be due to the smoothing behaviour of the surrogate, insufficient fidelity of the surrogate in areas of high performing configurations due to insufficient sampling, or that the stochasticity in the configuration benchmarks is not well matched in the surrogates.

4.7 Conclusion

In this chapter, we have investigated the idea of using surrogate configuration benchmarks for the meta-tuning of configurators and studying their parameters, using the example of irace. Six benchmarks with the cost metric of running time [94] and one with solution quality were used. The prediction quality of the surrogate models is sufficiently good. The homogeneity of the benchmarks regarding configurations of the target algorithm, and the list of important algorithm parameters from fANOVA [22] showed a reasonable level of consistency



(a) CPLEX-Regions100 (b) CPLEX-Regions200

(c) ILS (d) Lingeling

Figure 4.6: Test performance of irace using the default configuration and the best on returned by meta-irace, measured on a number of real configuration benchmarks that the meta-tuning never saw.

between the surrogate and the real benchmarks. The two meta-tuning experiments with irace configuring irace on single surrogate benchmark sets and across multiple surrogate benchmark sets indicate that statistically significant improvements over irace default settings are possible. We observed common trend in changes of irace's parameters shared among the best configurations. The contribution of those changes on the improvement was then further confirmed by a number of post-analyses, though the degree of the contribution seems to vary among different scenarios. Nevertheless, these results suggest that the current setting of irace might need to be changed, although further experiments with more benchmarks should be done before overall suggestions on what should be changed can be drawn. Tests of the tuned irace settings on real configuration benchmarks, however, led to mixed conclusions and could not fully confirm the performance advantages of the irace settings tuned on the surrogate benchmarks. Supplementary data of this work is available at *http://iridia.ulb.ac.be/supp/IridiaSupp2017-05*

There are a number of directions in which this work can be extended. The first one is on improving the surrogate modelling, as in the current results we do not always gain the same improvement in irace performance suggested

by the surrogates when testing on real algorithms. This indicates some discrepancy between the surrogates and the real configuration benchmarks. For this extension, new ways of collecting data to train the surrogates such as the one proposed in [99], where trajectories during multiple runs of the automatic algorithm configuration procedure are added, can be studied. The hyper-parameters of the surrogate models and the trade-off between a heavy and time-consuming but more accurate surrogate and a lighter but less accurate one should also be examined. The second extension is on gathering more surrogate algorithm configuration benchmarks, especially the ones for optimising solution quality. Recently 11 new algorithm configuration benchmarks have been proposed [99], but only 2 of them are for optimising solution quality. Running time and solution quality are both popular measures for optimisation algorithm performance, and the automatic algorithm configuration task of benchmarks corresponding to these two measures can be quite different. The third extension is about quantitatively defining representative characteristics of an algorithm configuration benchmark, besides the homogeneity measure using the Kendall concordance coefficient W. The fourth extension is about applying the same meta-tuning method to other state-of-the-art automatic algorithm configurators, such as SMAC [12], ParamILS [11], GGA [13], GGA++ [48]. And the last extension, deriving from all the ones mentioned above, could be an algorithm selection system for automatic algorithm configurators.

Chapter 5

Analysis of algorithm components and parameters

The contributions in this dissertation so far focus on the algorithm configuration task. In this chapter, we address another key aspect during the algorithm engineering cycle: analysing the influence of different algorithm components and parameters on algorithm performance, in order to gain insights into the roles of the components and how they work. Here we look at this aspect as a next step after the algorithm configuration phase. More specifically, in order to find a well-performing setting of a (highly) parametrised algorithm, normally a large number of algorithm runs with different parameter settings have to be executed, either by manual tuning or by using an automatic algorithm configurator. This performance dataset can be re-used to gain deeper understanding of the algorithm by a number of analysis techniques. The general idea is to build a regression model for predicting algorithm performance using the given performance dataset, and analyse the influence of algorithm components and parameters based on the prediction model. Analysis methods along this line are forward selection [21], fANOVA [22], and ablation analysis with surrogate models [23]. Each one looks at a different type of analysis. This chapter illustrates their application to a number of case studies and discusses the advantages of combining them.

The content of this chapter is organised as follows. An introduction to the context of this work is presented in Section 5.1. The three analysis methods [21, 22, 23] are briefly described in Section 5.2, 5.3 and 5.4. The application and combination of the analysis methods on three case studies are then explained in Section 5.5, 5.7 and 5.6. The chapter ends with conclusions in Section 5.8.

5.1 Introduction

5.1.1 Analysis methods

In this chapter, we focus on three algorithm parameter analysis techniques namely forward selection [21], fANOVA [22] and ablation analysis with surrogates [23]. These methods exploit the idea of building prediction models on algorithm performance data, and allow all types of parameters (including categorical ones). They have been tested on a number of cases with high-dimensional algorithm configuration spaces. Given a performance dataset of an algorithm, these methods build random-forest regression models to predict the algorithm performance over the whole configuration space. The models are then used to analyse the influence of the algorithm parameters on the predicted performance. Each one addresses a different analysis aspect. The forward selection method identifies a subset of key parameters by iteratively adding one parameter at a time to the subset so that the resulting prediction model yields the lowest loss in prediction accuracy compared to the model built on the full set of parameters. The fANOVA method applies functional analysis of variance [59] on the prediction model to quantify the contribution of every subset of algorithm parameters on the overall performance variance. Due to the computational cost required, the maximum size of subsets being analysed is often small (two in the current implementation[1], which means only single parameter and pairwise interactions are studied). While the two above methods work with the whole algorithm configuration space, the ablation analysis focuses on the local path between two algorithm configurations (normally between a default configuration and the best one obtained after the algorithm configuration procedure) and recognises parameters that make the largest contribution to performance changes between them. The original ablation analysis [55] works on the real algorithm configuration space. In this dissertation, we use the version with surrogate (i.e., prediction) models [23] due to its lower computational cost.

Automatic algorithm configurators such as irace [10], ParamILS [11], SMAC [12], and GGA [13] have been widely applied in various studies. During the configuration process, several algorithm runs are done, resulting in a normally large performance dataset of the configured algorithm. This dataset can then be used by the analysis methods mentioned above. Since these methods are based on the assumption that random-forest regression models can predict well the performance of algorithm configurations, the analysis results might not always perfectly reflect the ground-truth performance space. Nevertheless, the small amount of time needed to do the analyses makes them appealing. Algorithm

[1] https://github.com/automl/fanova

developers can consider their results as suggestions, and can verify the ones that they find interesting later on using more expensive experiments.

The implementation of all the three analysis methods used in this chapter is from the PIMP package (`https://github.com/automl/ParameterImportance`), provided by the *ML4AAD Group*.[2] We do some modifications to get additional information for the analyses in our case studies.

5.1.2 Motivation

In contrast to automatic algorithm configuration, where the result obtained is a well-performing algorithm configuration that can be used for solving any problem instance of the same application domain, different analysis methods provide information on various aspects of algorithm performance. The question of when to apply these methods is related to the kind of information we want to gain for a deeper understanding of how the analysed algorithm works. It might not always be straightforward for an algorithm developer to interpret analysis results and decide what to use for improving his/her algorithm.

In the original papers of the three analysis methods [21, 22, 55, 23], interesting example applications in domains of machine learning, propositional satisfiability (SAT), MIP, answer set programming (ASP), and automated planning and scheduling (AI planning) have been discussed. The main aim of this chapter is to add to this discussion by illustrating the applications of these analysis methods on a number of case studies in optimisation algorithms, with an emphasis on metaheuristic algorithms. In particular, we demonstrate how to interpret analysis results and discuss the advantages of combining them for a thorough understanding of the algorithm under consideration. By these examples, we would like to promote the usage of these analysis methods in the metaheuristic community, as together with automatic algorithm configuration, they have real potential towards supporting an autonomous and comprehensive algorithm engineering cycle.

5.2 fANOVA

We have seen a number of example applications of this particular analysis technique in the previous chapter. In this part, for convenience, the method is introduced again (with more details). Moreover, the way the fANOVA toolbox is used in this chapter are also more advanced.

