A Methodological Framework for Evaluating Metaheuristics: An Application to Vehicle Routing

Acknowledgements

Four years ago, I was a graduate commercial engineer about to embark on a doctoral research journey covering a topic that I soon learned to be a challenging one. Not just challenging for the work I had to invest in it myself, but also in selling the message this research aims to bring across. I am proud to be able to present this dissertation and believe it has helped me to acquire and develop skills that will be an added value for any future challenges I will be confronted with. Luckily, I could also count on the support of colleagues, friends and family in bringing this journey to a successful end.

First and foremost, I would like to thank my supervisor Prof. dr. An Caris for giving me the opportunity to become a PhD student at our research group. An, I got to know you as a supervisor who cares and always stands up for her PhD students, who encourages and motivates them to excel in their work, who provides valuable suggestions to make sure the correct and strongest message is brought across in research papers and who, fortunately, regularly leaves room for laughter. I am grateful for taking this journey under your guidance. I would also like to thank my co-supervisor Prof. dr. Benoît Depaire for navigating me, with great enthusiasm, past (statistical) obstacles I encountered. Benoît, what I will always remember is that you never seem to run out of ideas. When I was performing my first analyses, got stuck and thought I had tried it all, you always had several suggestions to overcome the current obstacle. After meetings, I often wished to have recorded the entire discussion, scared to forget the many new insights I learned from you. Together with An, you formed an ambitious and inspiring duo. I cannot imagine to be counselled by a better supervising duo these past four years.

I would also like to thank Prof. dr. Kris Braekers for making valuable contributions as a member of my doctoral committee and for always being available

to provide advice. Thank you as well to Prof. dr. Kenneth Sörensen for being part of my doctoral committee, providing useful feedback during these meetings as well as on my first research paper and for being a supportive next presenter at the Orbel conference in Antwerp after a quite *pretentious* first conference presentation. I am also grateful to Prof. dr. Patrick De Causmaecker for being a part of my doctoral jury and for showing an interest in this research years ago that resulted in a successful research collaboration with dr. Nguyen Thi Thanh Dang. And thank you Prof. dr. Yves Crama for being a member of this doctoral jury and for providing useful feedback to improve the quality of this dissertation.

These past four years I have been part of a group of colleagues with whom I could share laughs and frustrations, on whom I could count for assistance when needed and who motivated me to deliver the best possible work. Being able to share my workday with such a fun group of people only helped me to find joy in the tasks I performed each day. I was also fortunate to make new friends and go, among others, on city trips, amusement park excursions and death-defying walks. Gert, Marijke, Niels and Stef, thank you for your friendship and the truckloads of good times.

Finally, I would like to thank my family and friends for their support, their genuine interest in what I was doing and for providing the necessary distraction. Special thanks go to my brothers for being examples of determination and perseverance, and, above all, to my parents for supporting me in everything I do, for being ready to provide help in any way they can at any moment, and for the values they have instilled in me, making me the person I am today.

Jeroen Corstjens
Gellik, August 2018

Contents

List of Tables

List of Figures

Chapter 1

Introduction and Problem Statement

1.1 Introduction

Combinatorial optimisation problems, a class of optimisation problems characterised for having a finite set of solutions, comprise among others routing and timetabling problems and have great economical relevance. This has inspired the development of a large number of algorithms able to solve these kinds of problems. The distinction is made between exact and approximate algorithms. Many of these combinatorial optimisation problems are considered to be difficult to solve, a characteristic that in the computational complexity theory is referred to as being NP-hard. Exact algorithms that guarantee finding the optimal solution for any problem instance could require an enormous amount of computation time to find such a solution. Consequently, these algorithms are only applicable on small problem instances and are computationally intractable for larger ones. Therefore, the majority of the developed algorithms are approximate or heuristic algorithms which often provide good solutions in a reasonable computation time without guaranteeing it is the optimal solution (Birattari, 2009).

How effective or efficient a proposed heuristic method is, has to be assessed in a study. Two approaches exist towards such a study: theoretical or empirical. The former relies on deductive mathematics, while the latter is based on computational experiments (Hooker, 1994). Although the emphasis used to be on the theory-based

worst-case and average-case analysis, nowadays empirical analysis has become the dominant approach because it is very hard to analyse heuristic algorithm behaviour analytically (Moret and Shapiro, 2001) and it provides a more clear link between theory and practice (Johnson, 2002).

In empirical analysis, heuristic algorithm performance on some optimisation problem is most commonly evaluated by solving the instances of one or several popular benchmark problem sets followed by a comparison of the obtained results with those of existing (state-of-the-art) heuristic algorithms. This type of evaluation has a competitive focus. The objective is to present a heuristic method that is 'better' than current state-of-the-art methods, meaning it is able to obtain an improved solution quality, requires less computation time to realise the same solution quality, or it achieves a better trade-off of both measures. This competitive approach, however, has several drawbacks. It does not allow to explain why the new method is better (Hooker, 1995). Is it due to a new ingenious operator employed within the algorithm? Or due to a certain combination of operators that works well? Or perhaps due to a more efficient implementation of an existing method? Do all components significantly contribute to the performance of the algorithm, or can certain elements be left out, thereby possibly increasing the efficiency of the method? These are all questions that often remain unanswered when a new method is presented. Even though some competition between researchers might spur innovation, it has been noted that true innovation builds on the understanding of how a heuristic algorithm behaves, and not on proof of competitiveness (Sörensen, 2015).

This does not mean no knowledge is obtained from years of competitive experimentation, but it is the efficiency of this knowledge acquisition that is questioned. Historically, innovations in the field of metaheuristics are primarily founded on a researcher's experience and intuition, not on in-depth studies of how heuristic algorithms behave. Methods like Simulated Annealing (Kirkpatrick et al., 1983), Ant Colony Optimisation (Colorni et al., 1991) or genetic algorithms are optimisation procedures inspired on some natural (optimisation) phenomenon — annealing in metallurgy, navigation of ants from nest to food source, and Darwinian evolution respectively. Similarly, the concepts of intensification and diversification, first introduced in Tabu Search (Glover, 1989), simulate the approach an intelligent human being would take to solve a problem and have become key components underlying many metaheuristic algorithms (Sörensen et al., 2018). Likewise, ideas to extend existing concepts in order to reduce, for example, the computation time

required to obtain an acceptable quality have been successfully proposed. Granular neighbourhoods (Toth and Vigo, 2003) and reactive schemes (Battiti and Tecchiolli, 1994) are examples of extensions to Tabu Search. All these innovations depended on a researcher coming up with an idea that shows to be a worthwhile contribution, with its effectiveness evaluated primarily on benchmark instances. Furthermore, the many ideas that were tried and failed, but which may contain valuable insights, are never reported. A detailed analysis of a heuristic algorithm in order to completely understand how it behaves for an entire population of problem instances and how the different elements off which it consists contribute to achieving a performance value can lead more swiftly to ideas for new components, search strategies or perhaps complete metaheuristic frameworks. It is a more structured and active approach to look for innovative breakthroughs rather than having to wait for an idea to pop up.

The ultimate goal of science is to understand, not to win a "horse race" (Sörensen, 2015). This observation was already made more than twenty years ago by Hooker (1995) who stated, "We have confused research and development; competitive testing is suited only for the latter". Hooker (1995) regrets the time spent on adjusting the heuristic parameters to values that perform well on the used benchmark sets and on developing the fastest possible code, time that he argues could otherwise be spent on productive investigation, on learning about the problem and the algorithm under study. A competitive evaluation methodology is useful when the objective is to develop the fastest possible procedure for a specific environment. When the goal is to understand how performance is obtained, to discover which elements in the heuristic algorithm contribute to its performance and to draw conclusions that are valid beyond the specific problem instances chosen, one has to rely on a statistical evaluation method (Rardin and Uzsoy, 2001).

The aim of this doctoral thesis is to promote an approach to heuristic experimentation that is focused on gaining insight and understanding of how heuristic performance is established. To fulfil this aim, an evaluation methodology is proposed based on the concepts of design of experiments and its use demonstrated through several experimental studies. Although the proposed methodology can be applied on a wide range of optimisation problems, in this thesis the focus is on analysing heuristic performance for the vehicle routing problem with time windows. Therefore, a brief introduction to vehicle routing is given in the next section.

1.2 Vehicle Routing

Transport is a crucial sector in today's economy and society. In the European Union (EU) it accounts for about 5% of the total gross value and employs around 11.2 million people, 5.2% of the total workforce based on 2015 data. Freight transportation, in particular, accounts for an important share in the economic activities. The majority of goods transported is still done by road vehicles — 49% of total EU freight transport (The European Commission, 2017). This stresses the importance of routing and scheduling problems.

In 1959, Dantzig and Ramser (1959) formulated the truck dispatching problem and started a routing research field that has since spurred an enormous amount of research work. In its basic form the vehicle routing problem (VRP) is the problem of finding a set of routes to distribute goods from a depot to a number of customers with the objective of minimising a total cost measure. This basic version of the vehicle routing problem has been extended through the years by incorporating, for example, the use of heterogeneous vehicles, multiple depots, split delivery, et cetera. One popular extension is the vehicle routing problem with time windows (VRPTW) in which a number of customers have to be served at minimum cost without violating the customers' time-window constraints and the vehicle-capacity constraints. Example applications are newspaper delivery, food and beverage delivery and industrial waste collection (Cordeau et al., 2007). Exact algorithms, searching for the optimal solution, usually minimise the total distance travelled, while most heuristic methods consider a hierarchical objective that first minimises the number of vehicles used and then the total distance travelled. Given the high complexity of the problem, exact algorithms are only suitable for small[1] problem instances and the emphasis typically is on heuristic solution procedures. In this thesis the focus is on the experimental analysis of heuristic algorithms applied to the VRPTW since the importance of the problem in many distribution systems has spurred intensive research efforts for both heuristic and exact optimisation approaches (Bräysy and Gendreau, 2005). Yet, relatively few research articles apply statistical techniques to evaluate heuristics for these problems or seek to understand how they influence heuristic algorithm behaviour.

[1] According to Gendreau and Tarantilis (2010), it can be intractably hard to solve a VRP instance optimally if it has more than 100 customers.

1.3 Research Objectives

Experimental research on heuristic methods for optimisation problems has thus far typically focused on improving performance results. The goal generally is to do better than competing algorithms. A next step, in my view, is to asses heuristic methods performance, not on their competitiveness with other methods, but on the elements of which they consist. We want to learn, for example, if a proposed search strategy works as hypothesised, how it can be improved, or whether it should be abandoned for other ideas (Chapter 4). We want to know which elements contribute most to performance and which elements worsen the performance results and should be left out (Chapter 5). Additionally, we want to know what influence the specific problem instance has on the performance impact of the various heuristic elements (Chapter 3). Does a certain heuristic operator perform equally well for all problem sizes, or does it perform differently for smaller and larger instance sizes? In short, we want to gain insight and understanding of how heuristic algorithm performance on a specific problem instance is established to generate knowledge. This knowledge can be used during the design and optimisation process as well as the comparative analysis. It should lead to the most efficient and effective heuristic design that is competitive for a given problem instance.

The primary objective of this thesis is to propose a statistical evaluation framework that facilitates an experimental study that can bring insight in the behaviour of a heuristic method on a specific problem instance. The focus is not on comparing the performance of different algorithms, but on analysing an individual algorithm and study the relationship between the elements of which it consists and the performance it obtains as well as how the performance impact of these elements depends on the specific instance of a problem that is to be solved. The aim is to enable researchers to gain a thorough understanding of the relationship between algorithm performance, algorithm parameters and components, and problem instance characteristics. A parallel can be drawn with petri dish studies that are commonly performed in microbiology. Such experimental studies are aimed at learning in a controlled laboratory environment how tissue or individual cells react to, for example, newly developed drugs before these are administered to human subjects in clinical trials and approved for commercial use (Gibco, 2016). The research performed in this dissertation, similarly, considers both a problem and solution method in a very controlled environment to learn about their interplay before applying it in a real-world context using the knowledge obtained from the 'laboratory' study.

The proposed framework focuses on setting up and conducting a rigorous experiment and subsequently analysing and interpreting the results in such way as to obtain valid and objective conclusions. The methodology consists of an iterative process starting with one or multiple observations that lead to questions for which answers are formulated and verified in consecutive experimental studies, which in turn often lead to new observations and questions. As pointed out by Box et al. (2005): learning is advanced by iteration. How to apply the proposed evaluation framework in different contexts, such as confirmatory analyses to explain specific performance observations or automatic algorithm configuration in which suitable parameter values are determined, is a secondary objective covered in this thesis.

1.4 Outline Thesis

The outline of this thesis is presented in Figure 1.1. Chapter 2 discusses the practice of experimenting with heuristic algorithms, the common competitive approach towards the assessment of heuristic performance results, and how it compares to the description of an experiment as found in the literature on design and analysis of experiments. It is elaborated what an approach focused on insight and understanding entails and, regardless of the approach, an overview is given of the different steps to follow when setting up an experimental study.

Chapter 3 presents a methodology to rigorously evaluate heuristic algorithms. The methodology is based on a multilevel experimental design to efficiently study how the different algorithm elements (i.e., parameters and components) are correlated to performance and how the problem instance correlates with these algorithm elements. The application of the methodology is illustrated in a case study that analyses the performance results of a large neighbourhood search framework obtained for solving a number of instances of the vehicle routing problem with time windows.

Chapters 4, 5 and 6 cover three different contexts in which the proposed multilevel methodology is applied. Chapter 4 is focused on explaining results. A detailed analysis is performed on one of the observed patterns in Chapter 3. Certain combinations of operators are found to perform differently. In particular, it is observed that the more simple operator is predicted to perform better than the operator employing a more sophisticated logic. This counterintuitive observation sparked the question why

the simpler operator performs better and answers are sought in the experimental studies performed in this chapter.

Chapter 5 extends the methodology of Chapter 3 and proposes a combined methodology that no longer considers all algorithm elements and problem instance characteristics, but rather concentrates the analysis on those elements that have the most important impact on the performance measure. It involves combining the proposed multilevel methodology with a functional Analysis of Variance (fANOVA) that provides an importance ranking of the various effects considered. Based on this ranking a multilevel regression model is fitted including only the most important effects, thereby reducing the complexity of the model and enabling explanatory studies to be performed on these important effects.

In Chapter 6, the potential of the multilevel methodology for obtaining well-performing algorithm configurations is explored. The regression analysis is used to derive decision rules for the various algorithm parameters and components. These decision rules take the problem instance characteristics as inputs and return a value that optimises heuristic performance. It is studied how the performance of these problem-specific algorithm configurations relates to the performance of a configuration obtained with *irace*. The latter is an automatic algorithm configurator tool that suggests a configuration that performs best on average across all problem instances.

In Chapter 7, finally, the conclusions of this thesis are presented along with opportunities for future research.

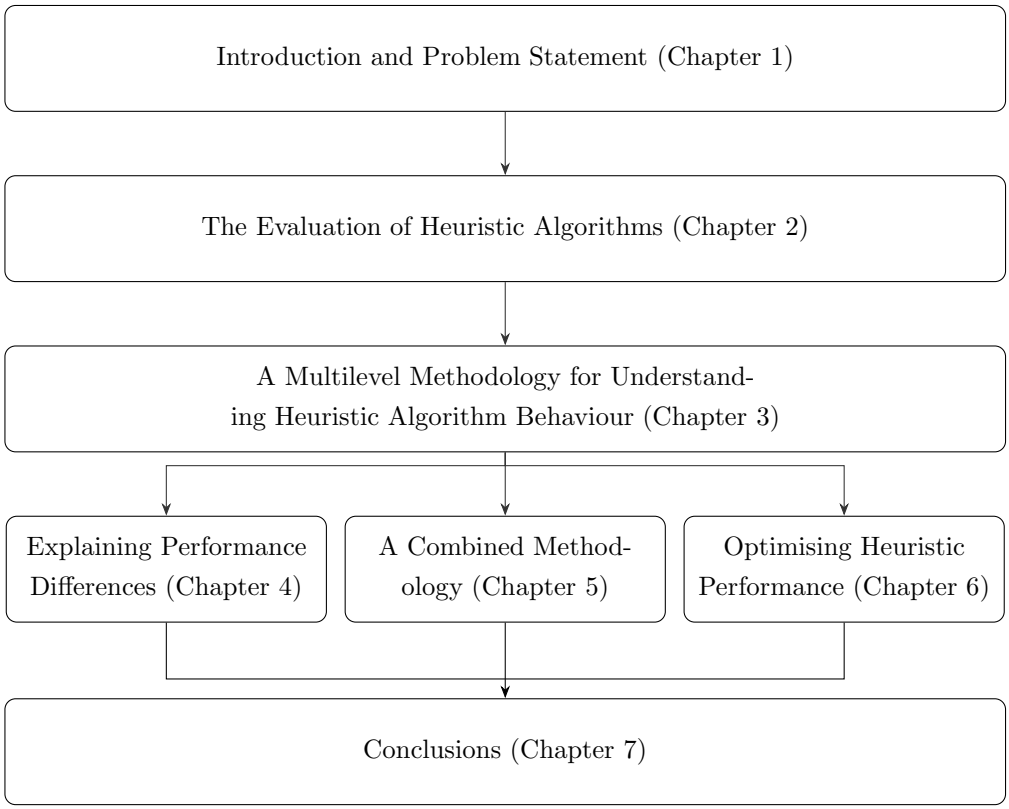


Figure 1.1: Outline of Thesis.

Chapter 2

The Evaluation of Heuristic Algorithms

2.1 Introduction

Heuristic algorithm performance is most commonly studied in an empirical analysis (Bräysy and Gendreau, 2005; Gendreau and Tarantilis, 2010), acquiring information based on evidence gathered from computational experiments. This evidence is never absolute, but expressed as likely evidence based on probability. Since heuristics — in particular metaheuristics — are formulated at a conceptual level, their evaluation occurs through computational experimentation on a specific implementation (Barr et al., 1995). According to the literature on experimentation, such an experiment is defined as a (series of) test(s) performed while controlling for conditions that might have an effect on the output. The aim is to observe the output and identify the reasons for possible differences in this output when input factors change. These reasons can be discovered by formulating hypotheses and verified through statistical analysis of the data. The result of this analysis will give a more profound knowledge about a certain process (Montgomery, 2012). Barr et al. (1995) similarly state that "investigators in all fields of study perform experiments to demonstrate theory, to uncover knowledge about a particular process, and to measure the effect of one or more factors on some phenomena". But, as will be discussed, such a "true" experiment is rare in the literature on combinatorial optimisation.

This chapter reviews the practice of experimentation with heuristic algorithms and

the assessment of heuristic performance (Figure 2.1). First, the common competitive approach towards evaluating experimental results is discussed in Section 2.2. It is followed by an introduction of the approach issued in this thesis, one that is focused on understanding how the performance results are obtained (Section 2.3). Both approaches require a rigorous experimental set-up, but this is often lacking (Section 2.4). Therefore, an overview of the different steps in the experimental process is provided in Section 2.5 — as stipulated in the literature on experimentation —, and they are discussed in the context of heuristic experimentation to enable a valid and objective evaluation of experimental results. These steps will be relied on when setting up experimental studies in the following chapters.

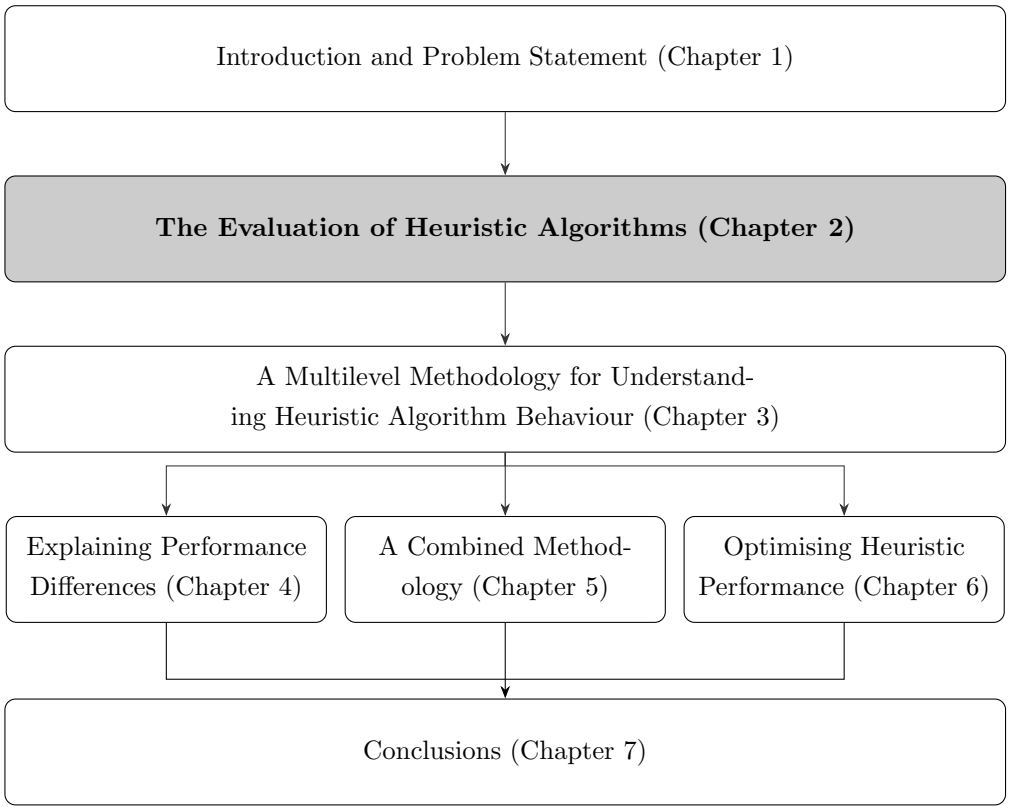


Figure 2.1: Outline of Thesis — Chapter 2.

2.2 Experimentation in a Competitive Context

The purpose of a computational experiment with a heuristic algorithm can be to show that the method is able to obtain a feasible solution for some newly defined (variant of an) optimisation problem that is identified by a researcher. In time, new problems become known problems and simply finding a feasible solution no longer suffices. The goal of an experimental study shifts to showing that a presented heuristic method is 'better' in some way than competing methods. So initially it might be satisfactory to find any feasible solution, but eventually it becomes the search for better — or at least equally good — solutions than previously found best solutions. The goal of an experimental study with heuristic methods, therefore, usually focuses on beating the competition for some performance measure.

The set up of an experiment involves making a number of choices for which the researcher often has a lot of leeway. For example, the experimenter can freely decide what computer environment is used, how to implement the algorithm, which problem instances and algorithm parameter values to select, and how to perform this selection, what performance measures to use, how to report the results, etc. (Barr et al., 1995).

Regarding the assessment of performance, there are various criteria to base the evaluation on (Cordeau et al., 2002; Bräysy and Gendreau, 2005): ease of implementation, robustness to varying problem characteristics or flexibility to cope with changes in the constraints or objective function. *Ease of implementation* refers to the complexity of the algorithm. The more complex it is to understand and code, the less likely researchers are to adopt it (Gendreau and Tarantilis, 2010). *Robustness* implies the ability of a heuristic algorithm to perform well on a wide range of problem instances and not just a single instance (Barr et al., 1995). *Flexibility* relates to the heuristic method being able to cope with various side constraints present in many real-life applications (Gendreau and Tarantilis, 2010), for example, limitations on goods that can be transported together or certain customers having to be visited before others (Kilby et al., 2000). The two most frequently applied criteria for comparing performance results are the *solution quality* — measured by the objective function — and *run time*, along with their trade-off since the performance value a heuristic method obtains is a compromise between a better solution quality and a shorter computation time. These measures are most commonly evaluated in an experiment that involves the computational testing of a heuristic algorithm by having it solve a set

Table 2.1: Example comparison table

Benchmark instance (class)	Heuristic 1	Heuristic 2	Heuristic 3
1			
2			
3			
4			
5		performance values	
6			
7			
8			
9			
10			
Computer		CPU information	
Run time		run time values	

of problem instances from one or several popular benchmark libraries. The results of such an experiment are typically presented in a table similar to table 2.1 with the best performance value per benchmark instance highlighted. Three relevant factors to performance in an empirical analysis are the test environment, the problems studied and the parameter values chosen for the algorithm.

2.2.1 Test Environment

The test environment factors that can influence the experimental results obtained are the hardware and software used as well as the programmer (Barr et al., 1995). Hardware concerns, among others, the number of processors used, CPU speed and the memory that is required. This information has to be reported to enable other researchers to set up a comparison based on similar resources. Software details include the operating system used (e.g., Windows, Linux), the programming language (e.g., C++, Java, Python) the heuristic algorithm is coded in, the compiler, etc. Finally, the programmer has an impact. Not all programmers are equally skilled at programming an algorithm and, therefore, different methods are most likely not equally efficiently implemented. Likewise, when manually tuning the algorithm parameters, the more experienced experimenter will probably find better values faster than someone who is less competent in parameter tuning. It will often be a question of how much time an experimenter wishes to invest in coding and tuning.

It is not necessary to report all details since this will only divert a reader's attention from the actual experimental results. A practitioner should include the information that is directly relevant for the conclusions drawn. All other information can be provided through different channels or in a working paper (McGeoch and Moret, 1999; Rardin and Uzsoy, 2001).

2.2.2 Benchmark Instances

Heuristic experiments typically solve one or several benchmark instance sets. Benchmark instances are computationally difficult problem instances — either randomly generated or based on some real-world applications — which are used by researchers to test the efficiency or effectiveness of some optimisation method (Sifaleras, 2014). The use of these existing benchmark sets enables the comparison of results on a by-instance basis and the calculation of the relative gap between two heuristics (Silberholz and Golden, 2010). In the VRPTW context, the most popular benchmark set for comparing performance results is that of Solomon (1987). His benchmark of 56 problem instances each have 100 customers, a central depot, capacity constraints, time windows on the time of delivery, and a total route time constraint. Furthermore, a distinction is made with regard to the positioning of customers and scheduling horizon. Other popular benchmark sets for the VRPTW are the problem instance set provided by Gehring and Homberger (1999), which is an extension of the Solomon instances with 200, 400, 600, 800 and 1000 customers; and the real-world based benchmark set provided by Russell (1995). The results obtained on these instances are then compared with best-known solutions provided by other heuristic methods. The aim is to do better in the sense of obtaining new best solutions or reduced computation time for the same average solution value.

2.2.3 Algorithm Design Choices

An important contributing factor in this aim are the values chosen for the various algorithm parameters. This is especially relevant for metaheuristic methods as they often contain a large number of parameters. The set of parameter values a heuristic method uses to solve a problem instance is referred to as a parameter setting or algorithm configuration.

In the past parameter values were predominately determined in a time-consuming ad-hoc way (Adenso-Díaz and Laguna, 2006; Johnson, 2002), based on a trial-and-

error approach, tests on a limited sample of benchmark instances or values were quoted from the literature without any rigorous examination of their suitability in the used context (Ridge and Kudenko, 2007). The importance of a parameter setting for performance has spurred research efforts to find more rigorous ways of determining these values. This has led to the formulation of the algorithm configuration problem.

The algorithm configuration problem is formalised by Birattari (2002). Given an algorithm with a parameter space Θ, a set of problem instances I and a performance metric $C(\theta)$ measuring the performance of the algorithm on problem instance set I given configuration θ, the solution to the configuration problem is the configuration θ^* such that:

$$\theta^* = \operatorname*{argmin}_{\theta} C(\theta) \tag{2.1}$$

Several automated methods have been developed to solve this problem and replaced the manual effort with machine effort. These automatic configurators have led to significant time savings in the development of complex algorithms, and to a more fair comparison of multiple algorithms (Hutter et al., 2009). They are commonly divided into two groups: model-free and model-based procedures. Both approaches try to find parameter settings that perform well over a large set of problem instances. The model-free procedures search the configuration space without relying on some predictive model to guide the search, but on heuristic rules, randomness and simple experimental designs (Dobslaw, 2010). Model-based procedures use information from previously evaluated parameter settings to build a prediction model of the algorithm configuration space and then select new candidate configurations to be evaluated in subsequent iterations (Hutter et al., 2011).

Model-free Examples of model-free approaches are numerical black-box optimisation procedures such as Covariance Matrix Adaptation Evolution Strategy (CMA-ES) (Hansen and Ostermeier, 2001) and mesh adaptive direct search (MADS) (Audet and Orban, 2006) that can be applied in scenarios where all parameters are numerical. Besides these direct search methods there is relevance estimation and value calibration (REVAC), an evolutionary algorithm based on information theory to estimate the relevance of parameters (Nannen and Eiben, 2007). Gender-based Genetic algorithm (GGA) divides the population in a competitive (directly evaluated) and non-competitive (serving as a source of diversity) part and recombines genomes from both parts in a crossover operator to generate new candidate configurations (Ansótegui et al., 2009). Parametric iterated local search (ParamILS), finally, is an

iterated local search algorithm with random perturbations and restart strategies that can solely handle categorical and ordinal parameters (Hutter et al., 2009).

Model-based An example of a model-based approach is iterated F-Race (I/F-Race)(Balaprakash et al., 2007). It is a racing procedure, a class of algorithms applied in machine learning to tackle the model selection problem. Each iteration candidate algorithm configurations are sampled from a probabilistic model and evaluated on a set of problem instances. Candidates are discarded once they are significantly inferior to other configurations and the surviving configurations are used to update the probabilistic model in order to increasingly focus sampling towards the most promising configurations. The method is implemented in the R package *irace* (López-Ibáñez et al., 2016). It is a model-based approach since it uses information from previous evaluations to select new candidate configurations in subsequent iterations, but it does not map the dependency of a parameter setting to algorithm performance. Model-based approaches that do fulfil this feature belong to the more generic approach known as sequential model-based optimisation (SMBO) and operate within the black-box optimisation problem setting (Hoos, 2011). Their development stems from the fact that the evaluation of configurations is computationally demanding since the algorithm has to actually run each configuration to know its performance. The idea behind SMBO is to reduce the computational effort by using surrogate models that can predict how a certain configuration will perform on one or multiple problem instances and then use these predictions to determine which configurations are worthwhile to evaluate. The actual evaluations are then used to update the prediction model (López-Ibáñez et al., 2016). Sequential Parameter Optimisation (SPO) (Bartz-Beielstein et al., 2005) and Sequential Model-based Algorithm Configuration (SMAC) (Hutter et al., 2011) are two examples of SMBO methods relying on surrogate models. SPO is a general framework that iteratively evaluates configurations, uses this information to build a surrogate model — e.g., a classical regression or tree-based model — that predicts the performance of not previously evaluated configurations. Each iteration the surrogate model is updated based on the information obtained from the previous iteration (Bartz-Beielstein et al., 2005). The method is implemented in the R package *Sequential Parameter Optimization Toolbox (SPOT)* (Bartz-Beielstein et al., 2010). SMAC relies on random forest models to predict the performance of configurations and on an expected improvement criterion to select promising configurations to be evaluated. Contrary to SPO, SMAC can handle categorical variables and optimisation for sets of instances (Hutter et al., 2011). It is available as a python package.

The success these configurators achieved has also inspired a software design paradigm called Programming by Optimisation in which programmers are encouraged to parameterise design choices as much as possible instead of committing to one design alternative. This leads to a rich design space from which an automated configurator can then determine the best design alternative for a specific problem. The idea is to allow researchers to focus their efforts on the creative process of thinking about, for example, new search strategies and mechanisms for problems and use machine effort for the process of determining what works best in a given problem context (Hoos, 2012).

All these methods have helped in achieving improved performance results and the dominant focus, therefore, remains on boosting the performance measure. It is not common to investigate the reasons for possible differences — better or worse — in the observed performance. The experimental goal of understanding performance and characterising how design choices influence heuristic algorithm behaviour is an objective often disregarded by practitioners. It shows that a traditional combinatorial optimisation experiment is different from an experiment as described by Montgomery (2012). The focus is much more on winning a "horse race" for some performance measure. As a next step, this doctoral thesis focuses on performing experimental research that aims to provide insight and understanding in the heuristic method used as well as the problem to be solved. The approach to such an experimental study is discussed in the next section.

2.3 A Focus on Insight and Understanding

The description of an experimental study as given in the introduction of this chapter (Section 2.1) summarises what is known as the scientific method or hypothetico-deductive method, the most common description of a set of steps scientists use to search for answers to research questions (cf. Figure 2.2) (Dodig-Crnkovic, 2002). There are roughly four steps.

Observe An experimental investigation most frequently starts by observing some phenomenon that sparks one or several research questions. In the context of heuristic experimentation, observations will most likely be made after first performing one or multiple screening experiments.

Theory Next, the search for explanations is initiated by asking questions which result in proposed theories that clarify what is going on. These ideas for clarifying the observed phenomenon can be based on, for example, a review of the literature or personal observations.

Hypothesis From these theories a set of conjectures or hypotheses are deduced that can, if not rejected, provide an answer to the observed phenomenon. In order to test whether or not some factor contributes to the observed phenomenon, a null hypothesis is usually formulated stating it has no significant influence on the phenomenon while the alternative hypothesis states it does have a significant influence.

Experiment The hypotheses are validated in controlled experiments that provide statistical evidence whether or not the null hypotheses can be rejected. Based on these results, the decision is made whether to reject a suggested theory or whether different conjectures are to be formulated. If rejected, a modified or completely new theory is formulated and the process repeated. If not rejected, the new insights acquired might expose other patterns that again lead to new research questions, new theories, hypotheses and so on. It shows that experimentation is not a one-time effort, but should be considered as an iterative process in which each iteration gains a deeper knowledge of the algorithm, while raising questions that need to be answered (Barr et al., 1995; McGeoch, 1996; Dodig-Crnkovic, 2002; Montgomery, 2012; Bartz-Beielstein and Preuß, 2014).

2.4 Towards a More Rigorous Experimentation

Whether the experimental goal has a competitive focus or seeks to obtain a better understanding of how heuristic performance is established, in either case, a rigorous approach towards experimentation is important since the reproducibility and strength of conclusions are key elements in any scientific or engineering discipline.

In order to interpret any value properly, rigorous experiment designs should be used, along with clear reporting and explicit hypotheses which are supported by the appropriate statistical tools. As Barr et al. (1995) point out “When an experiment’s results can vary with the test conditions (problems solved, computer, parameter settings), statistical methodology is the only objective approach to analysis”.

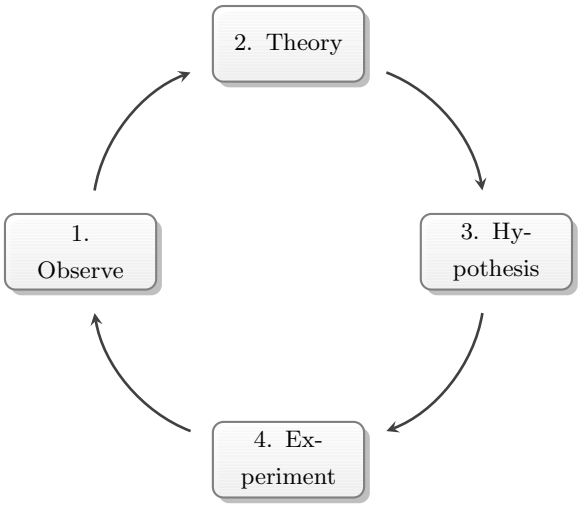


Figure 2.2: The hypothetico-deductive method.

Nonetheless, such proper practice is currently more an exception than a custom (Ridge and Curry, 2007), even though the use of design of experiments (DOE) methods is common practice within — among others — the physical sciences. Scanning the literature on vehicle routing, a small number of publications are found that use either DOE techniques or statistical tools for exploring data and testing hypotheses. These papers are listed in Table 2.2. This scarcity of papers is even more astonishing given that the need for more scientific rigour in the operations research and heuristics community was already appealed for many years ago. In the 1970s, Lin and Rardin (1979) already noted that the techniques of statistical experimental design are a basis to structure computational experiments on. In the 1980s, Golden et al. (1986) discussed the use of statistical methods to correct the arbitrary and subjective practice applied in empirical analysis. In the 1990s, Amini and Racer (1994) stated that conclusions on the performance of heuristic algorithms are questionable if the experimental data is not obtained using a statistically valid approach. Similar appeals are made by Barr et al. (1995), Hooker (1994, 1995), McGeoch (1996), Ahuja and Orlin (1996), Rardin and Uzsoy (2001) and Ridge and Kudenko (2006).

The topic of proper experimentation with heuristic algorithms has received increased attention over the years, within and beyond the field of vehicle routing. The majority of the VRP publications listed in Table 2.2 have been published after 2012. These research works rely on experimental designs — mostly a full factorial

Table 2.2: Vehicle Routing Papers that apply DOE

Author	**Year**	**Experimental design and statistical test**	**Purpose**
Abdoli et al.	2017	Fractional factorial, ANOVA	Parameter setting
Assis et al.	2013	Randomized block design, Gore test	Performance comparison
Coy et al.	2000	Fractional factorial, linear regression	Parameter setting
McNabb et al.	2015	Cluster analysis, least squares regression	Effect analysis operators
Karakatič and Podgorelec	2015	Friedman test, Nemenyi post hoc test	Performance comparison
Palhazi Cuervo et al.	2014	Full factorial, mixed-effects ANOVA	Identifying key elements, Parameter setting
Rahimi-Vahed et al.	2013	Fractional factorial, Friedman test	Parameter setting, Performance comparison
Rodríguez and Ruiz	2012	Full factorial, ANOVA	Effect problem factors
Saremi et al.	2007	Full factorial, ANOVA	Parameter setting
Silva et al.	2013	Factorial design, ANOVA	Performance comparison
Sörensen and Schittekat	2013	Full factorial, multi-way ANOVA, paired t-test	Parameter setting Performance comparison
Talarico et al.	2015	Full factorial, ANOVA	Parameter setting
Tyasnurita et al.	2017	Fractional factorial, ANOVA	Parameter setting

design — in which various combinations of parameter values are considered, followed by a statistical test such as an Analysis of Variance (ANOVA) to verify whether the parameter setting has an influence on the performance measure or not. Additional post-hoc tests are necessary to identify which specific values are responsible for better performance. Beyond the field of vehicle routing, relevant papers are Chiarandini and Goegebeur (2010), Hutter et al. (2013, 2014), Fawcett and Hoos (2015) and Smith-Miles et al. (2014) that focus on identifying which algorithm elements or element values contribute most to a good performing heuristic method. Chiarandini and Goegebeur (2010) rely on mixed effects models — as in this thesis — to distinguish the effects of algorithm parameters and problem instance characteristics for a 2-edge connectivity augmentation problem. Hutter et al. (2013, 2014) and Fawcett and Hoos (2015) introduce approaches for investigating the importance of algorithm parameters. Hutter et al. (2013) identify important parameters of algorithms for the propositional satisfiability problem (SAT), mixed integer programming (MIP), and the travelling salesman problem (TSP) by repeated model learning using forward selection. Hutter et al. (2014) rely on a functional analysis of variance framework (fANOVA) to assess the relative importance of algorithm parameters and showed that performance variability can often largely be attributed to a small number of algorithm parameters. Fawcett and Hoos (2015)

present an automated technique that iteratively modifies parameter settings from a source to a target configuration in order to identify which parameter level changes induce the largest performance differences between the two configurations for SAT, MIP, and AI planning problems. Smith-Miles et al. (2014) extend the algorithm selection model of Rice (1976) and focus on the problem instance characteristics that are discriminating of algorithm performance. Their proposed methodology seeks to identify within a defined problem instance space where an algorithm has a unique advantage or disadvantage over other algorithms such that a recommendation can be formulated of which algorithm is expected to perform well for which part of the problem instance space.

Furthermore, the surgence of the previously discussed automated algorithm configurators over the last decade, often inspired on concepts from machine learning, has notably acknowledged the importance of a proper experimental methodology (Birattari, 2002). Introducing a more formal procedure to determine parameter settings compared to the tedious and error-prone trial-and-error approach (Birattari, 2009), these configurators have led to subtantial time savings in the development of complex algorithms, and to a more fair comparison of multiple algorithms (Hutter et al., 2009). Nonetheless, they often do not provide much insight on why identified elite parameter configurations perform better than other ones. SMAC, for example, relies on the machine learning algorithm random forest to predict the performance of a certain parameter setting. While it does provide accurate predictions, interpretation is limited to an exploratory analysis. It is possible to deduct information on variable importance and visually analyse the kind of relationship parameters have with a performance measure using partial dependence plots (Jones and Linder, 2015). If a confirmatory analysis with hypotheses testing is to be performed on the prediction data, parametric statistical models such as classical linear regression models are better suited. In addition, SMAC applies a guided, and thus non-random, sequential sampling approach for the parameter settings. This makes it uncertain whether any claims made hold on average across the entire parameter setting space (Hutter et al., 2013). The methodology that will be presented in Chapter 3 uses an unbiased data set by randomly sampling parameter settings and problem instances. This is noted as a characteristic of a good experimental design and allows valid statements to be made for the entire configuration and problem space considered (Barr et al., 1995).

Experimenters might be reluctant towards using statistics as it requires an addi-

tional time investment and since the techniques often rely on a number of assumptions concerning the data that might not reflect reality (Chiarandini and Goegebeur, 2010). The lack of a proper statistical approach may result, however, in drawing conclusions that are only valid for the observed data and cannot be generalised.

The necessary means to conduct such a rigorous experiment are provided by the field of Design of Experiments. It offers established experimental designs and statistical analysis tools in order to collect the appropriate data and to draw valid and objective conclusions with mathematical preciseness (Adenso-Díaz and Laguna, 2006; Montgomery, 2012). In the next section a review of the different steps required to set up such an experiment are given and discussed in the context of heuristic optimisation.

2.5 The Experimental Process

The steps in the experimentation process can be roughly identified as (i) stating the goal of the experiment, (ii) choosing the performance measures and input factors, (iii) designing and executing the experiment, (iv) analysing the results and drawing conclusions, and finally (v) reporting the experimental findings (Barr et al., 1995; Montgomery, 2012). Furthermore, as pointed out in the hypothetico-deductive method in Section 2.3, experimentation is not a one-time effort, but should be considered as an iterative process in which each iteration gains a deeper knowledge of the algorithm, while raising questions that need to be answered (Barr et al., 1995; McGeoch, 1996; Bartz-Beielstein and Preuß, 2014).

2.5.1 Setting the Experimental Goal

The experimental goal provides guidance and focus in setting up an experiment and should therefore be explicitly stated beforehand. The questions to be answered should be made clear. The two distinct experimental goals have already been highlighted previously. Rardin and Uzsoy (2001) pose it as *research vs. development*. The former puts the experimental focus on gaining insight into what does or does not work and why this is the case. These insights should not be limited to the specific problem studied, but are transferable to other problems. The latter purpose focuses on producing the best performing heuristic algorithm for a specific environment. This puts details on the algorithm implementation and problem application in the center of attention since the aim is to fine-tune these details such that the best performance is obtained. Experimenters still spend a lot of time and effort on development issues

rather than on characterising heuristic algorithm behaviour.

Once it is known which questions are to be answered, the appropriate performance measures as well as problem and algorithm parameters are chosen since the analysis will be performed on the observations of these factors and measures.

2.5.2 Performance Measures

The data collected from an experiment are the values measured for some response variable, which should provide useful information for the objectives set in the first step (Dean et al., 1999). The most commonly chosen performance measures for comparing heuristic methods are the solution quality and computation time. Ideally, the solution quality of a heuristic method is assessed by its deviation from optimality — expressed as a percentage gap. Given multiple problem instances, an average percentage error over all problems can then be calculated. Such a metric requires that the optimal solution for every problem instance tested is known, which seldom is the case. Measurement of the deviation from optimality is only possible for small problem instances that can still be solved by exact algorithms. As an alternative, experimenters most commonly evaluate solution quality by the deviation from best-known solutions obtained on the problem instances of available benchmark libraries (Silberholz and Golden, 2010).

The computational effort a heuristic algorithm requires to find superior solutions is measured as the CPU time from initialisation until some stopping criterion is reached. Making sure that a comparative analysis for run times is fair — a crucial requirement in any comparison (Bräysy and Gendreau, 2005; Gendreau and Tarantilis, 2010) – can be an issue. First and foremost, in some cases the number of runs or CPU time required to get the result is not reported. This makes it impossible to conclude anything about the efficiency of methods or even start a comparison (Bräysy and Gendreau, 2005). Furthermore, as the use of heuristic algorithms is motivated by the trading of solution quality for reduced computation time, reporting on run times is therefore relevant to understand how the trade-off works out for the investigated algorithms (Johnson, 2002). Nonetheless, simply reporting total run time is insufficient as additional implementation details are required to make the best possible evaluation. This entails providing access to algorithm source code enabling the experimenter to compile and run the algorithms on the same computer system. The issue of different programming languages can be overcome through

the use of computer run time comparison tables (e.g., Dongarra (1993)), allowing a meaningful comparison to be made. The downside is that these run time multipliers only provide a rough idea of run time performance (Silberholz and Golden, 2010) and it is preferable to make comparisons within the same computer environment (Barr et al., 1995).

Yet, an evaluation based on run time still remains troublesome for the reproducibility of results as Johnson (2002) points out. He argues that reporting the best found solution after running the algorithm for, say, an hour does not ensure reproducibility of results. If a different computer with a different processor or operating system, or a more/less efficient implementation of the algorithm on the same computer is used, it is possible to obtain solutions with different levels of quality (Xu and Kelly, 1996). This impedes the reproducibility of any heuristic algorithm comparison since the use of a faster machine will not only lead to presumably improved results for all algorithms, but may also substantially change their ranking. Johnson (2002) therefore proposes to employ a measurable combinatorial count as the stopping criterion, to make a fair comparison that is also reproducible. This count measure can, for example, be the number of neighbourhoods searched, the number of branching steps, etc. In this way, run time and solution quality can be measured as a function of the combinatorial count and this will result in a well-defined algorithm. Ridge and Kudenko (2007), however, express concerns about this measure, because the use of a hard iteration stopping criterion risks biasing results. Fixing the number of iterations does not imply that a more difficult problem can be solved as well as an easier problem, especially when problem instance characteristics — that are hypothesised to affect performance — vary. The authors propose using stagnation as a stopping criterion, referring to a number of iterations in which no solution improvement is observed. This measure offers the reproducibility of a combinatorial count, but avoids the likelihood of bias. It incorporates problem difficulty in the stopping criterion of the algorithm.

2.5.3 Input Factors

The input factors that are hypothesised to have an effect on the performance measure are divided in two categories: factors characterising the problem instance to be solved and factors belonging to the heuristic algorithm that determine how search proceeds.

2.5.3.1 Test Instances

The problem factors or characteristics (e.g., problem size) are often a fixed list of values as the chosen test instances usually belong to a well-known library set. The use of existing benchmark sets allows comparisons on a by-instance basis and calculating the relative gap between two heuristics (Silberholz and Golden, 2010), but the risk in using the same benchmark problem set over and over again, is that the algorithm gets "fine-tuned" to obtain good results on these problem instances. Such practice would risk producing algorithms overspecialised on a given instance set and possibly performing much worse on other instances that differ only slightly. Moreover, these instances are only a sample of the population and could therefore contain patterns which might disappear in the entire population and might lack patterns present in the population. As a consequence, developing and optimising algorithms which outperform other algorithms for a particular sample of instances, might overfit the sample. In that case, the results are not necessarily generalisable to instances not observed in the sample (Birattari, 2009). Hence, a heuristic algorithm that performs well on a set of standard benchmark problems instances might not perform well on other instances (Bertsimas and Simchi-Levi, 1996). This issue of a lacking robustness can be resolved by using an independent second problem instance set, the test set, which produces an unbiased performance estimate. In addition, a problem instance generator could be used to randomly produce a new set of instances for each evaluation performed. Birattari (2009) prefers the latter approach whenever real-world instances are not available. Such a generator should be able to produce large problems instances such that optimal solutions cannot be calculated in reasonable run times. It also gives the experimenter control over the problem characteristics, enabling the creation of a diverse set of instances covering all parts of the problem space (Rardin and Uzsoy, 2001). It is important to provide others access to the developed problem instances or even to the instance generator in order to make comparisons easier (Silberholz and Golden, 2010).

The use of the same benchmark problem sets over and over again has also long ignored the importance of the problem instance characteristics for optimising heuristic performance. In machine learning Rice (1976) formulates the algorithm selection problem and acknowledges the no-free-lunch theorem by Wolpert and Macready (1997) that there is not a single best algorithm for all problem instances. This research area focused on identifying the strengths and weaknesses of algorithms by examining regions of the problem instance space where algorithms perform respec-

tively strong or weak and on identifying the characteristics relevant to problem instance difficulty (Smith-Miles and Bowly, 2015). Smith-Miles and Lopes (2012) and Shukla et al. (2013) apply it for combinatorial optimisation problems. Shukla et al. (2013), for example, formulated an efficient frontier for a portfolio of algorithms targeted at minimising computational cost and maximising solution quality for a class of instances of the vehicle routing problem with stochastic demand.

2.5.3.2 Parameterised Design Choices

The algorithm factors or parameters (e.g., stopping criterion, move-generating operator) are typically more freely controlled by the experimenter. The search for a performance-optimising set of parameter values, referred to as the algorithm configuration problem, has already been discussed in Section 2.2.3. It is still not uncommon to tackle this issue based on trial-and-error, testing on a limited sample of benchmark instances or quoting from the literature without any rigorous examination of their suitability in the used context instead of some rigorous statistical procedure (Ridge and Kudenko, 2007). Such work method gives rise to several issues. Information is seldom given on what human and machine resources are used for tuning the algorithm. This is nonetheless important information when comparisons are made with alternative algorithms. A simpler heuristic could, for example, already have been run multiple times while the more complex heuristic was still in the phase of ad-hoc parameter tuning. A second issue involves the misrepresentation of algorithm potential when parameter tuning is not performed in a clearly defined and methodical way. Even small adjustments in the parameter tuning can lead to large shifts in performance (Ridge and Curry, 2007). The ad-hoc approach in setting parameter values makes it difficult to estimate the robustness of proposed algorithms to changes in problem characteristics. A heuristic algorithm that works well on a set of standard test problems does not necessarily perform well on any particular data set and the resulting algorithm is usually extremely sensitive to changes in the data.