[2] http://www.ml4aad.org/

The *fANOVA* method proposed in [22] is an approach for analysing the importance of algorithm parameters for algorithm performance using a random forest prediction model and the *functional analysis of variance* [59]. The approach studies the contribution of every single parameter and every parameter interaction on the performance of the algorithm. In particular, given a data set of performance values of different algorithm configurations on a number of problem instances, fANOVA first builds a random forest-based prediction model to predict the average performance of every algorithm configuration over the whole problem instance space. Afterwards, the functional analysis of variance is applied on the prediction model to decompose the overall algorithm performance variance into additive components, each of which corresponds to a subset of the algorithm parameters. The ratio between the variance associated with each component and the overall performance variance is then used as an indicator of the importance of the corresponding algorithm parameter subset. In addition, fANOVA also provides some insights on which regions are good and bad (with a degree of uncertainty) for each parameter inside the subset through marginal plots. Given a specific value for each algorithm parameter in the subset, the corresponding marginal prediction value is the average performance value of the algorithm over the whole configuration space associated with all parameters not belonging to the subset. A marginal plot shows the mean and the variance of the marginal prediction values given by the random forest's individual trees.

An implementation of fANOVA is provided by the authors of the approach as a Python package at `https://github.com/automl/fanova`, and is wrapped by the PIMP package (`https://github.com/automl/ParameterImportance`). As a choice of implementation, the package only lists the contribution percentage and shows marginal plots of all single parameters and pairwise interactions. It is also possible to acquire the contribution percentage of a specific higher-order interaction and the corresponding marginal prediction given a fixed value for each parameter in the interaction.

5.3 Forward selection

In machine learning and statistics, forward selection is a popular method for selecting key variables in model construction. In [21], the authors use forward selection to identify a subset of key algorithm parameters. Given a performance dataset of an algorithm, the method first splits this dataset into training and validation sets. Starting from an empty subset of parameters, the method iteratively adds one parameter at a time to the subset, in such a way that the regression model built on the training set using the resulting parameter subset yields the lowest Root Mean Square Error (RMSE) on the validation set. As

the regression model, the authors used random forest, since it has been shown to be generally the best for predicting optimization algorithm performance in the literature [94]. Problem instance features can also be added into the analysis by treating them just as algorithm parameters. In some cases where hardness varies largely among instances, forward selection may predominantly prefer instance features over algorithm parameters during the selection path. When this happens, the authors proposed putting limits on both the number of algorithm parameters and the number of instance features being chosen during the selection.

It should be noted that the resulting selection paths of different runs of this analysis could be different, due to a number of factors: (i) the prediction model's randomness (different prediction models due to different training data subsamplings among runs, possible random factors in the prediction model in cases where the algorithms under study are stochastic (ii) the possibility of having correlated algorithm parameters (and instance features), which means that either of these variables can be selected for producing similar RMSE values at a time (iii) different splits of training and validation sets.

5.4 Ablation analysis with surrogates

If an algorithm developer already has a default algorithm configuration for their parametrised algorithm in mind, and receives a better performing algorithm configuration after tuning their algorithm with an automatic algorithm configurator, he/she might wonder which parameters in the default configuration should be modified in order to gain such performance improvement. The ablation analysis [55] answers this question by examining performance changes in the local path between the two configurations in the algorithm configuration space. Starting from the default configuration, the method iteratively modifies one parameter at a time from its default value to the corresponding value of the same parameter in the tuned configuration in such a way that the resulting configuration gains the largest amount of performance improvement. Important parameters in the local path can then be recognised by the percentage of their contribution to the total performance gains. The analysis can also be done in the reverse direction, from the tuned configuration to the default one. In this case, parameters that yield the smallest performance loss are chosen first. Since results of the reverse path can be different from the original path, the authors recommended doing ablation in both directions.

The original ablation method [55] performs real algorithm runs during its search. Instead, in this chapter, we take the surrogate-based version [23], where

Algorithm parameters

Name	Type	Domain	Conditions
algorithm	categorical	{as, mas, eas, ras, acs}	
localsearch	categorical	{0, 1, 2, 3}	
alpha	continuous	[0.01, 5]	
beta	continuous	[0, 10]	
rho	continuous	[0, 1]	
ants	integer	[1, 100]	
nnls	integer	[5, 50]	localsearch ∈ {1, 2, 3}
dlb	continuous	[0, 1]	localsearch ∈ {1, 2, 3}
q0	continuous	[0, 1]	algorithm ∈ {acs}
rasrank	integer	[1, 100]	algorithm ∈ {ras}
elitistants	integer	[1, 750]	algorithm ∈ {eas}

Table 5.1: Eleven algorithm parameters of ACOTSP

algorithm performance is provided by a prediction model. This allows re-using the algorithm performance dataset given by automatic algorithm configuration, hence reducing the computational cost of the ablation analysis significantly.

5.5 Case study 1: Ant Colony Optimization algorithms for the Travelling Salesman Problem

In this case study, we consider ACOTSP [98], a software package that implements various Ant Colony Optimization algorithms for the symmetric Travelling Salesman Problem. The algorithm has 11 parameters, listed in Table 5.1. The automatic algorithm configuration tool irace [10] is used to configure the algorithm with a configuration budget of 5000 runs, each of which has a maximum running time of 20 seconds. The training set is composed of 50 problem instances with 5 different problem sizes from 1000 to 3000 cities. The default configuration of ACOTSP and the five best algorithm configurations returned by irace (sorted by their performance rankings) are listed in Table 5.2.

The main aim of the analyses in this case study is to answer the following questions: Why does irace choose those configurations as the best? Which parameters and interactions are essential for irace's choices?

5.5.1 fANOVA analysis

We first apply fANOVA on the performance dataset given by irace to find the most important single algorithm parameters and pairwise interactions. There

algorithm	localsearch	alpha	beta	rho	ants	nnls	q0	dlb	rasrank	elitistants
(Best configurations)										
acs	3	2.31	8.77	0.48	34	10	0.45	1		
acs	3	2.95	8.64	0.49	48	14	0.54	1		
acs	3	2.67	7.78	0.46	47	12	0.51	1		
acs	3	2.98	9.58	0.14	31	15	0.81	1		
acs	3	2.44	8.13	0.48	42	13	0.53	1		
(Default configuration)										
mmas	1	1.64	4.65	0.5	50	31		0		

Table 5.2: The five best configurations returned by irace and the default configuration of ACOTSP

is a difference between fANOVA and irace (default version) in the way the algorithm performance is measured. fANOVA evaluates performance of an algorithm configuration as the mean of performance values across all problem instances. The default setting of irace, on the other hand, uses the Friedman test as the statistical test for identifying bad configurations, which means that the performances of different configurations on a problem instance set are converted to ranks before being compared. The advantage of this statistical test is that when the ranges of performance values greatly vary among different instances (for example, due to different problem sizes), the test avoids the dominance of instances with large performance values. Due to this difference, in order to use fANOVA results to explain irace's decisions, performance values should be normalised so that they belong to the same range across different problem instances before they are given to fANOVA. For each problem instance, the normalisation value is calculated by equation 5.1:

$$normalised_cost = \frac{cost - min_cost}{max_cost - min_cost} \tag{5.1}$$

where *min_cost* and *max_cost* are the smallest and the largest cost values in the performance dataset for the corresponding problem instance .

Following is a part of fANOVA's output on the normalised performance dataset:

```
All single-parameter effects: 87.63%
All pairwise interaction effects: 11.72%
localsearch: 77.85%
rho: 4.86%
... (remaining effects: < 4%)
```

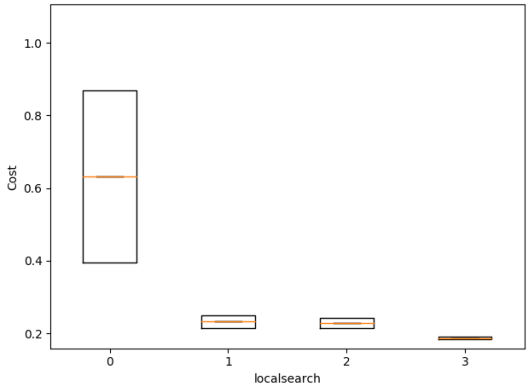


Figure 5.1: Marginal plot of parameter *localsearch* given by fANOVA analysis on the performance dataset obtained when applying irace on ACOTSP. Each box-plot shows the mean and the variance of the marginal prediction.

Results indicate that single parameters and pairwise interactions can explain 99.35% of the total performance variance. The categorical parameter *localsearch*, which defines the choice of the local search used inside ACOTSP, is obviously the most important parameter. Its effect clearly dominates all the others, as it explains 77.85% of the total algorithm performance variance across the whole algorithm configuration space. This parameter has four possible values 0,1,2,3, expressing the choices of using no localsearch, 2-opt, 2.5-opt, and 3-opt local search, respectively. Its marginal plot shown in Figure 5.1, which represents the average predicted performance of all algorithm configurations across the whole algorithm configuration space given a specific choice for the *localsearch* parameter, points out that using no local search is the worst choice and the 3-opt local search is the best choice. This explains the consistent choice of *localsearch* $= 3$ in the best configurations returned by irace.

5.5.2 Ablation analysis

Next, we apply ablation analysis (with surrogate) on the path from the default configuration to the best configuration returned by irace. The performance values are also normalised before being analysed. Figure 5.2 shows the order of parameters chosen during the ablation path from the default configuration to the best configuration, and their corresponding percentage of performance gain. Again, the *localsearch* parameter shows a clear dominance over the others,

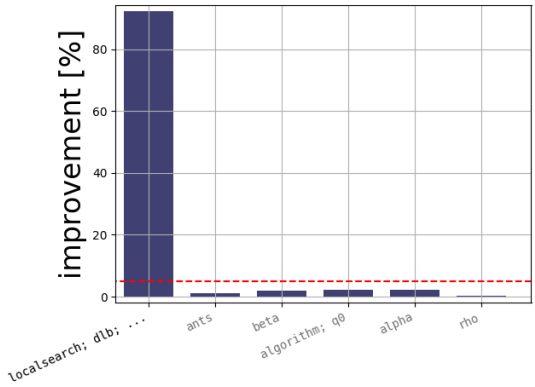


Figure 5.2: Ablation analysis (with surrogate) on the path from ACOTSP default configuration to the best configuration returned by irace. The y-axis shows the percentage of performance gain when switching the value of a parameter on the ablation path.

which is in line with fANOVA findings. In this local region between the two configurations, the influence of this parameter is quite strong, as it provides 92% of the improvement gain when switching from the 2-opt local search (default choice) to the 3-opt.

It should be noted that the finding of fANOVA and ablation analysis on the most important parameters is not necessarily consistent in all cases, as will be shown in the next case study. fANOVA analysis works on the whole configuration space, while ablation analysis focuses on the local path between two specific configurations. Therefore, the dominance of parameter *localsearch* given by both analyses emphasises the importance of this parameter on both the global and the local scale.

5.5.3 Summary

In this case study, we have applied fANOVA and ablation analyses with surrogates on the performance dataset obtained from tuning the ACOTSP algorithm with the automatic algorithm configuration tool irace. The aim of the analyses is to explain the choice of irace on selecting the best algorithm configuration. Both analyses show the clear dominance of the parameter *localsearch* on the algorithm performance. The consistent choice of irace on the best value for this parameter is also confirmed. These observations imply a

consistently important impact of this parameter on both the global algorithm configuration space and the local space in the path between the default and the best algorithm configurations.

In the next section, we will demonstrate an opposite situation where the analyses become more complicated and the impact of parameters on irace's choices is quite local.

5.6 Case study 2: tuning irace on a CPLEX surrogate configuration benchmark

In this case study, we re-use the experimental results of configuring the automatic algorithm configurator irace on one of the CPLEX surrogate benchmarks described in Chapter 4, namely *CPLEX-REG*. Recall that we use irace to configure the nine parameters of irace (Table 5.3) on an algorithm configuration benchmark, a procedure that we called meta-tuning. To avoid confusion, the higher-layer irace is named meta-irace. In an algorithm configuration setting, the tuned irace is simply considered as a parametrised optimisation algorithm, and the algorithm configuration benchmark plays the role of a problem instance. The aim of the analyses in this section is to understand the choice of the automatic algorithm configurator (here, meta-irace) on the best irace algorithm configuration. The default configuration of irace and the five elite configurations returned by meta-irace are shown in Table 5.4.

Although irace is not a meta-heuristic, this case study is an interesting example to illustrate the complementary usage of the fANOVA and ablation analyses. Unlike the ACOTSP case study in section 5.5, where there is a dominant parameter over both the global space and the local regions between the default and the best algorithm configurations, the analysis results in this case study indicate that the importance of irace parameters and the choice of the meta-irace are very local.

5.6.1 Ablation analysis

In the original ablation paper [55], it was recommended to do ablation in both directions between two algorithm configurations. The reason is due to the possibility of interactions between parameters: there are cases where important performance improvements can only be done if more than one parameter is

Parameter	Type	Domain
nbIterations	int	[1, 50]
mu	int	[1, 20]
firstTest	int	[4, 20]
minNbSurvival	int	[1, 20]
testType	categorical	{F-test, t-test, t-test-holm, t-test-bonferroni}
confidence	real	[0.5, 0.99]
softRestart	categorical	{true, false}
elitist	categorical	{true, false}
elitistInstances	int	[1, 10]

Table 5.3: Nine parameters of irace

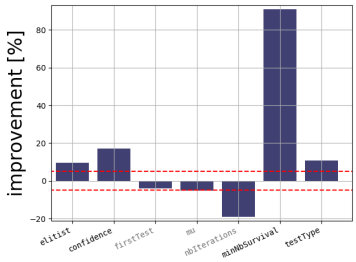
modified at a time. This case study is a clear example of this situation. Given the default configuration and the best one provided by the meta-irace, we will show that the reverse ablation path can give interesting information that is not really obvious in the original path.

First, we apply the ablation from the default irace configuration to the best one for ten times. One thing we notice from the results is that the orders in which the parameters are chosen as well as their contribution on the performance gain vary quite a bit among different runs. This can be seen in the plots of three example paths shown in Figure 5.3. As an example, *nbIterations* introduces even a negative performance gain in run 1, while it acquires the largest gain of nearly 50% along the path in run 2. Because of those fluctuations, it is difficult to draw any conclusions from the ablation analysis in this direction.

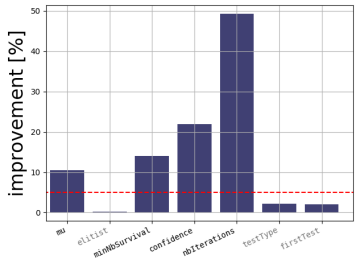
We apply the ablation another ten times, now on the reverse path from the best configuration to the default configuration. At each step, the parameter that introduces the least performance loss when changing its value from the best configuration to the default one is chosen. Figure 5.4 shows the reverse paths with the same random seeds as in the ones used in Figure 5.3. We can see that the orders in which the parameters are chosen along the paths are now much more consistent among different runs. In particular, the four parameters *nbIterations*, *confidence*, *minNbSurvival* and *elitist* are always chosen at the end of the path, which means that changing their values in the best configuration will cause more performance loss than all other parameters. Indeed, most of the performance loss on the paths are due to the three parameters *nbIterations*, *confidence*, *minNbSurvival*. Parameter *elitist*, although it is not explicitly associated with a big loss in performance, is constantly the last one chosen in the ten paths. This indicates two things: (i) the importance of setting this parameter at the right value for achieving the good performance of the best configuration, as changing its value in the path between the best configuration

nbIterations	minNbSurvival	confidence	elitist	elitistInstances	testType	mu	firstTest	softRestart
(Elite configurations)								
35	2	0.52	false		t-test-holm	4	6	true
29	1	0.55	false		t-test-holm	4	5	true
37	1	0.51	false		t-test-holm	4	5	false
29	1	0.55	false		t-test-holm	3	5	true
12	1	0.77	false		t-test	7	5	true
(Default configuration)								
8	8	0.95	true	1	t-test	5	5	true

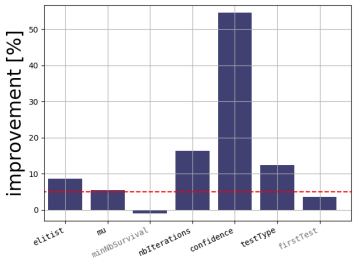
Table 5.4: Elite configurations (sorted in ascending order of ranks) for irace obtained from a meta-tuning experiment on the CPLEX-REG surrogate configuration benchmark and the corresponding default configuration of irace.



(a) Run 1



(b) Run 2



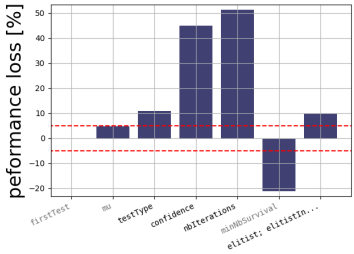
(c) Run 3

Figure 5.3: Three ablation paths from the irace default configuration to the best one returned by meta-irace on CPLEX-REG surrogate benchmark. The plots show the percentage of performance gain when the corresponding irace parameter is changed along the paths. The red line indicates the 5% performance gain threshold. Any variables with contribution below this line are marked as non-bold for visualisation purpose.

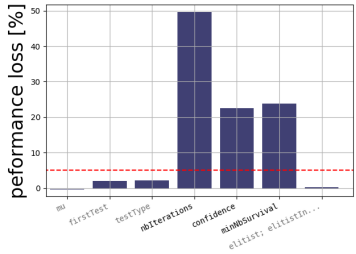
to the default one will induce larger performance loss than any other parameters, (ii) the strong dependency of this parameter and the others, especially the three parameters *nbIterations*, *confidence*, *minNbSurvival*; or, in other words, the possibly complicated interactions between these parameters in the local region between the two configurations under study.