Automated algorithm configurators can provide clarity on these issues. However, careful tuning of an algorithm to a specific problem set, can lead to unfair comparisons on these instances. It is recommended to pick a representative subset to train parameters on, the training set, and use the complementary subset, the testing set, for testing and comparing metaheuristics. This will avoid overfitting the algorithm to a particular data set (Silberholz and Golden, 2010). The configurator *irace*, for example, applies this strategy.

2.5.4 Choice of Experimental Design and Execution of Experiment

The field of Design of Experiments (DOE) provides established experimental designs and statistical analysis tools in order to collect the appropriate data and to draw valid and objective conclusions with mathematical preciseness (Adenso-Díaz and Laguna, 2006; Montgomery, 2012). The choice of experimental design relates to determining the randomisation to apply to decrease possible bias, the number of replications to run, and whether blocking should be applied to prevent any undesired variability from nuisance factors. It is also important to keep the experimental goal in mind (Montgomery, 2012). Desirable characteristics of a good experimental design to test heuristic methods are – according to Barr et al. (1995) — unbiasedness, realisation of the experimental goals, clear demonstration of process performance, exposure of causes for performance, a justifiable rationale, generation of supportable conclusions, and reproducibility. Factorial or latin-squares are examples of experimental designs that are able to produce results in line with the aforementioned characteristics.

2.5.5 Analysis of Results

After execution of the experiment, the collected numerical data is to be converted into information through analysis and interpretation. Data analysis should rely on statistical methods to ensure objective results and conclusions. The experimental goals formulated at the start of the experimental process should be addressed and the questions posed are to be verified through hypothesis testing and estimation of confidence intervals. These questions can range from "Does method A obtain a better average solution quality than method B?" to "Which value(s) for algorithm parameter A works best on average on instance i and why?". In the latter case it can be useful to formulate empirical models to investigate the relationship between performance and the relevant input factors. Further, it is always helpful to visualise experimental results in plots. The availability of software programs facilitates this data analysis process (Montgomery, 2012).

It is important to keep in mind that statistical methods do not prove the causal relationships between input factors and performance, but they do provide indications of reliability and validity of results. It can be calculated how much confidence can be conferred to conclusions or what the likely error is. Again, statistical methods bring objectivity to reported findings and lead to sound conclusions (Montgomery, 2012).

The analysis and interpretation of experimental results might expose previously unobserved phenomena that spark new research questions for which answers are sought in new experiments. As mentioned, experimentation is not a one-time effort, but should be considered an iterative process in which each iteration gains a deeper knowledge of the algorithm, while raising questions that need to be answered (Barr et al., 1995; McGeoch, 1996; Bartz-Beielstein and Preuß, 2014).

2.5.6 Reporting Results

In this final step of the experimental process the scientific merit of the experiment should be argued by providing sufficient information that convinces the reader of the validity of the conclusions. A proper reporting is also essential for the reproducibility of results. This includes presenting information on how the experiment is conducted (e.g., computing environment), sources and characteristics of problem instances, significance and variability of result values, and sharing algorithm code or at least a sufficiently detailed description of the implementation (Rardin and Uzsoy, 2001). The latter does not need to be included in a research paper, but can be made available in, for example, an online repository. Furthermore, it is equally important to report possible negative results that are obtained next to positive results as these may point out, for example, specific types of problems on which the algorithm performs a lot worse than on others (Barr et al., 1995). In practice, however, negative results are rarely reported (Hooker, 1994; Smith-Miles and Bowly, 2015). Whenever new best solutions are found, they will be reported as a victory of the algorithm. Yet, when results are worse the argumentation often focuses on how small the deviations are from the best-known solution or results are possibly not even published. The primary reason for this lack of negative results in reporting is probably because they are unattractive to publish and hard selling points for a research paper that seeks journal publication (Hooker, 1994).

Another issue in reporting is that often only the best results obtained during multiple executions are given. When algorithms contain random elements, however, each execution for the same problem instance will provide different results. A best found solution is in such a situation less probable to be reproducible than an average since a best solution is a sample from the tail of a distribution. Moreover, such 'best solutions' are usually reported with the run time of a single algorithm run, instead of the cumulative number of runs needed to obtain this 'best solution' (Johnson,

2002). Therefore, only average results based on multiple executions would be a good basis for the comparison of non-deterministic methods, supported with additional measures on the distribution of results, such as minima, maxima and standard deviations. It occurs nonetheless too often that only the best result is reported rather than an indication of the central tendency of the observations (Johnson, 2002; Birattari and Dorigo, 2007).

The reporting of results forms the concluding step in the experimental process. The presented set of steps offer a researcher guidance in setting up and conducting a rigorous experimental analysis. This chapter discusses them in the context of heuristic optimisation. A more general and detailed discussion can be found in various books on design and analysis of experiments, such as Box et al. (2005) and Montgomery (2012). The practice of rigorous experimentation contributes in improving the design, optimisation and comparative analysis phase of heuristic algorithms.

2.6 Conclusion

Heuristic experimentation typically has a competitive focus, methods rivalling with each other for the top spot in the horse race. Little research has been conducted in which an experiment is set up to find out which elements operating within the heuristic algorithm actually contribute to performance or what the reasons for observed performance differences are. It is a research gap this doctoral thesis aims to fill.

The obtained insights and knowledge from the analysis can be useful in the design, optimisation and comparison of heuristic methods. In the design phase insights should lead to the inclusion of only those elements that are crucial to its performance and exclude non-essential elements that could lead to inefficiencies. Further, the insights are also useful when optimising existing heuristic algorithms, as the deployment of these methods often involves selecting appropriate values for a multitude of algorithm parameters. It is shown that applying a rigorous procedure to determine parameter values results in a better performing heuristic algorithm compared to parameter values that are determined using a trial-and-error approach or limited testing (Birattari, 2009). Finally, an algorithm with an optimal setting of its parameters can be compared against other heuristic methods. The comparative analysis usually relies on widely used benchmark problem sets. As mentioned in Section 2.2.2, the popularity of these instances makes it easier to compare different

algorithms, but the risk is that the algorithm gets overfitted to the specific problem set. The comparison of different methods on particular benchmark sets is also usually done without the use of the proper techniques necessary to make these comparisons statistically valid. Such techniques are nevertheless necessary to be able to evaluate whether any observed differences in performance are due to differences in the algorithms, or to chance.

In the next chapter a statistical evaluation framework is proposed for analysing the relationships between algorithm parameters, problem instance characteristics and heuristic performance. The framework is model-based, works on an unbiased data set, takes into account the problem instance influence and enables the formulation and validation of hypotheses. It shares the advantages of SMAC, a model-based automatic algorithm configurator. The use of models enables identification of possible correlations between parameter settings and algorithm performance. They allow interpolation of performance between parameter settings, extrapolation to unseen regions of the parameter space and quantification of the importance of each parameter, as well as of parameter interactions (Hutter et al., 2011). In addition, an unbiased data set is used by random sampling parameter settings and problem instances, in line with the guidelines for setting up a good experimental design, enabling valid statements to be made for the entire configuration and problem space considered.

In this doctoral thesis the framework's use for the design and optimisation of heuristic algorithms is shown. It is applied to identify and explain algorithm elements' correlation with performance such that suitable design choices can be made given the problem instance to be solved (Chapters 3 to 5). A second application is to optimise heuristic performance by selecting appropriate parameter values (Chapter 6).

Chapter 3

A Multilevel Methodology for Understanding Heuristic Algorithm Behaviour

3.1 Introduction

This chapter[1] introduces the statistical framework that is used throughout this thesis for generating and analysing experimental data (Figure 3.1). It is summarised in Figure 3.2. First, a data set of scenarios is generated, with a scenario defined as a combination of a certain problem instance with a certain parameter setting. A parameter setting is interpreted as a set of values and included operators. It is common for the algorithm parameters to be under control of the designer while the characteristics of a problem instance to be solved usually are not. In the proposed framework, however, both groups of factors are under the experimenter's control. The data set is created according to a two-phase sampling procedure in which first a number of problem instances are randomly generated and then a number of parameter settings are randomly defined for each problem instance. This results in the formulation of a multilevel experimental design (Section 3.2). The algorithm runs each scenario returning a desired performance measure. The scenarios with performance results are analysed by fitting a multilevel regression model (Section 3.3).

[1]This chapter is based on the paper: Corstjens, J., Depaire, B., Caris, A., Sörensen, K., in review. A multilevel evaluation method for heuristics with an application to the vrptw. Manuscript submitted for publication.

From the regression output, the relationships between performance, the algorithm parameters and the problem instance characteristics can be investigated and interpreted.

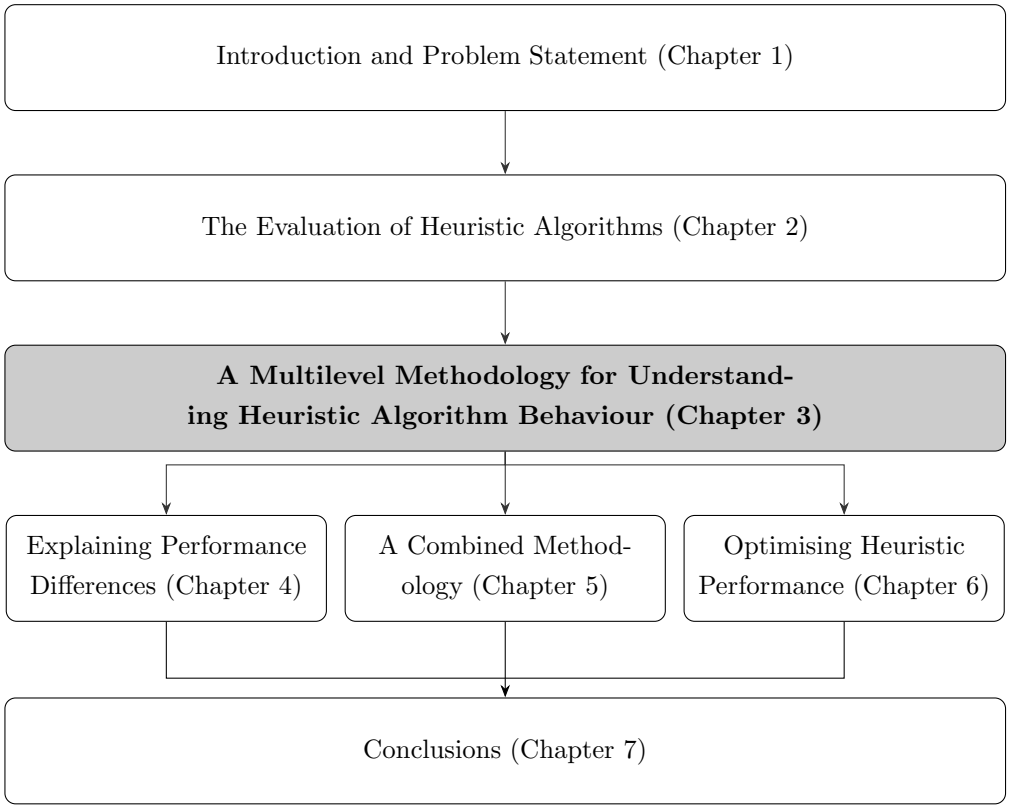


Figure 3.1: Outline of Thesis — Chapter 3.

The methodology is applied in a case study (Section 3.4) that examines the performance results obtained when solving instances of the vehicle routing problem with time windows (VRPTW) using a large neighbourhood search (LNS) algorithm. The aim is to identify how the various LNS parameters impact the obtained VRPTW solutions, positively or negatively, and how these effects vary across different parts of the problem space. In this chapter the first steps of the hypothetico-deductive process (Section 2.3) will be performed, i.e. looking for patterns in the performance data and start formulating hypotheses that might explain them. The remaining steps are treated in Chapter 4 where the focus is on explaining one of the observed patterns.

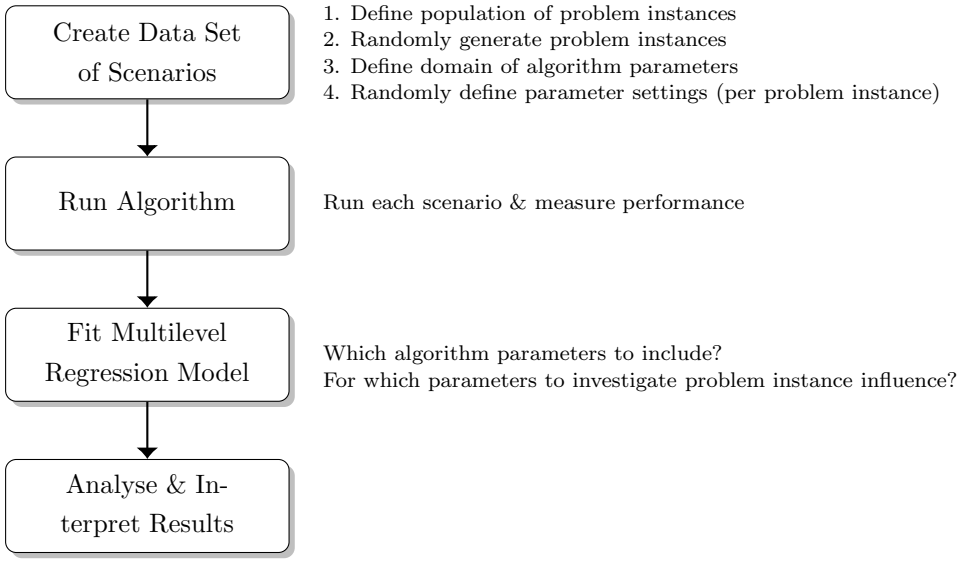


Figure 3.2: Diagram of Multilevel Methodology.

3.2 Experimental Design

The aim of the proposed framework is to expose how the various algorithm parameters relate to performance and how the problem instance influences the performance impact of the algorithm parameters. In order to be able to study these relationships effectively many different combinations of parameter settings and problem instances have to be analysed. A way of reducing the number of combinations and thereby the computational effort required without losing statistical power is by introducing a hierarchical structure in the data, i.e., testing several different parameter settings on a single problem instance such that it is clear that any performance differences observed are due to the algorithm parameters and not due to the problem instance. Doing this for multiple problem instances enables the exposure of the problem instance influence. Therefore, the methodology relies on multilevel models to efficiently study how effects vary by the group (or in this case problem instance) they belong to. These types of models are regularly applied in social research to investigate how individuals interact with the social contexts they belong to. For example, some research may focus on investigating the nationwide test scores of pupils. Individual skill levels will have an important effect on the obtained score, but since pupils attend specific schools, this might also have an influence (e.g., one school might have a better math teacher) (Hox et al., 2010). In the context of heuristic experimentation,

Table 3.1: Multilevel Experimental Design

	Instance Characteristics				**Algorithm Parameters**			
Scenario i	X_1	X_2	...	X_p	Z_1	Z_2	...	Z_k
1	0.5	10	...	0	0.2	12	...	0
2	0.5	10	...	0	0.4	5	...	1
3	0.5	10	...	0	0.9	15	...	1
4	0.5	10	...	0	0.9	6	...	0
5	1.2	8	...	1	0.6	10	...	1
6	1.2	8	...	1	0.1	1	...	1
7	1.2	8	...	1	0.5	5	...	1
8	1.2	8	...	1	0.2	3	...	0
...	...	...	...	...	...	...	...	...

it is similarly reasoned that the performance impact of, for example, a heuristic operator might depend on the specific problem instance to be solved and only has a beneficial effect on performance when the problem size is large (e.g., more than 300 customers) or it is better at coping with tight time windows than other operators.

The chosen multilevel experimental design considers two levels. First, the population of problem instances is defined by specifying the probability distributions (e.g., uniform, normal, ...) for different problem instance characteristics (e.g., problem size, time window width, vehicle capacity, customer demand, service time, ...). Next, from this population, a random sample of artificial instances is drawn. On a second level the algorithm parameters are considered. Likewise, probability distributions are specified for the different algorithm parameters. Multiple parameter settings will then be created by randomly selecting values and components. The problem instance characteristics and algorithm parameters are further discussed in Sections 3.4.2 and 3.4.3.

An illustration of a multilevel experimental design is given in Table 3.1. Each randomly generated problem instance is solved by a heuristic with a fixed number of randomly chosen parameter settings. Variables X_1 to X_p represent the different problem instance characteristics and variables Z_1 to Z_k represent the different algorithm parameters. For scenario i going from one to four the values for the problem instance characteristics remain the same while the values for the algorithm parame-

ters change. In other words, four different parameter settings are tested on the same problem instance. For scenario i going from five to eight a second problem instance is defined with again four different parameter settings for this second instance. Each row in the table represents a unique combination of problem instance characteristics and algorithm parameter values, and is referred to as a scenario.

3.3 Regression Model

The analysis of the multilevel design is performed by relying on regression models, to obtain complete insights over the full range of algorithm parameter values and problem instance characteristics. These models have the added benefit over classical ANOVA — the common analysis approach — that statements can be made for the complete range of values. Classical ANOVA, on the other hand, is limited to categorical variables and therefore limits insights into performance to the algorithm parameter levels that are measured. Further, the multilevel data structure in Section 3.2 demands a multilevel regression analysis since it violates the assumption of independent error terms made by traditional regression analysis. Performance observations for parameter settings on the same problem instance are expected to be more similar than observations for parameter settings on different problem instances. This means that individual observations are not completely independent and might cause problems when using standard statistical tests, which rely on the assumption of independent error terms. The violation of this assumption could lead to inaccurate statistical estimations due to biased (i.e., underestimated) standard errors that result in spuriously significant results (Hox et al., 2010). Classical regression models including interaction terms also do not allow the inclusion of both problem-level indicators (specifying the problem instance) as well as problem-level predictors (i.e., the characteristics defining a problem instance, e.g., number of customers) because this would cause collinearity of the predictors. Therefore a multilevel regression analysis is applied that takes the hierarchical structure of the data into account and that provides a clear model that accounts for both individual- and group-level effects.

In theory, a multilevel model could be fit using only a single parameter setting per problem instance, but as this could lead to imprecise estimations, multiple parameter settings are included (Gelman and Hill, 2006). In social research, there are some rules of thumb to be found regarding the sample size, depending on the interest of the experimenter. If the interest is mostly in the estimates that do not

vary per group, a '30/30' rule is a good guideline, i.e., 30 groups with 30 individuals per group (Hox et al., 2010). However, if one is interested in cross-level interactions, for example, interactions between algorithm parameters and problem instance characteristics, a '50/20' rule is suggested. Even more groups are preferred if the accuracy of variance and covariance estimates is important, leading to a '100/10' rule (Busing, 1993; Van der Leeden and Busing, 1994; Hox et al., 2010). All these rules account for the computational cost when collecting data and therefore decrease the number of individuals per group when the number of groups is increased (Hox et al., 2010). The general conclusion is that the number of groups seems to be more important than the number of individuals per group (Maas and Hox, 2005). The analysis in Section 3.4 considers 200 problem instances, so this is well above the suggested minimum numbers in order to have accurate estimates.

Similar to classical regression, multilevel regression typically relies on a number of assumptions. The residual terms are assumed to be independent — which is guaranteed through the multilevel design — and to follow a normal distribution. The latter is considered to be the least important assumption and Gelman and Hill (2006) even advise against testing this assumption. The third and final assumption of equal error variance states that the residual terms should be unrelated to any variable and is verified by plotting the residuals.

The multilevel design in Table 3.1 is translated into the following multilevel regression model. The general formulation below considers all variables as numerical, but algorithm element variables can also be boolean to reflect the decision on whether to activate an algorithm component or not, or can be categorical. Since the aim is to show how to formulate a multilevel regression model, no non-linear effects or variable interactions (within the same level) are included in order to focus on the multilevel aspect. The regression model used for the analysis in Section 3.4.4 does consider non-linear effects and variable interactions.

$$Y_i = \alpha_{j[i]} + \sum_{k \in K} \beta_{kj[i]} Z_{ki} + \epsilon_i \tag{3.1}$$

$$\alpha_j = \mu_0^{\alpha} + \sum_{p \in P} \mu_p^{\alpha} X_{pj} + \eta_j^{\alpha} \tag{3.2}$$

$$\beta_{kj} = \mu_0^{\beta_k} + \sum_{p \in P} \mu_p^{\beta_k} X_{pj} + \eta_j^{\beta_k} \qquad \forall k \in K \tag{3.3}$$

with

$i \in I$	scenario, a combination of a certain problem instance with a certain parameter setting
$j \in J$	problem instance
$k \in K$	algorithm parameter; associated variables are $Z_k i$
Z_{ki}	variable for algorithm parameter k in scenario i
$p \in P$	problem instance characteristic; associated variables are X_p
X_{pj}	variable for problem instance characteristic p in problem instance j
$j[i]$	index variable to code problem instance membership ($j[i] = j$), e.g., $j[90] = 5$ means the 90th scenario involves problem instance 5
Y_i	objective function value of scenario i
$\alpha_{j[i]}$	varying regression intercept, representing the objective function value given scenario i and problem instance j when $Z_{ki} = 0 \; \forall \; Z_{ki}$
μ_0^{α}	global intercept value
$\beta_{kj[i]}$	varying effect of algorithm parameter k on Y given scenario i and problem instance j
$\mu_0^{\beta_k}$	mean effect of algorithm parameter k on Y
$\mu_p^{\beta_k}$	effect of problem instance characteristic p on the coefficient β of algorithm parameter k
η_j	error at the problem instance level and is assumed to be $\sim N(0,\sigma^2)$
ϵ_i	error at the parameter setting level and is assumed to be $\sim N(0,\sigma_e^2)$

Equation (3.1) represents the regression model at the parameter setting level and estimates the impact of the algorithm parameters (Z_k) on the objective function value Y (e.g., total distance covered for a routing problem) as expressed by the regression coefficients (the β's). The α coefficient in this equation represents the regression intercept and is defined as the expected objective function value when all included predictor variables (i.e., the Z variables) are set equal to zero. If the parameters that are represented by the Z variables cannot be zero, the intercept has no meaningful value. This does not have to be an issue since the researcher is interested in the relationship between the Z variables and the Y variable, and the

regression intercept does not provide information on this relationship. Nevertheless, a meaningful value for the regression intercept in such a scenario can be obtained by centring the Z variables on their mean value such that, for example, $Z_1 = 0$ corresponds to the mean value for parameter 1 and the regression intercept can be interpreted as the expected objective function value when the algorithm parameters are at their mean value. The analysis in Section 3.4 will mean centre all numerical variables. The β coefficients quantify the change in the objective function value Y when the associated Z variables change by one unit.

Equations (3.2) and (3.3) represent the regression models at the problem instance level and measure the influence of the problem instance characteristics (X_p) on the intercept α and the performance impact the algorithm elements have (i.e., the β's). The multilevel model thus contains the algorithm parameters at the lowest level — the parameter setting or observation level — and are structured within a certain group or, in this case, problem instance level, where the problem instance characteristics are included. The coefficient μ_0^α in equation (3.2) represents the global intercept value when the problem instance characteristics are fixed at zero (i.e., $X_p = 0$). Similar to the algorithm elements, a zero value for these characteristics (e.g., number of customers) is not meaningful and by mean centring these variables μ_0^α can be interpreted as the expected objective function value for an average problem instance and with the algorithm parameters at their mean value. Likewise, $\mu_0^{\beta_k}$ in equation (3.3) is the expected effect of parameter k on the objective function value given an average problem instance. The μ_p coefficients quantify how α_j and β_{kj} change when the associated X_p variables change by one unit (e.g., when an additional customer has to be served).

Note that not all algorithm element coefficients (the β's) need to be modelled as a varying or random effect, but can also be modelled as having a constant impact across all problem instances, also known as a fixed effect. In this case the β coefficient will not be determined by a regression model at the problem instance level.

The set of equations (3.1) to (3.3) can also be re-expressed in a single regression equation by substituting (3.2) and (3.3) in equation (3.1). Equation (3.5) shows how the moderating effect of the X_p variables (at the problem instance level) on the relationship between the algorithm parameters (Z_k) and the dependent variable Y (at the parameter setting level) is expressed as a *cross-level interaction* in a single

equation version of the model.

$$Y_i = [\mu_0^\alpha + \sum_{p \in P} \mu_p^\alpha X_{pj[i]} + \eta_{j[i]}^\alpha] + \sum_{k \in K} [\mu_0^{\beta_k} + \sum_{p \in P} \mu_p^{\beta_k} X_{pj[i]} + \eta_{j[i]}^{\beta_k}] \times Z_{ki} + \epsilon_i \quad (3.4)$$

Or

$$\begin{aligned} Y_i =& [\mu_0^\alpha + \sum_{p \in P} \mu_p^\alpha X_{pj[i]}] + [\sum_{k \in K} \mu_0^{\beta_k} \times Z_{ki} + \sum_{k \in K} \sum_{p \in P} \mu_p^{\beta_k} X_{pj[i]} \times Z_{ki}] + \\ & [\epsilon_i + \eta_{j[i]}^\alpha + \sum_{k \in K} \eta_{j[i]}^{\beta_k} \times Z_{ki}] \end{aligned} \quad (3.5)$$

This regression model allows us to analyse how a single algorithm parameter has an impact on performance, under the influence of the problem instance characteristics. The focus is not on specific problem instances or particular algorithm parameter values. Instead the interest lies in the whole population of instances and value ranges. The aim is to gain a better understanding of how an algorithm parameter or component works and for which problem instance characteristics it performs well or not.

3.4 Experimental Analysis

In this section the statistical evaluation framework is illustrated on a case study that is introduced in Section 3.4.1, followed by a description of how the problem instances used in the experiments are generated (Section 3.4.2) and a discussion of the heuristic algorithm of which it is aimed to gain a better understanding (Section 3.4.3).

3.4.1 Case

An analysis is performed on the experimental results of a large neighborhood search (LNS) algorithm run on a number of instances for the vehicle routing problem with time windows (VRPTW). All experiments are performed on Intel Xeon E5-2680v2 CPUs (2.8 GHz, 25 MB level 3 cache) with 20 GB of RAM per core under Red Hat Enterprise Linux ComputeNode release 6.4 (Santiago), 64 bit. These resources are available from the infrastructure of the Flemish Supercomputer Center (`www.vscentrum.be`).

3.4.2 Problem Instance Generation

A problem instance generator is developed to create a desired number of artificial VRPTW instances. The instances provided by known benchmark problem sets, such

as the Solomon (1987) instances, are not used due to the previously (Chapter 2) mentioned concerns of overfitting and often unknown probability distributions of the characteristics of these instances.

The random generation of test instances enables proper statistical statements to be made about the experimental results. By applying a valid experimental design and statistical analysis, inference from a sample to all possible problem instances producable by the instance generator can be made (Lin and Rardin, 1979). Random sampling is preferred over some form of guided sampling as the interest of this research lies in investigating the entire problem instance space instead of focusing on mapping a small part of this area for which good performance measures are obtained. In this case, validity is more important than efficiency (Brus and De Gruijter, 1997). Rardin and Uzsoy (2001) point out the conveniences of using randomly generated problem instances. A properly designed instance scheme is able to produce a diverse population of instances since the researcher has complete control over the problem instance characteristics. The benefit of this diversity is that parts of the problem space are included that may not be expressed in available real data or benchmark problem sets. A well-documented generator also creates clarity on all problem instance characteristics, which may not be the case in existing benchmarks.

Some risk exists when using randomly generated instances. The design of the problem instance generator should ensure that the instances are sufficiently difficult and representative for the kind of problems the researcher aims to solve. Moreover, the question of which values to test needs to be answered. These risks are accounted for in the selection of the value ranges which are discussed in the next paragraphs. Whether a problem instance generator can produce problem instances that are also realistic is a research question on its own and is beyond the scope of this thesis. A possible approach in future research might be analysing some realistic data set provided by one or multiple companies to find indications of whether or not the assumed probability distributions for each of the instance characteristics correspond with reality.

The characteristics for which the combined values constitute a single problem instance are listed in Table 3.2. These values are partly based on the characteristics of the instances in the problem set of Solomon (1987). His representative benchmark set consists of problems containing one hundred customers, a central depot, capacity constraints, time windows on the time of delivery, and a total route time constraint.

All these features are included in the scheme. Not all values are determined randomly, some are kept constant for simplicity. The number of vehicles available is chosen to equal the number of customers in order to guarantee feasibility of the problem instance. The capacity of each vehicle is arbitrarily fixed at 150 units. The depot is also determined to be open during a fixed time window.

Table 3.2: Problem Instance Characteristics

Characteristic	**Type**	**Value Ranges**
Number of Customers	Integer	U[25, 400]
Capacity Vehicle	Integer	150
x/y-coordinates Customer	Integer	U[0,500]
Demand Customer	Integer	U[10,50]
Service Time Customer	Integer	TRIA(min,max)
		min∼U[10,30]
		max∼U[30,50]
Time Window Depot	Integer	Start = 0; End = 900
Time Window Customer		
- Time Window Centre	Integer	U[0 + Travel Time, 900 - Travel Time - Service Time]
- Time Window Width	Integer	TRIA[min,max]
		min∼U[20,50]
		max∼U[50,80]
- Start		Centre - 0.5*Width
- End		Centre + 0.5*Width
Run Time	Integer	TRIA(60,1800)

The problem instance characteristic values that are determined stochastically are either drawn from a uniform distribution or from a (symmetric) triangular distribution. The value ranges are given in Table 3.2. Unlike Solomon's instances, clustered or semi-clustered customers are not considered in order to limit the number of characteristics under investigation in this example. This can however, be easily incorporated in the generator. The service time for each customer is drawn randomly from a symmetric triangular distribution with a minimum and maximum value drawn from a uniform distribution. A triangular distribution is chosen, because the assumption is made that the time necessary to unload goods is the same at every customer. The time window constraints are constructed in a similar way as for the Solomon benchmark set. The maximum CPU time the algorithm is allowed to run

on the problem instance is defined as a problem instance characteristic and not as an algorithm parameter since a context in which a problem instance has to be solved within a certain time frame is assumed. This makes it typical for the problem instance and not a parameter that can be set to obtain the best performance results.

Further assumptions made are that the triangle inequality holds, the travel cost between two nodes is the same in both directions (i.e., symmetry), and the common assumption of constant speed (Cordeau et al., 2007) is made so that distances, travel times and travel costs have the same proportions.

In Appendix A more details are given on the generated problem instance sample as well as the Python script.

3.4.3 Large Neighborhood Search

The heuristic algorithm under investigation is a simplified version of the Adaptive Large Neighbourhood Search (ALNS) algorithm developed by Pisinger and Ropke (2007). This popular heuristic method is applied in literature to multiple variants of the vehicle routing problem. What makes the ALNS a very interesting algorithm to investigate is the fact that it is a generic solution method with many algorithm parameter choices that can be analysed for their impact on performance.

The ALNS is based on a local search framework, e.g. simulated annealing as in the implementation used here. After constructing an initial solution, the algorithm iteratively generates a new solution by removing and reinserting customers based on a destroy and repair neighbourhood, randomly selected from a set of destroy and repair neighbourhoods. A candidate solution that improves upon the last accepted solution is always accepted, but if the candidate solution is worse, an acceptance probability is calculated based on the size of the deterioration and the likeliness to accept worse solutions. This process is outlined in lines 5 to 14 of Algorithm 1 and is repeated until some stopping criterion is met. The adaptive mechanism operating within the ALNS adjusts the weights of the destroy and repair neighbourhoods based on their past performance. The more an operator has contributed to finding a better solution, the greater the probability it will be chosen in future iterations. Finally, the algorithm has a two-stage approach that first seeks to minimise the number of vehicles by iteratively removing one route and scheduling the customers from this route into the remaining ones. If the algorithm is no longer

able to find a solution that can serve all customers, it continues with the last found feasible solution. A second phase is targeted at minimising the total distance travelled.

When planning experiments it is recommended to start small (Lawson and Erjavec, 2016). Therefore, the analysis is performed on a simplified version of the algorithm. The adaptive mechanism that updates the weights of the operators is removed and all operators are assigned an equal probability of being selected each iteration. In other words, in this experimental study a Large Neighbourhood Search (LNS) algorithm is considered. The number of operators is also scaled down. The set of destroy heuristics is limited to random, worst and related removal. Random removal is the simplest destroy operator and removes q randomly selected customers. Worst removal removes customers with the highest cost, while related removal looks for customers that are in some way related to each other (e.g., in terms of distance as in Pisinger and Ropke (2007)) and therefore easy to interchange. The q number of customers to remove is determined randomly each iteration and varies between 10% and 50% of the total number of customers. The set of considered repair heuristics are basic greedy search and regret-2. The greedy operator inserts customers in the cheapest route, while regret-2 looks ahead by also accounting for the second cheapest route. All operators as well as the remaining algorithm parameters are listed in Table 3.3. Pisinger and Ropke (2007) use a maximum number of iterations as a stopping criterion. As mentioned in the discussion of the problem instances (Section 3.4.2), a limited computation time is considered to solve a problem instance and, therefore, this value is applied as a stopping criterion for the algorithm. It is arbitrarily determined that 20% of this maximum run time is assigned to the vehicle minimisation phase. The pseudocode is given in Algorithm 1 and the implementation is available on `https://github.com/corstjens/lns.git`.

The *determinism parameter* serves as an input for the destroy operators worst and related removal. It is a measure of the amount of randomness involved in the selection of customers to remove from a solution. The higher this value, the more the selection is based on the ranking established in these operators. For worst removal, a high determinism value means removing customers with a high cost, while for related removal it means removing customers that are close to each other. Shaw (1998), the author who introduced the related removal operator, found that values less than 3 and greater than 30 performed poorly, therefore the interval used here takes 30 as an upper bound. This initial range of values can later be altered if analysis results indicate a wider range should be considered. The *noise parameter* controls the fraction of noise

Algorithm 1 Large Neighbourhood Search

Input: Problem instance j, Parameter setting θ

Output: Best found solution x^{best}

Initialization: initial solution x constructed by regret-2 heuristic

Stage 1: Vehicle Minimisation

repeat

  Remove one route from x

  Schedule removed requests into remaining routes (as in Stage 2)

until 20% of maximum run time met

Stage 2: Minimisation of total distance covered

repeat

  select destroy and repair methods $d \in \Omega^-$ and $r \in \Omega^+$ using probabilities ρ^- and ρ^+

  $x^{candidate} = r(d(x))$

  if $x^{candidate}$ is accepted **then**

    $x = x^{candidate}$

  end if

  if $c(x^{candidate}) < c(x^{best}))$ **then**

    $x^{best} = x^{candidate}$

  end if

until maximum run time met

that is used in the repair heuristics. The noise amount is calculated as the maximum distance between two nodes in a problem instance multiplied by the noise parameter. It brings randomness to the moves these repair heuristics make. The *cooling rate* and *start temperature control parameter* are part of the local search framework simulated annealing operating within the LNS algorithm. The *cooling rate* determines how quickly the algorithm progresses towards accepting solely solution improvements. A standard exponential cooling rate is considered. The start temperature is set such that a solution that is s% worse than the start solution still has an acceptance probability of 50%, with s being the *start temperature control parameter*. According to Aarts et al. (2005) the values for the *cooling rate* are typically between 0.80 and 0.99. Finally, it is determined which *destroy* and *repair operators* to include. Each parameter setting should use at least one repair and one destroy operator, otherwise the algorithm cannot function. There are three possible scenarios for the repair heuristics, each with an equal probability of occurence: either greedy insertion or regret-2 is used

Table 3.3: Algorithm Parameters and Components

Parameter	**Type**	**Value ranges**
Determinism Parameter	Integer	U [1, 30]
Noise Parameter	Discrete	U [0, 1]
Cooling Rate	Continuous	U [0.80,0.99]
Start Temperature Control Parameter	Discrete	U [0.01,0.10]
Destroy Operators		U [1,7]
1. Random Removal	Dummy	[0,1]
2. Worst Removal	Dummy	[0,1]
3. Related Removal	Dummy	[0,1]
4. Random and Worst Removal	Dummy	[0,1]
5. Random and Related Removal	Dummy	[0,1]
6. Worst and Related Removal	Dummy	[0,1]
7. Random, Worst and Related Removal	Dummy	[0,1]
Repair Operators		U [1,3]
1. Greedy	Dummy	[0,1]
2. Regret-2	Dummy	[0,1]
3. Greedy and Regret-2	Dummy	[0,1]

alone, or both operators are included. A similar logic is used to determine which out of seven possible destroy operator combinations to include. More information on all algorithm parameters of an (A)LNS can be found in Pisinger and Ropke (2007).

3.4.4 Data Set and Model Formulation

The data set serving as an input for the algorithm has 4000 scenarios[2], consisting of 200 randomly generated problem instances and 20 randomly created parameter settings per problem instance (Table 3.4)[3]. The performance measure recorded is the total distance travelled by the vehicles. The analysis performed investigates how the

[2]A random seed is generated for each scenario as an input for all random numbers generated during the run of the algorithm.

[3]Parameter settings are randomly sampled for each problem instance since continuous parameters are considered (e.g., noise parameter), resulting in a huge number of possible value combinations. In this case, it is important to sample broadly and not selectively. This provides a better coverage of the values within the numerical interval and enables more different combinations of values being considered in the analysis. By only sampling a limited set of values you implicitly assume that the values in-between do not matter.

Table 3.4: Multilevel Experimental Design Case Study

	Instance Characteristics			**Algorithm Parameters**			
Scenario	**Customers**	...	**Max Run Time**	**Determinism Parameter**	**Noise Parameter**	...	**Regret-2**
1	122	...	130.55	15	0.13	...	True
2	122	...	130.55	19	0.69	...	False
3	122	...	130.55	27	0.28	...	True
...	...	...	...	...	...	...	...
20	122	...	130.55	10	0.17	...	True
...	...	...	...	...	...	...	...
3981	269	...	749.45	6	0.07	...	False
3982	269	...	749.45	23	0.02	...	True
3983	269	...	749.45	17	0.34	...	True
...	...	...	...	...	...	...	...
4000	269	...	749.45	18	0.38	...	True

different algorithm parameters and problem instance characteristics influence this total distance measure. Since some of these problem instance characteristics (time window width, customer demand and service time) have different values for each customer, averages are taken over all customers in order to obtain a variable at the problem instance level. The geographical coordinates are excluded from the analysis.

The regression model in this section is fitted with varying (i.e., random) effects for all algorithm parameters and components, while interaction-coefficients are chosen to remain fixed and do not vary per problem instance. Hence, the coefficients of all individual algorithm element terms in the model have their own regression model at the problem instance level — as illustrated in equation (3.3) — and the estimate can thus vary for different values of the problem instance characteristics. The coefficients for the interaction terms do not have such a higher-level regression model and have the same impact on performance for all problem instances. The model is run using the *brms* package version 1.7.0 (Bürkner, 2017) in R version 3.2.5 (R Core Team, 2016). This package allows to fit a generalized (non-)linear mixed model, which incorporates both fixed-effects parameters and random (i.e., varying) effects in a (non-)linear predictor via full Bayesian inference using Stan, a probabilistic programming language for statistical inference written in C++.

A first linear[4] regression model showed not to satisfy all assumptions. The residuals (Figure 3.3) reveal the presence of heteroscedasticity[5]. While this issue is minor in most cases of moderate sample size (Gelman and Hill, 2006; Jacqmin-Gadda et al., 2007), a common approach is to apply a variance-stabilising transformation (Montgomery, 2012) which results in a non-linear model (equations (3.6) to (3.8)). It is empirically found that the reciprocal[6] transformation of the response varable together with the cube root of the problem instance characteristic *Customers* succeeds in resolving the heteroscedasticity (Figure 3.4). Further details on variable transformations can be found in, for example, Montgomery (2012).

$$\frac{1}{Y_i} = \alpha_{j[i]} + \beta_{1j[i]} Greedy_i + \beta_{2j[i]} Regret2_i + \cdots + \beta_{33j[i]} Noise_i + \epsilon_i \quad (3.6)$$

$$\alpha_j = \mu_0^{\alpha} + \mu_1^{\alpha} Customers_j^{\frac{1}{3}} + \cdots + \mu_5^{\alpha} Runtime_j + \eta_j^{\alpha} \quad (3.7)$$

$$\beta_{kj} = \mu_0^{\beta_k} + \mu_1^{\beta_k} Customers_j^{\frac{1}{3}} + \cdots + \mu_5^{\beta_k} Runtime_j + \eta_j^{\beta_k} \qquad \forall k \in K \quad (3.8)$$

In Section 3.3 it was mentioned that the assumption of normally distributed error terms is the least important assumption for estimating the regression line according to Gelman and Hill (2006). For the sake of completeness, this assumption is checked through a Q-Q plot showing the standardised residuals against their normal scores. If the residuals have a normal distribution, the plotted points should lie on a straight diagonal line. Figure 3.5 shows a Q-Q plot. The panel on the left side shows data generated from a normal distribution and serves as a reference plot. The panel on the right side shows the Q-Q plot for the residuals of the model formulated in equations (3.6) to (3.8). It shows the residuals to lie on the diagonal line in the middle, but curving off at the extremities, a phenomenon typically referred to as having "heavy tails". It means extreme observations are more extreme than is expected with a normal distribution (Montgomery and Runger, 2010).

[4]Linear is interpreted here as linear in the parameters and not as linear in the variables

[5]A key regression assumption is that the residuals are structureless, meaning unrelated to any other variable. The residuals in Figure 3.3 show an increasing error variance as the fitted value increases, indicating a violation of this assumption. This does not lead to biased effect estimates, but the standard errors can be biased. According to Gelman and Hill (2006), this issue is minor in most cases. It does not impair the inference on the fixed effects in case of moderate sample sizes, but variance parameter estimates and random effects may be biased when the covariance structure is misspecified (Jacqmin-Gadda et al., 2007; Montgomery, 2012).

[6]When the observations are all positive continuous values, the logarithmic transformation is typically applied (Gelman and Hill, 2006). However, the residual plot of the log-transformed values still shows increasing error variance, but not for the inverse values.

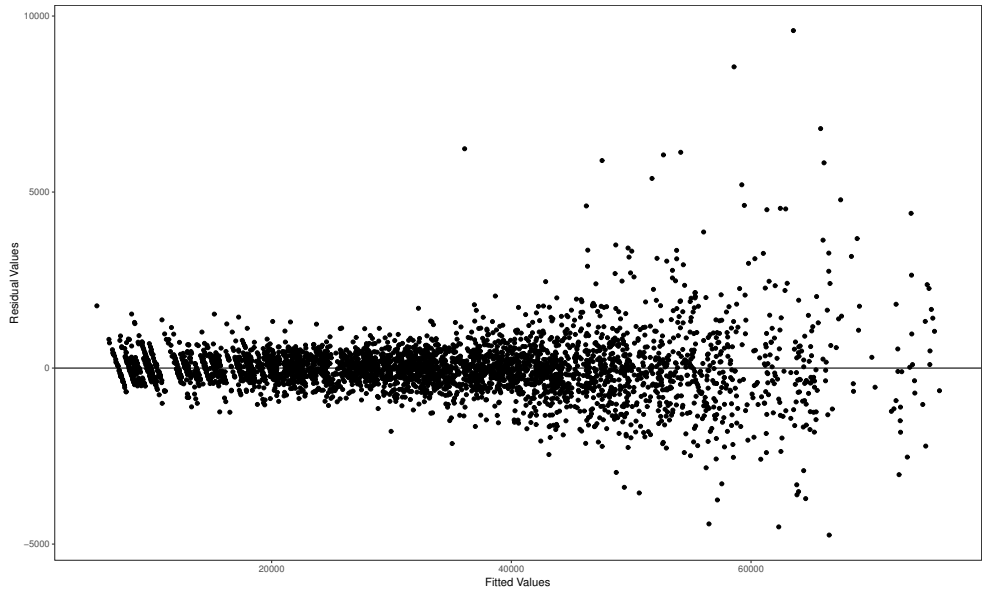


Figure 3.3: Fitted versus residual values for untransformed model

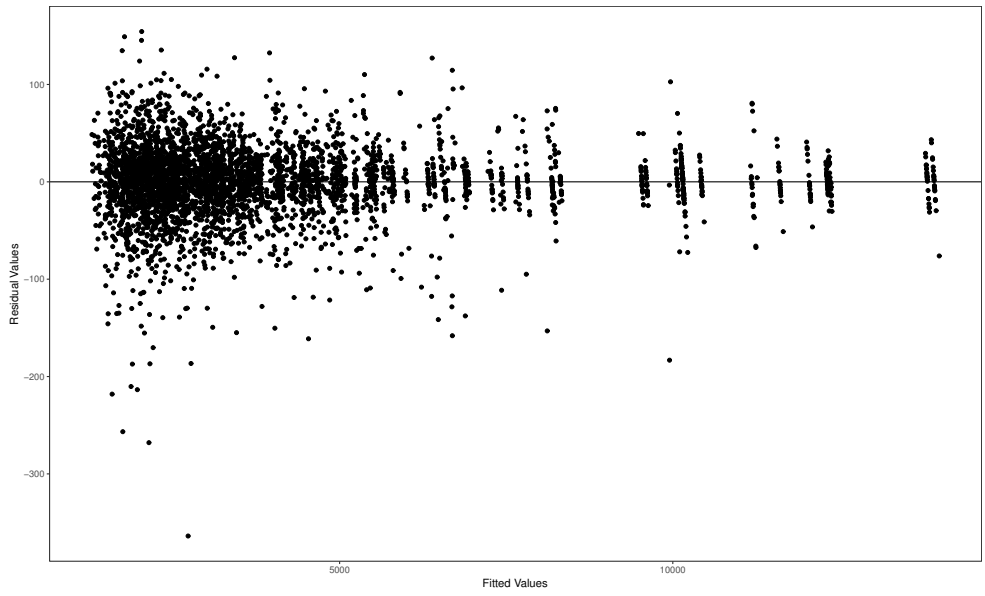


Figure 3.4: Fitted versus residual values for transformed model

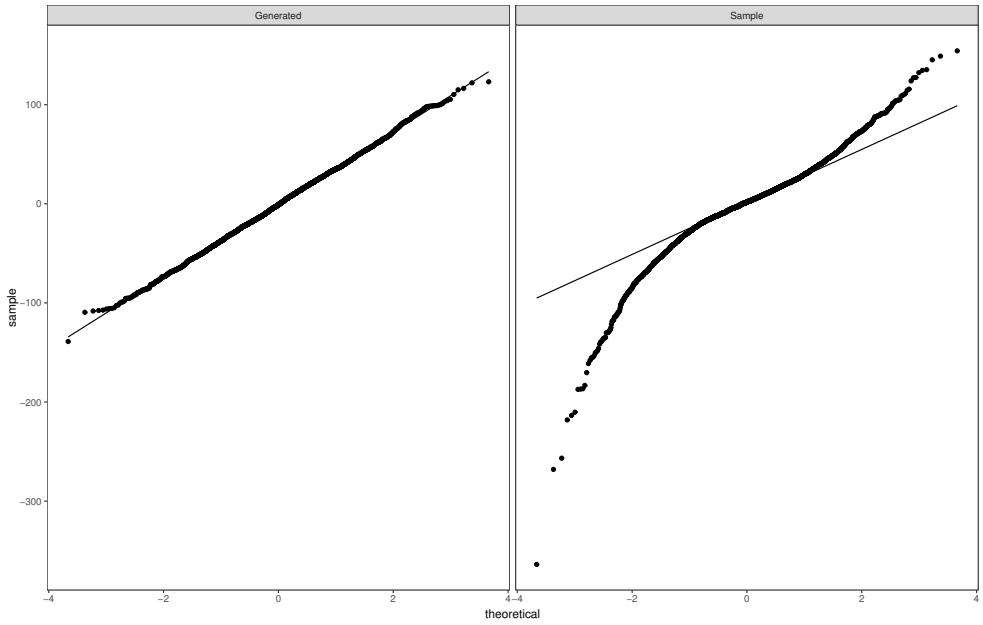


Figure 3.5: Q-Q plot

3.4.5 Analysis of Results

The hypotheses tested are whether the coefficients of the algorithm parameters, problem instance characteristics and their interaction are significantly different from zero. In other words, do they have a significant impact on the performance measure? A coefficient is considered to be significant if the 95% confidence interval (CI) for the coefficient estimate does not include zero. The output of the regression analysis indicates significant effects for all repair and most destroy operator combinations, the interaction of the determinism parameter with two individual destroy operators, the noise parameter, the interaction of the noise parameter with one individual repair operator and the start temperature control parameter. The operator effects and the effect of the noise parameter are also significantly moderated by certain problem instance characteristics. For all other algorithm parameters included in the model, no significant effects are found. Table 3.5 lists for all significant effects the estimated performance impact on $\frac{1}{Y}$, its standard error and 95% confidence interval. A complete summary table of the regression analysis is given in Table B.1 in Appendix B. Note that the effect estimates in this table cannot be interpreted as unconditional or average marginal effects like in a linear-additive model. Since the

model in equations (3.6) to (3.8) includes interaction terms, a specific effect estimate is conditional on the interacting variables (Brambor et al., 2006). Therefore, this analysis studies the marginal effect of parameters, which is calculated by taking the first derivative of the regression equation that includes all conditioning variables.

Before fitting the regression model, all problem variables are centred around their mean value such that the intercept estimate in Table 3.5 can be interpreted as the performance value obtained for an average problem instance rather than for a meaningless problem instance with zero customers or zero demand. The intercept estimate also accounts for the parameter setting that allows all repair and destroy operators to be used (i.e., *GreedyRegret2* and *RandomWorstRelated*, the reference levels for the operators)[7]. In this case, the expected cost is predicted to be 25 382.19[8] and will increase for larger problem sizes and narrower average time windows as indicated by the significant estimates for *Customers* and *Avg time window width* in Table 3.5. The principal interest of this research lies in investigating how this measure is further impacted by the different algorithm parameters and how the problem instance characteristics interact with these parameters. These results are discussed next.