5.6.2 fANOVA analysis

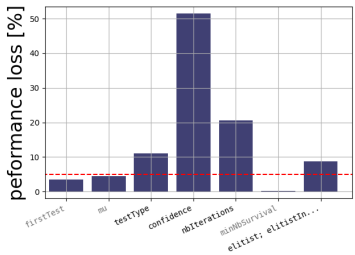
Next, we use fANOVA analysis to gain additional information on the findings given by the ablation. In particular, we want to see how parameter *elitist*



(a) Run 1



(b) Run 2



(c) Run 3

Figure 5.4: Three ablation paths from the best one returned by meta-irace on the CPLEX-REG surrogate benchmark to the default irace configuration. The plots show the percentage of performance loss when the corresponding irace parameter is changed along the paths. The red line indicates the 5% performance gain threshold. Any variables with performance loss below this line are marked as non-bold for visualisation purpose.

interacts with the others, especially the three parameters *nbIterations*, *confidence*, *minNbSurvival*, and whether their impacts are global.

We do two fANOVA analyses, one on the global space, dubbed *global-fANOVA* and one on the local space where the algorithm performance is not worse than the default one, namely *capped-fANOVA*. Following is a part of the global-fANOVA results:

```
All single-parameter effects: 38.01%
All pairwise interaction effects: 34.31%
```

```
nbIterations: 11.8%
minNbSurvival: 6.22%
firstTest: 5.73%
testType: 5.25%
confidence: 5.21%
... (remaining effects: < 4%)
```

and a part of the capped-fANOVA results:

```
All single-parameter effects: 11.69%
All pairwise interaction effects: 28.17%
firstTest x nbIterations: 3.89%
minNbSurvival: 3.28%
firstTest x minNbSurvival: 2.82%
firstTest: 2.55%
minNbSurvival x nbIterations: 2.26%
... (remaining effects: < 2%)
```

The differences between the two analyses are that in the local one, single parameter effects become less important, both single parameter and pairwise interaction effects have less contribution to the total performance variance, and their contribution is more widely spread. One possible explanation for these differences is that in the local space, the interactions of algorithm parameters are more complicated. Anyway, one clear thing is that *eliltist* parameter is not among the top list effects in both analyses.

In capped-fANOVA results, the total contribution of all single parameters and pairwise interactions on the overall performance variance is less than 40%, leaving a potential for some important higher-order interaction. We can check if this is the case for any of the interactions between *elitist* and the three parameters *nbIterations*, *confidence*, *minNbSurvival* by giving those four parameters to fANOVA and asking for the percentage of variance explained by all relevant interactions. Results show that the total contribution of all (pairwise and higher-order) interactions between *elitist* and the other three parameters is only 5.18%, and it generally is spread evenly among them. It means that there are no important interactions as expected. Since capped-fANOVA works on the configuration space where performance value is not worse than the default configuration, the observations in this analysis indicate that the importance of *elitist* is even more local, i.e., it only works when not only the three parameters *nbIterations*, *confidence*, *minNbSurvival* are set properly, but also the other ones have to be in the appropriate ranges. In fact, if we look at the values of the other parameters in the default and the best configurations, we can see that

they are already quite close to each other. For example, the parameter *firstTest* has the values of 5 and 6 in the default and the best configurations, respectively, while its domain is [4,20]. We can further confirm our conjecture as follows. First, we ask fANOVA for the predicted average cost of the default and the best configurations (here we get 138.92 and 135.81). Next, we modify the value of parameter *firstTest* in the best configuration to a more distant one, say 12. Then we ask fANOVA for the predicted cost of this new configuration (here we get 138.78). This increasing cost implies the importance of having *firstTest* near the local region around 6.

In summary, the conclusion from this fANOVA analysis is that the importance of irace parameters and the choice of the meta-irace are very local, i.e., although the four parameters *elitist*, *nbIterations*, *confidence*, *minNbSurvival* play essential roles in improving the performance between the default and the best configurations, the other parameters also need to be set in the appropriate ranges that are not far from their values indicated in the best configuration. Therefore, if irace's developers want to explain which changes in irace's behaviour (according to its parameters) improve the performance, they will need to associate those changes with all parameters instead of only a few of them.

5.6.3 Summary

In this case study, by applying ablation and fANOVA analyses to explain the choice of meta-irace when configuring irace on a CPLEX surrogate configuration benchmark, we have shown:

- an example where the ablation analysis in the two directions can provide different results; due to the complex interactions between irace parameters, the reverse path is more informative;
- a complementary usage of various fANOVA package's functionalities to gain more information on the findings given by the ablation analysis, more specifically, to understand how local the scope of the findings in the configuration space is.

5.7 Case study 3: a Large Neighbourhood Search for a Vehicle Routing Problem with Time Windows

The example analyses so far are only on the algorithm configuration space. In this case study, we illustrate the integration of problem instance features into fANOVA analysis, so that we can study not only the impact of algorithm parameters, but also their interactions with instance features.

The algorithm considered in this case study [1] is a Large Neighbhourhood Search metaheuristic algorithm for solving a typical Vehicle Routing Problem with Time Windows. It is a simplification version of the Adaptive Large Neighbourhood Search algorithm [100], a general algorithm for solving different variants of the Vehicle Routing Problem [101]. There are 200 problem instances generated randomly according to different problem features. For each problem instance, 20 random algorithm configurations are generated and tested. In total, the performance dataset contains 4000 algorithm configuration runs. The returned cost value is the total travelling distance of all vehicles.

The eight parameters of the algorithm and five problem instance features are listed in Table 5.5.

Algorithm parameters

Name	Type	Domain
random_seed	integer	[1, 1000000]
deterministic_parameter	integer	[1, 100]
noise_parameter	continuous	[0, 1]
cooling_rate	continuous	[0.01, 0.99]
start_temperature	continuous	[0.01, 1]
control_parameter	continuous	[0.01, 1]
destroy_operator	categorical	[Random, Worst, Related, RandomWorst, RandomRelated, WorstRelated]
repair_operator	categorical	[Greedy, Regret2, GreedyRegret2]

Problem instance features

Name	Type	Domain
customer_number	integer	[25, 400]
customer_demand	integer	[10, 50]
average_service_time	continuous	[10, 50]
average_time_windows	continuous	[20, 80]
max_running_time	integer	[60, 1800]

Table 5.5: Eight algorithm parameters and five problem instance features for the case study with a Large Neighbourhood Search metaheuristic algorithm for solving a Vehicle Routing Problem with Time Windows [1].

5.7.1 fANOVA analysis

We can add instance features into fANOVA analysis by treating them as algorithm parameters, i.e., each combination of algorithm configuration and instance feature setting is given to fANOVA as an algorithm configuration. In this way, fANOVA only sees one (virtual) instance. It must be noted that this integration can only be done when the *instance features are independent*, since this is one of the key assumptions fANOVA makes on its input variables. In this case study, the problem instances are generated based on random values drawn from all instance features in an independent way.

Following is a part of fANOVA's output on the percentage of performance variance explained by single parameters and pairwise interactions:

```
All single-parameter effects: 80.87%
All pairwise interaction effects: 12.23%
customer_number: 75.24%
average_service_time x customer_number: 5.24%
average_service_time: 3.78%
... (remaining effects: < 2%)
```

An obvious observation from the results above is that only instance features are listed as the most important variables. In particular, the feature *customer_number* has a clear dominant influence, as it explains 75.24% of the total performance variance. Looking at its marginal plot shown in Figure 5.5, we can see that the average performance values increase according to this feature. This means that the range of the algorithm performance values greatly depends on problem instance size (here it is the number of customers).