3.4.5.1 Effect Repair and Destroy Operators

When considering an average problem instance and with all other numerical variables (e.g., *cooling rate*) at their mean level, the results in Table 3.5 suggest to use random removal as sole destroy operator, since it has the largest positive performance impact (18.02) over the configuration with all destroy operators. Likewise, using regret-2 as sole repair operator is indicated as the best option since it significantly improves upon the performance of a configuration with both repair operators (16.64) while using greedy repair alone would lead to a deterioration of performance (-165.77). The significant interaction terms between *Greedy* and the different destroy operator combinations do not alter this conclusion. Furthermore, the results indicate that the configuration with either regret-2 as the sole repair operator or both repair operators is better able at repairing a solution that is destroyed by the random removal operator

[7]Multicollinearity may lead to inflated variance estimates and a high sensitivity of the coefficient estimates for changes in the model. This makes it difficult to interpret results as the estimates are unstable. In this case, including all destroy variables or all repair variables would lead to perfect multicollinearity. One variable of each needs to be left out and serve as a reference value which is represented in the regression intercept.

[8]The Intercept value in Table 3.5 is backtransformed to the original scale through division by 100 000 000 and taking the inverse of the resulting value.

Table 3.5: Significant Effects

Variable	Estimate	Est.Error	l-95% CI	u-95% CI
Intercept	$3,939.77^{**}$	121.21	3,706.59	4,180.24
Customers$^{\frac{1}{3}}$	-411.93^{**}	28.11	−466.81	−357.71
Avg time window width	42.24^{*}	18.14	7.02	77.87
Greedy	-165.77^{**}	6.29	−178.13	−153.45
Customers$^{\frac{1}{3}}$	-20.00^{**}	1.06	−22.11	−17.93
Avg service time	4.48^{**}	1.05	2.43	6.56
Avg time window width	-4.18^{**}	0.69	−5.52	−2.84
Runtime	3.04**	0.74	1.61	4.49
Regret2	16.64^{**}	4.45	7.89	25.45
Customers$^{\frac{1}{3}}$	2.10^{**}	0.40	1.30	2.88
Random	18.02^{**}	4.34	9.46	26.54
Worst	-15.23^{**}	5.09	−25.13	−5.19
Related	-39.05^{**}	4.76	−48.32	−29.72
Customers$^{\frac{1}{3}}$	-4.84^{**}	0.70	−6.21	−3.46
RandomWorst	4.41	4.66	−4.66	13.57
Avg service time	1.34^{*}	0.65	0.06	2.62
Runtime	-1.08^{*}	0.45	−1.96	−0.19
WorstRelated	-13.97^{**}	4.43	−22.74	−5.30
Start temperature control parameter	-66.34^{*}	28.23	−121.81	−10.49
Noise parameter	-9.37^{*}	4.25	−17.66	−0.94
Customers$^{\frac{1}{3}}$	-1.58^{*}	0.63	−2.82	−0.36
Avg time window width	-0.85^{*}	0.41	−1.64	−0.06
Worst×Determinism parameter	-0.97^{**}	0.32	−1.60	−0.35
Related×Determinism parameter	-1.88^{**}	0.32	−2.50	−1.25
Greedy×Noise parameter	-37.96^{**}	6.07	−49.90	−26.11
Greedy×Random	-70.62^{**}	6.40	−83.22	−58.10
Greedy×Worst	-85.48^{**}	6.78	−98.70	−72.17
Greedy×Related	59.20^{**}	6.80	45.87	72.44
Greedy×RandomWorst	-82.93^{**}	6.75	−96.14	−69.56
Greedy×RandomRelated	13.85^{*}	6.62	0.90	26.77

Note 1 : ** denotes significance at 1%, * denotes significance at 5%

Note 2 : Due to the reciprocal transformation, negative effect estimates imply an increase in total cost, while positive values indicate a decrease in total cost.

Note 3 : Since the reciprocal transformation of the response variable returned very small values causing difficulties in the sampling procedure of the *brms* package, all transformed (response) values were multiplied by a constant 100 000 000.

Note 4: The effects of Greedy & Regret-2 and Random, Worst & Related, the reference levels for the repair and destroy operator dummies, are accounted for in the Intercept.

(18.02) compared to a solution that is destroyed by the related removal operator (-39.05). The configuration with greedy as the sole repair operator, on the other hand, shows the opposite result. The latter can be derived from the estimates in Table 3.5: the performance impact of switching to random removal becomes negative when accounting for the interaction with greedy repair ($18.02 - 70.62 = -52.6$), while the impact of related removal turns positive ($-39.05 + 59.20 = 20.15$).

Figure 3.6 plots the expected total cost values for *GreedyRegret2* and *Greedy* with all destroy operator configurations. For example, the combination *Greedy* and *Random* has a predicted total cost of 26 871.61. The plot also shows the effect of switching from using both greedy and regret-2 to using only greedy as repair operator. The switch to greedy is expected to deteriorate the solution quality with all possible combinations of destroy operators ($-165.77+$ significant interaction term). The configuration with both repair operators expects its best performance when combined with random removal, while the highest average cost measure is predicted with related removal. The configuration with only greedy repair performs best with related removal or the combination of random and related removal, and obtains the highest total cost value with worst removal. A shift in the "ranking" of the destroy operator combinations can thus be observed going from all repair operators to greedy alone. When investigating the switch from both repair operators to using only regret-2 (Figure 3.7), an improvement in the performance measure is observed for all destroy operator combinations (16.64). Unlike the switch to greedy, there is no shift in the ranking of the destroy operators going from regret-2 and greedy to regret-2 alone. In both scenarios the lowest total cost value is expected when combined with random removal and the highest value with related removal.

These findings relate to an average problem instance, i.e., an instance with 216 customers, 29 minutes of average service time, an average customer demand of 38.5 units, an average time window width of 57 minutes, and on which the algorithm can run maximum 15 minutes – due to centring of these variables. Next, the influence these problem instance characteristics have on the performance impact of the repair operators is analysed. It is observed that it becomes more beneficial to use only regret-2 as repair operator and more detrimental to use only greedy as repair operator as the problem size increases. Furthermore, on the problem instances with up to 170 customers the performance difference between using regret-2 alone or together with greedy cannot be distinguished. Therefore, the results suggest that in order for the regret-2 operator to be effective (i.e., showing a positive effect on the performance

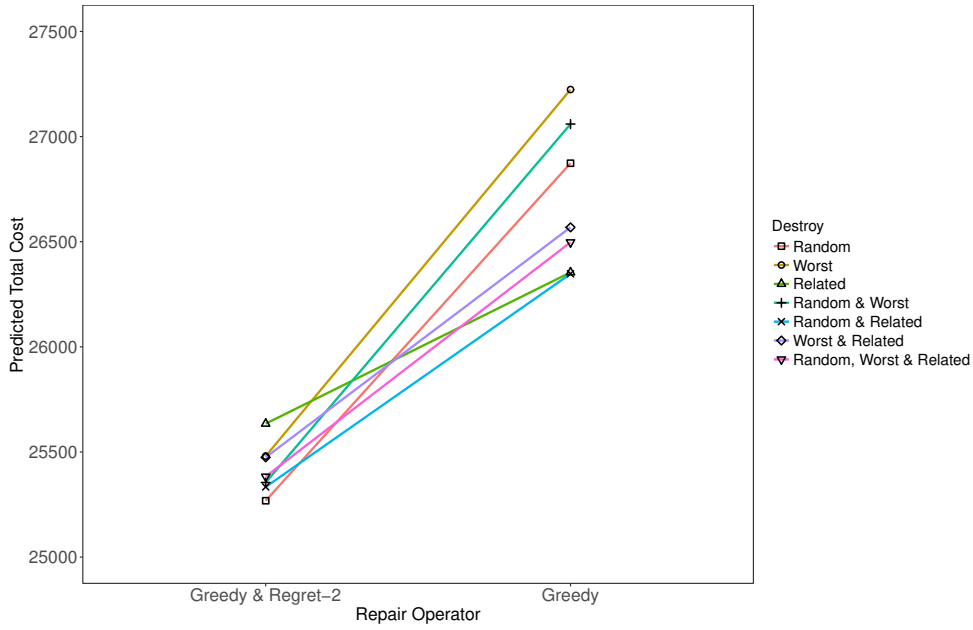


Figure 3.6: Total cost plot switching from Greedy & Regret-2 to Greedy.

measure), the number of customers in a problem instance matters. These conclusions are derived from studying the marginal effect using the estimates of Table 3.5. The marginal effect of *Greedy*, for example, given varying problem sizes is calculated as follows (with all other variables equal to zero).

$$Y_i = (\alpha_{j[i]} + \beta_{1j[i]} Greedy_i + \beta_{2j[i]} Regret2_i + \cdots + \beta_{33j[i]} Noise_i)^{-1} \quad (3.9)$$

$$\frac{\partial Y}{\partial Greedy} = -\frac{\beta_{1j[i]}}{(\alpha_{j[i]} + \beta_{1j[i]} Greedy_i)^2} \quad (3.10)$$

$$\frac{\partial Y}{\partial Greedy} = -\frac{\mu_0^{\beta_1} + \mu_1^{\beta_1} Customers_j^{\frac{1}{3}}}{(\mu_0^{\alpha} + \mu_1^{\alpha} Customers_j^{\frac{1}{3}} + (\mu_0^{\beta_1} + \mu_1^{\beta_1} Customers_j^{\frac{1}{3}}) Greedy_i)^2} \quad (3.11)$$

The impact of switching to greedy repair alone (when combined with random, worst and related removal) is assessed by entering the estimates for *Greedy* (-165.77) and the interaction term $Greedy \times Customers^{\frac{1}{3}}$ (-20.00) in equation (3.11).

$$\frac{\partial Y}{\partial Greedy} = -\frac{-165.77 - 20 * Customers_j^{\frac{1}{3}}}{(3939.77 - 411.93 * Customers_j^{\frac{1}{3}} - 165.77 - 20 * Customers_j^{\frac{1}{3}})^2} \quad (3.12)$$

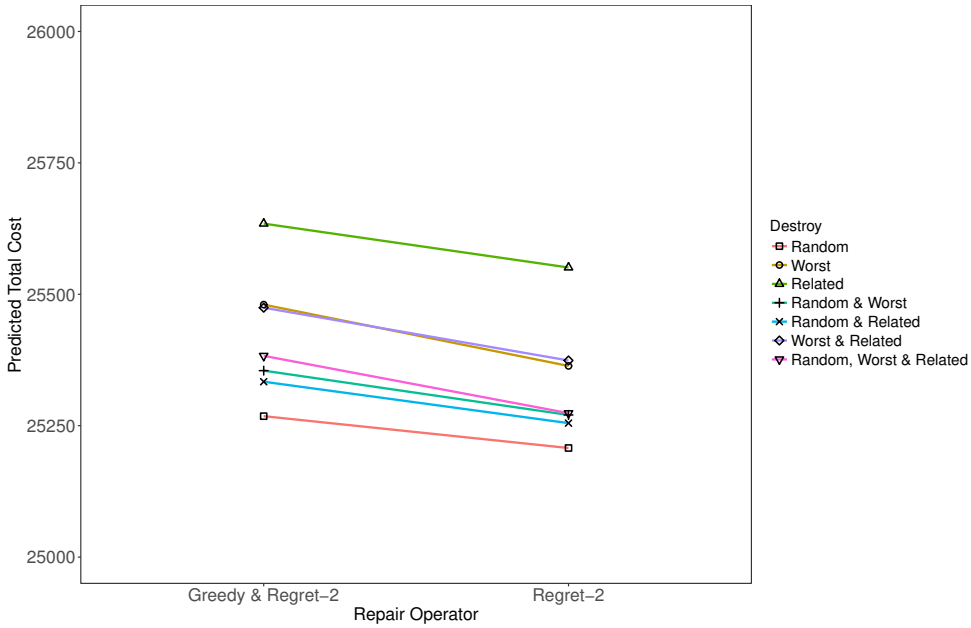


Figure 3.7: Total cost plot switching from Greedy & Regret-2 to Regret-2.

The impact estimate increases as more and more customers have to be served. For the combination with any other (set of) destroy operators, the estimate of the interaction term is also added, e.g., -70.62 is added for the combination with random removal.

$$\frac{\partial Y}{\partial Greedy} = -\frac{\mu_0^{\beta_1} + \mu_1^{\beta_1} Customers_j^{\frac{1}{3}} + \beta_2 Greedy_i \times Random_i}{(\mu_0^{\alpha} + \mu_1^{\alpha} Customers_j^{\frac{1}{3}} + (\mu_0^{\beta_1} + \mu_1^{\beta_1} Customers_j^{\frac{1}{3}}) Greedy_i + \beta_2 Greedy_i \times Random_i)^2} \tag{3.13}$$

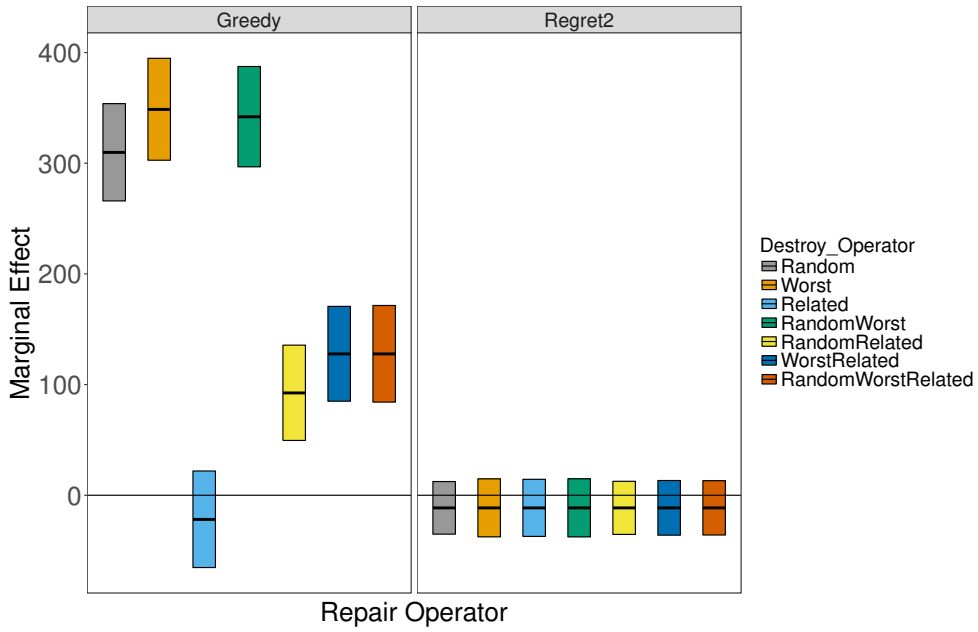
$$\frac{\partial Y}{\partial Greedy} = -\frac{-165.77 - 20 * Customers_j^{\frac{1}{3}} - 70.62}{(3939.77 - 411.93 * Customers_j^{\frac{1}{3}} - 165.77 - 20 * Customers_j^{\frac{1}{3}} - 70.62)^2} \tag{3.14}$$

For the impact of regret-2, calculations are similar and show estimates progressing towards zero as less and less customers have to be served. A threshold problem size — considering a 95% confidence interval — at which the impact becomes significantly positive is identified at 171 customers. This threshold value applies for all destroy operator combinations since there are no significant interactions between regret-2 and the destroy operators, making the marginal impact of regret-2 independent of the (set of) destroy operator(s) used. The marginal effects of *Greedy* and *Regret2*

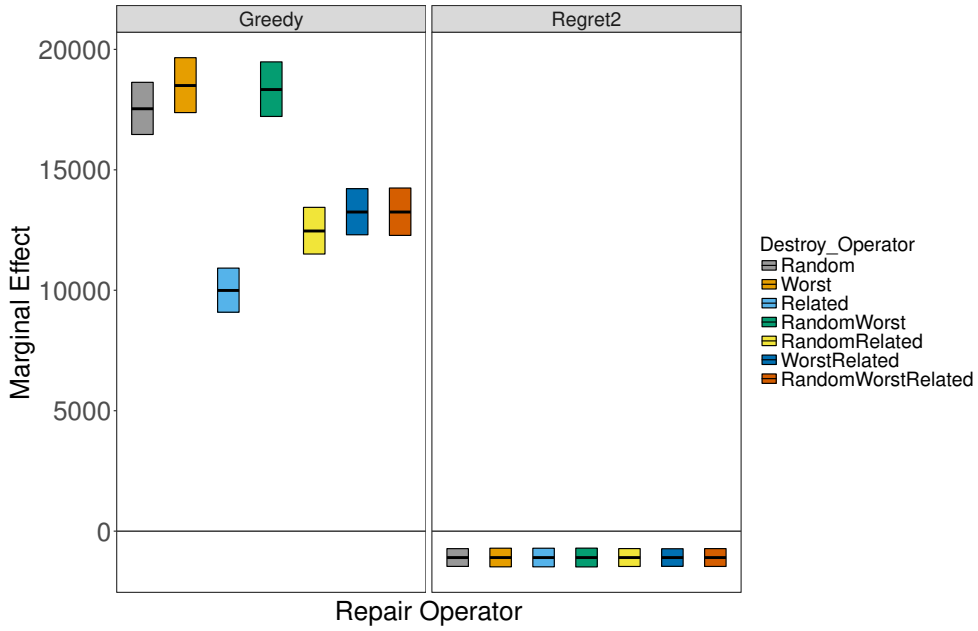
for the smallest and largest problem instance are plotted in Figure 3.8 in which the horizontal zero line represents the scenario with all repair operators enabled. Note the difference in scale on the vertical axis between panels (a) and (b).

Further, the effect of *Greedy* is also influenced by the average service time per customer (4.48), and the average time window width (−4.18). The more constraining these problem instance characteristics become, the smaller the performance differences between using greedy alone and using both repair operators together (Figures 3.9 and 3.10). However, the marginal effect shows the influence of the average time window width to be positive for the interaction of *Greedy* with *Random*, *Worst* and *RandomWorst* implying the marginal effect of *Greedy* benefits if this problem instance characteristic becomes less constraining. Finally, the effect of *Greedy* is positively influenced by the maximum run time given (3.04), meaning the longer the algorithm is allowed to search for better solutions, the smaller the differences between the use of greedy alone and the other repair operator configurations become (Figure 3.11). This is a logical deduction as you would expect performance to converge as run time increases.

In a similar way, it is investigated whether it is worth including all three destroy operators or whether there are conditions when a configuration with less destroy operators will give better results. The influence of the problem instance on these effects is limited. The analysis suggests that related removal performs relatively better for problem instances serving less than the average number of customers (i.e., 216), with the best performance expected for instances with 25 customers. This follows from the negative influence (−4.84) of problem size on the effect of *Related*. Recall that the effect estimates in Table 3.5 represent the switch from a configuration with random, worst and related removal to some other (set of) destroy operator(s) (given an average problem instance). The impact of switching to related removal is the same with both repair operators and with regret-2 alone (i.e., −39.05), meaning additional customers will further strengthen the negative impact. Due to the significant interaction with *Greedy*, the impact of switching to related removal for an average instance is positive (i.e., $-39.05 + 59.20 = 20.15$), meaning additional customers will diminish this positive impact and at about 224 customers the estimated impact can no longer be distinguished from the configuration with all destroy operators. The marginal effect of related removal for the smallest and largest problem size is plotted in Figure 3.12. On the other destroy operator combinations, the problem size has no real influence.



(a) 25 Customers



(b) 400 Customers

Figure 3.8: Marginal effects of Greedy and Regret-2 for the smallest and largest problem size.

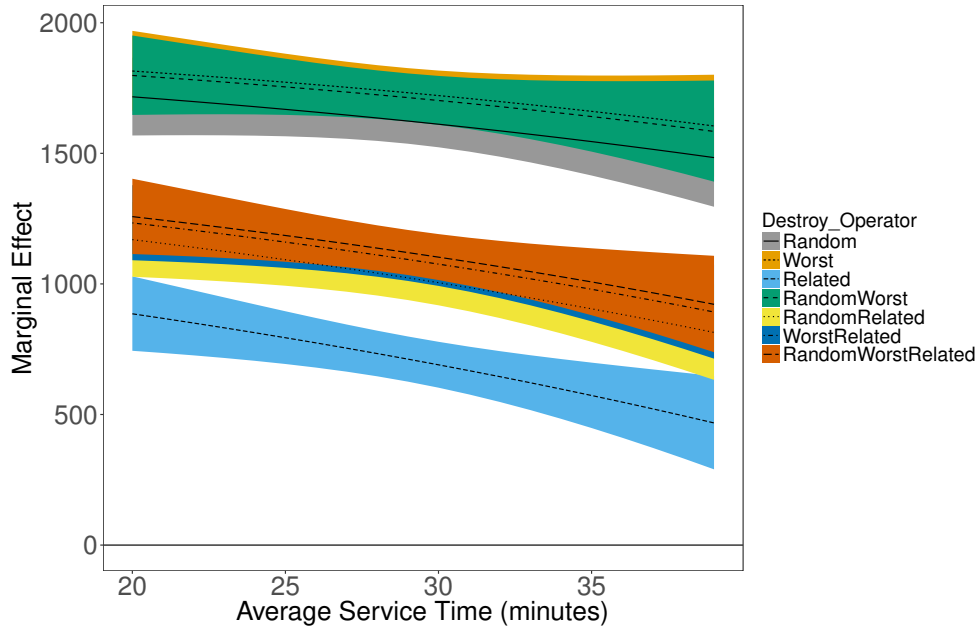


Figure 3.9: Marginal effect of Greedy for varying average service time values.

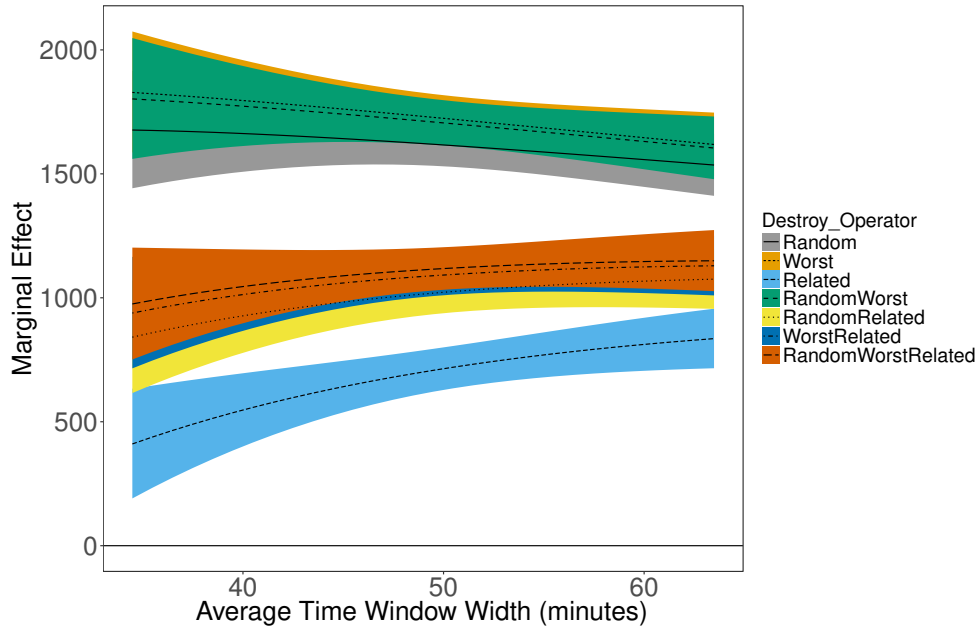


Figure 3.10: Marginal effect of Greedy for varying average time window width values.

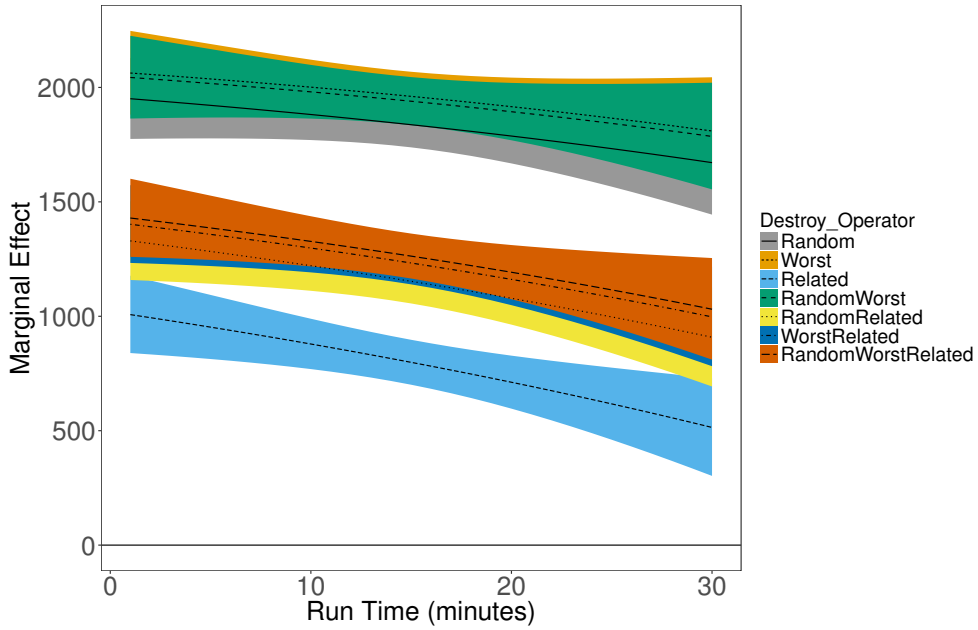


Figure 3.11: Marginal effect of Greedy for varying run time values.

The only other significant problem influence observed is on the impact of *RandomWorst*, which is positively moderated by the average service time (1.34) and negatively by the average run time (−1.08)(Figures 3.13 and 3.14). The impact for the combination with *Greedy* becomes less negative for additional service time and more negative for additional run time. For the combination with either *Regret2* or *GreedyRegret2* (overlapping in Figure 3.14 as there is no significant interaction between *RandomWorst* and *Regret2*), the impact remains mostly indistinguishable from the configuration with all destroy operators, but it is observed that *RandomWorst* performs significantly better for run times up to nine minutes. The latter can be seen in Figure 3.14, where the marginal effect line with 95% confidence interval does not include zero for run times between 1 and 9 minutes.

A final significant influence on the operator effects is the randomness element employed within the operators. The analysis results show that if all destroy operators are used in a single configuration, then complete randomisation in the selection of customers to remove should be left to the random removal operator, while the other operators should strictly focus on removing customers with a high cost (using

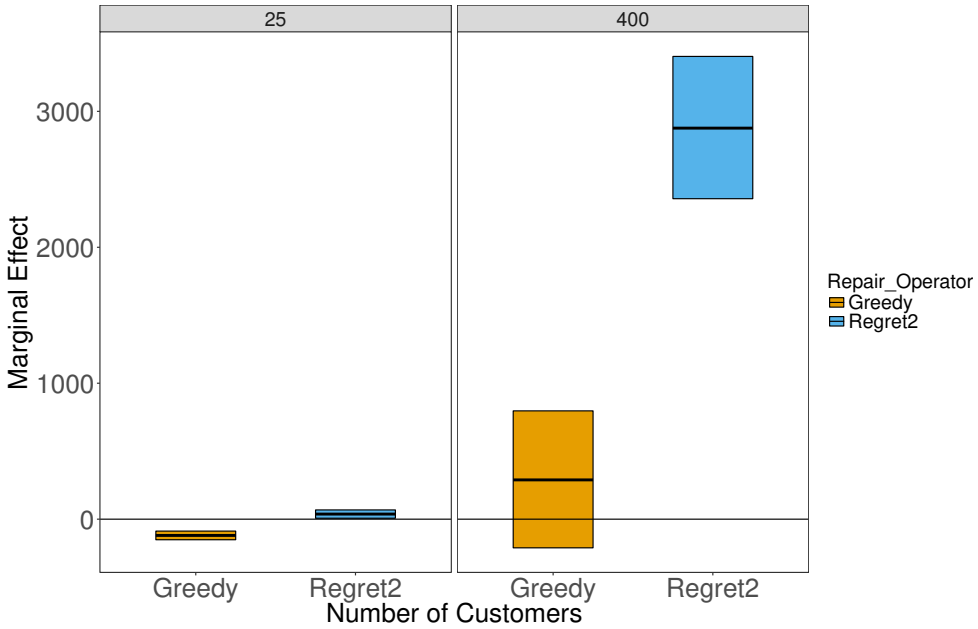


Figure 3.12: Marginal effect of Related for the smallest and largest problem size.

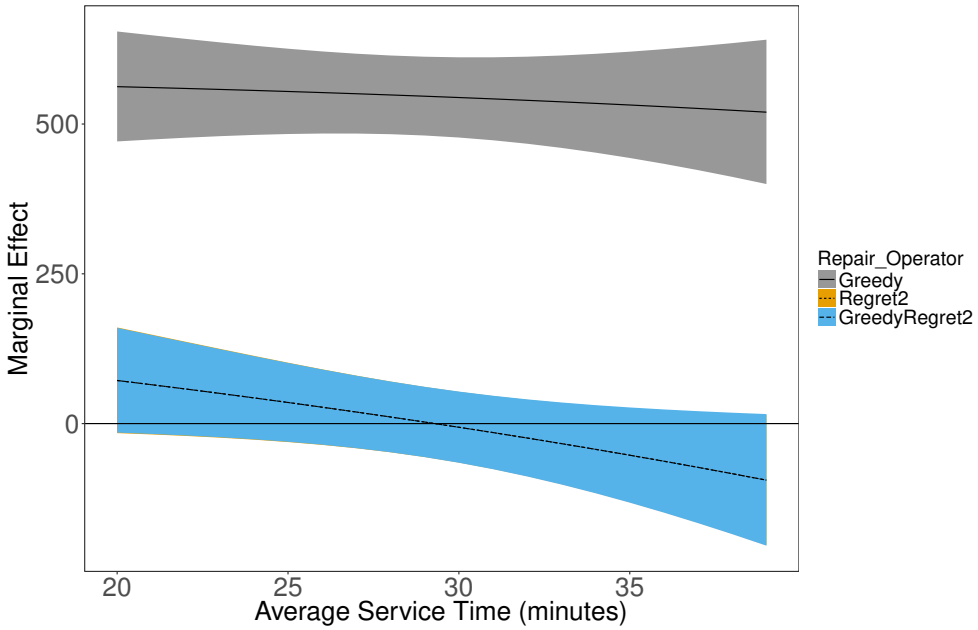


Figure 3.13: Marginal effect of RandomWorst for varying average service time values.

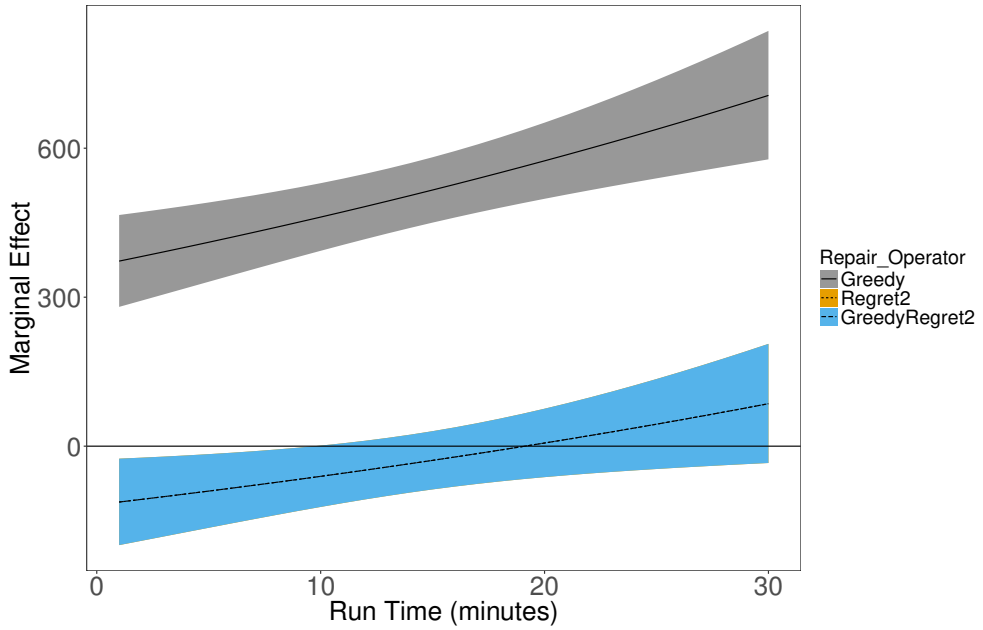


Figure 3.14: Marginal effect of RandomWorst for varying run time values.

worst removal) or customers that are easy to interchange (using related removal). On the other hand, if worst or related removal are used alone, it is preferable to add some randomisation in the selection of customers to remove. For the repair operators, the analysis results suggest it is never worthwhile to apply randomisation when reconstructing solutions. These conclusions are derived from the marginal effects for the determinism and noise parameter using the relevant effect esimtates in Table 3.5. The determinism parameter has a negative effect on the performance impact of *Worst* (−0.97) and *Related* (−1.88) (Figures 3.15 and 3.16), while the noise parameter negatively influences the impact of *Greedy* (−37.96) (Figure 3.17). The *noise parameter* has no significant influence on the performance impact of *Regret2* and, therefore, the parameter's effect estimate (−9.37) for *GreedyRegret2* is studied to conclude that adding noise in the repair phase is detrimental in all cases. The significant influence of the problem size (−1.58) and the average time window width (−0.85) does not alter this conclusion. The negative impact becomes larger for increasing problem sizes, while it becomes insignificant for smaller instances. The negative impact is also larger for wider average time windows, while for tight windows the impact becomes insignificant. The importance of randomisation to ALNS performance is further investigated by Hemmati and Hvattum (2017) who

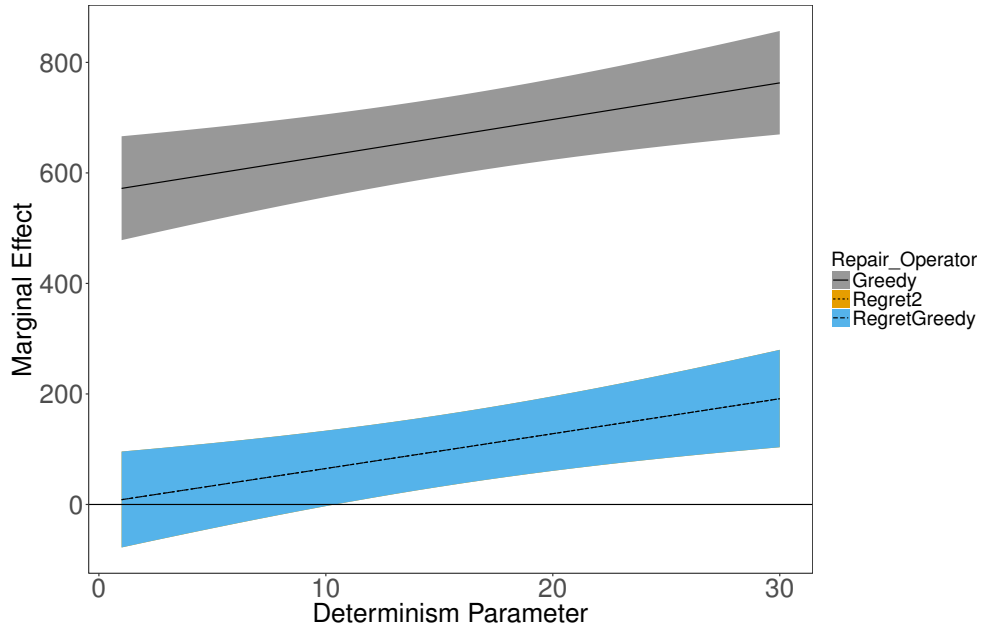


Figure 3.15: Marginal effect of worst removal for varying levels of randomness in the selection of customers to remove.

propose deterministic alternatives and found they perform mostly similar to the randomised variants.

Summarising the discussion on the effect of the operators, it can concluded that including all repair and destroy operators in a parameter setting does not necessarily lead to the best results. The analysis identified using regret-2 as the sole repair to be the best choice on average as it is expected to perform better than the other two repair operator configurations for larger problem sizes ($>$ 170 customers). The destroy operator combination that will obtain the best results with this repair operator is random removal. The results also showed that randomisation in the search for solutions works during the destroy process, but should be avoided when repairing solutions. The observation that regret-2 is the more effective repair operator (on larger instances) is not surprising as it is the more 'intelligent' one of the two, but it is a valuable insight to know this is not necessarily the case for the destroy operators where it is shown that a simple operator as random removal can outperform other, more 'intelligent' destroy operators.

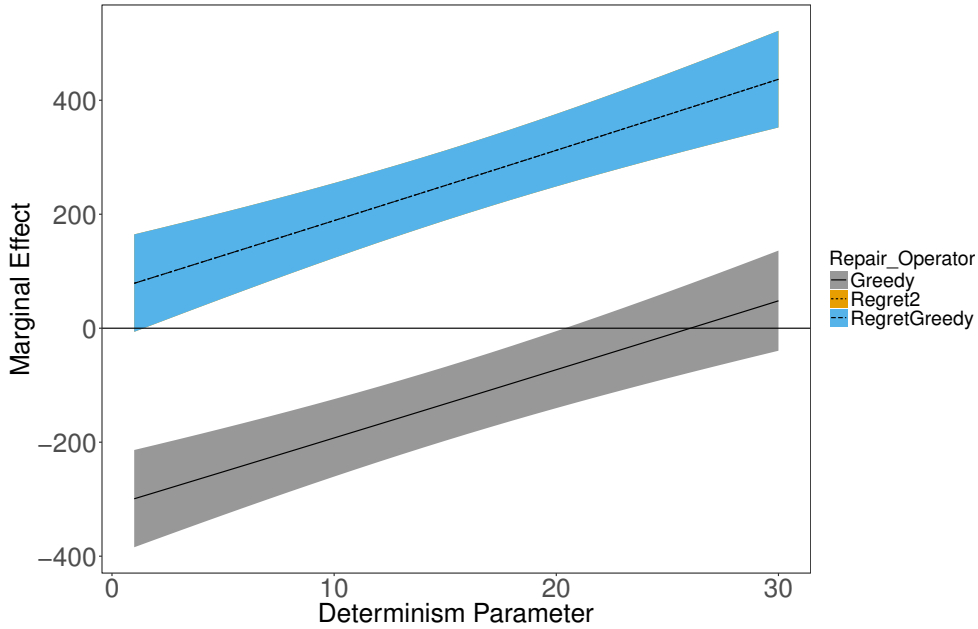


Figure 3.16: Marginal effect of related removal for varying levels of randomness in the selection of customers to remove.

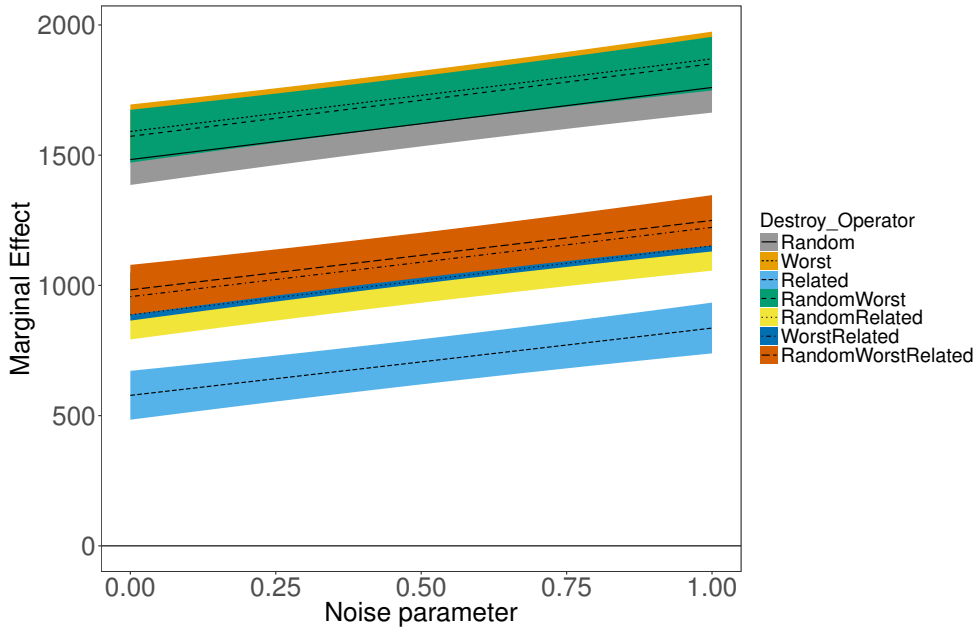


Figure 3.17: Marginal effect of greedy repair for varying levels of randomness in the selection of customers to insert.

3.4.5.2 Start Temperature Control Parameter

The start temperature control parameter, which operates within the simulated annealing framework that is part of the LNS, is the only other algorithm parameter included in the analysis with a significant impact on the performance measure. This parameter is used to set the start temperature at the beginning of both the vehicle and distance minimisation phase. The start temperature is set at a value such that a solution that is $s\%$ worse than the start solution [9] still has a 50 percent probability of being accepted, with s being the start temperature control parameter (Pisinger and Ropke, 2007). The percentage parameter is part of the analysis and has a significant effect (-66.34) on the objective function value. The higher the percentage the worse the performance measure becomes - for an average problem. A higher percentage means allowing more solutions that are worse than the start solution to have an acceptance probability of 50 percent. The parameter is not significantly influenced by any of the problem instance characteristics. This parameter should therefore be set to its lowest value regardless of the problem instance to be solved.

3.4.5.3 New Questions

The analysis exposed which combinations of operators work well for what parts of the problem space. This spurs new research questions. For example, what is so unique about the way related removal destroys a solution that makes it most difficult for regret-2 to repair it? The analysis results have led to several similar new questions. A logical next step would be to further investigate these observations by formulating new hypotheses and conducting further experiments. A single experiment will often not answer all questions posed and may raise new questions - as is the case in this experiment. It shows that experimenting is an iterative learning process: observing what works well in a first experiment, then finding out why it works well in consecutive experiments. Box et al. (2005) describe it as the iterative inductive-deductive process.

A first brainstorm on random and related removal resulted in the reasoning that removing a random selection of customers from a solution results in more "interesting" alternatives for the removed customers, meaning that the difference between their cheapest and second cheapest route (i.e., the regret value) is on average smaller compared to removing a group of geographically clustered customers. For

[9] For the vehicle minimisation phase, the start solution is the initial solution provided by regret-2. The last feasible solution of this phase is used as start solution for the phase minimising the total distance travelled.

the randomly removed customers, there might still be many routes nearby, while the removal of a cluster of customers might remove all nearby routes. So overall, a solution destroyed with the random removal operator has better alternatives for the cheapest route compared to a solution destroyed by the related removal operator. This gives algorithm configurations using random removal more flexibility in repairing a solution. From the previous, two hypotheses can be formulated to validate.

Hypothesis 1 (H1): When a cluster of geographically nearby customers is removed, each removed customer has on average less feasible routes to be inserted in compared to a customer that was removed at random.

Hypothesis 2 (H2): The average (maximum) regret value of the selected customer for insertion per iteration is lower when customers are removed at random compared to when a cluster of geographically nearby customers is removed.

Similarly, reasons why related removal has more trouble with larger instances than smaller instances can be investigated. Further, it is also observed that there is no significant performance difference between using regret-2 alone or together with greedy repair on the smaller instances. Using both repair operators implies regret-2 is used in half the iterations performed, while greedy is used in the other half. So even though no performance difference is observed, the configuration with only regret-2 will probably reach the best solution in fewer iterations and time than the configuration with both repair operators. It would be interesting to look into the gain achieved. Does it require half the time or even less?

3.5 Conclusion

In this chapter a statistical methodology is proposed for understanding heuristic algorithm performance. It enables investigation of correlations between algorithm performance and algorithm parameters and correlations between the latter and problem instance characteristics. This doctoral thesis considers it a next step in the experimental research on combinatorial optimisation problems to obtain a deeper understanding and insight in the effects of parameters and heuristic components on algorithm performance. The methodology is able to identify which algorithm parameters significantly impact the solution quality of a heuristic method and how the problem instance characteristics influence these effects. It enables researchers to

make statements about an entire population of problem instances, not just a small set of benchmark instances. Different recommendations for different parts of the problem space can be obtained.

In an analysis of a large neighbourhood search algorithm on instances of the vehicle routing problem with time windows it is observed that including all repair and destroy operators in a parameter setting does not necessarily lead to the best results. The analysis identified using regret-2 as the sole repair to be the best choice on average as it is expected to perform better than the other two repair operator configurations for larger problem sizes. The destroy operator combination that will obtain the best results with this repair operator is random removal. This analysis of the performance impact of the operators considered the moderating effect of each significant problem instance characteristic ceteris paribus, but these parameters can off course divert simultaneously from their average level. Which operator combinations work well and which do not depends on the unique combination of characteristics that constitute a problem instance. The multilevel methodology offers guidance and insights for both an 'average' problem instance as for a specific problem instance with certain characteristics (cf. Chapter 6).

In the next chapters the proposed methodology is applied in three different contexts. First, the analysis results have led to new questions that are to be answered in future research by formulating new hypotheses and setting up new controlled experiments. A single experiment will often not answer all questions posed and may raise new questions. It is a good illustration of the principle that learning is advanced by iteration. Chapter 4 performs a next iteration in the experimental analysis and focuses on answering one of those new questions. Secondly, regression model complexity grows with the number of variables added. Chapter 5 looks into possibilities of limiting the set of included variables through some kind of preliminary importance analysis. Finally, in Chapter 6 the multilevel methodology is applied to obtain a set of parameter values and component choices that optimise metaheuristic performance depending on the specific problem instance to be solved.

Chapter 4

Explaining Heuristic Performance Differences

4.1 Introduction

Chapter 3 proposes a methodological framework for evaluating (meta)heuristic algorithms. The framework is applied in a first experimental study to expose which and how algorithm parameters are correlated to performance. The objective was to discover patterns in the performance data, to see which (combinations of) parameters and components have a beneficial or detrimental effect on the objective function value. This chapter builds on these findings by going a step further than the exploratory analysis (Figure 4.1). Explanations are sought for patterns that are observed so that we can understand why there is a difference in performance. What can be learned about the differences in components, strategies or implementations that resulted in the observed pattern. This chapter aims to address this question. Such a detailed study of how metaheuristic performance is established is rarely the focus of a research paper in operations research literature.

This chapter[1] fulfils the hypothetico-deductive process of which the first steps are performed in Chapter 3. The study proceeds by formulating possible explanations

[1]This chapter is based on the proceedings paper: Corstjens, J., Caris, A., Depaire, B., in press. Explaining heuristic performance differences for vehicle routing problems with time windows. In: Kotsireas, I., Pardalos, P. (Eds.) Learning and Intelligent Optimization, Lecture Notes in Computer Science. Springer Berlin Heidelberg.

and translating them in hypotheses, which are validated in experimental studies. The results will indicate whether suggested explanations are correct, need alteration or are to be abandoned completely and replaced by other ones.

The structure of this chapter is as follows: Section 4.2 details the pattern observed in Chapter 3 to be explained and understood. Experimental data is generated (Section 4.3) and used to validate hypotheses in Section 4.4. A summary of what is learned is given in Section 4.5.

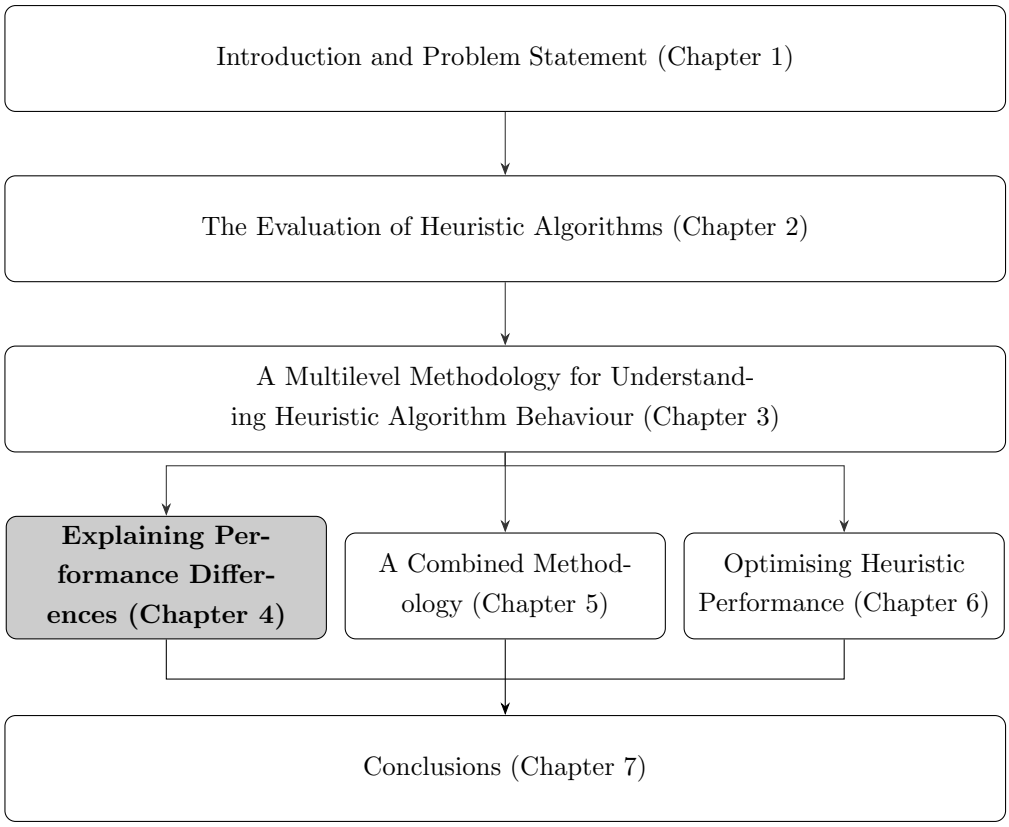


Figure 4.1: Outline of Thesis — Chapter 4.

4.2 Ask a Question

The experimental study in Chapter 3 on a large neighbourhood search algorithm applied on instances of the vehicle routing problem with time windows exposed how

parameter settings are correlated with performance. Further explanation about why a certain parameter setting is correlated in one direction or the other is not given. Nonetheless, such knowledge can be useful, especially when the impact (combinations of) parameters have on performance is different from what is expected. In such a case the question 'why?' is raised, as done in this chapter. The study presented here seeks to explain the performance difference observed in Chapter 3 between two LNS configurations applied on instances of the vehicle routing problem with time windows. It concerns the relative performance of some combinations of operators employed within the heuristic algorithm. The results predict certain combinations of these operators to perform better than others. However, these results did not provide any further indications as to why it is that these combinations perform differently. More specifically, it is observed that iteratively removing customers at random from a solution (i.e., random removal) and reinserting them using a difficulty measure (i.e., regret-2) results in a better predicted performance than when iteratively removing geographical clusters of customers (i.e., related removal). This is an unexpected observation and seems counterintuitive since removing clusters of customers involves applying a logic for selecting these customers that is aimed to work better than pure random selection. Yet, the contrary is observed. Hence, the experiment spurred new research questions that are to be answered in consecutive experiments. Recall from Section 3.4.5.3 that this iterative process is described by Box et al. (2005) as the iterative inductive-deductive process: experimenting is an iterative learning process, each time gaining knowledge and at the same time raising new questions. In this chapter the focus is on answering one of those new questions raised and investigate the correlations between certain operators. In particular, the aim is to explain why removing customers at random works better than removing clusters of customers given the use of a certain repair logic.

Therefore, the destroy and repair operators are of interest in this follow-up research. Two destroy operators — random and related removal — are considered and three repair operators — regret-2, regret-3 and regret-4. The repair operators are three variations on the same repair logic that is more generally formulated as the regret-k operator. Definitions of the relevant destroy and repair operators are discussed in the following sections.

4.2.1 Random Removal

The random removal destroy operator randomly selects q customers to remove. The idea is to diversify the search towards a better solution.

4.2.2 Related Removal

The related removal destroy operator removes q customers that are 'related'. How the relatedness between customers is defined, is determined by the experimenter. Shaw (1998) introduced the strategy of choosing related customers and noted 'related' should be suitably defined. The aim is to obtain solution improvements and the relatedness measure should therefore provide good insert opportunities to reach this objective. One definition of the relatedness measure is to base it on distance, looking at the geographical closeness of two customers, as used by Pisinger and Ropke (2007) in their experiments on the Rich Pick-up and Delivery Problem with Time Windows. The idea is that customers close to each other can more easily switch routes and/or positions, while more distant customers have a higher chance of being inserted back in their original position. Other definitions of relatedness are whether two customers are in the same route or have similar time windows during which they can be visited (Shaw, 1998). Ropke and Pisinger (2006) base the measure on four terms that account for distance, time, capacity and which vehicles can serve both customers. In this chapter, relatedness is defined in terms of distance, as in Pisinger and Ropke (2007).

4.2.3 Regret-k

The removed customers are reinserted in the solution using a repair heuristic. In this chapter a regret-k operator is considered, which prioritises customers that are considered 'difficult'. This parallel route building algorithm for the VRPTW is proposed by Potvin and Rousseau (1993). The difficulty of an insertion is determined by calculating the difference between a customer's best insertion route (i.e., insertion in this route adds the smallest distance to the overall travelled distance) and its second and third, ... up until its k-th best insertion route. This difference is referred to as the regret value and can be formulated as in equation (4.1). The term f_c^1 represents the change in the objective function value when inserting customer c at its best position in its cheapest route, while the term f_c^k indicates the change when inserting customer c at its best position in its k-cheapest route. Customers having high regret values are considered difficult to insert, in the sense that the additional cost incurred of not choosing the cheapest route for insertion is high. These customers should therefore

be prioritised and inserted in their cheapest route at their minimum cost position. If there are any ties in the calculated regret value, these are broken by choosing the customer with the lowest insertion cost.

$$\sum_{n=2}^{k}(f_c^k - f_c^1) \tag{4.1}$$

Regret-2 is the only regret variant used in the case study of Chapter 3. In the experiments performed in this chapter the variants regret-3 and regret-4 are included to find out whether the observed pattern also prevails when looking further ahead in the repair process.

4.3 Experimental Set-Up

Since the focus shifts from analysing a complete metaheuristic to analysing individual operators, they are extracted from the metaheuristic framework. The objective function value of a single destroy and repair iteration is measured to allow a detailed analysis of the destroy and repair process. A general outline of a destroy and repair iteration is given in Algorithm 2.

A data set of 10 000 observations is generated following the same multilevel experimental design as employed in Chapter 3. It consists of 200 artificial VRPTW instances and 50 random parameter settings tested per problem instance. A parameter setting in this experiment is defined as a combination of a single destroy operator — either random or related removal — with a single repair operator — either regret-2, regret-3 or regret-4. The experimental design is depicted in Table 4.1. Information on the values for the problem instance characteristics and algorithm parameters can be found in Tables 3.2 and 4.2. Each scenario is run on Xeon E5-2680v3 CPUs (2.5 GHz, 30 MB level 3 cache) with 128 GB RAM per compute node under Red Hat Enterprise Linux ComputeNode release 6.5 (Santiago), 64 bit. In order to analyse results, multilevel regression models are fitted using the statistical R packages *lme4* (Bates et al., 2015) and *brms* (Bürkner, 2017).

4.4 Analysis of Results

For 6124 of the 10 000 scenarios (61.24%) the destroy and repair iteration leads to an improvement over the initial solution. Since each scenario only performs one iteration

Table 4.1: Multilevel Experimental Design for Single Destroy and Repair Iteration

Instance Characteristics				**Algorithm Parameters**				
Scenario	**Customers**	...	**Average Demand**	**Random Removal**	**Related Removal**	**Regret-2**	**Regret-3**	**Regret-4**
1	169	...	29.79	True	False	False	True	False
2	169	...	29.79	False	True	False	False	True
3	169	...	29.79	True	False	False	True	False
...	...	...	...	...	...	...	...	...
50	169	...	29.79	True	False	True	False	False
...	...	...	...	...	...	...	...	...
9951	224	...	31.04	True	False	False	True	False
9952	224	...	31.04	False	True	False	True	False
9953	224	...	31.04	False	True	False	True	False
...	...	...	...	...	...	...	...	...
10000	224	...	31.04	False	True	True	False	False

Table 4.2: Operators

Operator	**Type**	**Value Ranges**
Destroy operators		U[0,1]
- Random removal	Dummy	True(0)/False(1)
- Related removal	Dummy	True(1)/False(0)
Repair operators		U[0,2]
- Regret-2	Dummy	True(0)/False(1,2)
- Regret-3	Dummy	True(1)/False(0,2)
- Regret-4	Dummy	True(2)/False(0,1)

Algorithm 2 Destroy and Repair Iteration

Input: Problem instance j, Parameter setting θ

Output: Solution x

Initialisation: initial solution x^{start} constructed by greedy heuristic

Function *Destroy*(θ, x^{start});
  Choose a random number q from interval [0.1n, 0.5n] (n = problem size)
  $x^d = x^{start}$
  repeat
    Select customer c to remove (given the destroy neighbourhood in θ)
    $x^d = x^d - c$
    Add customer c to request bank array
  until q customers are removed
return x^d
Function *Repair*(θ, x^d);
  $x = x^d$
  repeat
    Select customer r from request bank (given the regret-k variant in θ)
    $x = x + r$
    Remove customer r from request bank array
  until Request bank is empty or no more customers can be feasibly inserted
return x

and customers are to be served by a limited number of vehicles — i.e., at most the number of vehicles used in the initial solution —, it can occur that during the repair of the solution the final customer(s) to reinsert has(have) no feasible insertion and are placed in a request bank with a penalty cost of 100 000 assigned to the objective function value for each customer not reinserted. This is the case for 3239 scenarios in the data set. Such occurrences can be accounted for when fitting regression models through a request bank variable, but it still proved to be problematic to obtain a model that complies with the underlying assumptions of independence, normality, and homoscedasticity of the errors. The choice is therefore made to only consider the scenarios that were able to reinsert all removed customers. This is also the only relevant type of solution since it corresponds with a regular LNS solution. So, a truncated data set of 6761 observations is considered.

The analyses performed in the following sections have three stages. First, an ex-

ploratory stage verifies whether the operator pattern observed in Chapter 3 is also present in the new experimental data, and starts the search for explanations by looking into the customer difficulty measure that is used to select customers to reinsert (Section 4.4.2). A second stage concerns the 'theory construction' regarding customer prioritisation by formulating and verifying several statements coming out of the exploratory stage (Section 4.4.3). Finally, the 'theory validation' is performed by means of a regular LNS experiment (Section 4.4.4). Residual plots of all regression models are provided in Appendix C.

4.4.1 Verifying Observed Operator Pattern

A first analysis verifies whether the observed operator pattern in Chapter 3 is confirmed in this new experiment. A multilevel regression model is formulated in equations (4.2) to (4.4) predicting the total cost of a solution in terms of the total distance travelled (Y_i). The factors affecting this performance measure are the choice of destroy and repair operator (variables *Related, Regret-3 and Regret-4*), the percentage of customers removed (*Percentage removed*) and the total number of customers to be served (*Customers*)[2]. Variables for random removal and regret-2 are not included because of multicollinearity issues[3]. Therefore, they are the baseline operator choices for which the effect is accounted in the regression intercept and the effects of all other operators are relative to this baseline operator scenario. All continuous variables are centred around their mean value to facilitate interpretation of effect estimates. Random effects are considered up to two-way interaction effects.

$$\begin{aligned}
\sqrt{Y_i} &= \alpha_{j[i]} + \beta_{1j[i]} Related_i + \beta_{2j[i]} Regret3_i + \beta_{3j[i]} Regret4_i + \\
&\quad \beta_{4j[i]} Percentage\ removed_i + ... + \\
&\quad \beta_{11} Related_i \times Regret4_i \times Percentage\ removed_i + \epsilon_i \qquad (4.2) \\
\alpha_j &= \mu_0^{\alpha} + \mu_1^{\alpha} Customers_j + \eta_j^{\alpha} \qquad (4.3) \\
\beta_{kj} &= \mu_0^{\beta_k} + \mu_1^{\beta_k} Customers_j + \eta_j^{\beta_k} \qquad (4.4)
\end{aligned}$$

[2]The regression analyses in this chapter only consider a single problem instance characteristic (i.e., *Customers*). The analysis of Chapter 3 showed this characteristic to be the most influential and this will also be confirmed by the importance analysis of Chapter 5.

[3]Multicollinearity may lead to inflated variance estimates and a high sensitivity of the coefficient estimates for changes in the model. This makes it difficult to interpret results as the estimates are unstable. In this case, including all destroy variables or all repair variables would lead to perfect multicollinearity. One variable of each needs to be left out and serves as a reference value which is represented in the regression intercept.

with

$i \in I$	scenario, a combination of a problem instance with a parameter setting
$j \in J$	problem instance
$k \in K$	algorithm parameter
$j[i]$	index variable to code problem instance membership ($j[i] = j$), e.g., $j[10] = 5$ means the 10th scenario solves problem instance 5
Y_i	objective function value of scenario i
$\alpha_{j[i]}$	varying regression intercept, representing the objective function value given scenario i and problem instance j when the value for all k parameters is 0
$\beta_{kj[i]}$	varying effect of algorithm parameter k on Y given scenario i and problem instance j
β_k	fixed effect of algorithm parameter k on Y given scenario i
$\mu_0^{\beta_k}$	mean effect of algorithm parameter k on Y
η_j	error at the problem instance level and is assumed to be $\sim N(0,\sigma^2)$
ϵ_i	error at the parameter setting level and is assumed to be $\sim N(0,\sigma_e^2)$

The regression results in Table 4.3 confirm the previous findings of Chapter 3: random removal performs on average better than related removal when customers are reinserted with the regret-k operator. Figure 4.2 plots the predicted performance for the various operator combinations given a problem instance with an average number of customers (i.e., about 184 customers). The predicted total cost when using random removal and regret-2 is given by the intercept value $174.60^2 = 30\,485.16$ and will increase to $(174.60 + 1.56)^2 = 31\ 032.35$ when using related removal instead of random removal. For the combination with regret-3 and regret-4 the predicted value is $(174.60 + 0.08)^2 = 30\,513.10$ with random removal, while the predictions are respectively $(174.60 + 1.56 + 0.08 - 0.25)^2 = 30\,972.48$ and $(174.60 + 1.56 + 0.08 - 0.24)^2 = 30\,976$ with related removal. Regret-3 and regret-4 perform a little worse than regret-2 with random removal, while they perform a little better with related removal. This is also clear from the coefficient estimates. The estimates for *Regret-3* (0.08) and *Regret-4* (0.08) are positive with random removal,

Table 4.3: Significant effects model pattern verification

Dependent variable: $\sqrt{\text{total cost}}$	Estimate	Std. Error	l-95% CI	u-95% CI
Intercept	174.60***	1.14	172.32	176.80
Customers	0.40***	0.01	0.38	0.43
Related	1.56***	0.07	1.42	1.71
Customers	0.01***	0.001	0.005	0.01
Regret-3	0.08**	0.05	−0.01	0.17
Customers	−0.001*	0.0004	−0.001	0.0002
Regret-4	0.08**	0.05	−0.01	0.17
Customers	0.0003*	0.0004	−0.0005	0.001
Percentage removed	−0.09***	0.003	−0.10	−0.09
Customers	−0.0001***	0.000	−0.0002	−0.0001
Related × Regret-3	−0.25***	0.08	−0.40	−0.09
Customers	−0.001	0.001	−0.002	0.001
Related × Regret-4	−0.24***	0.08	−0.40	−0.08
Customers	−0.001	0.001	−0.003	0.0003
Related × Percentage removed	0.09***	0.01	0.08	0.10
Customers	0.0002***	0.0000	0.0002	0.0003
Regret-3 × Percentage removed	0.005	0.004	−0.003	0.01
Customers	−0.0000	0.0000	−0.0001	0.0000
Regret-4 × Percentage removed	0.01	0.004	−0.002	0.01
Customers	−0.0000	0.0000	−0.0001	0.0000
Related × Regret-3 × Percentage removed	−0.02**	0.01	−0.03	−0.003
Related × Regret-4 × Percentage removed	−0.02**	0.01	−0.03	−0.003
Observations	6761			
Num. groups: problem instances	200			

Note 1: *p<0.1; **p<0.05; ***p<0.01

but become negative when accounting for the negative estimates for the interaction effects *Related* × *Regret-3* (−0.25) and *Related* × *Regret-4* (−0.24). These significant estimates can be found in Table 4.3. Further, the total cost measure significantly increases for larger problem sizes (0.40) and decreases for increasing percentages of customers removed (−0.09). Significant interaction effects are found between between related removal and the percentage of customers removed and between related removal, the repair operators and the percentage of customers removed. Finally, the problem size significantly influences all individual operator effects, the percentage removed effect and the latter's interaction with related removal. Having verified the presence of the destroy operator performance difference in the new experimental data, the explanatory study can begin.

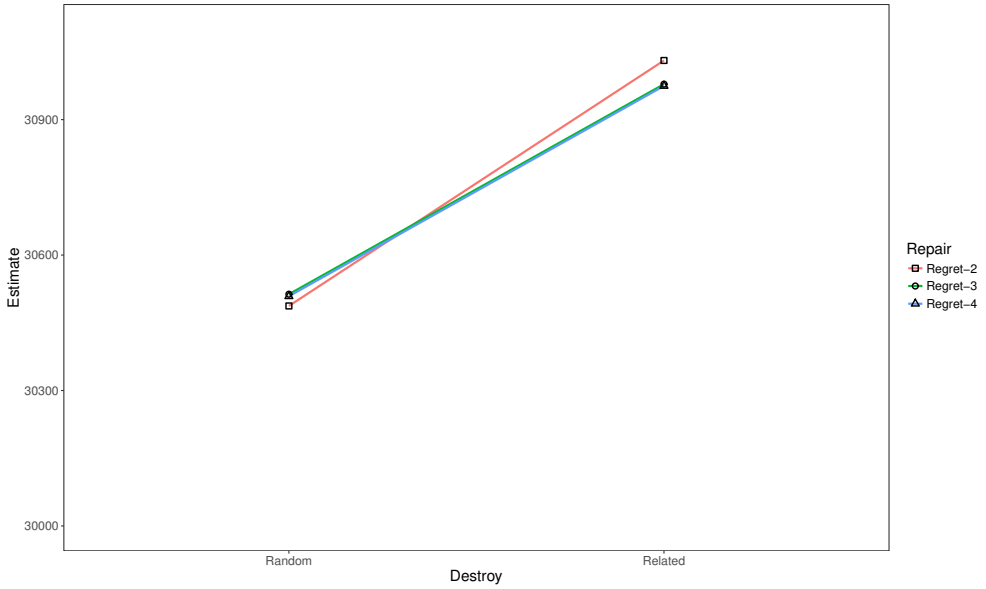


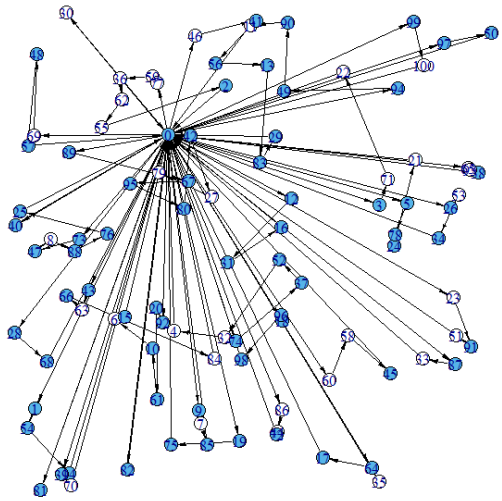
Figure 4.2: Predictions for a problem instance with an average number of customers.

4.4.2 Investigating Customer Difficulty

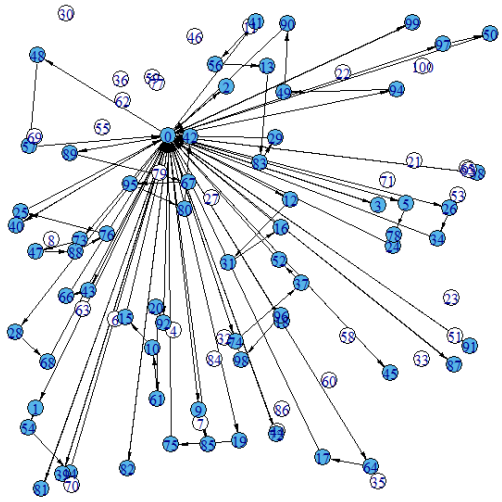
In the search for explanations for this counterintuitive result, the process of destroying and repairing a solution is decomposed and visualised for a problem instance with 100 customers in Figures 4.3 and 4.4. When customers are removed at random, a random sample of customer nodes are identified for removal (marked white in Figure 4.3a), no additional criteria are used. Related removal, on the other hand, defines a

relatedness criterion to base removals on. In these experiments, relatedness is defined in terms of distance, so this operator will remove a number of geographically nearby customers from the solution (marked white in Figure 4.4a). In both cases, customers are reinserted using a regret-k operator with k equal to 2, 3 or 4. The criterion used to decide which customer to iteratively insert is the additional cost incurred when not inserting a customer in its best route. The higher this additional cost, the higher the priority given to the customer. This measure is also referred to as the regret value. Hence, when customers have large regret values, this implies that the cost gap between the best insertion route for these customers and their second, third, fourth best route is large. Therefore, these customers should be considered first for insertion, since they only have a small number of interesting insertion alternatives. Customers having small regret values do not have to be immediately inserted since they can more easily be inserted in alternative routes for which the cost of insertion is not a lot higher compared to the best insertion route. Since the regret value is the key measure used in the repair process and applied in both removal scenarios, a first investigation focuses on this measure and how it differs between inserting randomly dispersed customers or a clustered group of customers.

It is observed in Figure 4.3b that removing customers at random affects many routes: 22 of the 32 routes have on average one or two customers removed. The repair phase reinserts about 70% of them in their original route at the same position — there are no intra-route position switches. So, any improvements in solution quality are due to the minority of removed customers that switch routes in Figure 4.3c. In the case of removing a geographical cluster of customers in Figure 4.4b less routes are affected (only 9) and, contrary to reinserting randomly removed customers, about 67% of the removed customers switch routes during the repair phase (Figure 4.4c). Furthermore, a (large) part of the solution is completely destroyed due to the removal of entire routes and has to be rebuilt from scratch. This is shown in the upper left corner of Figure 4.4b where there is not a single route left. For many of the removed customers, there are not a lot of existing routes nearby that are potential candidates to insert a customer. Consequently, for these customers the number of good alternative routes for their cheapest insertion route is expected to be small. This means that the cost difference between inserting a customer in its cheapest route and — depending on the k value — its second, third and fourth cheapest route is large. A randomly removed customer, on the other hand, has more existing routes nearby and thus better alternatives for the cheapest insertion route, resulting in a lower regret value.

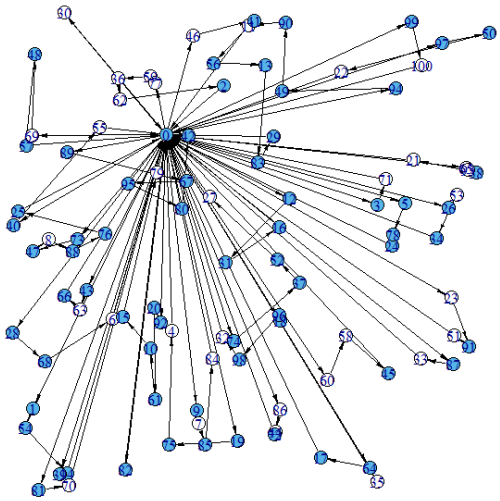


(a) Start Solution



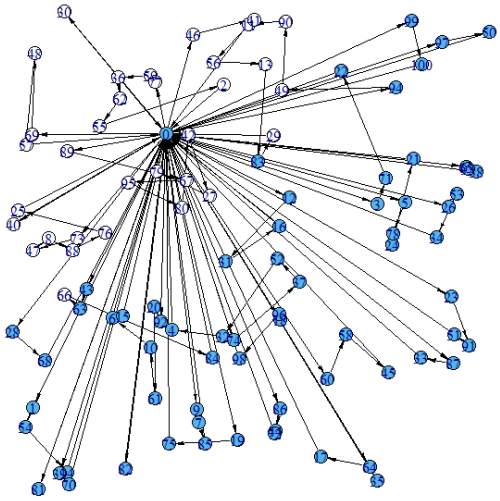
(b) Destroyed solution (34% removed)

Figure 4.3: Destroy and repair iteration when customers are removed at random.



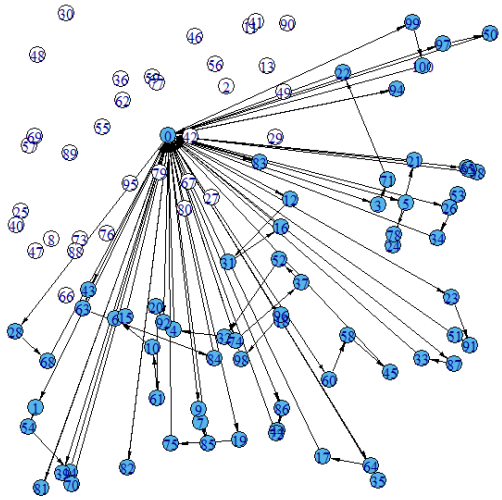
(c) Solution at the end of iteration 1

Figure 4.3: Destroy and repair iteration when customers are removed at random.

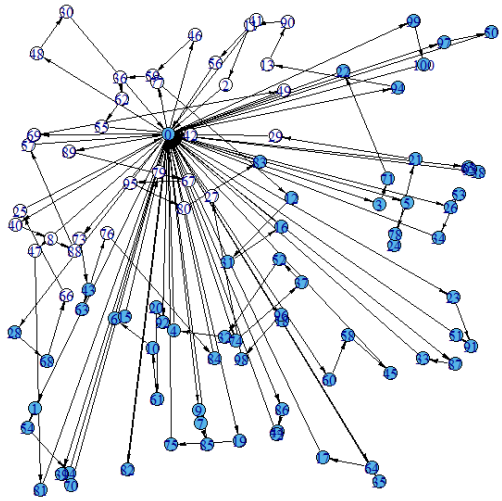


(a) Start Solution

Figure 4.4: Destroy and repair iteration when related customers are removed.



(b) Destroyed solution (34% removed)



(c) Solution at the end of iteration 1

Figure 4.4: Destroy and repair iteration when related customers are removed.

Hence, the observations from the exploratory stage of this study lead to the expectation that a customer who is part of a cluster of removed customers that are geographically nearby has fewer feasible alternatives compared to a randomly removed customer. Further, due to the difference in number of feasible alternatives, it is expected that the average regret value of the customer selected for insertion is higher for scenarios in which a cluster of customers has been removed. Statistical evidence for these expectations is sought such that the following two null hypotheses posing no difference can be rejected. It is the start of the theory construction stage.

Hypothesis 1 (H1): *The average number of feasible route options each removed customer can be inserted in is not different when the removed customer is selected at random or when it is part of a cluster of geographically nearby customers that is removed from the solution.*

Hypothesis 2 (H2): *The average (maximum) regret value of the customer selected for insertion is not different when the removed customer is selected at random or when it is part of a cluster of geographically nearby customers that is removed from the solution.*

The first hypothesis (H1) is validated by fitting a multilevel regression model predicting the average number of feasible insertions each removed customer has based on the choice of repair and destroy operators, the percentage of customers removed and the total number of customers to be served. All effect estimates are provided in Table 4.4. Given a scenario with an average number of customers, removing an average percentage of customers (i.e., about 28%) and using random removal and regret-2, each removed customer is expected to have 3.10 feasible insertion routes on average — this is the Intercept value in Table 4.4 back-transformed to the original scale, i.e. 1.76^2. If related removal is used instead of random removal, the number decreases to 2.53 ($= (1.76 - 0.17)^2$). The findings are similar for the combination with regret-3 and regret-4, but with slightly lower averages for scenarios relying on related removal. The decrease is statistically significant as indicated in Table 4.4 and the regression results therefore provide statistical evidence to reject Hypothesis 1. This rejection goes for all problem sizes. The more customers have to be served, the more negative the effect of related removal becomes (-0.001). Further, the effect is significantly negative for all removal percentages and also becomes more negative for increasing percentage values (-0.01).

Intuitively, a rejection of this hypothesis is logical since removing a cluster of geographically nearby customers most likely results in the removal of one or multiple entire routes and thus it is not surprising that in such a scenario the removed customers have less feasible options for insertion than in other scenarios. This regression analysis provides a statistical confirmation of what can be known by intuition.

Table 4.4: Regression Table Hypothesis 1

Dependent variable: $\sqrt{\text{avg \# of feasible insertions}}$	Estimate	Std. Error	l-95% CI	u-95% CI
Intercept	1.76^{***}	0.01	1.74	1.78
Customers	0.0013^{***}	0.0001	0.0011	0.0015
Related	-0.17^{***}	0.01	−0.19	−0.14
Customers	-0.001^{***}	0.0001	−0.0009	−0.0007
Regret-3	−0.01	0.01	−0.03	0.003
Customers	-0.0002^{***}	0.0001	−0.0003	−0.0001
Regret-4	-0.01^{*}	0.01	−0.03	0.002
Customers	-0.0004^{***}	0.0001	−0.0005	−0.0002
Percentage removed	0.004^{***}	0.001	0.003	0.005
Related × Regret-3	-0.06^{***}	0.01	−0.089	−0.034
Related × Regret-4	-0.06^{***}	0.01	−0.088	−0.034
Related × Percentage removed	-0.01^{***}	0.001	−0.007	−0.004
Regret-3 × Percentage removed	0.0003	0.001	−0.002	0.001
Regret-4 × Percentage removed	−0.001	0.001	−0.002	0.0008
Related × Regret-3 × Percentage removed	−0.001	0.001	−0.003	0.001
Related × Regret-4 × Percentage removed	−0.0004	0.001	−0.003	0.002
Observations	6761			
Num. groups: problem instances	200			

Note: *p<0.1; **p<0.05; ***p<0.01

The second hypothesis (H2) is validated in a similar fashion by fitting a multilevel regression model predicting the average regret value. Table 4.5 lists all effect estimates. A log transformation of the regret value variables is required in order to comply with the assumptions typically underlying regression models. The values can be back-transformed to their original scale by taking the exponential function. For example, the average[4] regret value when using random removal and regret-2 is $\exp(6.61) = 742.48$ while it is $\exp(6.61 - 0.22) = 595.86$ when using related

[4]The antilog of the arithmetic mean of log-transformed values is the geometric mean. For positively skewed data, which is the case for the experimental data in this study, the geometric mean will be less than the arithmetic mean. The geometric mean is often a good estimate of the original median.

removal and regret-2[5]. For the combination with regret-3 the predicted values are respectively 5884.05 and 8866.19, while for the combination with regret-4 they are 13 359.73 and 17 676.65. The regression output in Table 4.5 indicates that this average regret measure is significantly higher for scenarios using related removal, but for the combination with regret-2 the regression model indicates the measure to be significantly lower when using related removal. This suggests that the second best alternative each removed customer has is on average of a better quality (meaning the difference in insertion cost is smaller) when removing a geographical cluster of customer than when removing randomly dispersed customers. For a removed customer, part of a geographical cluster of removed customers, both the best and second best insertion route can be distant and, therefore, differ little in cost of inserting the customer in these routes. Note that this does not mean the insertion itself is cheap. Both insertion alternatives can be costly. For a randomly removed customer, on the other hand, there is probably more variation in the distance of the customer to the feasible insertion routes because of the randomness applied, but the insertion cost might be lower. This is confirmed in a regression analysis predicting the average insertion cost[6]. It indicates that insertions in scenarios relying on related removal to be significantly more expensive than in scenarios relying on random removal.

Investigation of the problem size influence (cf. coefficients 0.0007, 0.001, 0.0001 in Table 4.5) shows the average regret value being significantly lower with related removal up to 308 customers (with regret-2), and to be significantly higher for problem sizes of 110(regret-3)/165(regret-4) customers or more. So for problem instances with less than (roughly) 100 customers, the average regret value difference between random and related removal is not as expected. The average difference between best and second best insertion option is smaller when customers are removed according to a related logic compared to random removal. When also accounting for third and fourth best alternatives, there is no clear quality distinction between both destroy scenarios. This changes when more and more customers have to be served. The average difference between best and second best insertion option becomes increasingly smaller and eventually insignificant, while it alters to the expected larger difference — with related removal — when incorporating third and fourth best options.

[5]The predicted average regret value for the combination with *Related* can also be derived directly from its effect estimate -0.22, i.e. the predicted value is $\exp(-0.22) = 80.25\%$ of 742.48

[6]Table B.2 in Appendix B provides the regression output

The other interacting influence is that of the percentage of customers removed during the destroy process (cf. coefficients 0.02 and −0.01). The average regret measure is significantly lower with related removal for percentages up to 34% (with regret-2), 15% (with regret-3) and 22% (with regret-4). It becomes significantly higher for percentages of 45% (with regret-2), 21% (with regret-3) and 27% (with regret-4) or higher. So higher removal percentages negatively impact the quality of second, third and fourth best alternatives for a geographical cluster of removed customers.

Table 4.5: Regression Table Hypothesis 2

Dependent variable: Log(avg regret value)	Estimate	Std. Error	l-95% CI	u-95% CI
Intercept	6.61***	0.04	6.54	6.69
Customers	−0.004***	0.0003	−0.005	−0.003
Related	−0.22***	0.05	−0.31	−0.13
Customers	0.0007*	0.0004	−0.00016	0.0015
Regret-3	2.07***	0.03	2.00	2.13
Customers	0.0025***	0.0003	0.0019	0.0031
Regret-4	2.89***	0.03	2.82	2.96
Customers	0.0033***	0.0003	0.0028	0.0039
Percentage removed	−0.056***	0.001	−0.06	−0.05
Related × Regret-3	0.41***	0.05	0.31	0.50
Customers	0.001*	0.0004	−0.00002	0.0017
Related × Regret-4	0.28***	0.05	0.18	0.36
Customers	0.0001	0.0004	−0.00079	0.00095
Related × Percentage removed	0.02***	0.002	0.016	0.025
Regret-3 × Percentage removed	0.036***	0.002	0.033	0.039
Regret-4 × Percentage removed	0.045***	0.002	0.041	0.047
Related × Regret-3 × Percentage removed	−0.003	0.003	−0.0083	0.0023
Related × Regret-4 × Percentage removed	−0.01***	0.003	−0.016	−0.0044
Observations	6761			
Num. groups: problem instances	200			

Note: *p<0.1; **p<0.05; ***p<0.01

The fact that each customer has less feasible alternatives with related removal also explains the higher average regret value measured since a large cost (i.e., 9999)

is assigned to every infeasible insertion point within a route. This is most notable in the scenarios relying on regret-3 and regret-4. The large cost is assigned to prevent customers with more feasible alternatives having a larger regret value and thus higher priority than customers with less feasible alternatives.

4.4.3 Customer Prioritisation

Having found confirmation about the expectations on the difference in number of feasible alternatives in both removal scenarios, the sequence in which customers are reinserted back in the solution is investigated to see which customers are considered difficult and thus prioritised. For the problem instance plotted in Figure 4.4b customer 90 is inserted first, followed by customers 11, 25, 67, 80, 13, ... and finally customers 48 and 30. In this sequence it is noticed that the majority of the customers that are inserted first are almost all in near proximity of an existing route, while the more "isolated" customers (nodes 48 and 30 in Figure 4.4b) are all postponed to the final insertions. Characterising for these "isolated" customers is that they all have not one feasible insertion possibility in one of the existing routes. This means that the calculated regret value for these customers is zero[7] and they are thus assigned the lowest priority of all removed customers. The single alternative they have is to create a route straight from the depot to the customer and back. In the regret-k implementation used in these experiments, such an alternative is assigned a regret value of zero and thus the lowest priority since there is no immediate cost loss of postponing the insertion for that customer. The customer has no other alternatives to take into account, so its insertion is not urgent. Secondly, as the solution of routing problems typically not only strive to minimise the total distance travelled, but doing so with the fewest number of vehicles necessary, the current implementation does not favour the creation of additional routes. On the other hand, this observation raises the question what the impact on performance would be if these isolated customers are taken into account at the start of the repair process instead of initially being ignored. Perhaps their prioritisation might benefit other removed customers as this adds alternative routes in an area with few or no routes at all, maybe better alternatives than when the isolated customers are postponed to the final insertions. Therefore, it is expected that scenarios removing clusters of customers lead to more isolated customers than scenarios that remove customers at random. Secondly, the

[7]Recall that routes which are infeasible insertion options get assigned a large cost of 9999, so when a removed customer has not a single feasible insertion option, the regret-2 value, for example, is calculated as $9999 - 9999 = 0$.

solution quality is analysed to see how it differs between both removal strategies if isolated customers are prioritised instead of postponing their insertion. Hypotheses 3 and 4 are formulated to validate these expectations.

Hypothesis 3 (H3): *The number of removed customers which, at the start of the repair phase, have no other feasible insertion possibility than an individual route from the depot to the customer and back is not different when these customers are selected at random or when they belong to a geographically clustered group of removed customers.*

Hypothesis 4 (H4): *After performing a single destroy and repair iteration, there is no difference in the average solution quality for scenarios removing geographical clusters of customers if priority is given to customers which have no other feasible insertion possibility than an individual route from the depot to the customer and back or if these customers are ignored until the final insertions.*

Hypothesis 3 is validated by a model predicting the number of isolated customers available at the start of the repair process. Since this model involves predicting a (discrete) count variable, a Poisson regression model is typically fitted rather than the standard Gaussian model (Gelman and Hill, 2006; Cameron and Trivedi, 2013). The Poisson distribution assumes $E(Y) = \phi var(Y) = \phi\mu$ with $\phi = 1$, thus that the mean equals the variance. Therefore, the usual assumption of homoscedasticity is not appropriate for Poisson models. However, the mean equals variance relationship is often violated, i.e., the variance is often larger than the mean ($\phi > 1$). This is called overdispersion and is present when fitting a Poisson model on the experimental data. The overdispersion is estimated at $\phi = 3.54$. To accommodate for overdispersion in count data, a good alternative is a negative binomial regression model. It is a generalisation of the Poisson regression that relaxes the assumption of the variance being equal to the mean (Cameron and Trivedi, 2013). The output of the negative binomial regression is given in Table 4.6. There are no repair operator effects included in this model since the number of isolated customers is measured at the start of the repair phase before any insertion has occurred.

For a problem instance with an average number of customers (i.e., 184), these results show that scenarios using random removal on average have $exp(0.42) = 1.52$ isolated customers while scenarios using related removal have a significantly higher average of $exp(0.42 + 1.76) = 8.85$ isolated customers. The larger the percentage of customers removed, the higher this number, but also the larger the difference

between random and related removal as indicated by the estimate for *Percentage removed* (0.05) and the estimate for *Related* × *Percentage removed* (0.01). Likewise, for a fixed percentage of customers removed, the number of isolated customers will increase for larger problem instances (0.002) as well as the gap between random and related removal (0.003).

Table 4.6: Regression Table Hypothesis 3

Dependent variable: isolated customers	Estimate	Std. Error	l-95% CI	u-95% CI
Intercept	0.42***	0.05	0.33	0.50
Customers	0.002***	0.0004	0.001	0.003
Related	1.76***	0.04	1.68	1.84
Customers	0.003***	0.0004	0.002	0.004
Percentage removed	0.05***	0.002	0.04	0.05
Customers	−0.00002	0.00001	−0.00004	0.00001
Related × Percentage removed	0.01***	0.002	0.01	0.02
Observations	6761			
Num. groups: problem instances	200			

Note: *p<0.1; **p<0.05; ***p<0.01

The final hypothesis (H4) is validated by changing the prioritisation mechanism in regret-k such that the most isolated customers are now no longer ignored at the start, but are assigned a higher priority. The regret value for these customers is calculated as the difference between a large cost value of 9999 and twice the distance from the customer to the depot[8]. The 10 000 scenarios are run again using the modified prioritisation. An improvement over the initial solution is found for 8212 scenarios (82.12%), an increase of almost 20% compared to the previous data set. In 1687 scenarios one or multiple customers failed to be reinserted and remain in the request bank, which is almost half the number observed in the previous experiment. Hence, the truncated data set that will be analysed has 8313 observations, consisting of the previously mentioned 8212 scenarios and 101 scenarios that were able to reinsert all removed customers, but could not improve the initial solution.

Concerning customer prioritisation, for the problem instance in Figure 4.4

[8]For regret-3 and regret-4 this difference is respectively doubled and tripled.

customers 48 and 30 are now inserted a lot sooner resulting in a predicted total cost of 19 282.93 compared to 19 821.75 in the original prioritisation. The predicted total cost for the scenario using random removal (plotted in Figure 4.3) remains at 19 335.85. A statistical analysis is performed by fitting the regression model formulated in equations (4.2) to (4.4) on the new performance results. The effect estimates are listed in Table 4.7. The average performance for the different destroy and repair combinations is plotted in Figure 4.5. Note that, compared to Figure 4.2, the average performance values are higher for all operator scenarios, but this is not a completely fair comparison since the new experimental data includes 1552 observations more, which might explain the higher averages.

The important observation in Figure 4.5 is that the majority of the performance difference between random and related removal is reduced. A significant improvement is even observed for the combination with regret-3 for problem instance sizes up to 222 customers — $(0.11 + 0.002Customers) + (-0.23 - 0.001Customers) = -0.12 + 0.001Customers$. The combination of related removal with regret-2 or regret-4 performs significantly better on the smaller problem sizes (i.e., 25 to 65 customers), but still performs significantly worse than random removal for the larger instances (i.e., 208 customers or more) — $0.11 + 0.002Customers$. Nonetheless, the performance deterioration in this problem size range is largely reduced. The interaction with the removal percentage (estimates 0.03 and -0.01) makes the effect of switching from random to related removal negative — so an improvement in performance — for lower percentage values and positive for higher values. A solution improvement is observed up until 22% (with regret-2), 30% (with regret-3) and 24% (with regret-4), while related removal performs significantly worse when 30% (with regret-2), 44% (with regret-3) and 33% (with regret-4) or more of the solution is destroyed. Hence, related removal performs better when a limited number of customers are removed.

The two different prioritisations are compared in a simple multilevel regression model with a varying intercept per problem instance and a group predictor. Since this analysis compares the total cost value obtained for the same scenario in both experiments, only the 6432 scenarios are considered that both experiments have in common. The variable *Experiment* is a binary variable to indicate the experiment. The effect estimate for this variable in Table 4.8 is significantly negative (-0.011) implying the adjusted prioritisation leads to an expected improvement in performance. Hypothesis H4 is therefore rejected.

Table 4.7: Significant effects regression model Hypothesis 4

Dependent variable: $\sqrt{\text{total cost}}$	Estimate	Std. Error	l-95% CI	u-95% CI
Intercept	178.77***	1.36	176.17	181.52
Customers	0.40***	0.01	0.38	0.43
Related	0.11**	0.05	0.003	0.21
Customers	0.002***	0.0001	0.001	0.003
Regret-3	0.06	0.05	−0.03	0.15
Customers	−0.001	0.0004	−0.001	0.0002
Regret-4	0.06	0.05	−0.03	0.15
Customers	−0.0001	0.0004	−0.001	0.001
Percentage removed	−0.12***	0.003	−0.13	−0.11
Customers	−0.0002***	0.00	−0.0002	−0.0001
Related × Regret-3	−0.23***	0.07	−0.36	−0.09
Customers	−0.001*	0.001	−0.002	0.0001
Related × Regret-4	−0.06	0.07	−0.20	0.07
Customers	−0.0005	0.001	−0.002	0.001
Related × Percentage removed	0.03***	0.004	0.02	0.04
Customers	0.0001***	0.00	0.0001	0.0002
Regret-3 × Percentage removed	0.01	0.004	−0.00	0.01
Customers	−0.00*	0.00	−0.0001	0.00
Regret-4 × Percentage removed	0.003	0.004	−0.005	0.01
Customers	−0.00	0.00	−0.0001	0.00
Related × Regret-3 × Percentage removed	−0.01*	0.01	−0.02	0.0002
Related × Regret-4 × Percentage removed	−0.01	0.01	−0.02	0.01
Observations	8313			
Num. groups: problem instances	200			

Note 1: *p<0.1; **p<0.05; ***p<0.01

The expectations on why there is a performance difference are for the most part confirmed, but the experimental data used to validate the hypotheses is based on a single destroy and repair iteration performed on an initial greedy solution. This was done to enable a detailed study of the repair and destroy process. A large neighbour search typically runs a large number of iterations and returns the best found solution across these iterations. Whether a better performance is also observed when running

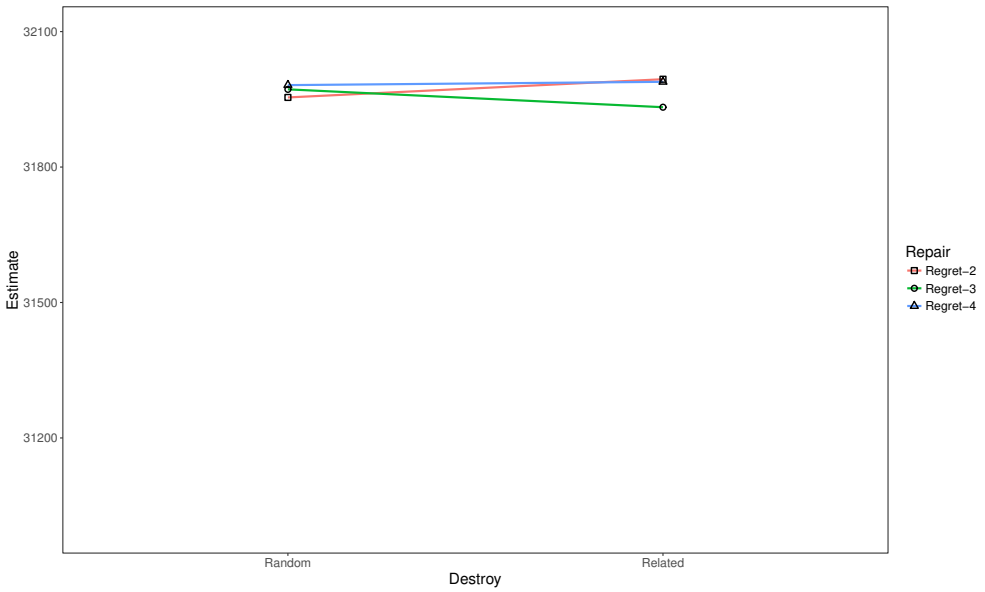


Figure 4.5: Predicted performance given a problem instance with an average number of customers (i.e., about 208)

Table 4.8: Regression Table for comparing experiments with different prioritisation

Dependent variable: Log(total cost)	Estimate	Std. Error	l-95% CI	u-95% CI
Intercept	10.36***	0.04	10.28	10.44
Experiment	−0.011***	0.0004	−0.012	−0.0107
Observations	12 864			
Num. groups: problem instances	200			

Note 1: *p<0.1; **p<0.05; ***p<0.01

Note 2: Experiment: original(0)/adjusted(1) prioritisation

Note 3: For 6432 scenarios both experiments were able to reinsert all removed customers.

multiple iterations is verified in a final experiment similar to the experiment performed in Chapter 3 using new sample data. This is the final stage of the explanatory study where the theory is validated.

4.4.4 Validation with Large Neighbourhood Search

A new data set of 4000 scenarios is generated by sampling 200 problem instances and 20 parameter settings per problem instance. A detailed description of all parameters can be found in Section 3.3 of Chapter 3. The restriction that a parameter setting has only one destroy and one repair operator is maintained. Two separate LNS experiments are run using the same input data: the first one gives low priority to isolated customers while the second one assigns a high priority to these customers.

The regression results for both LNS experiments are given in Tables B.3 and B.4 in Appendix B. An improvement in solution quality (26.33) is observed compared to the LNS experiment that ignores isolated customers (Table 4.9). Furthermore, prioritising isolated customers makes the LNS also more efficient (-0.64) on those instances where both experiments obtain the same best solution in that less iterations are required to reach this best found solution (Table 4.10). Nonetheless, the results still suggest that random removal performs significantly better than related removal when combined with a regret-k operator. But this does not hold for all problem sizes. As can be seen in Table 4.11, related removal now performs significantly better than random removal for the smaller problem sizes, when the destroy operator is combined with regret-3 or regret-4 (Figure 4.6). For the larger problem sizes random removal still performs significantly better (Figure 4.7). Looking at the repair operators, the effect estimates for regret-3 (-2.69) and regret-4 (-5.05) suggest they perform significantly worse than regret-2 for the combination with random removal. Studying their marginal effect for varying problem sizes regret-3 (0.74) is observed to perform significantly worse for problem instances up to 228 customers and regret-4 (0.07) for all problem sizes. For the combination with related removal (estimates 9.58 and 7.84) both repair operators significantly outperform regret-2 for problem sizes of respectively 72 customers (regret-3) and 177 customers (regret-4) or more (Table 4.12). The observation on the marginal effects of regret-3 and regret-4 with random removal on the smaller problem sizes again raises the question about why they perform worse than regret-2, given that the former operators 'look further ahead' when reinserting customers.

The takeaway of the explanatory study in this chapter concerning a destroy and repair process is that when customers have no feasible insertion possibility in one of the existing routes, but can only be inserted in a new route, their insertion should be given a high priority. Postponing their insertion is shown to be detrimental

Table 4.9: Regression Table for comparing the solution quality of two LNS experiments with different prioritisation

Dependent variable: total cost^{-1}	Estimate	Std. Error	l-95% CI	u-95% CI
Intercept	3858.16***	175.51	3513.38	4202.94
Experiment	26.33***	0.84	24.69	27.98
Observations	8 000			
Num. groups: problem instances	200			

Note 1: *p<0.1; **p<0.05; ***p<0.01

Note 2: Experiment: original(0)/adjusted(1) prioritisation

Table 4.10: Regression table for comparing the number of iterations required to reach the best found solution in two LNS experiments with different prioritisation

Dependent variable: Log(iterations)	Estimate	Std. Error	l-95% CI	u-95% CI
Intercept	6.86***	0.21	6.44	7.27
Experiment	−0.64***	0.807	−0.78	−0.51
Observations	1216			
Num. groups: problem instances	57			

Note 1: *p<0.1; **p<0.05; ***p<0.01

Note 2: Experiment: original(0)/adjusted(1) prioritisation

Table 4.11: Problem size ranges for which either random or related removal performs significantly better

Operator	Regret-2	Regret-3	Regret-4
Random significantly better than Related	≥ 135	≥ 221	≥ 215
Related significantly better than Random	never	≤ 178	≤ 153

Table 4.12: Problem size ranges for which either regret-2, regret-3 or regret-4 performs significantly better

Operator	Random	Related
Regret-2 significantly better than Regret-3/-4	≤ 228/always	never
Regret-3 significantly better than Regret-2	never	≥ 72
Regret-4 significantly better than Regret-2	never	≥ 177

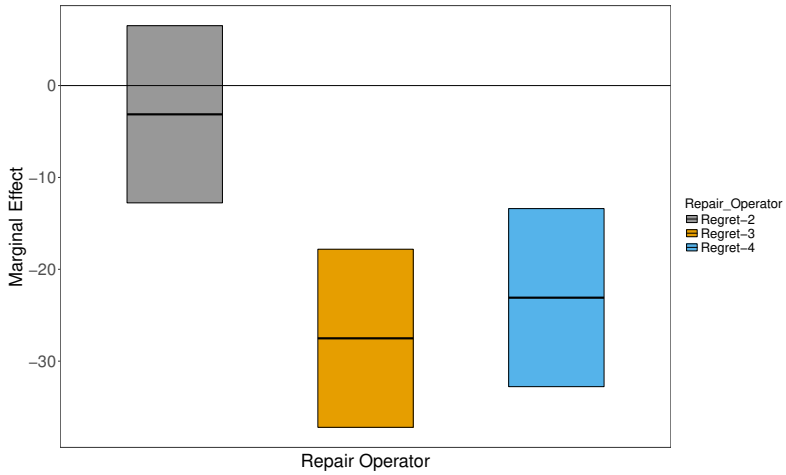


Figure 4.6: Marginal effect of related removal for a problem instance with 25 customers

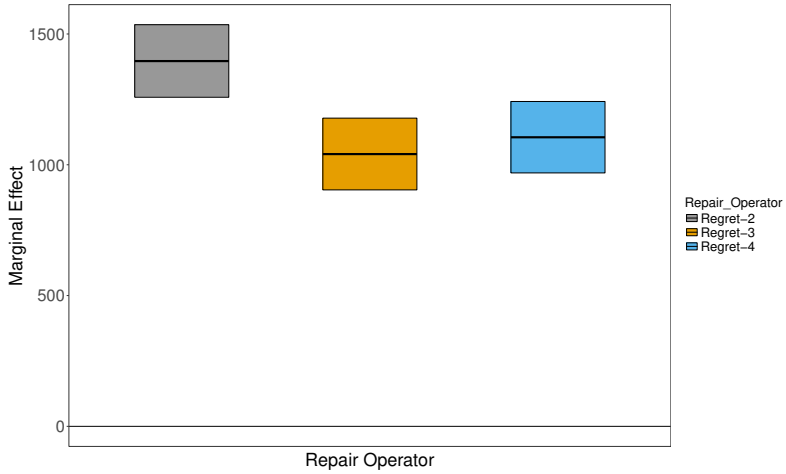


Figure 4.7: Marginal effect of related removal for a problem instance with 400 customers

to algorithm performance — when the repair phase applies a difficulty measure that accounts for cost increases of postponing customer insertions. These kinds of 'isolated' customers occur more often when removed customers form a geographical cluster. So the idea that such removals make it more easy for customers to shuffle within and between routes and that this would lead to a better performance than removing more distanced customers, has to be nuanced. The removal of clusters

will most likely result in the removal of entire routes, thereby removing feasible alternatives for the removed customers when they are to be reinserted.