In situations like this, depending on the aim of our analysis, normalisation can be necessary. Here we aim to study the algorithm performance without any bias towards particular instances, which is similar to what irace's default setting does as described in section 5.5. Therefore, we re-run the fANOVA analysis, now with normalised performance values. A part of the new results is listed below:

```
All single-parameter effects: 60.74%
All pairwise interaction effects: 15.38%
repair: 52.68%
destroy x repair: 4.42%
customer_number x repair: 4.19%
customer_number: 4.0%
```

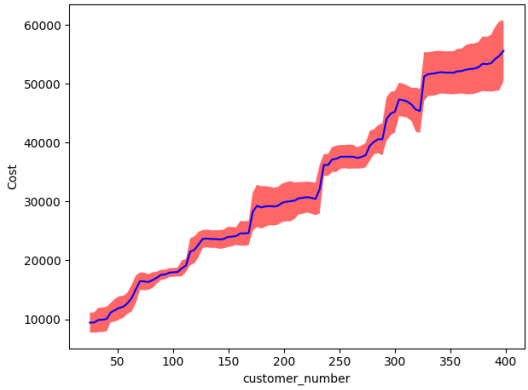
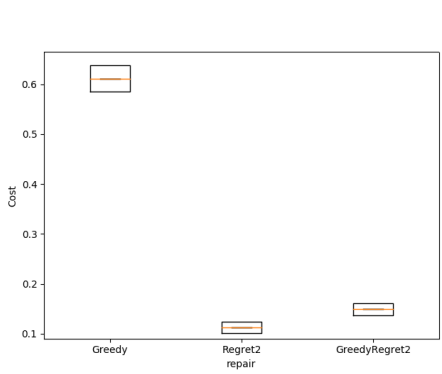


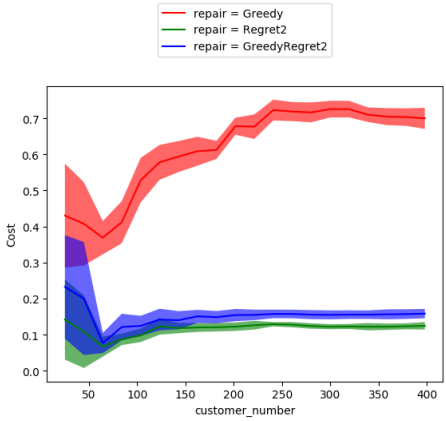
Figure 5.5: Marginal plot of the problem instance feature *customer_number* given by fANOVA analysis

```
destroy: 3.14%
... (remaining effects: < 3%)
```

The impact of problem instance feature *customer_number* is now significantly reduced, so that the influence of algorithm parameters becomes visible. The two categorical algorithm parameters *repair* and *destroy*, which represent the choices of the repair and the destroy operators inside the Large Neighbourhood Search, now involves in the top list of important parameters and pairwise interactions. In particular, the high contribution percentage of parameter *repair* (52.68%) on the total performance variance and its marginal plot (Figure 5.6a) show that the repair operators Regret2 and Greedy work generally best and worst, respectively, across the whole problem instance space. In addition, we can also see how much influence the choice is according to different problem sizes by looking at the marginal plot of the pairwise interaction *repair* x *customer_number* (Figure 5.6b): on instances with the number of customers less than 50, the bad performance of Greedy is not as obvious as on instances with larger sizes; for the other two choices, Regret2 and GreedyRegret2, their overall performance difference also gets clearer when the number of customers increases. Other problem instance features do not have important interactions with algorithm parameters.



(a) repair



(b) customer_number x repair

Figure 5.6: Marginal plot of algorithm parameter *repair*, and pairwise interaction between algorithm parameter *repair* and problem instance feature *customer_number* given by fANOVA analysis with performance normalisation.

5.7.2 Forward selection

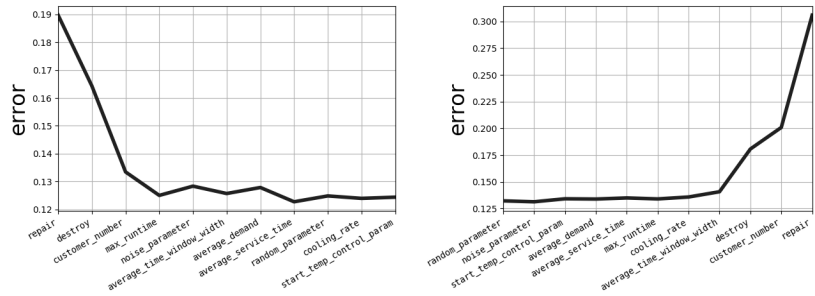
Although the fANOVA analysis provides interesting insights into interactions between algorithm parameters and instance features, such an analysis, in this case study, takes a large amount of time to run (16 hours on a single core Intel CPU 2.4Ghz)[3]. The reason is due to the heavy computation of all pairwise interactions, since the number of variables is not small. In this part, we demonstrate a cheaper alternative for this particular case using a combination of fANOVA and forward selection. First, we use fANOVA to calculate the importance of single variables. Next, we use forward selection to identify the key subset of variables, and then use fANOVA to examine interactions of the variables in that set only. In this way, the whole analysis takes less than half an hour.

Applying fANOVA on single parameter effects will give exactly the same results as in the normal fANOVA, but without pairwise interactions. This analysis

[3]This amount of running time is reported on the Python-based fANOVA package linked by PIMP. The previous fANOVA version, which is Java-based, is faster, although it still needs several hours to finish this analysis. Since all other methods are developed in Python and share the same Random Forest implementation with the new Python-based fANOVA, we use this new version for analysis consistency.

takes less than one minute to finish. Now we apply forward selection to the normalised performance dataset. As described in section 5.1, results of the analysis might vary among different runs due to a number of factors. Therefore, we repeat it ten times, which takes about 3 minutes in total. The visualisation of one run's result, which is a path of sequentially adding one variable to the model at a time so that the resulting root mean square error on the validation set is minimised, is shown in Figure 5.7a. We can see that the first three parameters chosen are *repair*, *destroy* and *customer_number*. This happens to be the case for all the ten runs, these three parameters are always chosen first, while the ordering of the remaining parameters in the path can vary among different runs. This indicates the high relevance of these parameters on the prediction model.

Since these experiments are computationally cheap, we also apply backward elimination, the reverse procedure of forward selection, where starting from the whole variable set, one variable resulting the least decrease in the prediction error is removed sequentially. This can help to take into account dependency between the parameters. These runs takes about 5 minutes in total. The chosen path and the error values in one of the ten runs is shown in Figure 5.7b. The three parameters *repair*, *destroy* and *customer_number* are the last ones to be chosen, and this is again consistent among the ten runs. This implies the importance of these parameters' interactions.



(a) Forward selection (b) Backward elimination

Figure 5.7: Prediction error (on validation set) of the model trained on a subset of variables chosen during one run of forward selection analysis and one run of backward elimination analysis.

Given those indications, we can now focus on analysing the interactions between these three variables using fANOVA. Results of the importance quantification and marginal plots given the three variables *repair*, *destroy* and *customer_number* are obviously the same as the original fANOVA, with an

addition of the three-way interaction between those variables. However, since the importance of this interaction is rather small (2.3%), we can simply ignore it. This particular fANOVA analysis takes about two minutes. In summary, in this case study, the combination of fANOVA and forward selection helps to significantly reduce computational time while still gaining the same analysis results as using fANOVA solely.

5.7.3 Summary

In this case study, we have illustrated:

- the integration of problem instance features into fANOVA analysis, which reveals interesting insights into how the influence of an important algorithm parameter might change according to different problem sizes;
- an example on the complementary usage of fANOVA and forward selection that can help to greatly reduce the computation time required, while still gaining the same analysis results as the expensive one using only fANOVA for this particular case. Forward selection was used to get hints on the most important parameters, and fANOVA was then applied on this subset instead of the whole set.

5.8 Conclusion

In this chapter, we have demonstrated and discussed the complementary usage of different parameter analysis techniques, namely the forward selection [21], fANOVA [22] and ablation [23] analyses. This is done using three case studies, each of which represents some special situations that the analyses can fall into. In the first and the second case study, where the aim is to use fANOVA and ablation analyses to understand the choice of the best configuration given by an automatic algorithm configuration tool (here irace), analysis conclusions diverge: in the first one, there is a dominant parameter in both the global configuration space and the local space between the default and best algorithm configurations, so that the algorithm developers can focus on this particular parameter for explaining the improvement over the default configuration; in the second one, the impact of parameters are very local and the algorithm performance improvement gained requires all parameters to be in their reasonable ranges of values, which means that the developers need to look at all parameter values as a whole instead of focusing on a few of them. In the third case study, the integration of problem instance features into fANOVA analysis is illustrated. This shows how the range

of algorithm performance values changes according to problem instance sizes; hence, it emphasises the necessity of normalising performance data to avoid bias. The interaction between algorithm parameters and problem instance features also provides interesting insights into how the impact of important algorithm parameters can vary among different problem sizes. Moreover, a complementary usage of the forward selection analysis that can help to significantly reduce the computation cost of fANOVA analysis in this particular case study is also presented.