The analysis managed to explain the majority of the performance gap observed between random and related removal given the combination with a regret-k repair operator. There is still a small significant performance difference for certain problem size ranges. It might be worthwhile to pursue a further optimisation of the prioritisation mechanism or perhaps a preparatory step, that creates individual routes for some difficult — to be defined — customers, might prove beneficial. Such ideas have to be verified in future experiments. As mentioned in Section 4.2, experimentation is an iterative process with each iteration providing a better understanding of the studied process.

4.5 Conclusion

This chapter continued the experimental study started in Chapter 3 and focused on one of the observations of that analysis. An experimental study sought to explain why two destroy operators perform different when combined with the same type of repair operator. It is found that removing geographical clusters of customers reduces the number of insertion alternatives to choose from during the repair phase. Several customers do not even have a single feasible insertion option in one of the existing routes and can therefore be considered isolated cases (at the start of the repair phase). Postponing the insertion of these isolated customers is found to have a detrimental impact on the solution quality. It is tested what the effect is of assigning these customers a higher priority by allowing their insertion in an individual route from the depot to the customer and back, an option that was previously considered as a last alternative. Permitting these individual routes to be created sooner in the repair process adds good insertion alternatives for many other removed customers and thus enables the regret operator to make better choices. Hence, a regret operator will make a better estimation of customer difficulty and consequently a better prioritisation if each individual customer has existing routes nearby in which it can be feasibly inserted. So, when a removed customer cannot be reinserted in one of the existing routes, its insertion (in a new route) should not be considered as less important, but rather as one that is urgent.

An idea coming out of these findings is to introduce and test a new destroy oper-

ator that removes maximally diverse customers. It would remove a first customer at random and then subsequently choose to remove the customer that is located furthest from the customers that are already taken out. This would probably reduce the occurrence of isolated customers and, therefore, result in a better performing operator than the related removal operator. Whether this maximum diversity can also improve upon the random diversity produced by the random removal operator has to be tested.

The study explained the majority of the performance difference between both destroy scenarios. A small significant difference still remains for which an explanation might be found by further improving how customers are prioritised or by using a preparatory step in the regret operator. In addition, it might be interesting to incorporate a number of customers per route variable, since routes with many customers are probably never completely destroyed. A customer removed from such a route is less likely to become 'isolated' compared to a customer removed from a route serving only a few customers. This can further nuance the conclusions obtained in this chapter, since the problem instances generated for the explanatory study show little variation in the average number of customers per route — on average five per route — and so the findings pertain to solutions with 'short' routes.

Another extension of the analysis would be to keep track of operator performance at each iteration of a large neighbourhood search run and investigate whether the conclusions drawn at the end of the run also hold at each moment during the search for solution improvements. It might be possible that some operator performs better at the start of a run and at some point should be replaced by another one.

The next chapter returns to the exploratory analysis that initiated the explanatory study of this chapter, but looks to perform this analysis in a more efficient way by combining methodologies.

Chapter 5

A Combined Approach for Analysing Heuristic Algorithms

5.1 Introduction

Chapter 3 introduced an evaluation methodology for algorithms based on multilevel models. It allows the analysis of the algorithm elements' (i.e., parameters and components) impact on performance as well as the influence of the problem instance characteristics on these algorithm elements. The case study involved an exploratory analysis that exposed patterns in the data. This was followed by a confirmatory analysis in Chapter 4 explaining one of the exposed patterns.

In this chapter[1] (Figure 5.1), the focus shifts again to the exploratory analysis. The regression model formulated in Chapter 3 considers all algorithm parameters and components as well as all problem instance characteristics. Several interaction terms between algorithm elements are included. For example, the interaction of the destroy operators with the amount of determinism employed within these operators. The result is an extensive model with 100 variable (interaction) terms since there is no preliminary knowledge of which algorithm elements are most relevant to performance. The more extensive the regression model, the more complicated inter-

[1]This chapter is based on the paper: Corstjens, J., Dang, N., Depaire, B., Caris, A., De Causmaecker, P. in press. A combined approach for analysing heuristic algorithms. Journal of Heuristics.

pretation becomes, especially when correlated effects impact the obtained estimates. Furthermore, the more effects have to be estimated, the more time is required to fit the model. The aim for this chapter is to perform a more efficient exploratory analysis that focuses on the variable (interaction) terms that are responsible for the major variations in performance.

Previous research methodologies have already focused on evaluating the relevance of algorithm elements (Bartz-Beielstein et al., 2004; Chiarandini and Goegebeur, 2010; Fawcett and Hoos, 2015; Hutter et al., 2013, 2014; Nannen and Eiben, 2007). One of these methodologies is considered and we explore the opportunity for a complementary use with the multilevel methodology proposed in Chapter 3. First, it is investigated whether it is worthwhile to combine both approaches by verifying for the VRPTW-LNS case study performed in Chapter 3 whether observed correlations are consistent with those from the multilevel evaluation methodology. If conclusions were very different, there is little to gain from a joint methodology. Secondly, the combined analysis is compared with an individual multilevel regression analysis. The focus of this chapter therefore lies on answering the following questions. How can both methodologies be formulated in a combined methodology and what is the added value of jointly applying both approaches? Are the insights obtained from the analysis results of both approaches consistent or are there any differences observed? How do the conclusions of the joint methodology compare to the conclusions obtained when applying a multilevel regression analysis individually?

The investigated methodology is functional analysis of variance (fANOVA) by Hutter et al. (2014) since it is a lot more computationally efficient than forward selection (Hutter et al., 2013), is able to detect interaction effects, and, contrary to ablation analysis (Fawcett and Hoos, 2015), is not bound by a specific default configuration (van Rijn and Hutter, 2018). fANOVA quantifies the relative importance of the algorithm elements and their interactions according to the amount of variance in the performance data they explain, giving an indication of which effects are most important to performance. In comparison, the multilevel regression methodology explicitly separates the performance impact of algorithm elements and problem instances. It is focused on quantifying the relationship between algorithm elements and performance and how this relationship is moderated by the characteristics of a specific problem instance.

Each methodology has its own advantages. fANOVA does not rely on statistical

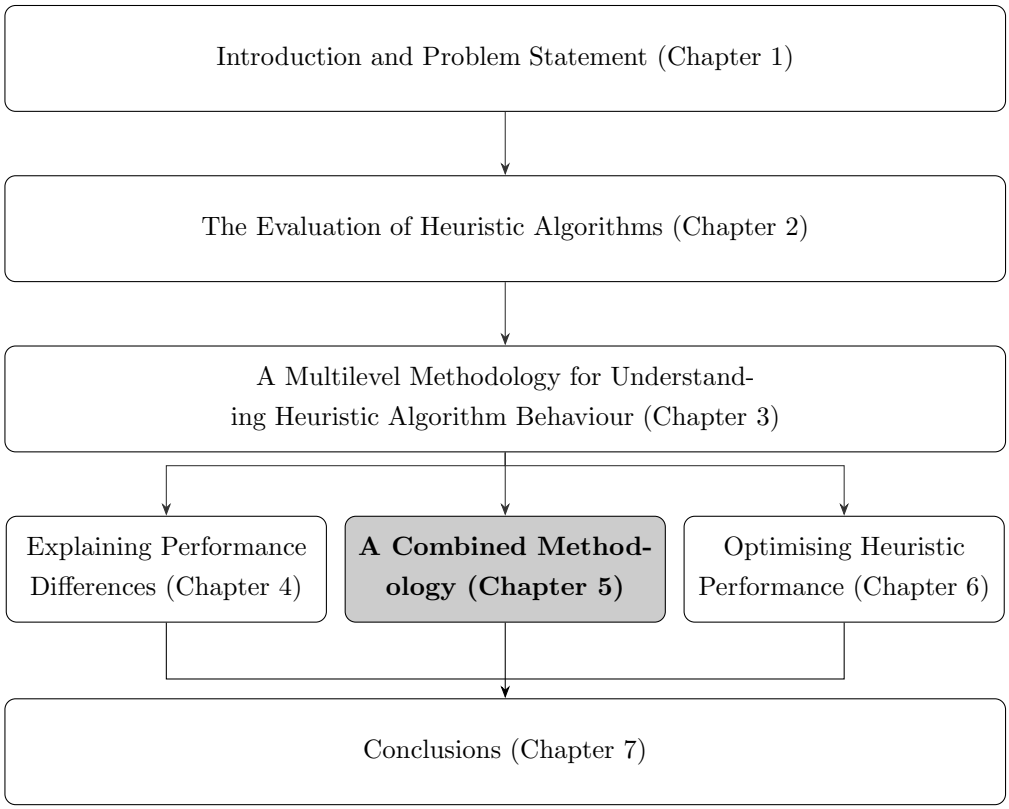


Figure 5.1: Outline of Thesis — Chapter 5.

assumptions, such as independence, normality and homoscedasticity, that are required by multilevel regression models. It is also computationally cheaper. The multilevel regression model, on the other hand, offers a more detailed analysis of the algorithm, since it can investigate algorithm element effects for a specific setting of the other elements and a specific problem instance. Moreover, the interpretation of effects in fANOVA is carried out through variance percentages and visual inspection of plots. The latter might be difficult for interaction effects. The interpretation of the regression analysis is based on quantified effects, and plots are merely used to support and visualise interpretation. The regression analysis also provides a statistical significance test to indicate whether there really is a link between the values chosen for a specific algorithm parameter and the obtained performance, or whether any observed relationships are likely due to chance.

In this chapter, both methodologies are combined by relying on the importance analysis provided by fANOVA to formulate a proper regression model for the multilevel methodology. This prevents an overly complex regression model with many variable (interactions) that contribute little to performance. This regression model can then be used for a more detailed analysis of the important effects and for confirmatory analyses with hypotheses testing. Both approaches and their combination are demonstrated on a case study in which a number of algorithm configurations for a Large Neighbourhood Search (LNS) algorithm are tested on a number of problem instances for the Vehicle Routing Problem with Time Windows (VRPTW).

The fANOVA methodology is introduced in Section 5.2. The proposed combination with the multilevel level methodology is motivated in Section 5.3. Section 5.4 details the experimental set-up for the case study, followed by the applications of fANOVA and multilevel regression analysis in Section 5.4.2. Finally, conclusions are given in Section 5.5.

5.2 fANOVA

The *fANOVA* method proposed in Hutter et al. (2014) is an approach for analysing the importance of algorithm elements on performance using a random forest prediction model and the functional analysis of variance (Hooker, 2012). The approach studies the contribution of every single element and every element interaction on the performance of the algorithm. 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 entire problem instance space. A random forest model is a collection of regression trees — i.e., decision trees with continuous values at the leaves instead of class labels — that provides more accurate predictions than individual trees and a measure of uncertainty for each predicted value. Each tree is built using a random sample of performance observations (Breiman, 2001). Random forest models have shown to perform very well for model-based optimisation in complex configuration spaces (Hutter et al., 2014).

Once the random forest prediction model is built, functional analysis of variance (Hooker, 2012) is applied on the prediction model to decompose the overall algorithm

performance variance into additive parts, each one corresponding to a subset of the algorithm elements. The ratio between the variance associated with each component and the overall performance variance is used as an indicator of the importance of the corresponding algorithm element subset. For example, the following illustrates the output fANOVA could provide when applied on Large Neighbourhood Search:

```
Sum of fractions for main effects 75.10%
Sum of fractions for pairwise interaction effects 6.26%
72.62% due to main effect: repair
1.72% due to main effect: destroy
1.60% due to interaction: repair x destroy
```

The interpretation of the first two lines is that 75.10% of the algorithm performance variance can be explained by single elements, and 6.26% by the interactions of every pair of algorithm elements. The remaining 18.64% is explained by higher-order ($\geq$3) interactions and error inherent in the model. The third line indicates that the *repair* component is the most important element, as the component itself can explain a huge part (72.62%) of the overall algorithm performance variance. The other component *destroy* and their pairwise interaction *repair* and *destroy* are less important.

In addition to the value indicating the importance of each algorithm element subset, fANOVA also provides some insights on which regions are good and bad (with a degree of uncertainty) for each element inside the subset through a marginal plot. Given a specific value for each algorithm element in the subset, the corresponding marginal prediction value is the average performance value of the algorithm over the entire configuration space associated with all elements 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. Figure 5.2 shows the marginal plot of *repair*. This categorical element has a domain of three values: *Greedy*, *Regret2* and *GreedyRegret2*, each of which is associated with a boxplot in Figure 5.2. The boxplot for *Greedy*, for example, shows the mean and the variance of the average algorithm performance values over the entire problem instance space of all algorithm configurations having *repair* = *Greedy*. The plot implies that *Regret2* is the best choice for the element *repair*.

Although fANOVA generally focuses on analysing the importance of algorithm elements, it can also be used to study the interaction between elements and problem

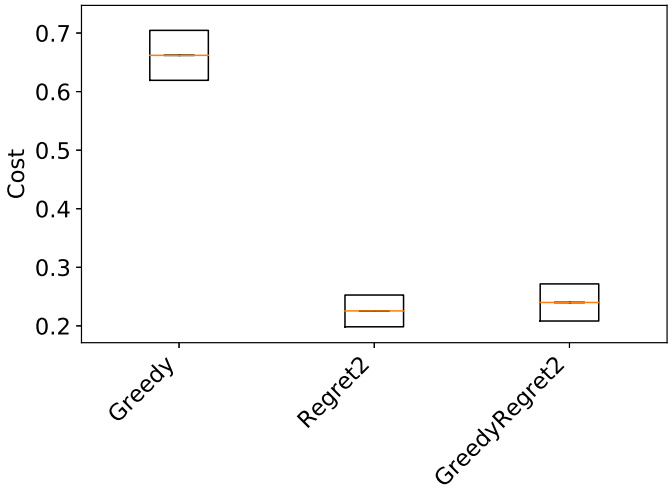


Figure 5.2: fANOVA's marginal plot for the main effect of parameter *repair*.

instance features — as we do in our case study — by simply adding those features into the prediction model of fANOVA. The features are treated exactly in the same way as the algorithm elements. It must be noted that such an integration can only be valid if the features are independent. In other words, instance features might be correlated, but their values in the problem instance set under study must be arbitrarily chosen (Hutter et al., 2013). In our case study, instances are generated by randomly sampling values from all features' domains, which satisfies this requirement of independency.

An implementation of fANOVA is provided by the authors as a Python package at `https://github.com/frank-hutter/fanova`. As a choice of implementation, the package only gives analysis results on the single and pairwise interaction effects. The higher-order (≥ 3) interactions are left out, probably due to potentially expensive computation time required and the fact that in many practical cases, single and pairwise interaction effects are usually sufficient to explain the majority of algorithm performance variance.

5.3 A Combined Methodology

A methodology is proposed that combines the use of fANOVA and multilevel regression analysis. The idea is to use the ranking of effects fANOVA provides

to formulate a multilevel regression model that includes only the most important effects. The search for a suitable regression model that includes all relevant variables can often be a cumbersome task (Gelman and Hill, 2006). The more variables and variable interactions are included, the more complex the model becomes and the more arduous it is to interpret the estimated effects. The challenge thus lies in deciding which explanatory variables and interactions to include in the model. fANOVA can provide assistance with this issue. Since the analysis gives a ranking on the importance of effects, a regression model could then be formulated using this ranking as a guideline in order to prevent an overly complex model with many variables. The regression analysis can then more easily focus on these important effects. A regression model that is less complex in terms of fewer variables and variable interactions included also implies time savings when fitting the model to the data.

The regression analysis facilitates a more detailed analysis since it provides effect estimates for a particular algorithm configuration and problem instance, while the fANOVA estimates marginal performance for a particular parameter value or component choice by averaging over all other elements and problem instance characteristics. Furthermore, the multilevel regression adds contribution on the importance analysis by calculating confidence intervals for each of the effects. This indicates which effects are actually statistically significant and which are likely due to chance. Finally, since regression models assume specific functional forms — linear or nonlinear (exponential, logarithmic, ...) —, it has the ability to extrapolate results, unlike the random forest prediction model fANOVA uses, which has an upper and lower bound for predicted values. Nonetheless, caution is advised when extrapolating results to regions in the configuration or problem instance space where there is no data to support them as the assumed trend does not necessarily hold for out-of-sample ranges (Gelman and Hill, 2006).

One might wonder whether using the two methodologies jointly affects the results obtained. If both fANOVA and multilevel regression are performed on independent data sets — like we do —, the only influence fANOVA exerts is on the choice of regression variables. This involves two risk. fANOVA can suggest variables to be important, but which do not have an effect. The inclusion of one or two variables that are unrelated to the dependent variable — also known as extraneous variables — (Osborne, 2014) is not really problematic. They might increase the imprecision of estimates, but have little effect on regression conclusions. When more extraneous variables are added, however, the presence of collinearity will be

induced (Frees, 2009). The second, more important, risk occurs when fANOVA incorrectly suggests variables to be unimportant while they do have a relevant effect on the dependent variable. The omitted variable becomes part of the error term and will be correlated with performance (i.e., the Y variable). If the omitted variable is also correlated with other variables that are included in the regression model, then the error term is no longer independent. The latter is referred to as the endogeneity problem and results in effect estimates being biased. This is not an issue in the analyses in this thesis since the random sampling used to generate the experimental design makes all included variables exogeneous (Stock and Watson, 2011).

In the next section the proposed combination of fANOVA and multilevel regression is applied on the VRPTW-LNS case study. The findings of both analyses are compared with the aim of showing their consistency.

5.4 Case Study

The experimental analysis performed here considers the case study of Chapter 3 on a large neighbourhood search algorithm solving instances for the vehicle routing problem with time windows.

5.4.1 Experimental Set-Up

A multilevel experimental design of 9000 observations is set up, consisting of 300 problem instances with 30 algorithm configurations per instance[2]. The value range of several parameters (*determinism*, *cooling rate* and *start temperature control parameter*) is widened such that any important effects that might have been missed with the reduced ranges, would show up in this analysis. An overview of all parameters and components with the adjusted value ranges is provided in Table 5.1.

[2]Two data sets were analysed, one with 4000 observations and one with 9000 observations. Increasing sample size will make estimates more precise, meaning their confidence intervals become narrower. Effects that are already significant will only become more significant. Whether or not an increased sample size will contribute much to the analysis is difficult to judge. As sample size increases, even the smallest effects become significant, but that does not make them important (Sullivan and Feinn, 2012). In this particular case, a larger sample size did not alter analysis conclusions other than adding more precision. It did require substantially more time to fit the regression models. So, it is a trade-off between increased precision and practicality. For the analysis presented here, the increased precision of estimates is preferred.

Table 5.1: LNS Parameters and Components

Parameter	**Type**	**Value ranges**
random seed	Integer	U[1, 1000000]
determinism parameter	Integer	U[1, 100]
noise parameter	Continuous	U[0, 1]
cooling rate	Continuous	U[0.01,0.99]
start temperature control parameter	Continuous	U[0.01,1]
destroy operators	Categorical	Random, Worst, Related, RandomWorst, RandomRelated, WorstRelated, RandomWorstRelated
repair operators	Categorical	Greedy, Regret2, GreedyRegret2

Two independent data sets of 9000 observations are created. One for the fANOVA and one for the multilevel regression analysis. The motivation is to prevent overfitting analysis findings to a single data set. Searching for a model that is the best fit for a single data set might risk fitting noise in the data — patterns present in the sample but not in the population — and might result in a model which performs poorly on other data points from the same population. A fitted model should be able to make accurate predictions for new data points instead of only the data points used to learn the model (Dietterich, 1995). Furthermore, a second data set also allows to gain more confidence on the effects that appear relevant in the first data set and to detect possible false positives. The latter implies that a variable (interaction) might show to have a contribution to performance in the sample data, while it does not in the population. Using a second sample reduces the risk of having false positives (Simmons et al., 2011). For these reasons, the multilevel regression analysis is performed using new sample data.

5.4.2 Analysis of Results

First, fANOVA is applied on the given algorithm performance data set. Then, a multilevel regression model is formulated based on the importance analysis provided by fANOVA, in particular all algorithm elements and problem instance characteristics having a contribution percentage value higher than 1% are included. As will be discussed, the conclusions of both approaches are consistent. However, not all effects taken from fANOVA are statistically significant according to the multilevel regression model.

5.4.2.1 Application of fANOVA

First, the cost is normalised on an instance-basis since the range of the cost values returned by the algorithm can vary among different instances.

$$p_j^c = \frac{(f_j^c - \min_{c' \in C_j,} f_j^{c'})}{(\max_{c' \in C_j,} f_j^{c'} - \min_{c' \in C_j,} f_j^{c'})} \tag{5.1}$$

where f_j^c and p_j^c are the original and normalised cost values of configuration c on problem instance j, and C_j is the set of all algorithm configurations that have been run on instance j.

The output generated by fANOVA, for the entire problem instance set, is as follows. In total, 67.91% of the performance variance is covered by main and pairwise interaction effects. The remaining 32.09% is due to higher-order interactions and error inherent in the model. The focus will be on the effects having a contribution of 1% or more. At this threshold 58.18% of the variance in performance is explained — of the 67.91% covered by fANOVA. A lower cut-off value would add little and only increase regression model complexity. On the other hand, a cut-off at, say 5%, would only explain about 45% of the performance variance and result in an uninteresting regression model that only included the repair operators and a main problem size effect, but not a single problem instance characteristic's influence on the repair operators.

```
Sum of fractions for main effects 49.44%
Sum of fractions for pairwise interaction effects 18.47%
37.54% due to main effect: repair
7.08% due to main effect: customers
4.80% due to interaction: repair x destroy
4.65% due to interaction: repair x customers
2.75% due to main effect: destroy
1.36% due to interaction: destroy x customers
0.98% due to interaction: destroy x average demand
0.81% due to main effect: average time window width
0.70% due to interaction: destroy x average time window width
0.68% due to interaction: repair x average demand
0.56% due to interaction: average time window width x average demand
0.55% due to main effect: average demand
0.47% due to interaction: repair x average time window width
0.39% due to interaction: max runtime x customers
```

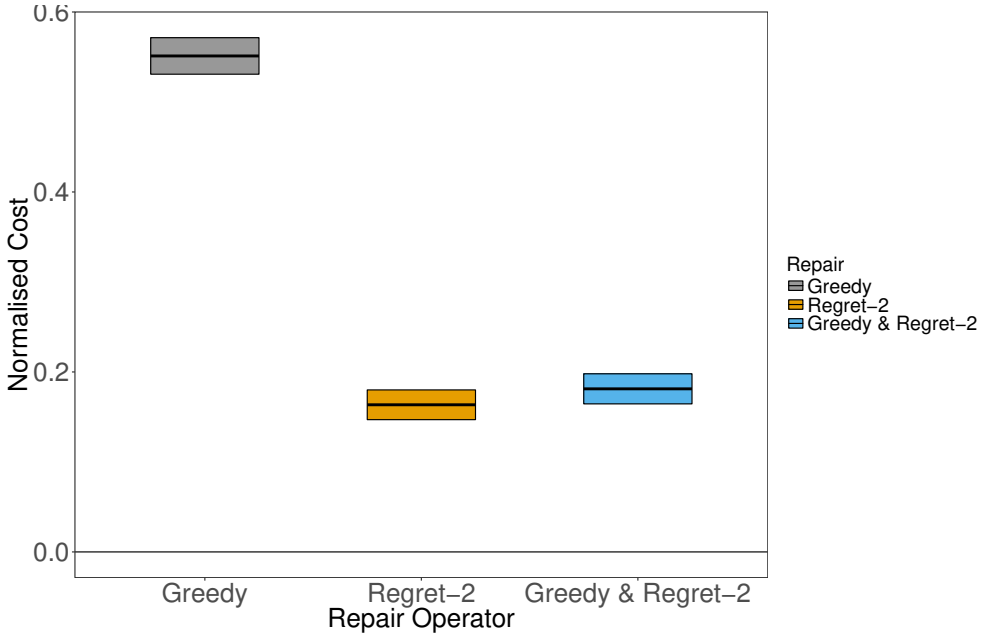
```
0.37% due to interaction: customers x average demand
0.32% due to interaction: customers x average time window width
...
```

The marginal plot for each effect is given in Figure 5.3. Since the interest is in the algorithm elements and their interactions with the problem instance characteristics, the main effect *customers* is omitted.

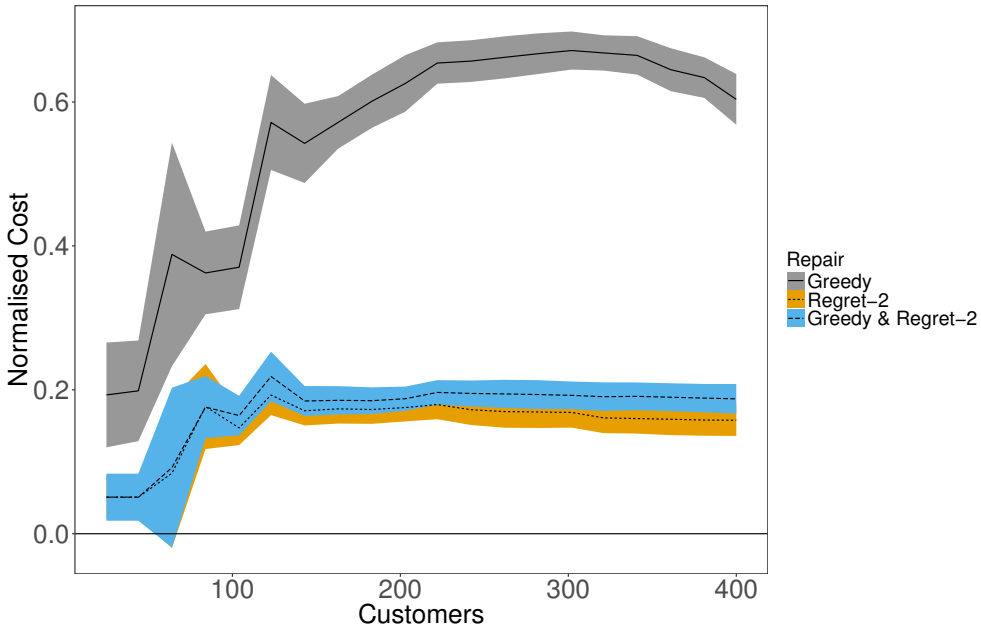
The single element *repair* explains a large part (37.54%) of the total algorithm performance variance, indicating that this element plays the most important role in the performance of the algorithm. Figure 5.3a shows that *Greedy* is clearly the worst choice. The difference between *Regret2* and *GreedyRegret2* is less clear, although *Regret2* has a slightly better normalised average performance value. How the impact of the chosen repair operator(s) changes given different problem instance sizes is explained in Figure 5.3b. The disadvantage of using the repair operator *Greedy* becomes more pronounced as the number of customers increases, especially when the number of customers is larger or equal to 100. There is no clear performance difference between the two repair operators *Regret2* and *GreedyRegret2*, but the difference does tend to increasingly grow in the advantage of *Regret2* for problem instance sizes of 200 customers or more.

The second categorical algorithm element, *destroy*, has much less importance than *repair* (2.75%). For this element, the choice of values, sorted in increasing order of marginal normalised cost values — i.e., from good to bad performance —, is as follows: *RandomRelated*, *RandomWorstRelated*, *Random*, *WorstRelated*, *RandomWorst*, *Related*, *Worst* (Figure 5.3c). The influence of different problem instance sizes on the impact of the chosen destroy operator(s) is depicted in Figure 5.3d, but this marginal plot is difficult to interpret visually.

The final marginal plot (Figure 5.3e) shows the interaction between the two categorical parameters (4.80%) and indicates consistency with the main effect observations: *Greedy* is always the worst choice, despite its combination with any destroy operator; and the choice between *Regret2* and *GreedyRegret2* is not very clear, but *Regret2* does have a slight advantage. Moreover, among all combinations of *repair* and *destroy* operators, using *Regret2* as a *repair* operator combined with the destroy operator *Random* is predicted to perform best.

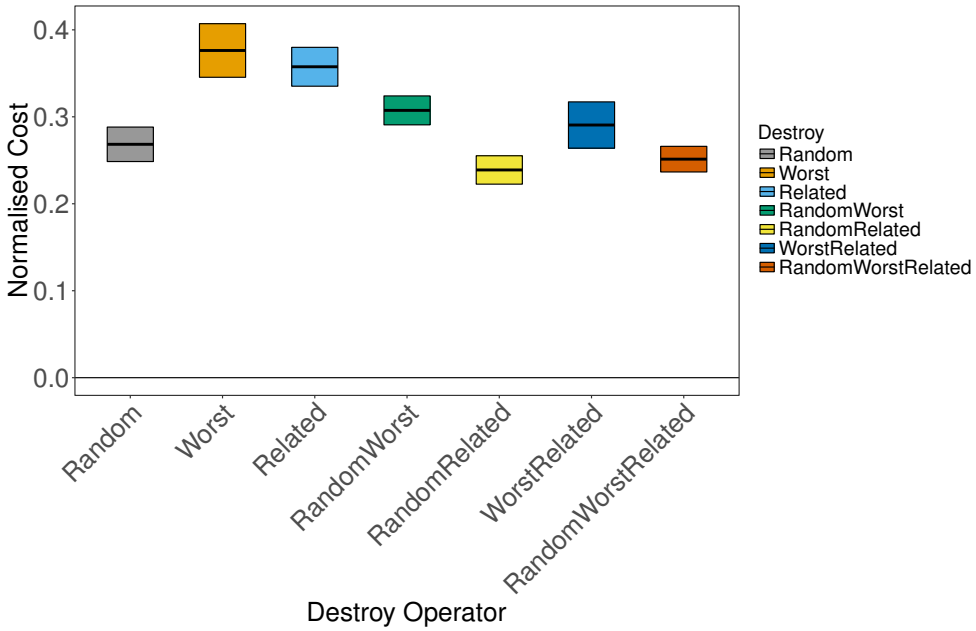


(a) Repair

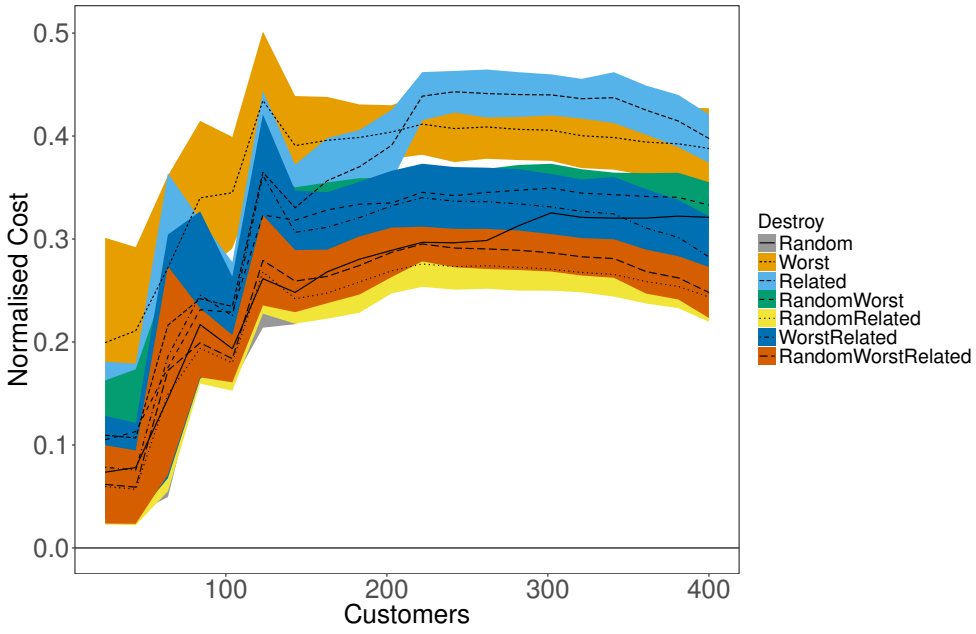


(b) Repair X Customers

Figure 5.3: Marginal plots of main and pairwise interaction effects of the fANOVA.

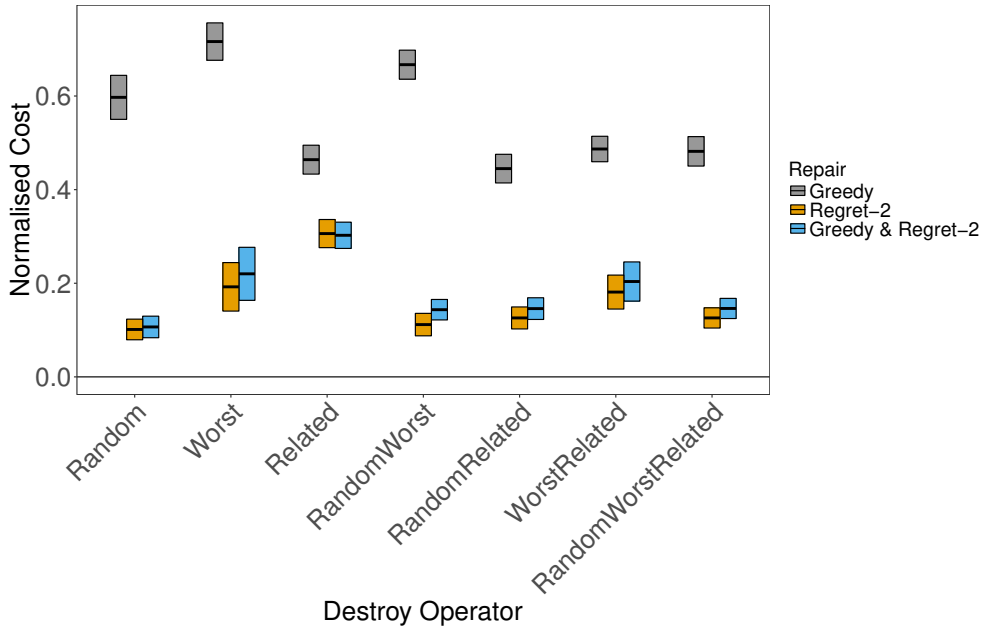


(c) Destroy



(d) Destroy X Customers

Figure 5.3: Marginal plots of main and pairwise interaction effects of the fANOVA.



(e) Repair X Destroy

Figure 5.3: Marginal plots of main and pairwise interaction effects of the fANOVA.

5.4.2.2 Application of Multilevel Regression

Based on the fANOVA output, a multilevel regression model is fitted. The resulting model in equations (5.2)-(5.4) describes a non-linear relationship between the performance measure and the algorithm elements explaining it[3].

[3]In contrast to the fANOVA, the regression model includes a binary or dummy variable for each possible combination of repair operators and each combination of destroy operators for which the impact on performance can be studied. So instead of one repair and one destroy variable, the regression model has respectively three and seven variables. Perfect multicollinearity prevents the inclusion of all repair and destroy dummies. Therefore, the configuration including all repair or destroy operators serves as a baseline configuration which is represented in the regression intercept. The effect estimates of all other operator dummies represent the change in total cost relative to the baseline.

$$\begin{aligned}\frac{1}{Y_i} =& \alpha_{j[i]} + \beta_{1j[i]} Greedy_i + \beta_{2j[i]} Regret2_i + \beta_{3j[i]} Random_i + \cdots + \\ & \beta_{8j[i]} RandomRelated_i + \beta_{9j[i]} Greedy_i \times Random_i + \cdots + \\ & \beta_{20j[i]} Regret2_i \times RandomRelated_i + \epsilon_i \end{aligned} \tag{5.2}$$

$$\alpha_j = \mu_0^{\alpha} + \mu_1^{\alpha} Customers_j^{\frac{1}{3}} + \eta_j^{\alpha} \tag{5.3}$$

$$\beta_{zj} = \mu_0^{\beta_z} + \mu_1^{\beta_z} Customers_j^{\frac{1}{3}} + \eta_j^{\beta_z} \tag{5.4}$$

All operator effects are modelled as varying effects depending on the problem instance characteristic *Customers*, as indicated by the output of fANOVA. The regression model fitted in Chapter 3 considered all algorithm element interaction terms as fixed effects. In this analysis, the interactions of destroy and repair operators are also considered as random effects. The reduced complexity offered by fANOVA allows us to more easily add such effects without making the model overly complex compared to a model that includes all algorithm elements. In the latter case, making interaction effects random would seriously complicate the regression model. Table 5.2 lists all significant effects. The complete regression table is given in Table B.5 in Appendix B. In the following paragraphs the regression findings are summarised.

Figure 5.4 shows the predicted objective function values for all repair and destroy operator configurations, all other variables fixed at their average value. Panel (a) displays the effect of switching to *Greedy* while panel (b) shows the impact of switching to *Regret2*. In agreement with the analysis in Chapter 3, the predictions show that using *Regret2* as the sole repair operator is expected to give the best performance results for all destroy operators it is combined with, while relying only on greedy repair is expected to give the worst results for all destroy operator combinations. It is also observed that the relative performance of the destroy operators per individual repair operator differs. The way a solution is destroyed has an impact on how good *Greedy* or *Regret2* is at repairing this solution. *Greedy* seems to have more difficulty in repairing a solution from which customers were removed randomly while *Regret2* is better able to cope with such a situation. *Regret2*, however, appears to find it more difficult to repair a solution from which related customers were removed — with relatedness interpreted in terms of distance as in Pisinger and Ropke (2007). These insights, again confirming the analysis of Chapter 3, spark a new research challenge to discover why certain operator combinations perform (relatively) different (cf. Chapter 4).

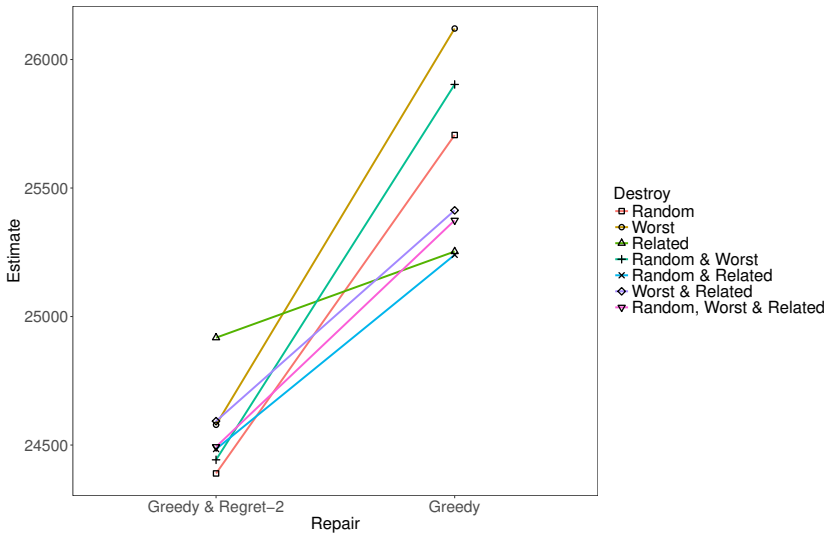
Table 5.2: Regression Table of significant effects[a]

Variable	Estimate	Est.Error	l-95% CI	u-95% CI
Intercept[b,c]	4,082.85	122.62	3,879.03	4,302.29
Customers$^{\frac{1}{3}}$	−428.11	26.96	−476.64	−373.45
Greedy	−141.97	4.39	−149.85	−134.36
Customers$^{\frac{1}{3}}$	−16.63	0.97	−18.46	−14.74
Regret2	12.67	2.46	7.81	17.51
Customers$^{\frac{1}{3}}$	1.71	0.57	0.60	2.80
Random	16.81	2.41	12.12	21.59
Customers$^{\frac{1}{3}}$	2.73	0.56	1.64	3.83
Worst	−13.90	2.45	−18.78	−9.20
Related	−69.67	3.16	−75.53	−63.84
Customers$^{\frac{1}{3}}$	−11.17	0.73	−12.59	−9.76
RandomWorst	7.96	2.36	3.34	12.61
Customers$^{\frac{1}{3}}$	1.80	0.57	0.70	2.90
WorstRelated	−16.26	2.43	−21.05	−11.48
Customers$^{\frac{1}{3}}$	−2.30	0.57	−3.40	−1.19
Greedy × Random	−67.38	5.26	−77.47	−57.11
Customers$^{\frac{1}{3}}$	−11.01	1.20	−13.34	−8.63
Greedy × Worst	−98.09	6.36	−110.17	−85.91
Greedy × Related	88.16	4.37	79.91	96.40
Customers$^{\frac{1}{3}}$	13.50	1.00	11.55	15.43
Greedy × RandomWorst	−87.72	5.54	−98.11	−77.07
Customers$^{\frac{1}{3}}$	−8.14	1.27	−10.59	−5.63
Greedy × WorstRelated	10.27	3.94	2.54	17.94
Customers$^{\frac{1}{3}}$	3.04	0.92	1.25	4.84
Greedy × RandomRelated	20.17	3.91	12.53	27.83
Customers$^{\frac{1}{3}}$	1.87	0.89	0.10	3.61
Regret2 × Related	−9.13	3.88	−16.58	−1.39
Customers$^{\frac{1}{3}}$	−2.13	0.91	−3.91	−0.32

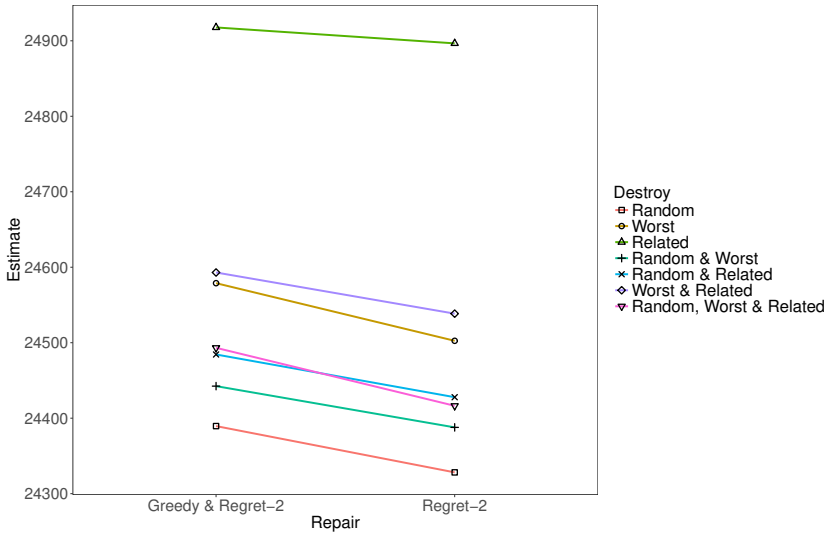
[a] Since the reciprocal transformation of the response variable returned very small values, causing difficulties in the sampling procedure of the *brms* package, all transformed (response) values were multiplied by 100 000 000.

[b] The effects of Greedy & Regret-2 and Random, Worst & Related, the reference levels for the repair and destroy operator dummies, are accounted for in the Intercept.

[c] The Intercept value is backtransformed to the original scale through division by 100 000 000 and taking the inverse of the resulting value.



(a)



(b)

Figure 5.4: Predicted total cost for an average problem instance (216 customers).

The influence of the number of customers is investigated by calculating the marginal effects. The calculation of these marginal effects is previously discussed in Section 3.4.5.1 in Chapter 3. In this analysis, the single new element in the marginal effect is the influence of *Customers* on the interaction between destroy and repair op-

erators. For example, the marginal impact on performance for *Greedy* with *Random* is derived as follows.

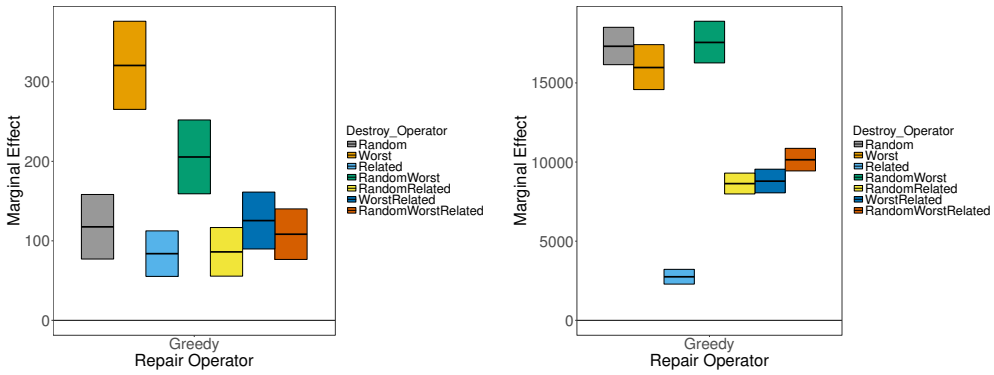
$$Y_i = (\alpha_{j[i]} + \beta_{1j[i]} Greedy_i + \cdots + \beta_{20j[i]} Regret2_i \times RandomRelated_i)^{-1} \tag{5.5}$$

$$\frac{\partial Y}{\partial Greedy} = -\frac{\mu_0^{\beta_1} + \mu_1^{\beta_1} Customers_j^{\frac{1}{3}} + (\mu_0^{\beta_2} + \mu_1^{\beta_2} Customers_j^{\frac{1}{3}})}{(\mu_0^{\alpha} + \mu_1^{\alpha} Customers_j^{\frac{1}{3}} + (\mu_0^{\beta_1} + \mu_1^{\beta_1} Customers_j^{\frac{1}{3}}) + (\mu_0^{\beta_2} + \mu_1^{\beta_2} Customers_j^{\frac{1}{3}}))^2} \tag{5.6}$$

$$\frac{\partial Y}{\partial Greedy} = -\frac{-141.97 - 16.63 Customers_j^{\frac{1}{3}} - 67.38 - 11.01 Customers_j^{\frac{1}{3}}}{(4082.85 - (141.97 + 67.38) - (428.11 + 16.63 + 11.01) Customers_j^{\frac{1}{3}})^2} \tag{5.7}$$

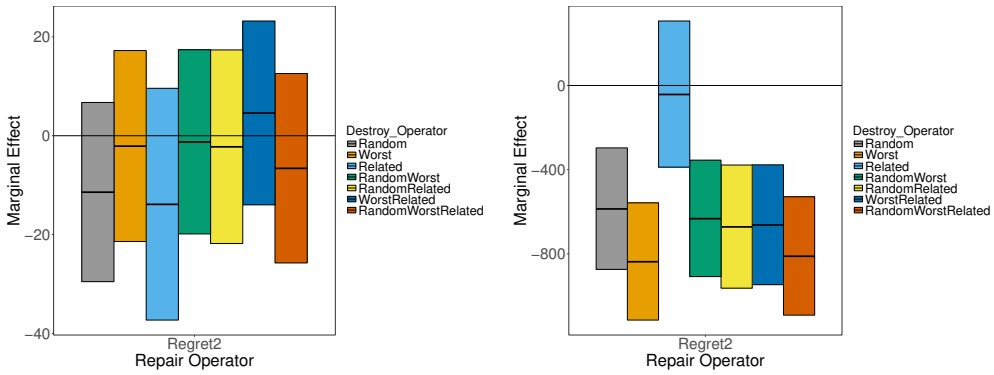
The problem instance size influence on the interaction effect is interpreted as follows: The combination greedy and random tends to perform increasingly worse than the combination of greedy with all destroy operators as more and more customers have to be served. Figures 5.5 and 5.6 show the marginal effect of *Greedy* and *Regret2* for the smallest (a) and largest (b) instance size for all destroy operator combinations. Note the different scale on the vertical axis between panels (a) and (b). Similarly, the influence of *Customers* on the effect of switching from *RandomWorstRelated* to any other (set of) destroy operator(s) can be investigated. Figures 5.7 and 5.8 show the marginal effects of the destroy operators for a problem instance with (a) 25 and (b) 400 customers. These plots have the same layout as Figures 5.5 and 5.6, but show effects from the perspective of the destroy operators rather than the repair operators.

Summarising the observations, regression results suggest to avoid relying only on greedy repair, as this is expected to give the worst results in all considered conditions. Concerning the sole use of regret-2, different conclusions are drawn for smaller and larger instances, due to the significant influence of the number of customers an instance has to serve. On the smallest instances, no significant improvement is observed over using both repair operators (Figure 5.6a). Furthermore, most of the performance differences between the various destroy operator configurations cannot be distinguished from each other as well, meaning the choice of destroy operators is irrelevant for these problem instance sizes. Only the combination with worst removal is indicated to perform significantly worse than the scenario with all destroy operators (Figure 5.7a). On the larger instances, performance differences are more pronounced. The threshold problem instance size at which it becomes beneficial to use regret-2 alone — i.e., where the effect of *Regret2* turns significantly positive because the 95%



(a) 25 customers (b) 400 customers

Figure 5.5: Marginal effect of Greedy for (a) 25 and (b) 400 customers.

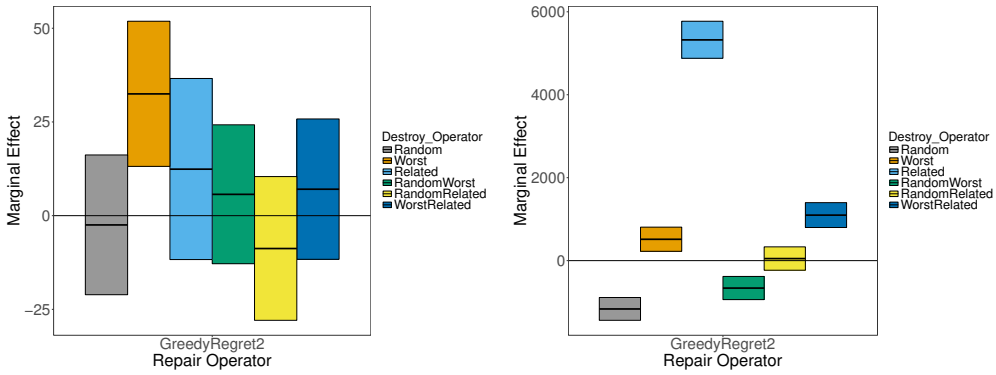


(a) 25 customers (b) 400 customers

Figure 5.6: Marginal effect of Regret-2 for (a) 25 and (b) 400 customers.

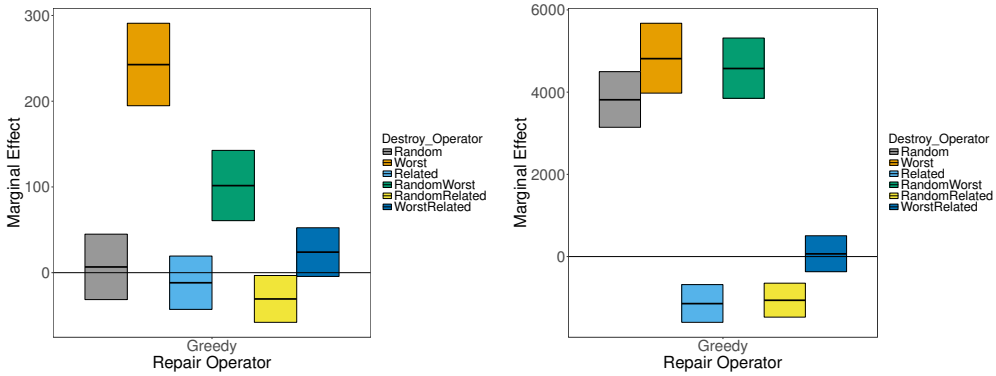
confidence interval no longer includes zero — is around 169 customers. Regret-2 performs significantly better for all destroy operator combinations, except for the combination with related removal (Figure 5.6b). The performance of regret-2 with related removal cannot be distinguished from the performance of both repair operators with related removal. Random removal is the preferred combination to use with regret-2 (Figure 5.7b).

Comparing the regression analysis to the findings of fANOVA, conclusions are consistent. On the larger problem instances, both approaches find that using



(a) 25 customers (b) 400 customers

Figure 5.7: Marginal effect of destroy operators (with GreedyRegret2 or Regret2) for (a) 25 and (b) 400 customers.



(a) 25 customers (b) 400 customers

Figure 5.8: Marginal effect of destroy operators (with Greedy) for (a) 25 and (b) 400 customers.

regret-2 alone combined with random removal is expected to perform best. The regression analysis is more clear about this observation than the fANOVA. For the smaller problem instances, there is consistency in that there is no clear performance difference between using either regret-2 alone or together with greedy. Concerning the destroy operators, the regression analysis cannot identify any (combination of) destroy operator(s) as the preferred one to use since their performances cannot be distinguished from each other. This is also observed in the marginal plot *Destroy*

x *Customers* provided by fANOVA, as there is no clear difference in performance between destroy operators on different problem sizes. Therefore, results are again consistent in both analyses.

Furthermore, the observations on the problem instance size influence in both analyses are deduced from different effects. In the fANOVA results, the 2-way interaction between operator(s) and *Customers* is considered an important effect. The multilevel regression analysis, however, also analyses the 3-way interactions between repair and destroy operators and *Customers*, an effect that the fANOVA tool does not take into account. Note that the fANOVA method itself can analyse any high-order interactions. This is just the implementation choice of the used fANOVA tool not to provide higher-order interactions. The observed consistencies make the regression analysis more robust, since these findings are confirmed by a methodology (fANOVA) which does not rely on the statistical assumptions of independence, normality and homoscedasticity of the error terms.

In addition, the regression model facilitates a more detailed analysis, since it provides effect estimates for a particular algorithm configuration and problem instance, while fANOVA estimates marginal performance for a particular parameter value by averaging over all other parameters and problem instance characteristics. The regression results are able to identify for each combination of repair and destroy operators an instance size interval for which a significant difference in performance is expected. For example, combined with both repair operators *Random* is expected to outperform *Worst* for 53 customers or more, *Related* for 91 customers or more, *RandomWorst* for 217 customers or more, *WorstRelated* for 154 customers or more, *RandomRelated* for 212 customers or more, and *RandomWorstRelated* for 161 customers or more.

Finally, the formulated regression model would have been different if fANOVA had not been performed in advance. A multilevel regression model would have been fitted that included all algorithm parameters and components and all problem-level predictors, since there would have been no prior knowledge on which elements have an important or significant impact on performance. Furthermore, parameter interactions would have been included as well. In short, this more extensive model would have had to estimate substantially more effects than the simpler model based on fANOVA. A possible extensive model was fitted (see Table B.6 in Appendix B) to illustrate our case.

First of all, the time required to fit the model is considerably longer for the extended model (about 100 hours) compared to the simple model (about 50 hours plus 12 hours for the fANOVA analysis)[4]. Then, comparing the significant effects of both models, it is observed that the effects significant in the simple model are also significant in the extended model. Furthermore, no large value changes are observed for these estimates of both models. This shows that the simple model does not miss any important variable which might bias the effect estimates, an issue known in statistics as 'omitted variable bias' (Stock and Watson, 2011).

Studying the predictive influence of every problem instance characteristic in the extended model, it can be concluded that the problem instance size is the most influential problem instance characteristic, as changing this factor leads to large changes in the total cost values. The other problem instance characteristics show influence as well for particular operator combinations, but the performance change they bring about is of much smaller magnitude than is the case for varying problem instance size values. Therefore, fANOVA understandably denoted the problem instance size as the most important problem instance characteristic. Furthermore, the predictions for varying problem instance sizes are almost the same in both models, so the larger model does not provide additional information that alters these predictions. However, it does provide additional information on operator behaviour for other problem instance characteristics. For example, for greedy repair, wider time windows increasingly worsen the solution quality with all destroy operators, except related removal, for which the deterioration becomes smaller. In conclusion, we believe the regression model based on fANOVA is a sufficiently detailed model that provides insight into the effects related to the largest shifts in algorithm performance.

5.5 Conclusions

This chapter showed how the multilevel methodology can focus on the algorithm elements and problem instance characteristics that are most relevant to performance. This focus is achieved by performing a functional analysis of variance (fANOVA) before fitting the multilevel regression model. The ranking of effects fANOVA provides will lead to a more concise regression model with less predictor variables. The multilevel methodology, on the other hand, provides a more detailed analysis of

[4]For the data set with 4000 scenarios the extended model required 44 hours compared to 22 plus 3.5 hours for the regression model based on fANOVA

the effects of algorithm elements and enables confirmatory analyses to be performed.

The two methodologies are applied on independent data sets drawn from the same algorithm element and problem instance characteristic distributions, thereby avoiding "overfitting" analysis findings. Experimental results on a case study for a large neighbourhood search algorithm applied on instances of the vehicle routing problem with time windows have shown to be consistent for both the fANOVA and the multilevel regression analysis, thereby making the regression analysis more robust. Furthermore, the regression analysis can help to give additional insights on the analysis results provided by fANOVA.

Chapter 6

Instance-Specific Optimisation of Algorithm Performance

6.1 Introduction

In Chapter 3 a multilevel regression analysis is performed to expose how algorithm parameters and components are correlated to performance and how the problem instance characteristics are correlated with these algorithm elements. This analysis was a starting point for the confirmatory analysis performed in Chapter 4 to explain one of the exposed correlations. This is one application of the multilevel methodology: observing patterns in the data and searching to explain them in order to gain insight and understanding of how a heuristic algorithm behaves on a specific problem instance. This emphasis on understanding has been the main focus of this thesis thus far. The regression analysis is not only useful for confirmatory testing, the exposed correlations can also be used to decide which components to include and which values to use for the various parameters such that performance is optimised.

In this chapter (Figure 6.1), the potential of applying the multilevel methodology in the context of automatic algorithm configuration is explored. For many years, values for the, often numerous, parameters employed within metaheuristic frameworks were determined in an ad-hoc way relying on personal experience and

intuition. This manual tuning of parameters is time-intensive and biased. Therefore, automatic algorithm configurators have been developed, providing a formal procedure to determine parameter values. The aim is to find an algorithm configuration that optimises some performance measure over a set of instances (Birattari, 2009). The obtained configuration(s) will perform well on average across all instances, but might not be the best one for each individual instance. The methodology introduced in Chapter 3 takes into account the influence of the problem instance characteristics on the performance impact of the algorithm parameters and components. Therefore, this methodology can provide an algorithm configuration that is predicted to perform best on average, but is additionally able to exploit the per-instance performance variation and specify an algorithm configuration for each individual problem instance.

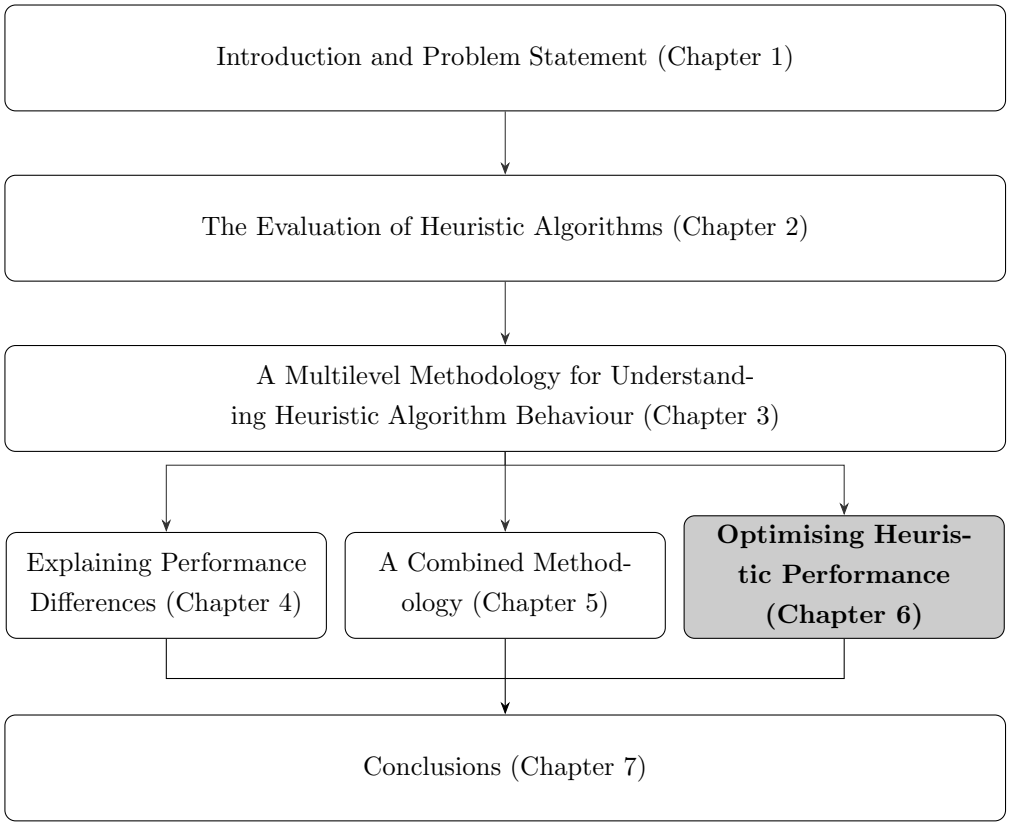


Figure 6.1: Outline of Thesis — Chapter 6.

This chapter starts with a refresher on automatic algorithm configuration and pre-

vious research work that has already incorporated problem instance characteristics when choosing appropriate parameter values (Section 6.2). The approach towards automatic algorithm configuration presented here derives decision rules from the multilevel regression model for each of the parameters and components (Section 6.3) which are used to determine optimal values. In order to verify whether it is worthwhile to choose an instance-specific configuration over an instance-oblivious configuration, an experiment is set up to compare both approaches to automatic algorithm configuration (Section 6.4). Experimental results are discussed in Section 6.5. Finally, conclusions are given in Section 6.6.

6.2 Automatic Algorithm Configuration

Section 2.2.3 discussed the algorithm configuration problem and the automated methods that have been developed to solve it. Birattari (2002) formalised the problem: Given an algorithm with a parameter space Θ, a set of problem instances I and a performance metric $C(\theta)$ measuring the performance of the algorithm on problem instance set I given configuration θ, the solution to the configuration problem is the configuration θ^* such that:

$$\theta^* = \operatorname*{argmin}_{\theta} C(\theta) \tag{6.1}$$

Rasku et al. (2014) compare seven state-of-the-art algorithm configuration methods on a number of vehicle routing metaheuristics and found the model-based configurator *irace* to be the most robust method. The considered approaches are CMA-ES, GGA, iterated F-Race, ParamILS, REVAC, SMAC, and Uniform Random Sampling (URS).

Irace relies on racing, which has its foundation in machine learning where it is used for solving model selection problems. Birattari (2002) was the first to apply the concept to the problem of selecting an algorithm configuration from a set of candidate configurations. The idea behind racing procedures is to sequentially evaluate candidate configurations on a set of problem instances and eliminate them from further consideration as soon as they become too inferior to the best performing candidate at a certain stage. More specifically, all candidate configurations solve one or multiple problem instance(s), after which statistically worse performing configurations are discarded and the procedure continues with the remaining configurations until a minimum number of remaining configurations, a maximum number of problem instances, or the specified maximum number of algorithm runs has been reached. Birattari

(2002) named their approach F-Race since they apply the Friedman two-way analysis of variance by ranks test to determine whether a candidate configuration should be discarded or not. Elimination of inferior configurations enables speeding up the procedure and increasing the reliability of the promising configurations as they can be evaluated on more problem instances. An iterative application of the procedure (I/F-Race) is proposed by Balaprakash et al. (2007) and consists of three steps. First, configurations are sampled by drawing values for each configurable parameter from independent probability distributions, a truncated normal distribution for numerical parameters and a discrete one for categorical parameters. At initialisation, configurations are uniformly sampled from the parameter space. In a second step racing is applied to identify the best performing candidate from the sampled configurations. Finally, these best candidates are used to update the probability distributions and, in this way, bias the sampling towards the best performing configurations. The iterated racing procedure is implemented in the R package *irace* by López-Ibáñez et al. (2016).

Automatic algorithm configurator methods deliver a single configuration that is expected to perform best on average across all problem instances. In 1976, Rice recognized that there is not a single best algorithm to solve a particular type of problem. Algorithms for NP-Hard problems have a lot of variation in run times and solution quality (Shukla et al., 2013). Therefore, different algorithms perform best on different problem instances. The problem of choosing the best-performing algorithm for a particular problem instance is formulated in the algorithm selection problem. The most common approach to tackle this problem had a "winner-take-all" strategy that implied running different algorithms on a given problem distribution and use only the best-performing one. This approach, however, has the drawback that it ignores algorithms that might do better than the average performance for a specific problem instance and are uncompetitive on all others (Leyton-Brown et al., 2003; Shukla et al., 2013). The focus has shifted from doing per-distribution algorithm selection to per-instance algorithm selection. This is achieved using prediction models that, based on the characteristics of a specific problem instance, predict the performance of each algorithm in a portfolio of algorithms and select the one that is expected to perform best (Leyton-Brown et al., 2003).

These portfolios typically consider a small set of fundamentally different algorithms to choose from and, therefore, ignore algorithm element choices. They only cover a single configuration of a heuristic algorithm with possibly numerous parameter and component choices. The multilevel methodology also relies on

predictions to make configuration decisions, but incorporates both the algorithm elements and problem instance characteristics. Hence, it can consider all possible combinations of parameter values and components.

The algorithm selection problem has inspired other per-instance approaches. Hutter et al. (2006) employ empirical hardness models to automatically tune algorithms for each problem instance to be solved. A function $g(x, c)$ is learned using ridge regression that takes as input the problem instance characteristics x of a problem instance, and the configuration c of an algorithm. The function estimates statistics (e.g., mean or median) of the algorithm's run time distribution given the problem instance and configuration combination. The configuration for which the function predicts the lowest median runtime on a specific problem instance is chosen. Average speed ups over the fixed best parameter settings of two stochastic local search algorithms (Novelty$^+$ and SAPS) are observed for a mixed SAT problem benchmark set containing both unstructured and structured problem instances.

Xu et al. (2010) introduced a method called *Hydra* that combines portfolio-based algorithm selection with automatic algorithm configuration. In a first step, an automatic algorithm configurator is applied to identify a single configuration that performs best on average across all (training) problem instances. In each of the consecutive iterations, it adds one configuration and, in such a way, assembles a portfolio of algorithm configurations. Configurations can also be dropped from the portfolio if they no longer contribute to the performance of the portfolio. The problem instance characteristics are only used to select the configuration from the portfolio that is predicted to perform best. The method is applied, using ParamILS as configurator and SATzilla as portfolio builder, on SAT benchmarks and is shown to outperform 17 state-of-the-art SAT solvers.

Kadioglu et al. (2010) present an automatic algorithm configurator called Instance-Specific Algorithm Configuration (ISAC), an integration of the Gender-Based Genetic Algorithm (GGA)(Ansótegui et al., 2009) and stochastic offline programming (Malitsky and Sellmann, 2010). They assume that problem instances with similar characteristics can be solved equally well using the same algorithm configuration. Therefore, they cluster problem instances in distinct instance sets based on similarity in the normalised problem instance characteristics. A performance-optimising algorithm configuration is then determined for each cluster using GGA. ISAC is evaluated on five high-performance solvers (a greedy set covering heuristic,

the dialectic search algorithm Hegel, Tabu Search, Cplex and SAPS) and three different problems (set covering, mixed integer programming and satistfiability). Performance improvements over the default configurations are found as well as over the GGA configurator.

The next section shows how the results of a multilevel regression analysis can be used to tune a heuristic algorithm given the specific problem instance to be solved.

6.3 Algorithm Optimisation with Multilevel Regression

Optimal parameter values and component choices are determined for the corresponding regression variables that show to have a significant (at 5%) impact on performance. The insignificant algorithm elements get assigned a random value. A configuration can be derived that is predicted to perform best for an average problem instance, i.e. fixing the value for all centred problem instance characteristics at zero. This corresponds to an instance-oblivious configuration, similar to the one provided by automatic algorithm configurators such as *irace*. If the problem instance characteristics are allowed to vary, instance-specific configurations can be obtained. Only significant problem instance characteristic influences are taken into account when making tuning decisions regarding parameters and components. For example, consider the following linear effect estimate for continuous parameter Z, significantly moderated by problem instance characteristics X_1 and X_2.

$$Y = \alpha + \beta Z \tag{6.2}$$

$$\beta = \mu_0^\beta + \mu_1^\beta X_1 + \mu_2^\beta X_2 \tag{6.3}$$

The decision rule for this effect is very straightforward. Given a minimisation problem, if $\beta < 0$, select the largest value for Z in the considered range of values as this will reduce the performance measure by the largest amount. If $\beta > 0$, choose the smallest value. Parameters, however, often do not have a linear performance impact, but might show to have a quadratic effect as well as significantly interact with other parameters or components. If a quadratic term is added, the model changes to equations (6.4) and (6.5), assuming the effect estimate for the quadratic term is a fixed effect and, hence, is not influenced by the problem instance characteristics.

$$Y = \alpha + \beta_1 Z + \beta_2 Z^2 \tag{6.4}$$
$$\beta_1 = \mu_0^{\beta_1} + \mu_1^{\beta_1} X_1 + \mu_2^{\beta_1} X_2 \tag{6.5}$$

In this case, the decision regarding which parameter value to select can no longer be based solely on the β_1 estimate, but the complete term $(\mu_0^{\beta_1}+\mu_1^{\beta_1}X_1+\mu_2^{\beta_1}X_2)Z+\beta_2 Z^2$ has to be optimised. The optimal Z value is obtained by looking at the marginal effect of Z. This marginal effect is calculated by taking the partial derivative of equation (6.4). The Z value at which this equation equals zero will be the optimal value for Z.

$$\frac{\partial Y}{\partial Z} = \mu_0^{\beta_1} + \mu_1^{\beta_1} X_1 + \mu_2^{\beta_1} X_2 + 2\beta_2 Z = 0 \tag{6.6}$$
$$Z = -\frac{\mu_0^{\beta_1} + \mu_1^{\beta_1} X_1 + \mu_2^{\beta_1} X_2}{2\beta_2} \tag{6.7}$$

Whether this optimum is a maximum or a minimum can be verified by calculating the second derivative $\frac{\partial^2 Y}{\partial Z^2} = 2\beta_2$. If the second derivative > 0, the optimum is a minimum and is used as the best parameter value. However, if this value is not part of the range of parameter values, the value that is closest to the optimum is chosen. If the second derivative < 0, the optimum is a maximum — a value you want to avoid given a minimisation objective — and the upper and lower bound of the parameter's value range are considered. The bound that has the most favourable impact on performance is chosen.

If the parameter additionally significantly interacts with a different parameter W, calculations are similar, but which value for Z optimises performance now depends on the value of W.

$$Y = \alpha + \beta_1 Z + \beta_2 Z^2 + \beta_3 Z \times W \tag{6.8}$$
$$\beta_1 = \mu_0^{\beta_1} + \mu_1^{\beta_1} X_1 + \mu_2^{\beta_1} X_2 \tag{6.9}$$
$$\beta_3 = \mu_0^{\beta_3} + \mu_1^{\beta_3} X_1 + \mu_2^{\beta_3} X_2 \tag{6.10}$$

The marginal effect is derived in equation (6.11) and the optimal value in equation (6.12).

$$\frac{\partial Y}{\partial Z} = \mu_0^{\beta_1} + \mu_1^{\beta_1} X_1 + \mu_2^{\beta_1} X_2 + 2\beta_2 Z + (\mu_0^{\beta_3} + \mu_1^{\beta_3} X_1 + \mu_2^{\beta_3} X_2) \cdot \times W \tag{6.11}$$
$$Z = -\frac{\mu_0^{\beta_1} + \mu_1^{\beta_1} X_1 + \mu_2^{\beta_1} X_2 + (\mu_0^{\beta_3} + \mu_1^{\beta_3} X_1 + \mu_2^{\beta_3} X_2) \cdot \times W}{2\beta_2} \tag{6.12}$$

6.4 Experimental Set-Up

Similar to the previous chapters, the case study in this chapter considers a large neighbourhood search algorithm applied on the vehicle routing problem with time windows. The parameters to be tuned include real-valued parameters such as the cooling rate, start temperature control parameter, noise parameter, ... as well as binary variables that determine which destroy and repair operators to activate. In all previous experiments, the number of customers to remove each iteration is determined at random in the range 10% to 50% of all customers to be served. In this experiment, the upper bound for this range is no longer fixed, but incorporated as a parameter to be tuned. A plenary talk by Stefan Röpke — one of the authors of ALNS — at the 2016 VeRoLog conference noted that the amount of destruction might be the most crucial parameter (Röpke, 2016). Consequently, the upper bound on the amount of destruction is added as a parameter for the tuning study. Table 6.1 lists the updated set of parameters and components with their value ranges. The problem instance characteristics that might be included in the decision rules derived from the multilevel regression analysis are the number of customers to be served, how long service at each customer takes on average, the average width of customer time windows, the average demand per customer and the maximum run time given to solve the problem instance.

Three different sets of problem instances are generated to be used for training the regression model. One set contains 100 randomly generated VRPTW instances, a second set 200 instances and a final set has 400 instances. An experimental design is set up for every problem instance set. Each problem instance is to be solved by 50 randomly generated parameter settings. The resulting experimental designs have respectively 5000, 10 000 and 20 000 scenarios for which the LNS returns a total distance travelled value.

The obtained configurations are tested on a set of 2000 randomly generated VRPTW instances. Each test instance will be solved by the suggested configuration for an average problem instance as well as a customised configuration the regression model proposes given the values for the problem instance characteristics. In addition, the best configuration obtained by the configurator tool *irace* for each set of training instances will be applied on the set of test instances.

All experiments are performed using Intel Xeon E5-2680v2 CPUs (2.8 GHz, 25 MB level 3 cache) under Red Hat Enterprise Linux ComputeNode release 6.4 (San-

Table 6.1: Algorithm Parameters and Components to be Tuned

Algorithm Element	**Type**	**Value ranges**
Determinism Parameter	Integer	U [1, 30]
Noise Parameter	Discrete	U [0, 1]
Cooling Rate	Continuous	U [0.80,0.99]
Start Temperature Control Parameter	Discrete	U [0.01,0.10]
Maximum percentage removed	Continuous	U [0.10, 0.50]
Destroy Operators		U [1,7]
1. Random Removal	Dummy	[0,1]
2. Worst Removal	Dummy	[0,1]
3. Related Removal	Dummy	[0,1]
4. Random and Worst Removal	Dummy	[0,1]
5. Random and Related Removal	Dummy	[0,1]
6. Worst and Related Removal	Dummy	[0,1]
7. Random, Worst and Related Removal	Dummy	[0,1]
Repair Operators		U [1,3]
1. Greedy	Dummy	[0,1]
2. Regret-2	Dummy	[0,1]
3. Greedy and Regret-2	Dummy	[0,1]

tiago), 64 bit. These resources are available from the infrastructure of the Flemish Supercomputer Center (`www.vscentrum.be`).

6.5 Discussion

In this section the decision rules derived from the multilevel regression analysis (Section 6.5.1) are presented, as well as the output of the *irace* runs (Section 6.5.2). The evaluation of the configurations on the test instances is discussed in Section 6.5.3.

6.5.1 Multilevel Decision Rules

After collecting performance values for the 5000, 10 000 and 20 000 combinations of problem instances and algorithm configurations, a multilevel regression model is fitted. The model in equations (6.13) to (6.15) is similar to the model in equations (3.6) to (3.8) (cf. Chapter 3). The variable *Max%Removed* is added as a random effect, as well as its interaction with individual operator dummies and its 3-way interaction

with both destroy and repair dummies. Additionally, a quadratic term is added for all real-valued parameters in order to expose any curvature that might be present in the parameter's effect on the objective function value.

$$\begin{aligned}
\frac{1}{Y_i} &= \alpha_{j[i]} + \beta_{1j[i]} Greedy_i + \beta_{2j[i]} Regret2_i + \beta_{3j[i]} Random_i + \\
&\quad \beta_{4j[i]} Worst_i + \beta_{5j[i]} Related_i + \beta_{6j[i]} RandomWorst_i + \cdots + \\
&\quad \beta_{9i} Greedy_i \times Random_i + \cdots + \beta_{20i} Regret2_i \times RandomRelated_i + \\
&\quad \beta_{21j[i]} Max\%Removed_i + \beta_{22} Max\%Removed_i^2 + \cdots + \beta_{29[i]} Noise_i + \\
&\quad \beta_{30} Noise_i^2 + \beta_{31i} Random_i \times Determinism_i + \cdots + \beta_{37i} Greedy_i \times Noise_i + \\
&\quad \beta_{38i} Regret2_i \times Noise_i + \beta_{39i} Random_i \times Max\%Removed_i + \cdots + \\
&\quad \beta_{47i} Greedy_i \times Random_i \times Max\%Removed_i + \cdots + \\
&\quad \beta_{58i} Regret2_i \times RandomRelated_i \times Max\%Removed_i + \epsilon_i \qquad (6.13) \\
\alpha_j &= \mu_0^\alpha + \mu_1^\alpha Customers_j^{\frac{1}{3}} + \cdots + \mu_5^\alpha Runtime_j + \eta_j^\alpha \qquad (6.14) \\
\beta_{kj} &= \mu_0^{\beta_k} + \mu_1^{\beta_k} Customers_j^{\frac{1}{3}} + \cdots + \mu_5^{\beta_k} Runtime_j + \eta_j^{\beta_k} \qquad \forall k \in K \qquad (6.15)
\end{aligned}$$

The effects of this model that show to have a significant impact on performance are used to determine what parameter value or set of operators to use, given an average problem instance or given a specific problem instance. In the next paragraphs, the decision process is illustrated for the model that is fitted on the training set with 200 problem instances. Table 6.2 lists the significant effect estimates for this training set. All complete regression tables are provided in Appendix B. Tuning decisions are made using the transformed effect estimates to prevent overcomplicated derivative calculations and since decisions do not depend on the scale used.

Table 6.2: Significant effects (at 5%) for training data with 200 instances

Variable	Estimate	Est.Error	l-95% CI	u-95% CI
Intercept[a,b]	4,481.95	139.19	4,212.48	4,760.17
Customers$^{\frac{1}{3}}$	−479.46	31.53	−543.12	−418.23
Avg service time	−71.36	33.82	−138.64	−5.37
Greedy	−122.60	3.87	−130.25	−115.04
Customers$^{\frac{1}{3}}$	−18.70	0.67	−20.02	−17.39
Avg time window width	−2.08	0.46	−2.98	−1.19
Runtime	1.53	0.48	0.57	2.46

Regret2	12.39	2.62	7.29	17.52
Customers$^{\frac{1}{3}}$	1.68	0.23	1.24	2.13
Runtime	−0.44	0.17	−0.76	−0.11
Random	8.86	2.67	3.67	14.13
Customers$^{\frac{1}{3}}$	0.81	0.36	0.11	1.53
Avg demand	−3.74	1.73	−7.11	−0.31
Worst	−22.86	3.05	−28.85	−16.80
Customers$^{\frac{1}{3}}$	−2.07	0.50	−3.05	−1.09
Related	−21.35	2.78	−26.82	−15.95
Customers$^{\frac{1}{3}}$	−4.14	0.39	−4.90	−3.38
Runtime	0.78	0.28	0.22	1.34
RandomWorst^{c}	−4.19	2.82	−9.72	1.27
Avg demand	−4.08	2.01	−8.00	−0.14
WorstRelated	−7.10	2.63	−12.24	−1.98
Customers$^{\frac{1}{3}}$	−1.57	0.35	−2.26	−0.89
Avg demand	−3.67	1.66	−6.92	−0.41
RandomRelated	7.20	2.66	1.98	12.47
Max%Removed	−0.03	0.16	−0.35	0.29
Customers$^{\frac{1}{3}}$	−0.19	0.01	−0.22	−0.17
Avg time window width	−0.02	0.01	−0.03	−0.002
Runtime	0.03	0.01	0.01	0.04
Cooling rate	21.70	7.79	6.21	36.84
Customers$^{\frac{1}{3}}$	3.83	1.77	0.38	7.30
Noise parameter	−11.19	2.59	−16.28	−6.09
Customers$^{\frac{1}{3}}$	−2.04	0.36	−2.76	−1.34
Determinism parameter	0.27	0.13	0.02	0.51
Customers$^{\frac{1}{3}}$	−0.05	0.01	−0.07	−0.03
Runtime	0.02	0.01	0.0004	0.03
Max%Removed^{2}	−0.01	0.003	−0.02	−0.01
Noise parameter^{2}	26.27	5.50	15.51	37.08
Greedy×Random	−26.45	3.77	−33.85	−19.17
Greedy×Worst	−58.73	3.79	−66.20	−51.38
Greedy×Related	37.26	3.76	29.87	44.59
Greedy×RandomWorst	−40.84	3.75	−48.19	−33.52
Greedy×RandomRelated	16.63	3.73	9.35	23.93
Regret2×Random	−7.65	3.68	−14.87	−0.44
Random×Determinism parameter	−0.38	0.17	−0.71	−0.04
Worst×Determinism parameter	−1.01	0.18	−1.35	−0.65

Related×Determinism parameter	−1.37	0.17	−1.71	−1.02
Greedy×Noise parameter	−25.75	3.52	−32.63	−18.91
Random×Max%Removed	0.46	0.22	0.02	0.89
Worst×Max%Removed	0.93	0.23	0.49	1.38
Related×Max%Removed	−1.37	0.23	−1.81	−0.93
RandomWorst×Max%Removed	0.76	0.23	0.32	1.21
Greedy×Max%Removed	−1.96	0.23	−2.41	−1.51
Greedy×Random×Max%Removed	−1.78	0.33	−2.42	−1.14
Greedy×Worst×Max%Removed	−0.85	0.33	−1.50	−0.21
Greedy×Related×Max%Removed	1.05	0.33	0.41	1.69
Greedy×RandomWorst×Max%Removed	−1.60	0.32	−2.23	−0.96

Note a: The effects of Greedy & Regret-2 and Random, Worst & Related, the reference levels for the operator dummies, are accounted for in the Intercept.
Note b: The Intercept value is backtransformed to the original scale through division by 100 000 000 and taking the inverse of the resulting value.
Note c: The estimate is insignificant and will be interpreted as having an effect of 0, which the average demand can influence.

First, the decision on which destroy and repair operators to use is made. The marginal effects are calculated to quantify the impact on performance when switching from a configuration that uses all destroy and all repair operators to one that uses, for example, only random removal and one that uses only regret-2, ceteris paribus. The equations below show the marginal effect on Y^{-1} for switching to random removal (equation (6.16)) or switching to regret-2 as sole repair operator (equation (6.17)).

$$\begin{aligned} \frac{\partial Y^{-1}}{\partial Random} &= \mu_0^{\beta_3} + \mu_1^{\beta_3} Customers^{\frac{1}{3}} + \mu_4^{\beta_3} Demand \\ &= 8.86 + 0.81 Customers^{\frac{1}{3}} - 3.74 Demand \qquad (6.16) \\ \frac{\partial Y^{-1}}{\partial Regret2} &= \mu_0^{\beta_2} + \mu_1^{\beta_2} Customers^{\frac{1}{3}} + \mu_5^{\beta_2} Runtime \\ &= 12.39 + 1.68 Customers^{\frac{1}{3}} - 0.44 Runtime \qquad (6.17) \end{aligned}$$

The performance impact when changing both the destroy and repair operator at the same time is calculated by summing these marginal effects and adding the interaction effect — if significant — for $Regret2 \times Random$ (-7.65). This is done for every combination of destroy and repair operators and the combination that delivers the largest positive performance impact is chosen. If no positive impact is found, this implies the configuration with all destroy and repair operators is predicted to

perform best and should be chosen.

Next, optimal values for the real-valued parameters are determined. The effect the parameter *cooling rate* has on Y^{-1} is formulated in equation (6.18). If its marginal effect is positive, *cooling rate* is fixed at 99%, otherwise at 80%. There is not enough evidence in the data to indicate a significant influence of the *start temperature control parameter* and, therefore, it gets assigned a random value in the range $[0.01, 0.1]$.

$$\begin{aligned} \frac{\partial Y^{-1}}{\partial CoolingRate} &= \mu_0^{\beta_{23}} + \mu_1^{\beta_{23}} Customers^{\frac{1}{3}} \\ &= 21.70 + 3.83 Customers^{\frac{1}{3}} \end{aligned} \tag{6.18}$$

The *determinism parameter* significantly interacts with the destroy dummies *Random*, *Worst* and *Related*. The marginal effect with *Related* is formulated in equation (6.19). The dot in the interaction term refers to the *Determinism* variable this equation is differentiated to.

$$\begin{aligned} \frac{\partial Y^{-1}}{\partial Determinism} &= \beta_{27_{j[i]}} + \beta_{33} Related \times \cdot \\ &= \mu_0^{\beta_{27}} + \mu_1^{\beta_{27}} Customers^{\frac{1}{3}} + \mu_5^{\beta_{27}} Runtime + \\ &\quad \beta_{33} Related \times \cdot \\ &= 0.27 - 0.05 Customers^{\frac{1}{3}} + 0.02 Runtime - \\ &\quad 1.37 Related \times \cdot \end{aligned} \tag{6.19}$$

If the marginal effect is positive, set determinism at 30 and rely on the ranking related removal established to select customers to remove. In the other case, when the marginal effect is negative, choose a determinism value of 1 and randomise customer selection. The *noise parameter* and *maximum percentage removed* both have a significant quadratic term. Equation (6.20) shows the marginal effect for *Noise* and its optimal value in (6.21). As explained in Section 6.3, it needs to be verified whether this optimum is a minimum or maximum. Note that due to the inverse transformation of the Y variable, the objective has switched from minimisation on the original scale to maximisation on the transformed scale. If the optimum is a maximum and lies within the value range, the *noise parameter* is fixed at this value. Otherwise, the boundary value (i.e., 0 or 1) closest to the optimum is chosen. If the optimum is a minimum, the boundary value with the most favourable impact is chosen. The

second derivative is 52.54, a positive number and, therefore, the found optimum is a minimum. This implies that one of the boundary values will be chosen.

$$\begin{aligned}
\frac{\partial Y^{-1}}{\partial Noise} &= \beta_{29j[i]} + 2\beta_{30} Noise \\
&= \mu_0^{\beta_{29}} + \mu_1^{\beta_{29}} Customers^{\frac{1}{3}} + 2\beta_{30} Noise \\
&= -11.19 - 2.04 Customers^{\frac{1}{3}} + 52.54 Noise \qquad (6.20) \\
Noise &= -\frac{-11.19 - 2.04 Customers^{\frac{1}{3}}}{52.54} \qquad (6.21)
\end{aligned}$$

The optimal maximum percentage of customers to remove is determined in a similar way. Equation (6.21) shows the marginal effect if both related removal and greedy repair are used. The second derivative -0.02 indicates a maximum and the optimal value for this parameter is therefore determined by equation (6.22) if within the value ranges.

$$\begin{aligned}
\frac{\partial Y^{-1}}{\partial Max\%Removed} &= \beta_{21j[i]} + 2\beta_{22} Max\%Removed + \beta_{45} Related \times \cdot + \\
&\quad \beta_{45} Greedy \times \cdot + \beta_{49} Greedy \times Related \times \cdot \\
&= \mu_0^{\beta_{21}} + \mu_1^{\beta_{21}} Customers^{\frac{1}{3}} + \mu_3^{\beta_{21}} TW\,Width + \mu_5^{\beta_{21}} Runtime + \\
&\quad 2\beta_{22} Max\%Removed + \beta_{45} Related \times \cdot + \beta_{45} Greedy \times \cdot + \\
&\quad \beta_{49} Greedy \times Related \times \cdot \\
&= -0.03 - 0.19 Customers^{\frac{1}{3}} - 0.02 TW\,Width + 0.03 Runtime - \\
&\quad 0.02 Max\%Removed - 1.37 Related \times \cdot - 1.96 Greedy \times \cdot + \\
&\quad 1.05 Greedy \times Related \times \cdot \qquad (6.22) \\
Max\%Removed &= -\frac{-0.03 - 0.19 Customers^{\frac{1}{3}} + \cdots + 1.05 Greedy \times Related \times \cdot}{-0.02} \qquad (6.23)
\end{aligned}$$

These decision rules are used to determine algorithm configurations for the 2000 test problem instances generated. If the influence of the significant problem instance characteristics is ignored, a single configuration is suggested for all instances. So, in the decision rules above the variables for the problem instance characteristics are all fixed at zero. Recall that these variables are centred around their mean value so that a value of zero implies the average value for the associated problem instance characteristics. The suggested configuration from the regression model will therefore be the predicted optimal one for an average problem instance. Table 6.3 indicates for each training data set the suggested best configuration for an average problem instance. Note that the larger sample size, the less destroy operators are used.

Table 6.3: Suggested best configuration for an average problem instance

Sample size	Determinism	Noise	Max perc removed	Cooling rate	Start temp control	Destroy	Repair
5000	U[1,30]	0	0.29	U[0.80,0.99]	U[0.01,0.1]	Random, Worst & Related	Regret-2
10 000	30	0	0.28	0.99	U[0.01,0.1]	Random & Related	Regret-2
20 000	1	0	0.41	0.99	U[0.01,0.1]	Random	Regret-2

Table 6.4: Operator Choices based on Multilevel Regression Analysis for Training Data with 100 Instances

	Greedy	Regret-2	Greedy & Regret-2
Random	0	418	0
Worst	0	0	0
Related	186	0	0
Random & Worst	0	0	0
Random & Related	0	942	0
Worst & Related	0	10	0
Random, Worst & Related	0	454	0

If instance characteristics are incorporated in the decision rules, like in the previous paragraphs, each of the 2000 test problem instances will have its own customised configuration. Table 6.5 indicates how frequent each combination of operators is chosen. The dominant operator choice is the combination of random and related removal with regret-2. For 246 problem instances, related removal is combined with greedy repair. These are most often the smaller problem instances, having less than 100 customers. Regarding the parameters, the *noise parameter* is assigned a value of 1 in 36 cases, all having 30 customers or less. All 246 choices for related removal have a determinism parameter value of 1. The majority of the configurations including multiple destroy operators set this parameter value at 30. The *cooling rate* is fixed at 99% for all test problem instances. The parameter *maximum percentage removed* is set at many different values, so it is difficult to make statements about the instance-specific tuning of this parameter just by looking at the values.

Tables 6.4 and 6.6 provide operator frequencies for tuning on the training sets with respectively 100 and 400 problem instances.

Table 6.5: Operator Choices based on Multilevel Regression Analysis for Training Data with 200 Instances

	Greedy	Regret-2	Greedy & Regret-2
Random	0	212	18
Worst	0	0	0
Related	246	0	0
Random & Worst	0	18	0
Random & Related	0	1495	1
Worst & Related	0	10	0
Random, Worst & Related	0	0	0

Table 6.6: Operator Choices based on Multilevel Regression Analysis for Training Data with 400 Instances

	Greedy	Regret-2	Greedy & Regret-2
Random	0	1639	51
Worst	0	0	0
Related	310	0	0
Random & Worst	0	0	0
Random & Related	0	0	0
Worst & Related	0	0	0
Random, Worst & Related	0	0	0

6.5.2 Output irace

The *irace* configurator tool identifies four elite configurations[1]. They are listed in Tables 6.7, 6.8 and 6.9 from best to worst according to the sum of ranks. Inspection of these elite configurations show somewhat different recommendations regarding the way customers are to be removed. Given 100 training instances, the configurator tool suggests removing customers at random works best. Additionally, this removal strategy can be combined with a strategy that selects customers based on a relatedness criterion. The experiment with 400 instances similarly indicates the combination of random and related removal to be the best choice of destroy operators in three out of the four elite configurations. On the set of 200 instances, however, *irace* selected the combination worst and related removal to perform best on average. The latter contradicts the findings in Chapter 3 that indicated this combination of destroy operators to be one of the worst performing ones. But that analysis investigated

[1] A tuning budget of 1000 algorithm runs is assigned.

Table 6.7: Elite *irace* configurations given 100 training instances

Ranking	Determinism	Noise	Max perc removed	Cooling rate	Start temp control	Destroy	Repair	Mean solution quality
1	26	0.33	0.37	0.98	0.02	Random	Regret-2	29 225.06
2	28	0.07	0.38	0.87	0.07	Random & Related	Regret-2	29 284.01
3	23	0.46	0.31	1.00	0.01	Random	Regret-2	29 273.76
4	11	0.45	0.21	0.85	0.01	Random & Related	Regret-2	29 250.89

operator effects in isolation, varying only a single problem instance characteristic.

The output tables show that the *determinism parameter* gets assigned a lower and lower value for increasing training problem sets, implying that the established ranking worst and related removal make, should be relaxed. This makes sense when random removal is not part of the suggested set of destroy operators to introduce some randomness in the customer selection procedure, but it seems odd when random removal is included, like on the 400-instance training set. It raises the question, by introducing a lot of randomness in the customer selection related removal makes in addition to the randomness produced by random removal, why it would be even worthwhile to use related removal. If randomness is so beneficial to performance, why not rely solely on random removal to destroy a solution? This observation contradicts the findings of Chapter 3: when multiple destroy operators are used, complete randomisation of customer selection should be left to random removal.

Tables 6.7 and 6.9 both recommend to set the maximum on the number of customers to remove each iteration at around one third. Table 6.8, on the other hand, indicates a value around 15%.

All elite configurations suggest to repair a solution using regret-2 as the sole repair operator. This is in line with the conclusions of Chapter 3. What amount of noise to include in the repair phase is less clear as the values for the *noise parameter* differ substantially between the configurations, but they do suggest not to include too much noise since most values are well below 50%. Finally, the elite values for the parameters operating within the local search framework simulated annealing suggest a *cooling rate* around 98% for the 200-instance training set and around 90% for the 400-instance training set. On the smallest training set, elite *cooling rate* values vary substantially.

Table 6.8: Elite *irace* configurations given 200 training instances

Ranking	Determinism	Noise	Max perc removed	Cooling rate	Start temp control	Destroy	Repair	Mean solution quality
1	5	0.06	0.14	0.98	0.03	Worst & Related	Regret-2	24 718.38
2	14	0.02	0.18	0.94	0.03	Worst & Related	Regret-2	24 804.46
3	6	0.16	0.14	0.98	0.02	Worst & Related	Regret-2	24 830.11
4	5	0.04	0.15	0.99	0.02	Random & Related	Regret-2	24 772.76

Table 6.9: Elite *irace* configurations given 400 training instances

Ranking	Determinism	Noise	Max perc removed	Cooling rate	Start temp control	Destroy	Repair	Mean solution quality
1	4	0.33	0.39	0.89	0.06	Random & Related	Regret-2	29 672.81
2	6	0.17	0.28	0.93	0.06	Random & Related	Regret-2	29 662.48
3	1	0.11	0.38	0.93	0.03	Related	Regret-2	29 682.38
4	4	0.04	0.28	0.92	0.04	Random & Related	Regret-2	29 684.76

In most cases, a low *start temperature control parameter* value around 2 or 3% is suggested to perform best.

6.5.3 Evaluation of Configuration Performance

In this section the tuning capabilities of the multilevel methodology on the test problem set are evaluated. First, the performance of the configuration for an average problem instance is assessed by comparing it to the performance of the best *irace* configuration, both ignoring the problem instance specificities. Table 6.10 lists for each training data set the number of times the multilevel regression configuration performs better than the best *irace* configuration and vice versa, or if both configurations performed equally well.

The frequencies lead to different conclusions for the three training data sets. For the smallest sample of 100 training instances, they show the best *irace* configuration performing better on more test problem instances than the multilevel 'average instance' configuration. The difference is small though, as indicated by both the frequencies as the absolute and relative average performance differences. For the two larger training sets, the multilevel configuration for an average problem instance is

Table 6.10: Instance-Oblivious Tuning Performance on 2000 Test Instances

	100 training instances	200 training instances	400 training instances
Multilevel regression configuration performs better	758	1061	1159
Best *irace* configuration performs better	830	581	368
Same performance	412	358	473
Average performance gap (absolute)[a]	16.51	-87.47	-224.10
Average performance gap (%)[b]	0.08	-0.25	-0.52

[a] Calculated as the difference of the performance of the multilevel regression configuration and the performance of the best *irace* configuration.

[b] Calculated as the absolute difference divided by the performance of the best *irace* configuration.

able to obtain a better performance than the configuration suggested by *irace* for the majority of test instances. The absolute and relative differences do show that the gap in solution quality is small, but it is a significant[2] difference. Hence, multilevel tuning performs better for larger data sets, which makes sense as these samples contain more information the regression model can use to make a recommendation. Note that the worst results are obtained when including all destroy operators, while tuning performance excels when using only random removal. Part of the explanation might be the number of iterations the configuration can run. Random removal, being the simplest destroy operator, requires less computation time per iteration and, therefore, the multilevel configuration is able to perform significantly more iterations than the *irace* configuration which applies both random and related removal, given the same maximum run time. On the other hand, one would expect the logic behind an operator like related removal to be able to reach the same solution quality as a simpler operator in less iterations. But, as the analyses of Chapter 4 showed, operators do not always function as expected.

The choice of operators is not the only difference in both configurations. Other parameter choices diverge as well and might contribute to the 'average instance' configuration performing better. The *irace* configuration suggests to apply some noise (33%) during the repair phase and intensifies the search more quickly due to the higher cooling rate (89%), while the regression analysis indicates no noise should

[2]Significance tested using the Wilcoxon signed rank test.

be implemented and chooses the slowest cooling schedule. Furthermore, the search behaviour of *irace* itself is controlled by parameters. For the experiments in this chapter, these parameter are kept at their default values, but performance could be improved by tuning *irace* itself (Dang et al., 2017). The configurator is also given a tuning budget of 1000 algorithm runs, considerably less than the training data sets used to fit the regression model. Nevertheless, *irace* applies guided sampling when choosing configurations to test and should, therefore, need less algorithm run evaluations to make a recommendation on the set of components and parameter values to use.

The previous has shown that the multilevel methodology is able to produce an algorithm configuration for an average problem instance that performs in line with the configuration of an established tuning method as *irace*. A next step is to analyse whether it is worthwhile to consider instance-specific configurations in this case study. Table 6.11 compares the performance of the customised configurations with the configuration for an average problem instance. It shows the instance-specific configuration performing best for the smallest training sample. The difference with the 'average' configuration is not significant though. Similar to the comparison with the *irace* configuration, the 'average instance' configuration significantly outperforms the customised configurations for the two larger training samples and this for the majority of test instances. Hence, these findings show no real benefit of customising the algorithm configuration per individual problem instance.