For future work, more case studies can be investigated, especially the ones for local search metaheuristics and evolutionary algorithms in solving combinatorial optimisation problems, where the applications of these analysis techniques are still scarce. As illustrated in this chapter, the interpretation of analysis results is not always straightforward. There can be complex interactions or different angles that can be looked at together to yield a thorough understanding on how algorithms work. Therefore, we believe that discussions on how to use these analysis methods in different cases will be beneficial to the main theme of this research: providing support for algorithm design.

The second extension is to study the impact of the hyper-parameters of the internal prediction models used in those methods. The current implementations of all these three analyses make use of random forest, which have been shown to be the best models for predicting running time of optimisation algorithms [15] [94] and quality of machine learning algorithms in most cases. As pointed out in [99], the prediction quality of a random forest can be quite sensitive to its hyper-parameters. This study is particularly necessary when the algorithm under study is for solving combinatorial optimisation problems and the performance metric is solution quality, as those cases have not been thoroughly investigated. Such kind of study will also be useful for the work in chapter 4, as the algorithm configuration surrogates currently utilise exactly the same prediction model (random forest), and one of the main future works in chapter 4 is to extend the meta-tuning to more benchmarks with solution quality as performance measure.

As a third extension, the integration of problem instance features into the combined analyses can be further investigated. Since performance of optimisation algorithms might greatly vary according to different types of problem instances, insights acquired by studying both algorithm parameters and problem instance features would provide useful information for algorithm developers.

Chapter 6

Conclusions

In the era of increased computational power, collecting large amounts of data on the performance of optimisation algorithms has become easy. This naturally leads to the rise of applications of various techniques from data science to assist the algorithm development process. Automatic algorithm configuration, the focus of the first two chapters in this dissertation, is one of the most popular applications in this direction. It allows algorithm developers to creatively design several potential components for their algorithms, and leaving the tedious task of choosing the best components and their specific parameter settings to be done automatically using algorithm configuration tools.

The case study presented in chapter 3 is an example for the support of automatic algorithm configuration during algorithm design. We consider the development of a local search metaheuristic [75] for solving the Swap-Body Vehicle Routing Problem [74], a variant of the conventional Vehicle Routing Problem [101] where each vehicle can change its capacity by dropping or picking up a "swap-body" at some fixed locations along its route. In order to solve this problem, besides the neighbourhoods specially designed for dealing with the swap-body properties, the algorithm can make use of several neighbourhoods that have been proposed in the VRP literature. We assume that the integration of neighbourhoods into the algorithm is controlled by a scheme where the probability that a neighbourhood is chosen at each algorithm iteration is defined by a weight value. With the help of an automatic algorithm configuration tool, the algorithm developers can then add as many neighbourhoods to the algorithm as they wish, and let the configurator choose the best weights automatically. Hence, the support that automatic algorithm configurators offer can help to reduce a lot of manual efforts in designing such an algorithm. We enhance this support by

proposing a method to automatically recognise similar neighbourhoods based on their search behaviour on a given problem instance set. This helps to reduce the number of parameters (i.e., the number of weights) that need to be tuned, so that the efficiency of the automatic algorithm configurator is improved. Since the method is both independent of the underlying local search and the problem being solved, future work can simply apply this method or extend it further to other cases where the same scheme for choosing a neighbourhood at each iteration is used.

The topic of automatic algorithm configuration continues to be studied in chapter 4. This time we focus on the configurators themselves. By making use of surrogate modelling, we show, for the first time, the feasibility of a systematic study on the parameters of irace [10], a general-purpose automatic algorithm configuration tool, which has been shown to successfully improve or generate well-performing algorithms in several cases [9] [50]. The study suggests potentials for significantly improving the configurator's performance by configuring its parameters automatically using surrogate configuration benchmarks through a procedure called meta-tuning. This, in turn, offers benefits to the algorithm design process due to the broad application range of this tool. Although the results we obtained are promising, there are still cases where we reach inconsistent conclusions when testing on the real benchmarks. Therefore, in future work, the surrogate modelling should be improved to reduce the discrepancy between the surrogate benchmarks and the real ones. In addition, more algorithm configuration benchmarks on optimising solution quality can be collected to make the study more generalizable. Another necessary extension is about quantitatively defining characteristics of algorithm configuration benchmarks and understanding the relations between these characteristics and the performance of algorithm configurators in a systematic way. This can finally lead to the ultimate aim of this research direction - algorithm selection for automatic algorithm configurators - where the advantages of all configurators can be combined.

During the algorithm development process, besides obtaining a well-performing algorithm configuration, it is also important to gain insights into how an algorithm works, which algorithm parameters have the most influence on the algorithm performance, and how. This is the main focus of the last contribution in this dissertation, chapter 5. In particular, we demonstrate and discuss the complementary usage of three analysis techniques, namely forward selection [21], fANOVA [22] and ablation analysis [55] [23], on three case studies, each of which represents a specific situation where the advantage of combining different analyses is illustrated. As it is not always straightforward to interpret analysis results, in future work, more case studies should be investigated. We believe that discussion on the usage of different analysis techniques through various

examples will be beneficial, especially for the metaheuristic community, where these methods have not been popularly used in publications to support the development of algorithms.

Besides the individual future works, the research lines in this dissertation can be used in collaborative ways. Firstly, the algorithm parameter tuning problem in chapter 3 can be used as a benchmark for studying the configurator's parameters in chapter 4. Secondly, the analyses in chapter 5 can be applied at the end of the algorithm development process of the local search metaheuristic scheme with multiple neighbourhoods considered in chapter 3. This can help to understand the importance of the neighbourhood groups found by the clustering method proposed, as well as possible interactions between these groups. Finally, these analysis methods are also useful for the study of automatic algorithm configurators' parameters presented in chapter 4, as illustrated in one of the case studies in chapter 5 where the complicated impact of irace's parameters on the improvement over the default irace parameter setting was revealed. This collaboration is especially necessary for the future work in chapter 4 when relations between algorithm configuration benchmarks' characteristics and configurators' parameters are investigated.

In summary, the contributions in this dissertation have added to the rich application list of data analytics for optimisation in general, and for algorithm design in particular. In the bigger picture, we believe that this field will continue to grow rapidly. We expect a more extensive application range and more combinations of prominent techniques in this field to design of algorithms for combinatorial optimisation problems, especially real-world ones, which are often sophisticated and hard to solve. To serve this aim, two research directions can be further investigated. Firstly, knowledge extracted from both the algorithm being designed and the problem being solved can be integrated into the current supporting tools to improve their performance. Our first contribution on the neighbourhood characterisation is an example of this "opening the box" idea. Some other potential examples include providing hints on identifying less useful problem instances during the configuration process, or learning to recognise low-performing algorithm runs that can be stopped in advance during the configuration. Secondly, more guidelines on how to apply the current tools effectively and efficiently are needed. The other two contributions of the dissertation are oriented towards this direction. One of the main future works of the second contribution on configuring irace's parameters is to provide guidelines on how to set irace's parameter properly according to the characteristics of the configuration scenario being tackled. The third contribution serves as illustrating examples on how to combine different analysis methods for better analysis results.

Bibliography

[1] Jeroen Corstjens, An Caris, Benoit Depaire, and Kenneth Sörensen. A multilevel methodology for analysing metaheuristic algorithms for the VRPTW. 2016.

[2] Michael Jünger, Gerhard Reinelt, and Giovanni Rinaldi. The traveling salesman problem. *Handbooks in operations research and management science*, 7:225–330, 1995.

[3] Andrea Schaerf. A survey of automated timetabling. *Artificial intelligence review*, 13(2):87–127, 1999.

[4] Michael R Garey and David S Johnson. *Computers and intractability*, volume 29. wh freeman New York, 2002.

[5] Kenneth Sorensen, Marc Sevaux, and Fred Glover. A history of metaheuristics. *arXiv preprint arXiv:1704.00853*, 2017.

[6] Gerhard J Woeginger. Exact algorithms for NP-hard problems: A survey. *Lecture notes in computer science*, 2570(2003):185–207, 2003.

[7] David L Applegate, Robert E Bixby, Vašek Chvátal, William Cook, Daniel G Espinoza, Marcos Goycoolea, and Keld Helsgaun. Certification of an optimal TSP tour through 85,900 cities. *Operations Research Letters*, 37(1):11–15, 2009.

[8] Thomas Stützle and Rubén Ruiz. Iterated greedy. *Handbook of Heuristics*, pages 1–31, 2017.

[9] Holger H. Hoos. Automated algorithm configuration and parameter tuning. In Y. Hamadi, E. Monfroy, and F. Saubion, editors, *Autonomous Search*, pages 37–71. Springer, Berlin, Germany, 2012.

[10] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Thomas Stützle, and Mauro Birattari. The irace package: Iterated racing

for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[11] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, October 2009.