Analysing the frequencies per destroy and repair configuration shows the instance-specific tuning clearly 'underperforming' for the combination of related removal with greedy repair (Figure 6.2). For the 100-instance training set, the regression analysis suggested this operator combination for 186 test instances. In only 8 instances an improvement is observed over the 'average instance' configuration, which performs better for 138 test instances — the remaining 40 showing an equal performance. For the 200- and 400-training set, the observations are similar for this combination of operators: respectively 12 and 24 improvements against 173 and 184 deteriorations compared to the 'average instance' configuration. For all other operator combinations, such an extreme underperformance is not observed.

Recall from the exploratory analyses of Chapters 3 and 5 that related removal is the preferred destroy operator to use with greedy repair, but that relying only on greedy repair is not recommended. These observations were made looking at

Table 6.11: Instance-Specific Tuning Performance on 2000 Test Instances

	100 training instances	200 training instances	400 training instances
Instance-specific configuration performs better	846	543	464
Best configuration for average instance performs better	848	1130	1206
Same performance	306	327	330
Average performance gap (absolute)[a]	-6.07	147	182.98
Average performance gap (%)[b]	0.04	0.42	0.52

[a] Calculated as the difference of the objective function value obtained by the instance-specific configuration and the value obtained by of the 'average instance' configuration.

[b] Calculated as the absolute difference divided by the performance of the 'average instance' configuration.

effects in isolation and varying only a single problem instance characteristic. When accounting for all significant problem instance influences at the same time, the instance-specific tuning suggests to only use greedy repair for 10-15% of the test instance set. The majority of these instances need to serve less than 100 customers. Analysing some individual instances shows that this operator combination benefits from the large positive interaction effect $Greedy \times Related$ (37.26) that makes the combined operator effect positive. Similar to the analysis in Chapter 3, the interaction effects between destroy and repair operator are considered as fixed effects, meaning their performance impact is not influenced by the problem instance characteristics. Perhaps the treatment of these interaction terms as fixed effects incorrectly predicts greedy as sole repair operator to perform well for these problem instances. For this reason, a model is fitted that considers these interaction terms as random effects — as in Chapter 5. The results for this model fitted on the 200-instance training data set showed a drastic decrease in the choice for greedy repair. It is the suggested repair operator for only 29 test instances. However, overall instance-specific tuning performance is worse compared to the model of equations (6.13) to (6.15). Therefore, this model has been discarded.

Thus, instance-specific tuning does not seem worthwhile to apply in this case study. Still, past research — for example, Rice (1976) and Smith-Miles et al. (2014) — has acknowledged the no-free-lunch theorem by Wolpert and Macready (1997) that there is not a single best algorithm for all problem instances. Possible

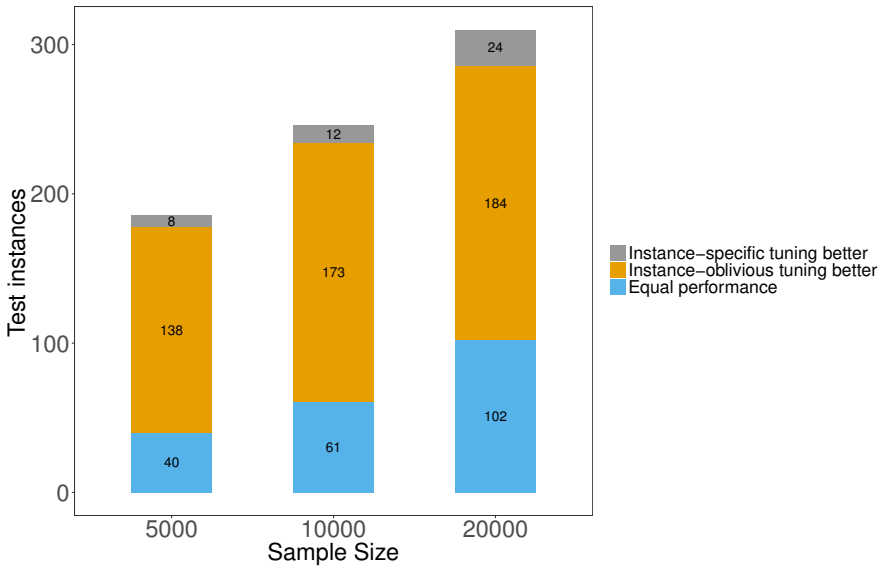


Figure 6.2: Frequencies of performance for configurations using related removal and greedy repair.

explanations why performance differences are so small relate to the diversity of the problem instances and the problem instance characteristics. It is possible that the generated VRPTW instances are quite homogeneous, allowing the instance-oblivious configuration to perform well. In terms of problem size, I believe instances show sufficient diversity, but perhaps the other problem instance characteristics do not bring about a clear performance distinction. The details on the sample instances generated for the analysis in Chapter 3 (see Appendix A) suggest, for example, given the uniformly distributed demand values and fixed vehicle capacity, the average number of customers per route is rather small and varies little. The same goes for the samples used in the experiments of this chapter. A second possible explanation might be that the analysis lacks problem instance characteristics that are distinctive for performance between instances. The choice was made to only incorporate those problem instance characteristics that are determined at random from some probability distribution, but other VRP specific features can be included. For example, descriptors of the distance matrix (lowest, highest, average, standard deviation edge cost, ...), the fraction of distinct distances, the ratio of total demand to total capacity or the depot location (Rasku et al., 2016). These are worthwhile ideas to pursue to show the benefit of instance-specific tuning.

This chapter demonstrates how the multilevel regression analysis can be applied in the context of automatic algorithm configuration using the significant effect estimates and that the obtained algorithm configurations result in similar solution quality as the configurator *irace*.

6.6 Conclusion

This chapter explored the potential of applying the multilevel methodology — presented in Chapter 3 of this thesis — to choose well-performing algorithm components and values for the algorithm parameters. It is possible to obtain a single best configuration to be used on all problem instances as well as customised algorithm configurations for each individual problem instance. The latter is possible since the multilevel methodology accounts for the problem instance characteristics in the tuning process. It is shown how decision rules can be extracted from the regression analysis for each algorithm parameter and component to determine which value or component to use given a specific problem instance. The decision rules are formulated using the significant problem instance influences for these algorithm elements.

An experimental study aimed at identifying optimal algorithm configurations for a large neighbourhood search algorithm solving instances of the vehicle routing problem with time windows. First, the performance of a configuration for an average problem instance is compared with the performance of a similarly instance-oblivious configuration provided by the automatic algorithm configurator *irace*. Three different sample sizes were considered to train the tuning approaches on. Results showed the 'average instance' configuration performing in line with the best configuration suggested by *irace*. It performs increasingly better for larger training samples. So the more data, the better multilevel tuning performs. Generating large data sets can, however, be impractical in terms of computational cost.

Secondly, it is investigated whether it is worthwhile to consider instance-specific configurations obtained from the regression analysis. Their performance is compared with the 'average instance' configuration. Results indicate instance-specific tuning does not improve upon the performance of the 'average instance' configuration. It shows that the LNS elements responsible for the major performance variations, like the repair operators, cannot benefit from instance-specific tuning since one single

repair configuration is the recommended choice for all problem instances.

Another consideration to make is how diverse the set of problem instances are on which you want to apply instance-specific tuning. If an instance set is quite homogeneous in terms of algorithm configurations that perform best, there is little to gain from an instance-specific tuning approach and a single general configuration can perform equally well.

The multilevel tuning approach has the benefit of comprehensibility when providing algorithm configurations. It presents a motivation for why a specific configuration is chosen since the regression model makes the relationship between parameter and performance explicit, thereby helping to understand tuning decisions. Automatic configurators such as *irace* are black-box approaches and do not generally motivate their configuration choices. Elite configurations can have widely varying values for the same parameter, thereby lacking any insight into the algorithm element's impact on performance. In addition, finding the multilevel decision rules is a one-time effort to be made and can be used for all problem instances of the same population. Black-box configurators, on the other hand, need to be re-run for each new problem instance sample.

Finally, the approach applied to decide on optimal parameter values and component choices can be considered naive since the decision rules for the operators do not consider the interaction of the destroy operators with the determinism parameter, the repair operators with the noise parameter and the interaction of all operators with the maximum percentage of customers that can be removed. Hence, the optimisation of the operator configuration ignores these parameters. Simultaneously optimising both the operator and parameter choices is rather complex. It may be best to consider all categorical variables first and make the calculations for every possible combination of operators, and then proceed with the continuous variables by optimising their value choices for every possible categorical scenario. Pick the scenario delivering the largest positive impact on performance. In this way both the choice of operators and parameters values are optimised simultaneously.

Chapter 7

Conclusions

This doctoral thesis promoted an approach to heuristic experimentation that is focused on gaining insight and understanding of how heuristic performance is established. This stands in contrast to the common competitive approach emphasising a heuristic method that can outperform other methods. Chapter 2 reviewed this manner of experimentation with heuristic algorithms and the issues regarding the generalisation of results obtained this way. Chapter 3 proposed a methodology based on the concepts of Design of Experiments to rigorously evaluate heuristic algorithms and Chapter 4 showed how it could be used to better understand observed performance differences. This methodology is combined with another evaluation methodology in Chapter 5 in order to perform a more efficient exploratory analysis that focuses on the effects that are most relevant to performance. Chapter 6 explored the potential of the methodology to choose well-performing parameter values and heuristic components. Finally, in this chapter (Figure 7.1) general conclusions and opportunities for further research are presented.

7.1 Final Conclusions

Since the introduction of the vehicle routing problem in 1959, an abundant amount of research on the topic has been published. Given the high complexity of the problem, solutions are mostly provided by heuristic approaches, since exact methods are only suitable for small instances. A lot of research effort has, therefore, been dedicated on developing heuristic methods such as 2-Opt and Or-Opt, but also on more general metaheuristic frameworks such as Simulated Annealing and Large Neighbourhood Search. When presenting a heuristic algorithm, the dominant approach towards

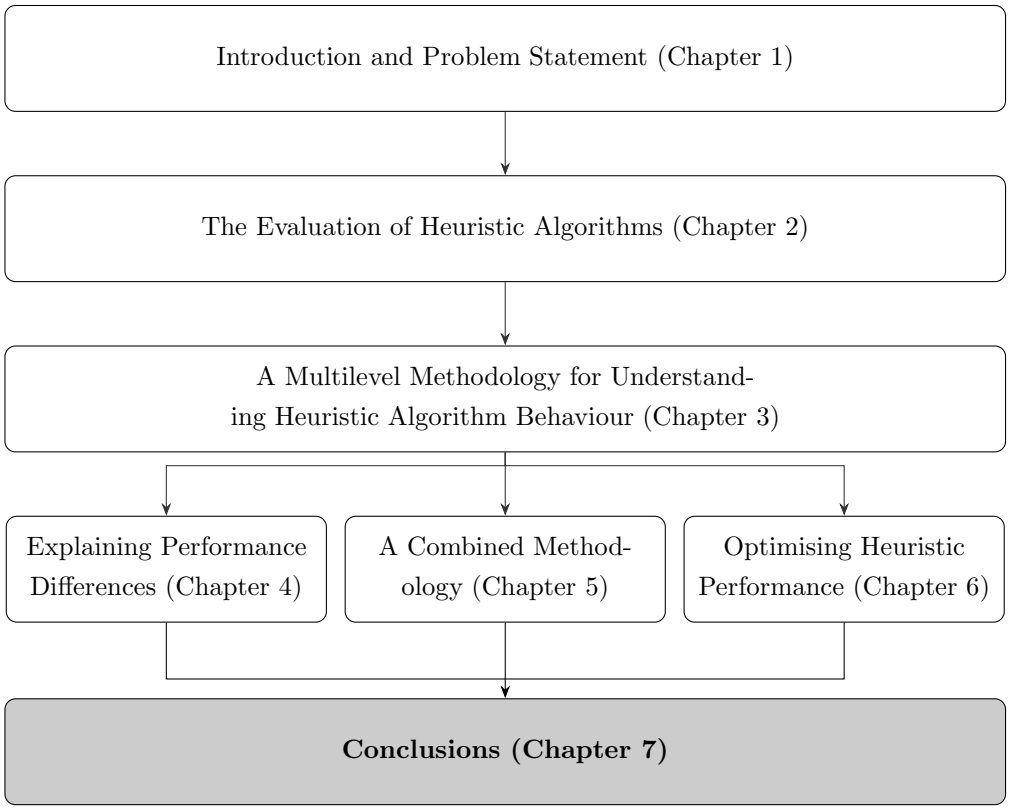


Figure 7.1: Outline of Thesis — Chapter 7.

assessing its performance is by comparing it with other competing methods on one or multiple publicly available benchmark problem sets. The aim often is to do better, in terms of computation cost, solution quality or a trade-off of both. Yet, if a better performance is obtained, it can rarely be explained why the method performs better. Is it because of a new neighbourhood that is implemented in the algorithm? Is it because of a combination of neighbourhoods that performs well together? Or did the experimenter find more efficient ways for applying existing search strategies? These types of questions are rarely answered when presenting the experimental results a heuristic algorithm obtained for solving some type of vehicle routing problem.

It is by answering these kinds of questions that one can learn about the interplay between an algorithm and problem. Having such an understanding of both the problem and the method to solve it, will provide researchers guidance in the design,

optimisation and comparison of heuristic algorithms.

This thesis provides a methodological framework that enables experimenters to find answers on questions relating to why two methods or configurations of a single method differ in the performance they obtain. This doctoral thesis sees it as a next step in the experimental research on vehicle routing problems to obtain a deeper understanding and insight in the effects of parameters and heuristic components on algorithm performance. The methodology is able to identify which algorithm elements significantly impact the solution quality of a heuristic method and how the problem instance characteristics influence these effects. Multilevel experimental designs are employed to efficiently study how the effects vary by the problem instance to be solved. It enables researchers to make statements about an entire population of problem instances, not just a small set of benchmark instances. Different recommendations for different parts of the problem space can be obtained. First, an exploratory analysis exposes how the algorithm elements are correlated with algorithm performance as well as how their performance impact is correlated with the characteristics of a problem instance. Second, a confirmatory analysis searches to explain observed correlations through hypotheses testing. The exploratory analysis is at first performed relying solely on the multilevel regression model. Then, it is shown how the multilevel methodology can focus on the algorithm elements and problem instance characteristics that are most relevant to performance. This focus is achieved by performing a functional analysis of variance (fANOVA) before fitting the multilevel regression model. The ranking of effects fANOVA provides will lead to a more concise regression model with less predictor variables. The multilevel methodology, on the other hand, provides a more detailed analysis of the effects of algorithm elements and enables confirmatory analyses to be performed.

An analysis of a large neighbourhood search algorithm's performance on instances of the vehicle routing problem with time windows showed that including all destroy and repair operators in an algorithm configuration does not necessarily lead to the best results. The use of regret-2 as the sole repair was identified as the best choice on average since it is expected to perform better than the other two repair operator configurations for larger problem sizes. The destroy operator combination that will obtain the best results with this repair operator is random removal. These findings are also confirmed by fANOVA, which is not bound by the assumptions regression models typically rely on, and makes the regression analysis more robust. This analysis of the performance impact of the operators

considered the moderating effect of each significant problem instance characteristic ceteris paribus, but these characteristics can of course divert simultaneously from their average level. Which operator combinations work well and which do not depends on the unique combination of characteristics that constitute a problem instance. The multilevel methodology offers guidance and insights for both an 'average' problem instance as for a specific problem instance with certain characteristics.

The exploratory analysis raised new questions regarding the LNS and VRPTW, such as why removing customers at random works better than removing geographical clusters of customers, given that the removed customers are reinserted using a difficulty measure. Therefore, explanations were sought for why two destroy operators perform different when combined with the same type of repair operator. It is found that removing geographical clusters of customers reduces the number of insertion alternatives to choose from during the repair phase. Several customers do not even have a single feasible insertion option in one of the existing routes and can therefore be considered isolated cases (at the start of the repair phase). Postponing the insertion of these isolated customers is found to have a detrimental impact on the solution quality. It is tested what the effect is of assigning these customers a higher priority by allowing their insertion in an individual route from the depot to the customer and back, an option that was previously considered as a last alternative. Permitting these individual routes to be created sooner in the repair process adds good insertion alternatives for other removed customers and thus enables the regret operator to make better choices. Hence, a regret operator will make a better estimation of customer difficulty and consequently a better prioritisation if each individual customer has existing routes nearby in which it can be feasibly inserted. Through this detailed analysis of a destroy and repair iteration, the majority of the performance difference between both destroy scenarios is explained.

The analyses went from looking into a complete metaheuristic framework to delving into the functioning of individual operators during a single iteration. Through this approach of iterative experimentation, each time examining issues at a more detailed level one can gain general insights that are not confined to a specific heuristic algorithm. The analyses of Chapter 4 showed the importance of isolated customers during solution construction. Initial solution construction for any metaheuristic framework might be improved by treating isolated customers as high priority insertions once the first route has been created.

The exploratory analysis cannot only be used as a starting point for subsequent confirmatory analyses that aim to explain observed correlations, but can also serve to choose well-performing parameter values and components for both an average problem instance as well as a specific problem instance. It is shown how the multilevel regression analysis can be used to derive decision rules for each algorithm parameter and component. These decision rules are formulated using the (significant) problem instance characteristics which enables the definition of algorithm configurations for each individual problem instance. It has been shown that there often is not a single algorithm (configuration) that performs best for each individual instance, but that heuristic algorithm performance varies over the set of problem instances solved. It is believed that the tuning of heuristic algorithms, using the multilevel regression analysis, can exploit such instance variations.

Results for a single LNS configuration that performs best for an average VRPTW instance show its performance is in line with that of an established automatic algorithm configurator as *irace*. The more training data, the better this 'average instance' configuration performs. The performance results for the instance-specific configurations, on the other hand, indicate no real benefit of defining instance-specific configurations. This is probably due to the fact that the generated problem instances are quite homogeneous and, therefore, do not provide opportunity for exploitation of problem instance performance variation.

The main takeaway of this doctoral dissertation is that science is about understanding, rather than some race to be won. Performing a thorough analysis of experimental results can provide valuable knowledge that is not limited to the specific experimental context applied, but is beneficial to all related research work. Hence, when faced with some vehicle routing problem, the decision regarding what solution strategies to apply or how to optimise them can be supported by previously performed analyses. These analyses should account for both the heuristic algorithm as the problem instances solved in order to be able to investigate their interplay. This doctoral thesis encourages to conduct research that is not confined to the specifics the experimenter has chosen to consider, but is generalisable to a wider whole of similar contexts. The dissertation provides the VRP community the means to do this and learn about both the problem and the method used to solve it.

7.2 Further Research

The aim of this thesis was to propose and illustrate the use of an evaluation methodology for gaining insight and understanding of the interplay between a heuristic algorithm and the problem instance it has to solve. A generic and well-studied vehicle routing variant was considered. Chapter 3 mentioned that providing a problem instance generator that can produce realistic problem instances is a research question on its own that includes investigating what suitable probability distributions are for the various problem instance characteristics. Such realistic problem instances often have additional side constraints, such as driving and rest times, of which the influence on algorithm parameters and components can be analysed. Opportunities for further research therefore lie in finding out which probability distributions best reflect real-world instances and in studying other vehicle routing variants that may have received far less attention and which have great potential to learn from.

A related subject is the identification of problem instance characteristics that are distinctive in the performance a heuristic algorithm obtains. A preparatory descriptive analysis on the sample of problem instances that will be used in experiments might be a good starting point for identifying problem instance characteristics to be included in regression analyses. Rasku et al. (2016) have already listed a number of characteristics that are worth investigating. Finding the proper characteristics is both beneficial for explanatory and subsequent confirmatory analyses to obtain knowledge and understanding, and for instance-specific tuning of heuristic algorithms.

The combined methodology proposed in Chapter 5 is not yet implemented in a single tool. At this moment, fANOVA is available as a Python package, and the multilevel regression needs to be formulated manually. In this thesis the R package *brms* was used. It would be more convenient to use if both approaches were integrated in a single tool that first provides the fANOVA importance ranking and then allows the experimenter to select the terms to be included in the multilevel regression model before fitting it to the data.

The detailed analysis of a destroy and repair iteration in Chapter 4 was able to explain the majority of the performance difference between two destroy scenarios, given the use of the same repair operator. A small significant difference still remains for which an explanation might be found by further improving how customers are prioritised or by using a preparatory step in the regret operator. Furthermore, this

analysis focused on the perspective of the regret operator to find out why there is a performance difference, but it might be interesting to take the perspective of related removal and find out what needs to change for the performance gap to close. This can reinforce the conclusions of Chapter 4.

The case study used throughout this thesis considered an existing metaheuristic framework with existing components. Yet, that does not mean the application of the methodology is limited to established algorithms and new methods should first be evaluated on their competitiveness. On the contrary, if a researcher has come up with a new innovative way of searching for solution improvements, the first step should be to verify whether the idea actually works as reasoned. Once one has a complete understanding what one is creating, the focus can switch to developing the best possible version of the idea.

The experiments performed in this dissertation were set up having control over both the algorithm configurations and problem instances to be solved. By relying on artificially generated problem instances the practitioner is able to make statements for an entire population of problem instances. The researcher has complete control when defining this population. One can generate a diverse population or choose to focus on a population of difficult problem instances. In either case the methodology is applied in the same way, but with conclusions being relevant to the population defined. If a company provides a number of real-world instances without having knowledge off which population these instances are a sample from, the practitioner no longer has control over the problem instances. This is referred to as an uncontrolled experiment. The limitation of the non-random sample of problem instances is that it is difficult to identify the problem instance population. Furthermore, significance tests and confidence intervals cannot be interpreted since they rely on the assumption of the data being a random sample from some population. These limitations might not be important if the practitioner does not have the intention to generalise regression results (i.e., external validity), but just wants them to be valid for the sample itself (i.e., internal validity) (Banerjee and Chaudhury, 2010).

Finally, this dissertation has focused analysis efforts on the vehicle routing problem with time windows. The proposed multilevel methodology, however, is not limited to the vehicle routing application, but can be used just as well for other problem contexts, which are undoubtedly equally interesting to analyse.

Appendix A

Details on Generated Problem Instances

The diversity of the problem instances can be assessed through summary statistics as well as plots such as histograms. This is done for the sample of problem instances used in Section 3.4.5 in Table A.1 and Figures A.1a to A.1l. These measures are not only provided for the problem instance characteristics listed in Table 3.2 are considered, but also for characteristics like the spatial distribution of customers which was noted an important aspect of a VRP problem by Tuzun et al. (1997). The latter show little variation in their values, which is logical since a fixed 500x500 area is considered for all problem instances. Further, given the uniformly distributed demand values and fixed vehicle capacity, the average number of customers per route is rather small and varies little, which makes these instances not representative for, say, small package delivery where a single route usually serves many customers. Yet, as mentioned earlier, it is not the focus of this paper to analyse real-life problem instances. The conclusions from this research should be seen in the context of problem instances with a diverse number of customers randomly dispersed in a 500^2 area without peak demand values, with small variations in the service time and time window width for each customer.

Table A.1: Summary Table sample problem instances

Problem characteristic	min	max	average	standard deviation
Number of customers	25	397	216.45	103.65
Average demand	27.06	35.11	30.02	1.06
Average service time	20.37	39.07	29.29	4.29
Average time window width	34.50	63.58	49.35	6.68
Maximum run time (seconds)	72.55	1707.81	901.82	373.42
Average edge distance	201.03	272.24	241.44	14.76
Standard deviation edge distance	98.43	128.86	115.64	6.17
Fraction of distinct distances	0.41	1	0.72	0.17
Centroid of the nodes: x coordinate	170	331	249	33.20
Centroid of the nodes: y coordinate	160	330	248	34.42
Average distance to centroid	145.31	201.95	176.54	11.47
Average number of customers on a route	4	5.47	4.92	0.19

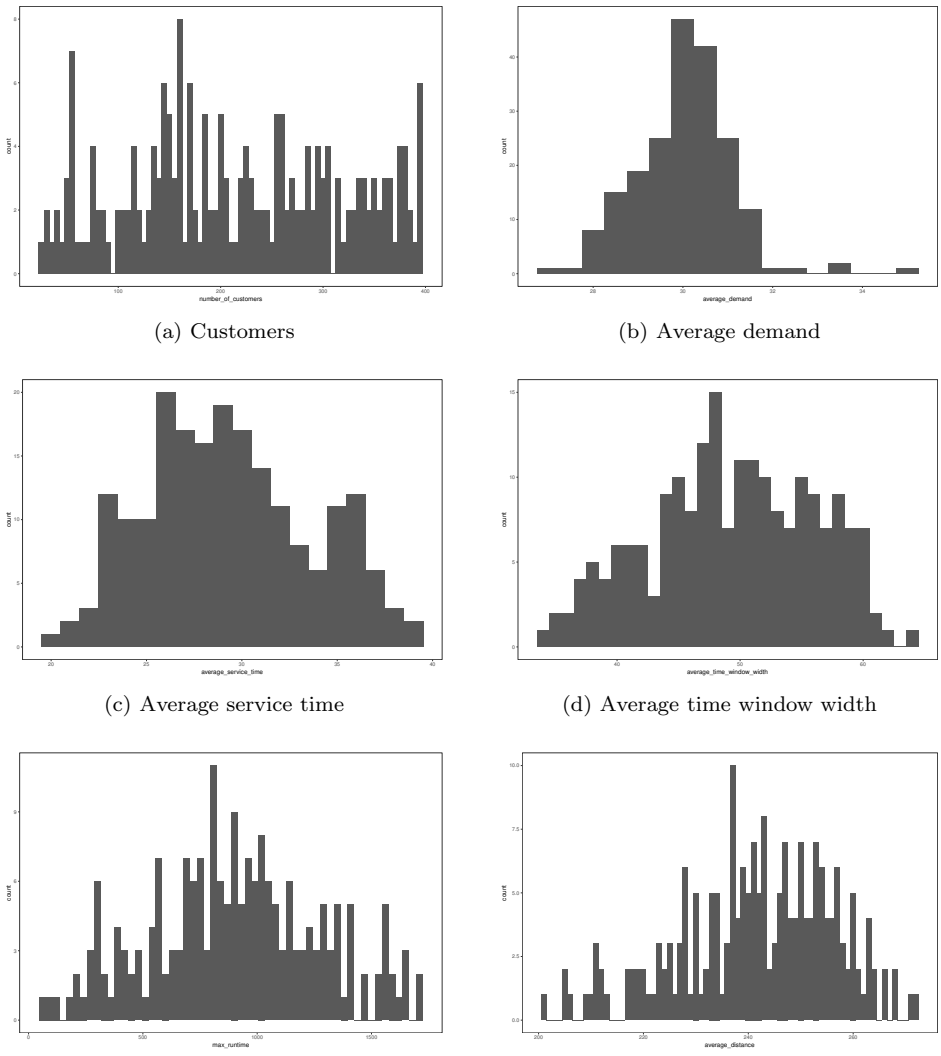
A.1 Problem Instance Generator

```
import random
import math
import os

#Specifcy folder where txt-files are to be saved
save_path = './Instances_Experiment_1'

#How many problem instances to generate
number_of_instances = 200

for instance_id in range(1,number_of_instances + 1):
    #Sample the number of customers to serve
    customers = random.randint(25,400)
    vehicles= customers
    #The capacity of each vehicle is fixed at 150 units
    vehicle_capacity = 150

    depot_id = 0
    depot_x_coord = random.randint(0,500)
    depot_y_coord = random.randint(0,500)
    depot_demand = 0
    #The depot is opened a fixed time window of 15 hours
    depot_start_tw = 0
```

```
depot_end_tw = 900
depot_service_time = 0

#Sample the maximum CPU time the algorithm gets to solve the problem
                                        instance
runtime = random.triangular(60,1800)

#For these 3 characteristics we keep track of totals in order to
#calculate an average measure across all customers
total_service_time = 0
total_time_window_width = 0
total_demand = 0

#create txt-file
#filename = Instance + id + geographical distribution of
#customers (Random/Clustered/Semi-clustered)
filename = 'Instance%d_' % (instance_id) + 'LNS_Random.txt'
complete_name = os.path.join(save_path, filename)
f = open(complete_name,'w')
f.write("%s\nruntime: %f" % (filename, runtime))
f.write("\n\tVEHICLE\n\tNUMBER\t\tCAPACITY\n")
f.write("\t%d\t\t\t%d\n\n" % (vehicle_number, vehicle_capacity))
f.write("CUSTOMERS\nCUST ID\t XCOORD\t YCOORD\t DEMAND\t
         START TIME WINDOW\t END TIME WINDOW\t SERVICE TIME\n\n")
f.write("%d\t%d\t%d\t%d\t\t%d\t\t\t%d\t\t\t%d\n" %
       (depot_id, depot_x_coord, depot_y_coord, depot_demand,
        depot_start_tw, depot_end_tw, depot_service_time))

#For these characteristics we sample minimum and maximum values
#from a uniform distribution
min_service_time = random.uniform(10,30)
max_service_time = random.uniform(30,50)
min_width = random.randint(20,50)
max_width = random.randint(50,80)

#Create uniformly distributed customers
for id in range(1, customer_number+1):
    feasible = False
    #As long as the generated problem instance is 'infeasible',
    #generate a new one
    while feasible == False:
        customer_id = id
        #Sample x and y coordinates for customer
        x_coord = random.randint(0,500)
        y_coord = random.randint(0,500)
        #Sample customer demand
```

```
            demand = random.randint(10, 50)

            #Sample service time at customer from triangular
                                                distribution
            service_time = int(random.triangular(min_service_time,
                                                max_service_time))
            distance = int(math.sqrt((depot_x_coord - x_coord) ** 2
                                + (depot_y_coord - y_coord) ** 2))

            if (depot_start_tw + distance < depot_end_tw - distance -
                service_time):
                tw_centre = random.randint(depot_start_tw + distance,
                                           depot_end_tw - distance -
                                           service_time)

                tw_width = random.randint(min_width, max_width)
                tw_start = time_window_centre - 0.5*tw_width
                tw_end = time_window_centre + 0.5*tw_width

                if ((tw_end + service_time + distance <= depot_end_tw)):
                    f.write("%d\t%d\t%d\t%d\t\t%d\t\t\t%d\t\t\t%d\n" %
                            (customer_id, x_coord, y_coord, demand,
                             tw_start, tw_end, service_time))
                    total_service_time += service_time
                    total_demand += demand
                    total_tw_width += tw_width
                    feasible = True
            else: feasible = False
    average_service_time = total_service_time/float(customers)
    average_tw_width = total_tw_width/float(customers)
    average_demand = total_demand/float(customers)
    f.write("average service time: %f - average time window width: %f
            - average demand: %f\n" %
           (average_service_time, average_tw_width, average_demand))
    f.close()
```

A.2 Histograms



(a) Customers

(b) Average demand

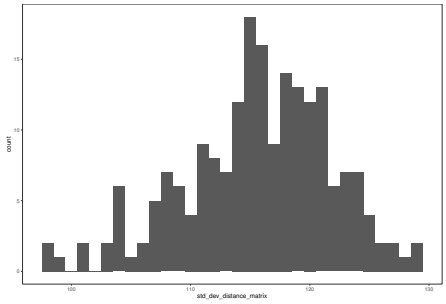
(c) Average service time

(d) Average time window width

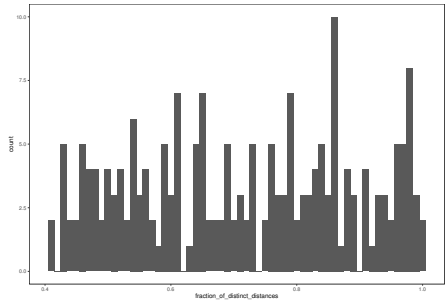
(e) Maximum run time

(f) Average edge distance

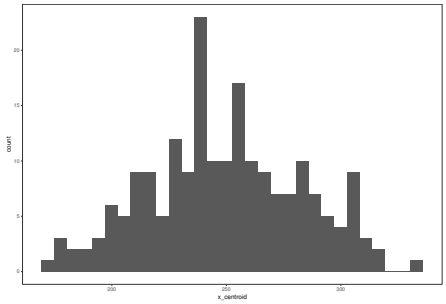
Figure A.1: Histograms



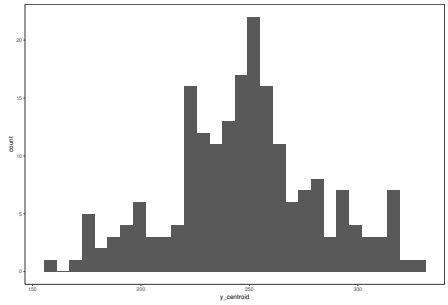
(g) Standard deviation edge distance



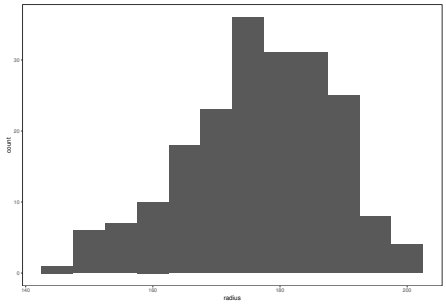
(h) Fraction of distinct distances



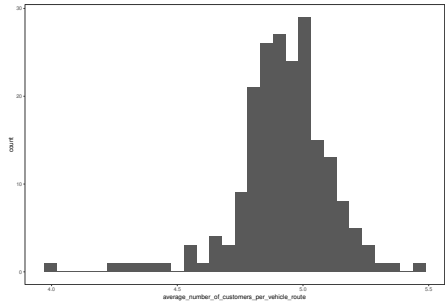
(i) Centroid (x coordinate)



(j) Centroid (y coordinate)



(k) Average distance to centroid



(l) Average number of customers on a route

Figure A.1: Histograms

Appendix B

Regression Tables

This appendix provides the complete regression tables for all models fitted in this doctoral thesis.

B.1 Chapter 3

Table B.1: Regression Table Chapter 3

Variable	Estimate	Est.Error	l-95% CI	u-95% CI
Intercept	$3,939.77^{**}$	121.21	3,706.59	4,180.24
Greedy	-165.77^{**}	6.29	−178.13	−153.45
Regret2	16.64^{**}	4.45	7.89	25.45
Random	18.02^{**}	4.34	9.46	26.54
Worst	-15.23^{**}	5.09	−25.13	−5.19
Related	-39.05^{**}	4.76	−48.32	−29.72
RandomWorst	4.41	4.66	−4.66	13.57
WorstRelated	-13.97^{**}	4.43	−22.74	−5.30
RandomRelated	7.55	4.55	−1.34	16.56
Cooling rate	−3.88	12.61	−28.49	20.97
Start temperature control parameter	-66.34^{*}	28.23	−121.81	−10.49
Noise parameter	-9.37^{*}	4.25	−17.66	−0.94
Determinism parameter	0.40	0.22	−0.03	0.84
Customers$^{\frac{1}{3}}$	-411.93^{**}	28.11	−466.81	−357.71
Avg service time	−24.37	27.98	−79.33	30.51
Avg time window width	42.24^{*}	18.14	7.02	77.87
Avg demand	78.66	114.13	−142.95	307.61
Runtime	−15.45	19.80	−54.78	23.81
Cooling rate × Start temperature control parameter	285.23	455.54	−600.17	1,179.29
Random × Determinism parameter	−0.23	0.30	−0.82	0.35
Worst × Determinism parameter	-0.97^{**}	0.32	−1.60	−0.35
Related × Determinism parameter	-1.88^{**}	0.32	−2.50	−1.25
RandomWorst × Determinism parameter	−0.26	0.31	−0.88	0.35
WorstRelated × Determinism parameter	0.09	0.30	−0.51	0.68
RandomRelated × Determinism parameter	−0.12	0.31	−0.73	0.48
Greedy × Noise parameter	-37.96^{**}	6.07	−49.90	−26.11

Regret2 × Noise parameter	2.94	5.87	−8.71	14.36
Greedy × Random	−70.62**	6.40	−83.22	−58.10
Greedy × Worst	−85.48**	6.78	−98.70	−72.17
Greedy × Related	59.20**	6.80	45.87	72.44
Greedy × RandomWorst	−82.93**	6.75	−96.14	−69.56
Greedy × WorstRelated	3.78	6.34	−8.58	16.25
Greedy × RandomRelated	13.85*	6.62	0.90	26.77
Regret2 × Random	−7.54	6.22	−19.77	4.62
Regret2 × Worst	1.30	6.50	−11.37	14.24
Regret2 × Related	−3.73	6.46	−16.26	8.91
Regret2 × RandomWorst	−3.86	6.55	−16.76	8.78
Regret2 × WorstRelated	−1.66	6.23	−13.88	10.54
Regret2 × RandomRelated	−4.43	6.21	−16.51	7.85
Customers$^{\frac{1}{3}}$ × Runtime	6.73	3.87	−0.86	14.16
Greedy × Customers$^{\frac{1}{3}}$	−20.00**	1.06	−22.11	−17.93
Greedy × Avg service time	4.48**	1.05	2.43	6.56
Greedy × Avg time window width	−4.18**	0.69	−5.52	−2.84
Greedy × Avg demand	−0.15	4.30	−8.66	8.20
Greedy × Runtime	3.04**	0.74	1.61	4.49
Regret2 × Customers$^{\frac{1}{3}}$	2.10**	0.40	1.30	2.88
Regret2 × Avg service time	−0.29	0.40	−1.08	0.48
Regret2 × Avg time window width	0.29	0.25	−0.21	0.79
Regret2 × Avg demand	−1.09	1.68	−4.44	2.17
Regret2 × Runtime	−0.18	0.28	−0.74	0.37
Random × Customers$^{\frac{1}{3}}$	0.10	0.61	−1.09	1.30
Random × Avg service time	0.75	0.61	−0.45	1.93
Random × Avg time window width	−0.13	0.40	−0.93	0.65
Random × Avg demand	−1.79	2.58	−6.79	3.28
Random × Runtime	−0.43	0.43	−1.26	0.40
Worst × Customers$^{\frac{1}{3}}$	0.12	0.77	−1.41	1.63
Worst × Avg service time	1.30	0.78	−0.23	2.81
Worst × Avg time window width	−0.77	0.51	−1.77	0.24
Worst × Avg demand	−0.71	3.32	−7.12	5.83
Worst × Runtime	−0.50	0.55	−1.57	0.60
Related × Customers$^{\frac{1}{3}}$	−4.84**	0.70	−6.21	−3.46
Related × Avg service time	0.54	0.70	−0.85	1.91
Related × Avg time window width	−0.84	0.46	−1.75	0.08
Related × Avg demand	0.56	2.87	−5.09	6.11
Related × Runtime	−0.17	0.49	−1.14	0.81
RandomWorst × Customers$^{\frac{1}{3}}$	−0.92	0.65	−2.21	0.34
RandomWorst × Avg service time	1.34*	0.65	0.06	2.62
RandomWorst × Avg time window width	−0.63	0.43	−1.47	0.20
RandomWorst × Avg demand	0.52	2.69	−4.72	5.77
RandomWorst × Runtime	−1.08*	0.45	−1.96	−0.19
WorstRelated × Customers$^{\frac{1}{3}}$	−0.87	0.60	−2.05	0.30
WorstRelated × Avg service time	0.27	0.61	−0.91	1.47
WorstRelated × Avg time window width	0.10	0.40	−0.68	0.88
WorstRelated × Avg demand	2.06	2.45	−2.75	6.82
WorstRelated × Runtime	−0.57	0.41	−1.38	0.23
RandomRelated × Customers$^{\frac{1}{3}}$	0.98	0.61	−0.23	2.18
RandomRelated × Avg service time	0.13	0.61	−1.07	1.33
RandomRelated × Avg time window width	−0.01	0.40	−0.79	0.78
RandomRelated × Avg demand	0.41	2.47	−4.46	5.19
RandomRelated × Runtime	−0.49	0.42	−1.31	0.33
Cooling rate × Customers$^{\frac{1}{3}}$	−3.81	2.90	−9.51	1.88
Cooling rate × Avg service time	0.16	3.00	−5.82	6.07
Cooling rate × Avg time window width	−0.44	1.91	−4.18	3.28
Cooling rate × Avg demand	19.22	11.85	−4.07	42.18

Cooling rate × Runtime	1.21	2.05	−2.84	5.21
Start temperature control parameter × Customers$^{\frac{1}{3}}$	1.35	6.56	−11.42	14.26
Start temperature control parameter × Avg service time	8.06	6.72	−5.09	21.23
Start temperature control parameter × Avg time window width	3.09	4.29	−5.29	11.44
Start temperature control parameter × Avg demand	−2.50	27.74	−56.41	52.58
Start temperature control parameter × Runtime	−2.17	4.60	−11.22	6.75
Determinism parameter × Customers$^{\frac{1}{3}}$	0.02	0.02	−0.02	0.06
Determinism parameter × Avg service time	−0.02	0.02	−0.06	0.02
Determinism parameter × Avg time window width	0.001	0.01	−0.02	0.03
Determinism parameter × Avg demand	0.10	0.08	−0.06	0.26
Determinism parameter × Runtime	0.0002	0.01	−0.03	0.03
Noise parameter × Customers$^{\frac{1}{3}}$	−1.58*	0.63	−2.82	−0.36
Noise parameter × Avg service time	0.05	0.62	−1.17	1.28
Noise parameter × Avg time window width	−0.85*	0.41	−1.64	−0.06
Noise parameter × Avg demand	0.48	2.57	−4.55	5.47
Noise parameter × Runtime	0.14	0.44	−0.71	1.01

Note: ** denotes significance at 1%, * denotes significance at 5%

B.2 Chapter 4

Table B.2: Regression Table Chapter 4 — Average Insertion Cost

Dependent variable: Log(avg insertion cost)	Estimate	Std. Error	l-95% CI	u-95% CI
Intercept	4.16***	0.03	4.11	4.22
Customers	−0.002***	0.0002	−0.003	−0.002
Related	0.48***	0.04	0.41	0.56
Customers	0.001***	0.0004	0.0003	0.002
Regret-3	0.02	0.02	−0.02	0.07
Customers	−0.0003	0.0002	−0.001	0.0001
Regret-4	0.01	0.02	−0.04	0.05
Customers	−0.0002	0.0002	−0.001	0.0002
Percentage removed	0.02***	0.002	0.01	0.02
Related × Regret-3	0.12***	0.05	0.04	0.21
Customers	0.001**	0.0004	0.0002	0.002
Related × Regret-4	0.14***	0.05	0.05	0.23
Customers	0.001***	0.0004	0.0006	0.002
Related × Percentage removed	−0.002	0.003	−0.01	0.003
Regret-3 × Percentage removed	−0.0005	0.002	−0.005	0.004
Regret-4 × Percentage removed	0.0002	0.002	−0.004	0.004
Related × Regret-3 × Percentage removed	0.0003	0.003	−0.006	0.007
Related × Regret-4 × Percentage removed	−0.002	0.003	−0.009	0.005
Observations	6761			
Num. groups: problem instances	200			

Note: *p<0.1; **p<0.05; ***p<0.01

Table B.3: Regression Table LNS experiment Chapter 4 — isolated customers not prioritised

Dependent variable: total cost^{-1}	Estimate	Std. Error	l-95% CI	u-95% CI
Intercept	3891.09^{***}	122.68	3647.76	4126.34
Related	-62.08^{***}	1.87	−65.75	−58.38
Regret-3	−0.15	1.50	−3.10	2.78
Regret-4	-4.66^{***}	1.50	−7.58	−1.73
Cooling rate	−5.07	8.00	−20.68	10.60
Start temperature control parameter	−27.27	16.65	−59.61	5.31
Determinism parameter	0.05	0.08	−0.11	0.20
Noise parameter	-14.32^{***}	2.60	−19.41	−9.23
Customers$^{\frac{1}{3}}$	-402.14^{***}	27.34	−456.11	−349.63
Avg service time	−16.54	29.17	−72.07	41.55
Avg time window width	24.07	19.97	−14.52	63.13
Avg demand	181.99	139.41	−90.64	456.67
Run time	9.28	19.75	−28.78	48.29
Customers$^{\frac{1}{3}}$ × Run time	0.68	4.34	−7.82	9.22
Cooling rate × Start temperature control parameter	-533.27^{*}	306.45	−1141.61	61.88
Related × Determinism parameter	-2.49^{***}	0.10	−2.68	−2.29
Regret-3 × Noise parameter	6.45^{*}	3.62	−0.68	13.50
Regret-4 × Noise parameter	5.56	3.59	−1.41	12.69
Related × Regret-3	14.92^{***}	2.11	10.77	19.02
Related × Regret-4	16.08^{***}	2.08	11.98	20.09
Related × Customers$^{\frac{1}{3}}$	-8.88^{***}	0.33	−9.53	−8.23
Related × Avg service time	0.69^{**}	0.34	0.02	1.37
Related × Avg time window width	-0.85^{***}	0.24	−1.31	−0.39
Related × Avg demand	0.83	1.64	−2.40	4.02
Related × Run time	1.51^{***}	0.24	1.05	1.98
Regret-3 × Customers$^{\frac{1}{3}}$	1.04^{***}	0.24	0.57	1.52
Regret-3 × Avg service time	−0.06	0.25	−0.56	0.44
Regret-3 × Avg time window width	0.30^{*}	0.17	−0.04	0.64
Regret-3 × Avg demand	1.79	1.22	−0.62	4.17
Regret-3 × Run time	−0.22	0.17	−0.56	0.12
Regret-4 × Customers$^{\frac{1}{3}}$	0.85^{***}	0.24	0.37	1.33
Regret-4 × Avg service time	0.42	0.26	−0.09	0.93
Regret-4 × Avg time window width	0.16	0.18	−0.19	0.51
Regret-4 × Avg demand	−0.83	1.23	−3.25	1.56
Regret-4 × Run time	−0.12	0.18	−0.47	0.23
Cooling rate × Customers$^{\frac{1}{3}}$	−1.58	1.80	−5.11	1.94
Cooling rate × Avg service time	1.41	1.88	−2.27	5.06
Cooling rate × Avg time window width	−1.63	1.29	−4.17	0.87
Cooling rate × Avg demand	−1.21	9.05	−18.88	16.68
Cooling rate × Run time	0.10	1.30	−2.47	2.62
Start temperature control parameter × Customers$^{\frac{1}{3}}$	3.73	3.73	−3.65	11.02
Start temperature control parameter × Avg service time	3.02	3.91	−4.62	10.76
Start temperature control parameter × Avg time window width	2.01	2.66	−3.23	7.19
Start temperature control parameter × Avg demand	37.39^{**}	19.07	0.27	74.87
Start temperature control parameter × Run time	−1.51	2.69	−6.77	3.73
Determinism parameter × Customers$^{\frac{1}{3}}$	-0.19^{***}	0.01	−0.22	−0.16
Determinism parameter × Avg service time	0.02	0.01	−0.01	0.05
Determinism parameter × Avg time window width	-0.03^{***}	0.01	−0.05	−0.01
Determinism parameter × Avg demand	−0.02	0.07	−0.15	0.12
Determinism parameter × Run time	0.01	0.01	−0.01	0.03
Noise parameter × Customers$^{\frac{1}{3}}$	-1.46^{***}	0.35	−2.14	−0.77
Noise parameter × Avg service time	0.27	0.36	−0.44	0.98
Noise parameter × Avg time window width	−0.17	0.25	−0.66	0.31

Noise parameter × Avg demand	0.98	1.73	−2.44	4.36
Noise parameter × Run time	0.27	0.25	−0.22	0.75
Observations	4,000			
Num. groups: problem instances	200			

Note: *p<0.1; **p<0.05; ***p<0.01

Table B.4: Regression Table LNS experiment Chapter 4 — isolated customers prioritised

Dependent variable: total cost^{-1}	Estimate	Std. Error	l-95% CI	u-95% CI
Intercept	3905.83***	121.19	3666.74	4146.37
Related	−19.08***	1.37	−21.78	−16.41
Regret-3	−2.69**	1.09	−4.82	−0.55
Regret-4	−5.05***	1.12	−7.24	−2.86
Cooling rate	7.60	6.00	−4.14	19.47
Start temperature control parameter	−9.12	12.33	−33.48	15.07
Determinism parameter	0.09	0.06	−0.02	0.21
Noise parameter	−6.02***	1.93	−9.80	−2.24
Customers$^{\frac{1}{3}}$	−401.12***	27.60	−454.79	−345.54
Avg service time	−18.13	28.66	−73.98	38.62
Avg time window width	24.01	19.49	−13.89	62.22
Avg demand	179.67	135.38	−85.85	449.95
Run time	9.76	19.56	−28.24	48.16
Customers$^{\frac{1}{3}}$ × Run time	1.79	4.40	−6.90	10.37
Cooling rate × Start temperature control parameter	−11.09	226.59	−451.48	433.63
Related × Determinism parameter	−0.64***	0.07	−0.78	−0.49
Regret-3 × Noise parameter	3.41	2.70	−1.88	8.68
Regret-4 × Noise parameter	1.08	2.70	−4.16	6.38
Related × Regret-3	9.58***	1.56	6.51	12.66
Related × Regret-4	7.84***	1.55	4.79	10.87
Related × Customers$^{\frac{1}{3}}$	−3.46***	0.23	−3.91	−2.99
Related × Avg service time	0.38	0.24	−0.10	0.85
Related × Avg time window width	−0.63***	0.17	−0.96	−0.29
Related × Avg demand	0.09	1.16	−2.21	2.37
Related × Run time	0.52***	0.17	0.19	0.84
Regret-3 × Customers$^{\frac{1}{3}}$	0.74***	0.18	0.39	1.09
Regret-3 × Avg service time	0.15	0.18	−0.21	0.51
Regret-3 × Avg time window width	0.14	0.13	−0.11	0.39
Regret-3 × Avg demand	0.90	0.88	−0.81	2.62
Regret-3 × Run time	−0.47***	0.13	−0.71	−0.22
Regret-4 × Customers$^{\frac{1}{3}}$	0.07	0.18	−0.28	0.42
Regret-4 × Avg service time	0.11	0.19	−0.27	0.48
Regret-4 × Avg time window width	0.22*	0.13	−0.04	0.48
Regret-4 × Avg demand	0.05	0.92	−1.76	1.85
Regret-4 × Run time	−0.32**	0.13	−0.58	−0.06
Cooling rate × Customers$^{\frac{1}{3}}$	−1.13	1.34	−3.75	1.52
Cooling rate × Avg service time	2.61*	1.42	−0.21	5.41
Cooling rate × Avg time window width	−0.42	0.97	−2.32	1.49
Cooling rate × Avg demand	−4.57	6.71	−17.68	8.67
Cooling rate × Run time	0.36	0.97	−1.53	2.26
Start temperature control parameter × Customers$^{\frac{1}{3}}$	1.78	2.76	−3.68	7.14
Start temperature control parameter × Avg service time	−2.24	2.90	−7.92	3.40
Start temperature control parameter × Avg time window width	−0.00	1.97	−3.85	3.89
Start temperature control parameter × Avg demand	−15.36	14.08	−42.85	12.48
Start temperature control parameter × Run time	1.27	2.00	−2.67	5.17