[12] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In Carlos A. Coello Coello, editor, *Learning and Intelligent Optimization, 5th International Conference, LION 5*, volume 6683 of *Lecture Notes in Computer Science*, pages 507–523. Springer, Heidelberg, Germany, 2011.

[13] Carlos Ansótegui, Meinolf Sellmann, and Kevin Tierney. A gender-based genetic algorithm for the automatic configuration of algorithms. In Ian P. Gent, editor, *Principles and Practice of Constraint Programming, CP 2009*, volume 5732 of *Lecture Notes in Computer Science*, pages 142–157. Springer, Heidelberg, Germany, 2009.

[14] Lars Kotthoff. Algorithm selection for combinatorial search problems: A survey. *AI Magazine*, 35(3):48–60, 2014.

[15] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606, June 2008.

[16] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Hydra-MIP: Automated algorithm configuration and selection for mixed integer programming. Technical Report TR-2011-01, Department of Computer Science, University of British Columbia, Canada, 2011.

[17] Lars Kotthoff, Chris Thornton, Holger H. Hoos, Frank Hutter, and Kevin Leyton-Brown. Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. *Journal of Machine Learning Research*, 17:1–5, 2016.

[18] Edmund K Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.

[19] Nguyen Thi Thanh Dang and Patrick De Causmaecker. Characterization of neighborhood behaviours in a multi-neighborhood local search algorithm. In *International Conference on Learning and Intelligent Optimization*, pages 234–239. Springer, 2016.

[20] Nguyen Dang, Leslie Pérez Cáceres, Patrick De Causmaecker, and Thomas Stützle. Configuring irace using surrogate configuration benchmarks. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 243–250. ACM, 2017.

[21] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Identifying key algorithm parameters and instance features using forward selection. In *International Conference on Learning and Intelligent Optimization*, pages 364–381. Springer, 2013.

[22] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. An efficient approach for assessing hyperparameter importance. In *Proceedings of the 31th International Conference on Machine Learning*, volume 32, pages 754–762, 2014.

[23] André Biedenkapp, Marius Lindauer, Katharina Eggensperger, Frank Hutter, Chris Fawcett, and Holger H. Hoos. Efficient parameter importance analysis via ablation with surrogates. In Satinder P. Singh and Shaul Markovitch, editors, *AAAI Conference on Artificial Intelligence*. AAAI Press, February 2017.

[24] Nguyen Dang and Patrick De Causmaecker. Analysis of algorithm components and parameters: some case studies. 2018. Accepted at the International Conference on Learning and Intelligent Optimization (LION12).

[25] Nguyen Thi Thanh Dang and Patrick De Causmaecker. Motivations for the development of a multi-objective algorithm configurator. In *Proceedings of the 3rd International Conference on Operations Research and Enterprise Systems*, pages 328–333. SCITEPRESS, 2014.

[26] Shunji Tanaka and Shuji Fujikuma. A dynamic-programming-based exact algorithm for general single-machine scheduling with machine idle time. *Journal of Scheduling*, 15(3):347–361, 2012.

[27] Eckart Zitzler, Joshua Knowles, and Lothar Thiele. Quality assessment of pareto set approximations. *Multiobjective Optimization*, pages 373–404, 2008.

[28] Sara Ceschia, Nguyen Thi Thanh Dang, Patrick De Causmaecker, Stefaan Haspeslagh, and Andrea Schaerf. The second international nurse rostering competition. In *PATAT 2014: Proceedings of the 10th International Conference of the Practice and Theory of Automated Timetabling*, pages 554–556, 2014.

[29] Haroldo G Santos, Túlio AM Toffolo, Rafael AM Gomes, and Sabir Ribas. Integer programming techniques for the nurse rostering problem. *Annals of Operations Research*, 239(1):225–251, 2016.

[30] IBM. ILOG CPLEX optimizer. `http://www.ibm.com/software/integration/optimization/cplex-optimizer/`, 2017.

[31] Luca Di Gaspero and Andrea Schaerf. EASYLOCAL++: an object-oriented framework for the flexible design of local-search algorithms. *Software: Practice and Experience*, 33(8):733–765, 2003.

[32] Nguyen Thi Thanh Dang, Sara Ceschia, Andrea Schaerf, Patrick De Causmaecker, and Stefaan Haspeslagh. Solving the multi-stage nurse rostering problem. In *Proceedings of the 11th International Conference of the Practice and Theory of Automated Timetabling*, pages 473–475, 2016.

[33] T Stützle and M Dorigo. The ant colony optimization metaheuristic: Algorithms, applications, and advances. *Handbook of metaheuristics*, pages 251–285, 2002.

[34] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1):29–41, 1996.

[35] GI Zobolas, Christos D Tarantilis, and George Ioannou. Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm. *Computers & Operations Research*, 36(4):1249–1267, 2009.

[36] Tony Wauters, Fulgencia Villa, Jan Christiaens, Ramon Alvarez-Valdes, and Greet Vanden Berghe. A decomposition approach to dual shuttle automated storage and retrieval systems. *Computers & Industrial Engineering*, 101:325–337, 2016.

[37] Túlio AM Toffolo, Eline Esprit, Tony Wauters, and Greet Vanden Berghe. A two-dimensional heuristic decomposition approach to a three-dimensional multiple container loading problem. *European Journal of Operational Research*, 257(2):526–538, 2017.

[38] Rainer Kolisch and Sönke Hartmann. Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis. In *Project scheduling*, pages 147–178. Springer, 1999.

[39] Mauro Birattari, Thomas Stützle, Luís Paquete, and Klaus Varrentrapp. A racing algorithm for configuring metaheuristics. In W. B. Langdon et al., editors, *Proceedings of the Genetic and Evolutionary Computation*

Conference, GECCO 2002, pages 11–18. Morgan Kaufmann Publishers, San Francisco, CA, 2002.

[40] Mauro Birattari. *The Problem of Tuning Metaheuristics as Seen from a Machine Learning Perspective*. PhD thesis, IRIDIA, École polytechnique, Université Libre de Bruxelles, Belgium, 2004.

[41] Manuel López-Ibáñez and Thomas Stützle. Automated offline design of algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1038–1065. ACM, 2017.

[42] Belarmino Adenso-Diaz and Manuel Laguna. Fine-tuning of algorithms using fractional experimental designs and local search. *Operations research*, 54(1):99–114, 2006.

[43] Aldy Gunawan, Hoong Chuin Lau, et al. Fine-tuning algorithm parameters using the design of experiments approach. In *International Conference on Learning and Intelligent Optimization*, pages 278–292. Springer, 2011.

[44] Volker Nannen and Agoston E Eiben. Relevance estimation and value calibration of evolutionary algorithm parameters. In *IJCAI*, volume 7, pages 975–980, 2007.

[45] Thomas Back. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

[46] Nikolaus Hansen. The CMA evolution strategy: a comparing review. *Towards a new evolutionary computation*, pages 75–102, 2006.

[47] Zhi Yuan, Marco A Montes De Oca, Mauro Birattari, and Thomas Stützle. Continuous optimization algorithms for tuning real and integer parameters of swarm intelligence algorithms. *Swarm Intelligence*, 6(1):49–75, 2012.

[48] Carlos Ansótegui, Yuri Malitsky, Horst Samulowitz, Meinolf Sellmann, and Kevin Tierney. Model-based genetic algorithms for algorithm configuration. In Qiang Yang and Michael Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI-15)*, pages 733–739. IJCAI/AAAI Press, Menlo Park, CA, 2015.

[49] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Automated configuration of mixed integer programming solvers. In A. Lodi, M. Milano, and P. Toth, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 7th International Conference, CPAIOR 2010*, volume 6140 of *Lecture Notes in Computer Science*, pages 186–202. Springer, Heidelberg, Germany, 2010.

[50] Leonardo C. T. Bezerra, Manuel López-Ibáñez, and Thomas Stützle. Automatic component-wise design of multi-objective evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 20(3):403–417, 2016.

[51] Manuel López-Ibáñez and Thomas Stützle. The automatic design of multi-objective ant colony optimization algorithms. *IEEE Transactions on Evolutionary Computation*, 16(6):861–875, 2012.

[52] Leonardo CT Bezerra, Manuel López-Ibánez, and Thomas Stützle. Automatic design of evolutionary algorithms for multi-objective combinatorial optimization. In *International Conference on Parallel Problem Solving from Nature*, pages 508–517. Springer, 2014.

[53] A. R. KhudaBukhsh, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. SATenstein: Automatically building local search SAT Solvers from Components. *Artificial Intelligence*, 232:20–42, 2016.

[54] Holger H. Hoos. Programming by optimization. *Communications of the ACM*, 55(2):70–80, February 2012.

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

[56] Marco Chiarandini and Yuri Goegebeur. Mixed models for the analysis of optimization algorithms. *Experimental Methods for the Analysis of Optimization Algorithms*, 1:225, 2010.

[57] Thomas Bartz-Beielstein, Christian WG Lasarczyk, and Mike Preuß. Sequential parameter optimization. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 1, pages 773–780. IEEE, 2005.

[58] Thomas Bartz-Beielstein, C. Lasarczyk, and Mike Preuss. The sequential parameter optimization toolbox. In Thomas Bartz-Beielstein, Marco Chiarandini, Luís Paquete, and Mike Preuss, editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 337–360. Springer, Berlin, Germany, 2010.

[59] Giles Hooker. Generalized functional ANOVA diagnostics for high-dimensional functions of dependent variables. *Journal of Computational and Graphical Statistics*, 16(3):709–732, 2012.

[60] Kenneth Sörensen and Fred W Glover. Metaheuristics. In *Encyclopedia of operations research and management science*, pages 960–970. Springer, 2013.

[61] Scott Kirkpatrick, C Daniel Gelatt, Mario P Vecchi, et al. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[62] Fred Glover. Tabu search—part I. *ORSA Journal on computing*, 1(3):190–206, 1989.

[63] John H Holland. Genetic algorithms. *Scientific american*, 267(1):66–73, 1992.

[64] Pablo Moscato and C Cotta. Memetic algorithms. *Handbook of Applied Optimization*, pages 157–167, 2002.

[65] Robert Johannes Maria Vaessens, Emile HL Aarts, and Jan Karel Lenstra. Job shop scheduling by local search. *INFORMS Journal on Computing*, 8(3):302–317, 1996.

[66] Luca Di Gaspero and Andrea Schaerf. Multi-neighbourhood local search with application to course timetabling. In *International Conference on the Practice and Theory of Automated Timetabling*, pages 262–275. Springer, 2002.

[67] Kenneth Sörensen, Marc Sevaux, and Patrick Schittekat. Multiple neighbourhood search in commercial VRP packages: Evolving towards self-adaptive methods. *Adaptive and multilevel metaheuristics*, pages 239–253, 2008.

[68] Bruce L Golden, Subramanian Raghavan, and Edward A Wasil. *The vehicle routing problem: latest advances and new challenges*, volume 43. Springer Science & Business Media, 2008.

[69] Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Computers & operations research*, 24(11):1097–1100, 1997.

[70] Ender Özcan, Burak Bilgin, and Emin Erkan Korkmaz. A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis*, 12(1):3–23, 2008.

[71] Donald Ratliff. 10 rules for supply chain and logistics optimization, 2013.

[72] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5):533–549, 1986.

[73] Mustafa Mısır, Pieter Smet, and Greet Vanden Berghe. An analysis of generalised heuristics for vehicle routing and personnel rostering problems. *Journal of the Operational Research Society*, 66(5):858–870, 2015.

[74] W Heid, G Hasle, and D Vigo. Verolog solver challenge 2014–VSC2014 problem description. *VeRoLog (EURO Working Group on Vehicle Routing and Logistics Optimization) and PTV Group*, pages 1–6, 2014.

[75] Túlio AM Toffolo, Jan Christiaens, Sam Van Malderen, Tony Wauters, and Greet Vanden Berghe. Stochastic local search with learning automaton for the swap-body vehicle routing problem. *Computers & Operations Research*, 89:68–81, 2018.

[76] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search: Framework and applications. In *Handbook of Metaheuristics*, pages 363–397. Springer, 2010.

[77] Edmund K Burke and Yuri Bykov. A late acceptance strategy in hill-climbing for exam timetabling problems. In *PATAT 2008 Conference, Montreal, Canada*, 2008.

[78] Edmund K Burke and Yuri Bykov. The late acceptance hill-climbing heuristic. *European Journal of Operational Research*, 258(1):70–78, 2017.

[79] Georges A Croes. A method for solving traveling-salesman problems. *Operations research*, 6(6):791–812, 1958.

[80] Mustafa Mısır, Stephanus Daniel Handoko, and Hoong Chuin Lau. OSCAR: Online selection of algorithm portfolios with case study on memetic algorithms. *Learning and Intelligent Optimization*, page 59, 2015.

[81] Erik Pitzer and Michael Affenzeller. A comprehensive survey on fitness landscape analysis. In *Recent Advances in Intelligent Engineering Systems*, pages 161–191. Springer, 2012.

[82] Raivo Kolde, Sven Laur, Priit Adler, and Jaak Vilo. Robust rank aggregation for gene list integration and meta-analysis. *Bioinformatics*, 28(4):573–580, 2012.

[83] John Aitchison. A concise guide to compositional data analysis. In *2nd Compositional Data Analysis Workshop, CoDaWork'05,*, 2005.

[84] Juan José Egozcue, Vera Pawlowsky-Glahn, Glòria Mateu-Figueras, and Carles Barcelo-Vidal. Isometric logratio transformations for compositional data analysis. *Mathematical Geology*, 35(3):279–300, 2003.

[85] Charles Bouveyron, Stéphane Girard, and Cordelia Schmid. High-dimensional data clustering. *Computational Statistics & Data Analysis*, 52(1):502–519, 2007.

[86] Laurent Bergé, Charles Bouveyron, and Stéphane Girard. HDclassif: An R package for model-based clustering and discriminant analysis of high-dimensional data. 2012.

[87] Gideon Schwarz et al. Estimating the dimension of a model. *The annals of statistics*, 6(2):461–464, 1978.

[88] Edmund K Burke, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R Woodward. A classification of hyper-heuristic approaches. In *Handbook of metaheuristics*, pages 449–468. Springer, 2010.

[89] Frank Hutter, Manuel López-Ibáñez, Chris Fawcett, Marius Thomas Lindauer, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. AClib: a benchmark library for algorithm configuration. In Panos M. Pardalos, Mauricio G. C. Resende, Chrysafis Vogiatzis, and Jose L. Walteros, editors, *Learning and Intelligent Optimization, 8th International Conference, LION 8*, volume 8426 of *Lecture Notes in Computer Science*, pages 36–40. Springer, Heidelberg, Germany, 2014.

[90] Domagoj Babić and Frank Hutter. Spear theorem prover. In *SAT'08: Proceedings of the SAT 2008 Race*, 2008.

[91] Emmanuel Zarpas. Benchmarking SAT solvers for bounded model checking. In Fahiem Bacchus and Toby Walsh, editors, *International Conference on Theory and Applications of Satisfiability Testing*, volume 3569, pages 340–354, 2005.

[92] Leslie Pérez Cáceres, Manuel López-Ibáñez, and Thomas Stützle. An analysis of parameters of irace. In Christian Blum and Gabriela Ochoa, editors, *Proceedings of EvoCOP 2014 – 14th European Conference on Evolutionary Computation in Combinatorial Optimization*, volume 8600 of *Lecture Notes in Computer Science*, pages 37–48. Springer, Heidelberg, Germany, 2014.

[93] Kevin Leyton-Brown, Eugene Nudelman, and Yoav Shoham. Empirical hardness models: Methodology and a case study on combinatorial auctions. *Journal of the ACM (JACM)*, 56(4):22, 2009.

[94] Frank Hutter, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111, 2014.

[95] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, 1998.

[96] Katharina Eggensperger, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Efficient benchmarking of hyperparameter optimizers via surrogates. In *AAAI*, pages 1114–1120, 2015.

[97] Manuel López-Ibáñez, Leslie Pérez Cáceres, Jérémie Dubois-Lacoste, Thomas Stützle, and Mauro Birattari. The irace package: User guide. Technical Report TR/IRIDIA/2016-004, IRIDIA, Université Libre de Bruxelles, Belgium, 2016.

[98] Thomas Stützle. ACOTSP: A software package of various ant colony optimization algorithms applied to the symmetric traveling salesman problem, 2002.

[99] Katharina Eggensperger, Marius Lindauer, Holger H Hoos, Frank Hutter, and Kevin Leyton-Brown. Efficient benchmarking of algorithm configuration procedures via model-based surrogates. *arXiv preprint arXiv:1703.10342*, 2017.

[100] David Pisinger and Stefan Ropke. A general heuristic for vehicle routing problems. *Computers & operations research*, 34(8):2403–2435, 2007.

[101] Paolo Toth and Daniele Vigo. *The vehicle routing problem*. SIAM, 2002.



FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
CODES
Etienne Sabbelaan 53 box 7659
8500 Kortrijk
https://set.kuleuven.be/codes