Determinism parameter × Customers$^{\frac{1}{3}}$	-0.05^{***}	0.01	−0.07	−0.03
Determinism parameter × Avg service time	0.01	0.01	−0.01	0.03
Determinism parameter × Avg time window width	-0.02^{***}	0.01	−0.03	−0.00
Determinism parameter × Avg demand	−0.02	0.05	−0.12	0.07
Determinism parameter × Run time	−0.00	0.01	−0.02	0.01
Noise parameter × Customers$^{\frac{1}{3}}$	-1.05^{***}	0.26	−1.56	−0.53
Noise parameter × Avg service time	0.21	0.27	−0.31	0.73
Noise parameter × Avg time window width	-0.38^{**}	0.19	−0.74	−0.01
Noise parameter × Avg demand	−0.35	1.30	−2.91	2.21
Noise parameter × Run time	−0.06	0.18	−0.43	0.30
Observations	4,000			
Num. groups: problem instances	200			

Note: *p<0.1; **p<0.05; ***p<0.01

B.3 Chapter 5

Table B.5: Regression Table Chapter 5 — simplified model VRPTW-LNS

Variable	Estimate	Est.Error	l-95% CI	u-95% CI
Intercept	4,082.85	122.62	3,879.03	4,302.29
Greedy	−141.97	4.39	−149.85	−134.36
Regret2	12.67	2.46	7.81	17.51
Random	16.81	2.41	12.12	21.59
Worst	−13.90	2.45	−18.78	−9.20
Related	−69.67	3.16	−75.53	−63.84
RandomWorst	7.96	2.36	3.34	12.61
WorstRelated	−16.26	2.43	−21.05	−11.48
RandomRelated	1.20	2.46	−3.73	5.97
Customers$^{\frac{1}{3}}$	−428.11	26.96	−476.64	−373.45
Greedy × Random	−67.38	5.26	−77.47	−57.11
Greedy × Worst	−98.09	6.36	−110.17	−85.91
Greedy × Related	88.16	4.37	79.91	96.40
Greedy × RandomWorst	−87.72	5.54	−98.11	−77.07
Greedy × WorstRelated	10.27	3.94	2.54	17.94
Greedy × RandomRelated	20.17	3.91	12.53	27.83
Regret2 × Random	−2.12	3.47	−8.91	4.70
Regret2 × Worst	−0.59	3.47	−7.31	6.27
Regret2 × Related	−9.13	3.88	−16.58	−1.39
Regret2 × RandomWorst	−3.65	3.41	−10.34	3.15
Regret2 × WorstRelated	−4.47	3.46	−11.30	2.15
Regret2 × RandomRelated	−2.89	3.55	−9.85	4.19
Greedy × Customers$^{\frac{1}{3}}$	−16.63	0.97	−18.46	−14.74
Regret2 × Customers$^{\frac{1}{3}}$	1.71	0.57	0.60	2.80
Random × Customers$^{\frac{1}{3}}$	2.73	0.56	1.64	3.83
Worst × Customers$^{\frac{1}{3}}$	0.002	0.57	−1.12	1.13

Related × Customers$^{\frac{1}{3}}$	−11.17	0.73	−12.59	−9.76
RandomWorst × Customers$^{\frac{1}{3}}$	1.80	0.57	0.70	2.90
WorstRelated × Customers$^{\frac{1}{3}}$	−2.30	0.57	−3.40	−1.19
RandomRelated × Customers$^{\frac{1}{3}}$	−0.45	0.56	−1.55	0.65
Greedy × Random × Customers$^{\frac{1}{3}}$	−11.01	1.20	−13.34	−8.63
Greedy × Worst × Customers$^{\frac{1}{3}}$	−1.65	1.42	−4.43	1.13
Greedy × Related × Customers$^{\frac{1}{3}}$	13.50	1.00	11.55	15.43
Greedy × RandomWorst × Customers$^{\frac{1}{3}}$	−8.14	1.27	−10.59	−5.63
Greedy × WorstRelated × Customers$^{\frac{1}{3}}$	3.04	0.92	1.25	4.84
Greedy × RandomRelated × Customers$^{\frac{1}{3}}$	1.87	0.89	0.10	3.61
Regret2 × Random × Customers$^{\frac{1}{3}}$	−0.73	0.80	−2.29	0.85
Regret2 × Worst × Customers$^{\frac{1}{3}}$	0.23	0.81	−1.36	1.78
Regret2 × Related × Customers$^{\frac{1}{3}}$	−2.13	0.91	−3.91	−0.32
Regret2 × RandomWorst × Customers$^{\frac{1}{3}}$	−0.23	0.80	−1.80	1.34
Regret2 × WorstRelated × Customers$^{\frac{1}{3}}$	0.06	0.81	−1.52	1.66
Regret2 × RandomRelated × Customers$^{\frac{1}{3}}$	−0.18	0.82	−1.79	1.41

Table B.6: Regression Table Chapter 5 — large model VRPTW-LNS

Variable	Estimate	Est.Error	l-95% CI	u-95% CI
Intercept	4,070.47	169.20	3,860.59	4,288.77
Greedy	−141.80	5.84	−149.15	−134.63
Regret2	12.86	2.41	8.21	17.49
Random	17.23	2.43	12.56	21.93
Worst	−13.99	2.49	−18.70	−9.13
Related	−69.80	3.65	−75.63	−64.10
RandomWorst	8.33	2.36	3.71	12.95
WorstRelated	−16.41	2.45	−21.20	−11.71
RandomRelated	1.37	2.39	−3.22	6.14
Cooling rate	0.25	1.47	−2.65	3.12
Start temperature control parameter	−0.68	1.41	−3.40	2.13
Noise parameter	−13.40	2.39	−17.97	−8.77
Determinism parameter	0.12	0.04	0.05	0.19
Customers$^{\frac{1}{3}}$	−437.47	28.38	−488.87	−391.47
Avg demand	−334.42	91.96	−511.41	−151.55
Avg service time	−40.39	24.39	−87.73	10.63
Avg time window width	39.59	16.09	6.86	70.09
Run time	2.16	16.99	−31.53	34.67
Greedy × Random	−66.91	5.50	−76.90	−56.84
Greedy × Worst	−96.90	6.53	−108.11	−85.32
Greedy × Related	88.97	4.96	81.02	97.11
Greedy × RandomWorst	−88.48	6.02	−99.18	−78.20
Greedy × WorstRelated	10.55	3.92	2.73	18.26
Greedy × RandomRelated	21.04	3.83	13.46	28.46
Regret2 × Random	−3.31	3.39	−10.07	3.25
Regret2 × Worst	−0.60	3.36	−7.18	6.00
Regret2 × Related	−8.47	3.73	−15.82	−1.15
Regret2 × RandomWorst	−4.69	3.36	−11.28	1.90
Regret2 × WorstRelated	−5.19	3.41	−11.89	1.45
Regret2 × RandomRelated	−2.47	3.41	−9.26	4.20
Random × Determinism parameter	−0.12	0.05	−0.22	−0.02
Worst × Determinism parameter	−0.32	0.05	−0.42	−0.22

Related × Determinism parameter	−0.49	0.05	−0.59	−0.38
RandomWorst × Determinism parameter	−0.17	0.05	−0.26	−0.07
WorstRelated × Determinism parameter	−0.07	0.05	−0.17	0.02
RandomRelated × Determinism parameter	−0.02	0.05	−0.12	0.08
Greedy × Noise parameter	−22.70	3.46	−29.48	−16.01
Regret2 × Noise parameter	2.07	3.18	−4.16	8.31
Greedy × Customers$^{\frac{1}{3}}$	−16.99	1.03	−18.72	−15.28
Greedy × Avg demand	5.03	3.45	−1.70	11.83
Greedy × Avg service time	2.78	0.91	0.98	4.56
Greedy × Avg time window width	−2.69	0.60	−3.87	−1.54
Greedy × Run time	1.61	0.64	0.35	2.87
Regret2 × Customers$^{\frac{1}{3}}$	1.63	0.56	0.53	2.71
Regret2 × Avg demand	1.71	2.19	−2.59	5.90
Regret2 × Avg service time	−0.18	0.57	−1.30	0.95
Regret2 × Avg time window width	0.33	0.38	−0.39	1.06
Regret2 × Run time	0.05	0.41	−0.77	0.86
Random × Customers$^{\frac{1}{3}}$	2.75	0.55	1.65	3.84
Random × Avg demand	0.24	2.24	−4.17	4.70
Random × Avg service time	−0.07	0.58	−1.23	1.06
Random × Avg time window width	0.11	0.37	−0.62	0.85
Random × Run time	−0.15	0.40	−0.94	0.65
Worst × Customers$^{\frac{1}{3}}$	0.11	0.57	−1.03	1.24
Worst × Avg demand	−0.23	2.37	−4.90	4.35
Worst × Avg service time	−0.68	0.62	−1.88	0.52
Worst × Avg time window width	0.02	0.39	−0.74	0.78
Worst × Run time	0.05	0.41	−0.76	0.84
Related × Customers$^{\frac{1}{3}}$	−11.36	0.78	−12.71	−10.05
Related × Avg demand	2.48	2.70	−2.90	7.82
Related × Avg service time	0.31	0.70	−1.10	1.66
Related × Avg time window width	−1.57	0.47	−2.49	−0.66
Related × Run time	0.45	0.49	−0.52	1.40
RandomWorst × Customers$^{\frac{1}{3}}$	1.87	0.55	0.77	2.97
RandomWorst × Avg demand	0.90	2.20	−3.37	5.20
RandomWorst × Avg service time	−0.31	0.56	−1.41	0.79
RandomWorst × Avg time window width	0.15	0.36	−0.57	0.85
RandomWorst × Run time	−0.29	0.42	−1.12	0.53
WorstRelated × Customers$^{\frac{1}{3}}$	−2.44	0.55	−3.52	−1.34
WorstRelated × Avg demand	1.77	2.24	−2.53	6.15
WorstRelated × Avg service time	−0.27	0.58	−1.41	0.88
WorstRelated × Avg time window width	−0.18	0.39	−0.93	0.59
WorstRelated × Run time	0.63	0.40	−0.16	1.42
RandomRelated × Customers$^{\frac{1}{3}}$	−0.45	0.56	−1.55	0.64
RandomRelated × Avg demand	0.54	2.24	−3.87	4.92
RandomRelated × Avg service time	−0.47	0.57	−1.57	0.63
RandomRelated × Avg time window width	−0.49	0.38	−1.25	0.25
RandomRelated × Run time	0.62	0.39	−0.14	1.38
Cooling rate × Customers$^{\frac{1}{3}}$	0.54	0.34	−0.12	1.21
Cooling rate × Avg demand	0.20	1.32	−2.40	2.82
Cooling rate × Avg service time	−0.08	0.36	−0.79	0.65
Cooling rate × Avg time window width	−0.26	0.24	−0.73	0.19
Cooling rate × Run time	0.22	0.25	−0.26	0.71
Start temperature control parameter × Customers$^{\frac{1}{3}}$	0.33	0.33	−0.31	1.00
Start temperature control parameter × Avg demand	0.43	1.32	−2.24	3.01
Start temperature control parameter × Avg service time	−0.01	0.35	−0.69	0.66
Start temperature control parameter × Avg time window width	0.35	0.23	−0.10	0.80
Start temperature control parameter × Run time	0.11	0.24	−0.36	0.59
Determinism parameter × Customers$^{\frac{1}{3}}$	−0.001	0.003	−0.01	0.01
Determinism parameter × Avg demand	−0.01	0.01	−0.03	0.02

Determinism parameter × Avg service time	0.003	0.004	−0.004	0.01
Determinism parameter × Avg time window width	0.005	0.002	−0.0002	0.01
Determinism parameter × Run time	−0.003	0.003	−0.01	0.002
Noise parameter × Customers$^{\frac{1}{3}}$	−1.11	0.34	−1.76	−0.42
Noise parameter × Avg demand	−1.81	1.38	−4.50	0.85
Noise parameter × Avg service time	0.53	0.36	−0.18	1.23
Noise parameter × Avg time window width	−0.50	0.24	−0.96	−0.03
Noise parameter × Run time	0.50	0.26	−0.01	1.00
Greedy × Random × Customers$^{\frac{1}{3}}$	−10.65	1.22	−12.98	−8.33
Greedy × Random × Avg demand	−3.61	4.73	−13.01	5.64
Greedy × Random × Avg service time	3.07	1.25	0.61	5.55
Greedy × Random × Avg time window width	−2.35	0.80	−3.90	−0.77
Greedy × Random × Run time	1.00	0.86	−0.71	2.68
Greedy × Worst × Customers$^{\frac{1}{3}}$	−1.53	1.32	−4.14	1.09
Greedy × Worst × Avg demand	−0.34	5.28	−10.76	10.05
Greedy × Worst × Avg service time	5.59	1.44	2.79	8.42
Greedy × Worst × Avg time window width	−5.04	0.95	−6.88	−3.18
Greedy × Worst × Run time	−0.06	0.98	−1.98	1.82
Greedy × Related × Customers$^{\frac{1}{3}}$	14.31	1.05	12.46	16.18
Greedy × Related × Avg demand	−3.97	3.81	−11.47	3.51
Greedy × Related × Avg service time	0.09	0.99	−1.85	2.04
Greedy × Related × Avg time window width	1.90	0.64	0.65	3.17
Greedy × Related × Run time	−0.80	0.69	−2.14	0.56
Greedy × RandomWorst × Customers$^{\frac{1}{3}}$	−7.68	1.26	−10.10	−5.20
Greedy × RandomWorst × Avg demand	−3.19	4.78	−12.62	6.11
Greedy × RandomWorst × Avg service time	3.06	1.31	0.49	5.68
Greedy × RandomWorst × Avg time window width	−3.38	0.85	−5.02	−1.74
Greedy × RandomWorst × Run time	1.26	0.91	−0.55	3.03
Greedy × WorstRelated × Customers$^{\frac{1}{3}}$	3.52	0.90	1.73	5.30
Greedy × WorstRelated × Avg demand	−1.94	3.63	−9.14	5.17
Greedy × WorstRelated × Avg service time	1.14	0.97	−0.72	3.02
Greedy × WorstRelated × Avg time window width	0.22	0.63	−1.01	1.44
Greedy × WorstRelated × Run time	−1.06	0.65	−2.32	0.22
Greedy × RandomRelated × Customers$^{\frac{1}{3}}$	2.07	0.87	0.34	3.78
Greedy × RandomRelated × Avg demand	−3.58	3.42	−10.34	3.04
Greedy × RandomRelated × Avg service time	0.17	0.91	−1.59	1.94
Greedy × RandomRelated × Avg time window width	0.55	0.60	−0.62	1.74
Greedy × RandomRelated × Run time	−0.61	0.63	−1.85	0.64
Regret2 × Random × Customers$^{\frac{1}{3}}$	−0.72	0.78	−2.24	0.82
Regret2 × Random × Avg demand	0.28	3.08	−5.72	6.36
Regret2 × Random × Avg service time	−0.55	0.82	−2.15	1.04
Regret2 × Random × Avg time window width	0.36	0.55	−0.71	1.43
Regret2 × Random × Run time	−0.16	0.58	−1.28	0.98
Regret2 × Worst × Customers$^{\frac{1}{3}}$	0.27	0.79	−1.29	1.84
Regret2 × Worst × Avg demand	0.37	3.18	−5.76	6.59
Regret2 × Worst × Avg service time	0.78	0.85	−0.86	2.43
Regret2 × Worst × Avg time window width	−0.29	0.54	−1.33	0.79
Regret2 × Worst × Run time	−0.38	0.58	−1.51	0.75
Regret2 × Related × Customers$^{\frac{1}{3}}$	−1.77	0.87	−3.46	−0.06
Regret2 × Related × Avg demand	−1.61	3.49	−8.50	5.33
Regret2 × Related × Avg service time	1.35	0.90	−0.42	3.10
Regret2 × Related × Avg time window width	−0.55	0.59	−1.70	0.61
Regret2 × Related × Run time	0.63	0.63	−0.61	1.88
Regret2 × RandomWorst × Customers$^{\frac{1}{3}}$	−0.11	0.78	−1.62	1.44
Regret2 × RandomWorst × Avg demand	−0.84	3.10	−6.90	5.19
Regret2 × RandomWorst × Avg service time	−0.28	0.81	−1.90	1.29
Regret2 × RandomWorst × Avg time window width	−0.48	0.53	−1.52	0.55
Regret2 × RandomWorst × Run time	−0.20	0.58	−1.35	0.95

Regret2 × WorstRelated × Customers$^{\frac{1}{3}}$	0.22	0.79	−1.32	1.77
Regret2 × WorstRelated × Avg demand	−3.98	3.09	−10.03	2.12
Regret2 × WorstRelated × Avg service time	0.29	0.84	−1.39	1.92
Regret2 × WorstRelated × Avg time window width	−0.09	0.55	−1.18	0.98
Regret2 × WorstRelated × Run time	−0.34	0.57	−1.46	0.79
Regret2 × RandomRelated × Customers$^{\frac{1}{3}}$	−0.03	0.79	−1.60	1.52
Regret2 × RandomRelated × Avg demand	−2.25	3.12	−8.33	3.86
Regret2 × RandomRelated × Avg service time	0.21	0.82	−1.42	1.81
Regret2 × RandomRelated × Avg time window width	−0.23	0.55	−1.33	0.85
Regret2 × RandomRelated × Run time	−0.59	0.58	−1.73	0.56

B.4 Chapter 6

Table B.7: Regression Table Chapter 6 — training data with 100 instances

Variable	Estimate	Est.Error	l-95% CI	u-95% CI
Intercept	3,971.86	159.04	3,657.40	4,284.25
Greedy	−126.46	5.00	−136.15	−116.61
Regret2	8.82	3.42	2.11	15.45
Random	4.78	3.49	−2.14	11.55
Worst	−26.33	3.96	−34.13	−18.52
Related	−24.86	3.79	−32.32	−17.52
RandomWorst	−7.89	3.60	−14.98	−0.94
WorstRelated	−10.46	3.44	−17.27	−3.81
RandomRelated	3.30	3.56	−3.65	10.31
Max%Removed	−0.04	0.21	−0.45	0.38
Cooling rate	11.45	9.98	−8.22	31.03
Start temperature control parameter	2.09	22.23	−40.80	45.71
Noise parameter	−10.99	3.54	−17.95	−4.01
Determinism parameter	0.18	0.16	−0.13	0.50
Max%Removed^2	−0.01	0.004	−0.02	−0.01
Cooling rate^2	247.20	189.12	−126.14	610.44
Start temperature control parameter^2	694.45	852.44	−978.29	2,382.06
Noise parameter^2	32.66	7.27	18.28	46.76
Determinism parameter^2	−0.01	0.01	−0.02	0.01
Customers$^{\frac{1}{3}}$	−386.46	37.25	−460.22	−312.59
Avg service time	−50.54	42.26	−132.25	32.17
Avg time window width	76.88	24.99	26.61	126.66
Avg demand	−281.85	140.32	−554.31	−6.71
Runtime	−13.49	27.98	−67.97	41.96
Greedy × Random	−22.67	4.72	−31.90	−13.45
Greedy × Worst	−47.49	4.82	−56.97	−37.93
Greedy × Related	46.62	4.93	37.05	56.35
Greedy × RandomWorst	−36.64	4.83	−45.94	−27.26
Greedy × WorstRelated	5.28	4.79	−4.05	14.73
Greedy × RandomRelated	20.03	4.91	10.32	29.74
Regret2 × Random	−2.79	4.84	−12.22	6.88
Regret2 × Worst	2.82	4.91	−6.84	12.65
Regret2 × Related	0.98	4.87	−8.45	10.67
Regret2 × RandomWorst	2.45	4.89	−7.20	12.00
Regret2 × WorstRelated	3.55	4.87	−5.94	13.08
Regret2 × RandomRelated	0.88	4.88	−8.58	10.42
Cooling rate × Start temperature control parameter	63.18	369.51	−676.08	785.66
Random × Determinism parameter	0.05	0.23	−0.40	0.50
Worst × Determinism parameter	−0.64	0.23	−1.10	−0.20
Related × Determinism parameter	−0.96	0.24	−1.42	−0.50

RandomWorst × Determinism parameter	−0.22	0.22	−0.66	0.22
WorstRelated × Determinism parameter	0.05	0.22	−0.40	0.48
RandomRelated × Determinism parameter	−0.02	0.23	−0.46	0.42
Greedy × Noise parameter	−19.86	4.52	−28.67	−11.11
Regret2 × Noise parameter	5.64	4.52	−3.18	14.45
Random × Max%Removed	0.58	0.28	0.03	1.13
Worst × Max%Removed	0.87	0.30	0.29	1.46
Related × Max%Removed	−1.39	0.31	−2.00	−0.78
RandomWorst × Max%Removed	0.58	0.29	0.01	1.14
WorstRelated × Max%Removed	−0.12	0.29	−0.70	0.45
RandomRelated × Max%Removed	−0.39	0.30	−0.98	0.20
Greedy × Max%Removed	−1.82	0.29	−2.35	−1.25
Regret2 × Max%Removed	0.06	0.29	−0.51	0.63
$\text{Customers}^{\frac{1}{3}}$ × Runtime	10.26	5.68	−1.11	21.50
Greedy × $\text{Customers}^{\frac{1}{3}}$	−16.64	0.88	−18.37	−14.93
Greedy × Avg service time	2.34	1.02	0.32	4.34
Greedy × Avg time window width	−3.08	0.63	−4.32	−1.86
Greedy × Avg demand	−1.43	3.39	−8.11	5.14
Greedy × Runtime	1.33	0.69	−0.02	2.70
Regret2 × $\text{Customers}^{\frac{1}{3}}$	1.46	0.30	0.86	2.05
Regret2 × Avg service time	0.10	0.35	−0.60	0.79
Regret2 × Avg time window width	0.12	0.21	−0.30	0.54
Regret2 × Avg demand	0.84	1.17	−1.45	3.14
Regret2 × Runtime	−0.16	0.24	−0.62	0.30
Random × $\text{Customers}^{\frac{1}{3}}$	0.21	0.51	−0.78	1.20
Random × Avg service time	1.26	0.60	0.08	2.42
Random × Avg time window width	0.07	0.35	−0.62	0.75
Random × Avg demand	3.10	1.90	−0.59	6.87
Random × Runtime	0.33	0.40	−0.45	1.11
Worst × $\text{Customers}^{\frac{1}{3}}$	−0.96	0.62	−2.19	0.25
Worst × Avg service time	1.59	0.72	0.20	2.98
Worst × Avg time window width	−0.71	0.43	−1.55	0.11
Worst × Avg demand	2.49	2.37	−2.25	7.18
Worst × Runtime	0.86	0.47	−0.06	1.77
Related × $\text{Customers}^{\frac{1}{3}}$	−2.97	0.56	−4.07	−1.88
Related × Avg service time	−0.47	0.64	−1.72	0.76
Related × Avg time window width	−0.38	0.38	−1.12	0.37
Related × Avg demand	0.97	2.12	−3.20	5.12
Related × Runtime	0.29	0.43	−0.55	1.15
RandomWorst × $\text{Customers}^{\frac{1}{3}}$	−0.42	0.53	−1.46	0.60
RandomWorst × Avg service time	1.20	0.60	0.01	2.38
RandomWorst × Avg time window width	−0.52	0.36	−1.24	0.18
RandomWorst × Avg demand	0.70	1.99	−3.19	4.61
RandomWorst × Runtime	0.59	0.41	−0.20	1.38
WorstRelated × $\text{Customers}^{\frac{1}{3}}$	−1.41	0.46	−2.30	−0.53
WorstRelated × Avg service time	0.71	0.52	−0.32	1.71
WorstRelated × Avg time window width	−0.03	0.31	−0.64	0.57
WorstRelated × Avg demand	0.52	1.74	−2.85	3.92
WorstRelated × Runtime	0.37	0.35	−0.30	1.06
RandomRelated × $\text{Customers}^{\frac{1}{3}}$	1.49	0.49	0.54	2.46
RandomRelated × Avg service time	0.02	0.57	−1.09	1.14
RandomRelated × Avg time window width	0.09	0.34	−0.57	0.76
RandomRelated × Avg demand	3.17	1.84	−0.40	6.73
RandomRelated × Runtime	0.09	0.38	−0.65	0.84
Max%Removed × $\text{Customers}^{\frac{1}{3}}$	−0.17	0.01	−0.19	−0.14
Max%Removed × Avg service time	−0.02	0.02	−0.05	0.01
Max%Removed × Avg time window width	−0.01	0.01	−0.02	0.01
Max%Removed × Avg demand	−0.05	0.05	−0.15	0.05

Max%Removed × Runtime	0.02	0.01	−0.001	0.04
Cooling rate × Customers$^{\frac{1}{3}}$	0.33	2.27	−4.15	4.80
Cooling rate × Avg service time	0.94	2.66	−4.26	6.21
Cooling rate × Avg time window width	−1.79	1.61	−4.95	1.39
Cooling rate × Avg demand	−6.89	8.78	−24.12	10.54
Cooling rate × Runtime	0.66	1.76	−2.84	4.12
Start temperature control parameter × Customers$^{\frac{1}{3}}$	−5.43	5.02	−15.28	4.40
Start temperature control parameter × Avg service time	3.41	5.79	−7.91	14.73
Start temperature control parameter × Avg time window width	−0.32	3.53	−7.26	6.58
Start temperature control parameter × Avg demand	27.73	18.97	−9.52	65.18
Start temperature control parameter × Runtime	0.61	3.86	−6.97	8.07
Determinism parameter × Customers$^{\frac{1}{3}}$	−0.01	0.01	−0.04	0.02
Determinism parameter × Avg service time	0.004	0.02	−0.03	0.04
Determinism parameter × Avg time window width	−0.0004	0.01	−0.02	0.02
Determinism parameter × Avg demand	0.01	0.06	−0.10	0.12
Determinism parameter × Runtime	−0.02	0.01	−0.04	0.01
Noise parameter × Customers$^{\frac{1}{3}}$	0.03	0.57	−1.09	1.15
Noise parameter × Avg service time	−0.37	0.66	−1.66	0.90
Noise parameter × Avg time window width	−0.54	0.40	−1.33	0.24
Noise parameter × Avg demand	−3.19	2.16	−7.35	1.03
Noise parameter × Runtime	−0.22	0.43	−1.06	0.62
Greedy × Random × Max%Removed	−1.59	0.40	−2.39	−0.80
Greedy × Worst × Max%Removed	−0.77	0.40	−1.56	0.02
Greedy × Related × Max%Removed	0.91	0.42	0.09	1.74
Greedy × RandomWorst × Max%Removed	−1.75	0.40	−2.54	−0.97
Greedy × WorstRelated × Max%Removed	−0.12	0.41	−0.91	0.67
Greedy × RandomRelated × Max%Removed	0.08	0.42	−0.75	0.90
Regret2 × Random × Max%Removed	−0.24	0.41	−1.04	0.57
Regret2 × Worst × Max%Removed	−0.03	0.43	−0.86	0.81
Regret2 × Related × Max%Removed	0.08	0.42	−0.74	0.91
Regret2 × RandomWorst × Max%Removed	−0.02	0.42	−0.83	0.79
Regret2 × WorstRelated × Max%Removed	−0.31	0.41	−1.11	0.50
Regret2 × RandomRelated × Max%Removed	0.20	0.42	−0.61	1.01

Table B.8: Regression Table Chapter 6 — training data with 200 instances

Variable	Estimate	Est.Error	l-95% CI	u-95% CI
Intercept	4,481.95	139.19	4,212.48	4,760.17
Greedy	−122.60	3.87	−130.25	−115.04
Regret2	12.39	2.62	7.29	17.52
Random	8.86	2.67	3.67	14.13
Worst	−22.86	3.05	−28.85	−16.80
Related	−21.35	2.78	−26.82	−15.95
RandomWorst	−4.19	2.82	−9.72	1.27
WorstRelated	−7.10	2.63	−12.24	−1.98
RandomRelated	7.20	2.66	1.98	12.47
Max%Removed	−0.03	0.16	−0.35	0.29
Cooling rate	21.70	7.79	6.21	36.84
Start temperature control parameter	−8.57	15.49	−38.84	21.60
Noise parameter	−11.19	2.59	−16.28	−6.09
Determinism parameter	0.27	0.13	0.02	0.51
Max%Removed^2	−0.01	0.003	−0.02	−0.01
Cooling rate^2	101.84	156.85	−207.68	409.78
Start temperature control parameter^2	1,149.01	631.41	−90.97	2,389.64
Noise parameter^2	26.27	5.50	15.51	37.08
Determinism parameter^2	0.0000	0.01	−0.01	0.01
Customers$^{\frac{1}{3}}$	−479.46	31.53	−543.12	−418.23

Avg service time	−71.36	33.82	−138.64	−5.37
Avg time window width	40.24	21.33	−1.63	82.71
Avg demand	−275.12	150.21	−567.12	21.65
Runtime	−1.86	22.17	−46.60	40.55
Greedy1 × Random	−26.45	3.77	−33.85	−19.17
Greedy1 × Worst	−58.73	3.79	−66.20	−51.38
Greedy1 × Related	37.26	3.76	29.87	44.59
Greedy1 × RandomWorst	−40.84	3.75	−48.19	−33.52
Greedy1 × WorstRelated	−0.02	3.72	−7.29	7.15
Greedy1 × RandomRelated	16.63	3.73	9.35	23.93
Regret21 × Random	−7.65	3.68	−14.87	−0.44
Regret21 × Worst	1.81	3.71	−5.43	9.04
Regret21 × Related	−4.30	3.69	−11.56	2.89
Regret21 × RandomWorst	1.42	3.75	−5.98	8.69
Regret21 × WorstRelated	−0.58	3.66	−7.71	6.68
Regret21 × RandomRelated	−6.57	3.68	−13.72	0.62
Cooling rate × Start temperature control parameter	217.38	288.49	−341.15	781.97
Random × Determinism parameter	−0.38	0.17	−0.71	−0.04
Worst × Determinism parameter	−1.01	0.18	−1.35	−0.65
Related × Determinism parameter	−1.37	0.17	−1.71	−1.02
RandomWorst × Determinism parameter	−0.33	0.18	−0.67	0.03
WorstRelated × Determinism parameter	−0.28	0.17	−0.62	0.06
RandomRelated × Determinism parameter	0.19	0.17	−0.15	0.54
Greedy × Noise parameter	−25.75	3.52	−32.63	−18.91
Regret2 × Noise parameter	3.44	3.44	−3.28	10.21
Random × Max%Removed	0.46	0.22	0.02	0.89
Worst × Max%Removed	0.93	0.23	0.49	1.38
Related × Max%Removed	−1.37	0.23	−1.81	−0.93
RandomWorst × Max%Removed	0.76	0.23	0.32	1.21
WorstRelated × Max%Removed	−0.18	0.23	−0.62	0.26
RandomRelated × Max%Removed	−0.11	0.23	−0.56	0.33
Greedy × Max%Removed	−1.96	0.23	−2.41	−1.51
Regret2 × Max%Removed	−0.005	0.23	−0.45	0.44
$\text{Customers}^{\frac{1}{3}}$ × Runtime	−3.01	4.86	−12.48	6.88
Greedy × $\text{Customers}^{\frac{1}{3}}$	−18.70	0.67	−20.02	−17.39
Greedy × Avg service time	1.04	0.73	−0.41	2.47
Greedy × Avg time window width	−2.08	0.46	−2.98	−1.19
Greedy × Avg demand	−3.94	3.24	−10.39	2.40
Greedy × Runtime	1.53	0.48	0.57	2.46
Regret2 × $\text{Customers}^{\frac{1}{3}}$	1.68	0.23	1.24	2.13
Regret2 × Avg service time	−0.43	0.25	−0.91	0.06
Regret2 × Avg time window width	0.08	0.15	−0.22	0.38
Regret2 × Avg demand	−0.26	1.11	−2.43	1.91
Regret2 × Runtime	−0.44	0.17	−0.76	−0.11
Random × $\text{Customers}^{\frac{1}{3}}$	0.81	0.36	0.11	1.53
Random × Avg service time	0.33	0.39	−0.44	1.10
Random × Avg time window width	0.22	0.24	−0.27	0.69
Random × Avg demand	−3.74	1.73	−7.11	−0.31
Random × Runtime	0.12	0.26	−0.40	0.64
Worst × $\text{Customers}^{\frac{1}{3}}$	−2.07	0.50	−3.05	−1.09
Worst × Avg service time	0.78	0.54	−0.27	1.84
Worst × Avg time window width	−0.33	0.34	−0.99	0.34
Worst × Avg demand	−4.05	2.47	−8.87	0.77
Worst × Runtime	0.13	0.36	−0.59	0.83
Related × $\text{Customers}^{\frac{1}{3}}$	−4.14	0.39	−4.90	−3.38
Related × Avg service time	0.30	0.42	−0.52	1.13
Related × Avg time window width	−0.15	0.26	−0.67	0.37
Related × Avg demand	−2.63	1.89	−6.34	1.08
Related × Runtime	0.78	0.28	0.22	1.34

RandomWorst × Customers$^{\frac{1}{3}}$	−0.36	0.42	−1.18	0.48
RandomWorst × Avg service time	0.64	0.46	−0.25	1.54
RandomWorst × Avg time window width	0.03	0.28	−0.52	0.58
RandomWorst × Avg demand	−4.08	2.01	−8.00	−0.14
RandomWorst × Runtime	0.20	0.30	−0.39	0.78
WorstRelated × Customers$^{\frac{1}{3}}$	−1.57	0.35	−2.26	−0.89
WorstRelated × Avg service time	−0.11	0.37	−0.85	0.63
WorstRelated × Avg time window width	0.28	0.23	−0.17	0.74
WorstRelated × Avg demand	−3.67	1.66	−6.92	−0.41
WorstRelated × Runtime	0.23	0.25	−0.26	0.71
RandomRelated × Customers$^{\frac{1}{3}}$	0.65	0.36	−0.06	1.35
RandomRelated × Avg service time	0.02	0.39	−0.74	0.77
RandomRelated × Avg time window width	0.19	0.24	−0.28	0.66
RandomRelated × Avg demand	−0.83	1.72	−4.19	2.54
RandomRelated × Runtime	0.28	0.26	−0.24	0.79
Max%Removed × Customers$^{\frac{1}{3}}$	−0.19	0.01	−0.22	−0.17
Max%Removed × Avg service time	0.01	0.01	−0.02	0.03
Max%Removed × Avg time window width	−0.02	0.01	−0.03	−0.002
Max%Removed × Avg demand	0.03	0.06	−0.08	0.14
Max%Removed × Runtime	0.03	0.01	0.01	0.04
Cooling rate × Customers$^{\frac{1}{3}}$	3.83	1.77	0.38	7.30
Cooling rate × Avg service time	−3.47	1.92	−7.23	0.30
Cooling rate × Avg time window width	−1.27	1.19	−3.62	1.07
Cooling rate × Avg demand	0.88	8.47	−15.83	17.26
Cooling rate × Runtime	−0.99	1.28	−3.52	1.53
Start temperature control parameter × Customers$^{\frac{1}{3}}$	0.71	3.50	−6.15	7.63
Start temperature control parameter × Avg service time	1.08	3.80	−6.39	8.54
Start temperature control parameter × Avg time window width	0.49	2.32	−4.02	5.07
Start temperature control parameter × Avg demand	−11.36	17.01	−44.69	22.11
Start temperature control parameter × Runtime	1.70	2.56	−3.30	6.70
Determinism parameter × Customers$^{\frac{1}{3}}$	−0.05	0.01	−0.07	−0.03
Determinism parameter × Avg service time	0.002	0.01	−0.02	0.03
Determinism parameter × Avg time window width	0.004	0.01	−0.01	0.02
Determinism parameter× Avg demand	−0.01	0.05	−0.12	0.09
Determinism parameter × Runtime	0.02	0.01	0.0004	0.03
Noise parameter × Customers$^{\frac{1}{3}}$	−2.04	0.36	−2.76	−1.34
Noise parameter × Avg service time	0.18	0.40	−0.60	0.96
Noise parameter × Avg time window width	−0.41	0.24	−0.88	0.07
Noise parameter × Avg demand	−0.46	1.77	−3.92	3.01
Noise parameter × Runtime	0.03	0.26	−0.49	0.55
Greedy × Random × Max%Removed	−1.78	0.33	−2.42	−1.14
Greedy × Worst × Max%Removed	−0.85	0.33	−1.50	−0.21
Greedy × Related × Max%Removed	1.05	0.33	0.41	1.69
Greedy × RandomWorst × Max%Removed	−1.60	0.32	−2.23	−0.96
Greedy × WorstRelated × Max%Removed	0.12	0.32	−0.51	0.75
Greedy × RandomRelated × Max%Removed	0.05	0.33	−0.58	0.69
Regret2 × Random × Max%Removed	0.21	0.32	−0.42	0.84
Regret2 × Worst × Max%Removed	−0.10	0.32	−0.73	0.54
Regret2 × Related × Max%Removed	0.11	0.32	−0.53	0.73
Regret2 × RandomWorst × Max%Removed	−0.25	0.33	−0.90	0.39
Regret2 × WorstRelated × Max%Removed	−0.33	0.32	−0.96	0.29
Regret2 × RandomRelated × Max%Removed	0.02	0.32	−0.61	0.65

Table B.9: Regression Table Chapter 6 — training data with 400 instances

Variable	Estimate	Est.Error	l-95% CI	u-95% CI
Intercept	4,234.10	96.59	4,048.98	4,421.92
Greedy	−129.60	2.67	−134.85	−124.40
Regret2	8.71	1.79	5.19	12.23
Random	9.42	1.81	5.87	12.96
Worst	−22.92	2.22	−27.29	−18.63
Related	−26.76	1.98	−30.66	−22.86
RandomWorst	−4.26	1.95	−8.09	−0.43
WorstRelated	−14.21	1.94	−17.95	−10.39
RandomRelated	3.35	1.90	−0.40	7.12
Max%Removed	−0.01	0.11	−0.22	0.20
Cooling rate	27.34	5.43	16.69	38.03
Start temperature control parameter	2.15	12.60	−22.44	27.07
Noise parameter	−8.19	1.77	−11.66	−4.70
Determinism parameter	0.26	0.09	0.10	0.43
Max%Removed^2	−0.02	0.002	−0.02	−0.01
Cooling rate^2	51.39	109.05	−162.66	265.12
Start temperature control parameter^2	24.14	442.57	−834.70	895.54
Noise parameter^2	24.30	3.78	16.93	31.65
Determinism parameter^2	0.002	0.004	−0.01	0.01
Customers$^{\frac{1}{3}}$	−458.95	20.93	−499.10	−416.40
Avg service time	−58.38	24.11	−103.98	−11.23
Avg time window width	42.37	15.14	13.65	72.77
Avg demand	533.46	91.32	351.00	709.62
Runtime	−2.86	16.09	−33.76	29.51
Greedy × Random	−20.77	2.55	−25.90	−15.77
Greedy × Worst	−62.77	2.60	−67.88	−57.71
Greedy × Related	49.97	2.59	44.89	55.00
Greedy × RandomWorst	−44.92	2.62	−50.05	−39.74
Greedy × WorstRelated	6.92	2.60	1.80	12.04
Greedy × RandomRelated	22.56	2.56	17.56	27.61
Regret2 × Random	−3.45	2.58	−8.51	1.56
Regret2 × Worst	4.17	2.54	−0.82	9.12
Regret2 × Related	1.93	2.57	−3.12	7.00
Regret2 × RandomWorst	3.29	2.58	−1.79	8.35
Regret2 × WorstRelated	5.41	2.57	0.35	10.38
Regret2 × RandomRelated	0.84	2.55	−4.10	5.77
Cooling rate × Start temperature control parameter	53.85	202.57	−340.25	446.36
Random × Determinism parameter	−0.27	0.12	−0.51	−0.03
Worst × Determinism parameter	−0.65	0.12	−0.89	−0.41
Related × Determinism parameter	−1.11	0.12	−1.35	−0.87
RandomWorst × Determinism parameter	−0.35	0.12	−0.59	−0.11
WorstRelated × Determinism parameter	−0.36	0.12	−0.60	−0.12
RandomRelated × Determinism parameter	0.12	0.12	−0.11	0.37
Greedy × Noise parameter	−24.53	2.40	−29.28	−19.79
Regret2 × Noise parameter	−0.19	2.39	−4.82	4.52
Random × Max%Removed	0.39	0.15	0.09	0.69
Worst × Max%Removed	0.54	0.16	0.24	0.85
Related × Max%Removed	−1.67	0.16	−1.98	−1.38
RandomWorst × Max%Removed	0.58	0.15	0.28	0.88
WorstRelated × Max%Removed	−0.54	0.15	−0.85	−0.24
RandomRelated × Max%Removed	−0.30	0.15	−0.59	0.01
Greedy1 × Max%Removed	−1.96	0.15	−2.26	−1.67
Regret21 × Max%Removed	−0.05	0.15	−0.34	0.25
Customers$^{\frac{1}{3}}$ × Runtime	3.56	3.34	−2.94	10.14
Greedy × Customers$^{\frac{1}{3}}$	−18.25	0.45	−19.13	−17.38
Greedy × Avg service time	2.79	0.49	1.84	3.77

Greedy × Avg time window width	−3.17	0.32	−3.79	−2.55
Greedy × Avg demand	0.97	1.89	−2.74	4.60
Greedy × Runtime	1.91	0.34	1.24	2.58
Regret2 × Customers$^{\frac{1}{3}}$	1.64	0.17	1.31	1.97
Regret2 × Avg service time	−0.32	0.18	−0.66	0.04
Regret2 × Avg time window width	0.25	0.12	0.02	0.48
Regret2 × Avg demand	−0.55	0.70	−1.92	0.82
Regret2 × Runtime	−0.38	0.12	−0.61	−0.14
Random × Customers$^{\frac{1}{3}}$	0.85	0.24	0.38	1.31
Random × Avg service time	0.19	0.25	−0.31	0.70
Random × Avg time window width	−0.01	0.17	−0.34	0.32
Random × Avg demand	0.46	1.00	−1.53	2.41
Random × Runtime	−0.33	0.17	−0.68	0.004
Worst × Customers$^{\frac{1}{3}}$	−1.73	0.36	−2.46	−1.02
Worst × Avg service time	1.10	0.39	0.34	1.86
Worst × Avg time window width	−1.12	0.26	−1.61	−0.61
Worst × Avg demand	2.83	1.54	−0.18	5.80
Worst × Runtime	0.55	0.27	0.03	1.08
Related × Customers$^{\frac{1}{3}}$	−3.10	0.29	−3.68	−2.53
Related × Avg service time	−0.01	0.31	−0.61	0.60
Related × Avg time window width	−0.45	0.20	−0.84	−0.05
Related × Avg demand	1.49	1.22	−0.88	3.87
Related × Runtime	0.58	0.21	0.16	1.00
RandomWorst × Customers$^{\frac{1}{3}}$	−0.43	0.28	−0.97	0.12
RandomWorst × Avg service time	0.35	0.30	−0.24	0.95
RandomWorst × Avg demand	0.31	1.18	−1.97	2.65
RandomWorst × Runtime	0.12	0.21	−0.29	0.52
WorstRelated × Customers$^{\frac{1}{3}}$	−1.08	0.28	−1.64	−0.55
WorstRelated × Avg service time	−0.07	0.30	−0.66	0.50
WorstRelated × Avg time window width	−0.10	0.20	−0.48	0.29
WorstRelated × Avg demand	1.48	1.16	−0.81	3.72
WorstRelated × Runtime	0.36	0.21	−0.04	0.77
RandomRelated × Customers$^{\frac{1}{3}}$	1.13	0.27	0.60	1.67
RandomRelated × Avg service time	−0.68	0.29	−1.25	−0.11
RandomRelated × Avg time window width	0.04	0.19	−0.34	0.41
RandomRelated × Avg demand	1.61	1.14	−0.56	3.85
RandomRelated × Runtime	0.20	0.20	−0.18	0.59
Max%Removed × Customers$^{\frac{1}{3}}$	−0.19	0.01	−0.20	−0.17
Max%Removed × Avg service time	−0.003	0.01	−0.02	0.01
Max%Removed × Avg time window width	−0.01	0.01	−0.02	−0.002
Max%Removed × Avg demand	0.01	0.03	−0.05	0.08
Max%Removed × Runtime	0.02	0.01	0.01	0.03
Cooling rate × Customers$^{\frac{1}{3}}$	0.68	1.24	−1.76	3.10
Cooling rate × Avg service time	−3.68	1.29	−6.21	−1.18
Cooling rate × Avg time window width	0.41	0.85	−1.26	2.08
Cooling rate × Avg demand	2.71	4.98	−7.24	12.40
Cooling rate × Runtime	0.22	0.87	−1.48	1.92
Start temperature control parameter × Customers$^{\frac{1}{3}}$	1.01	2.79	−4.44	6.50
Start temperature control parameter × Avg service time	−0.87	3.02	−6.79	5.12
Start temperature control parameter × Avg time window width	−2.49	1.98	−6.36	1.39
Start temperature control parameter × Avg demand	21.71	11.92	−1.52	44.99
Start temperature control parameter × Runtime	−2.22	2.07	−6.30	1.86
Determinism parameter × Customers$^{\frac{1}{3}}$	−0.01	0.01	−0.03	−0.0003
Determinism parameter × Avg service time	0.01	0.01	−0.01	0.02
Determinism parameter × Avg time window width	0.004	0.01	−0.01	0.01
Determinism parameter × Avg demand	0.04	0.03	−0.03	0.10
Determinism parameter × Runtime	−0.002	0.01	−0.01	0.01
Noise parameter × Customers$^{\frac{1}{3}}$	−1.24	0.25	−1.72	−0.76

Noise parameter × Avg service time	−0.02	0.26	−0.53	0.50
Noise parameter × Avg time window width	−0.49	0.17	−0.82	−0.15
Noise parameter × Avg demand	1.72	1.01	−0.26	3.71
Noise parameter × Runtime	0.36	0.18	0.01	0.72
Greedy × Random × Max%Removed	−1.48	0.22	−1.90	−1.06
Greedy × Worst × Max%Removed	−1.24	0.22	−1.68	−0.80
Greedy × Related × Max%Removed	1.40	0.22	0.96	1.83
Greedy × RandomWorst × Max%Removed	−1.32	0.22	−1.75	−0.89
Greedy × WorstRelated × Max%Removed	0.58	0.22	0.15	1.01
Greedy × RandomRelated × Max%Removed	−0.04	0.22	−0.47	0.39
Regret2 × Random × Max%Removed	0.22	0.22	−0.21	0.64
Regret2 × Worst × Max%Removed	0.27	0.22	−0.18	0.71
Regret2 × Related × Max%Removed	0.07	0.22	−0.37	0.51
Regret2 × RandomWorst × Max%Removed	0.14	0.21	−0.28	0.56
Regret2 × WorstRelated × Max%Removed	0.17	0.22	−0.26	0.60
Regret2 × RandomRelated × Max%Removed	0.20	0.22	−0.22	0.62

Appendix C

Residual Plots

To assess whether a regression model is in line with the underlying assumptions of independence, normality and homoscedasticity of the error terms, the fitted values against the residual values are plotted. If the points in these plots show a random pattern, the analysis can proceed with the current model, otherwise a more suitable model has to be found. In this appendix the residual plots are provided for all regression models used throughout this thesis.

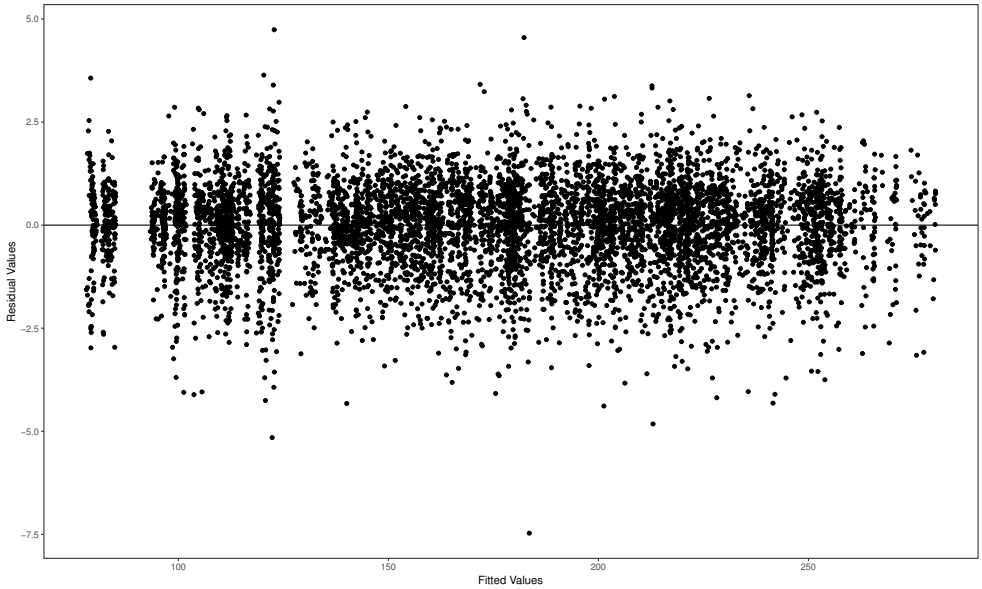


Figure C.1: Fitted versus residual values for regression Table 4.3.

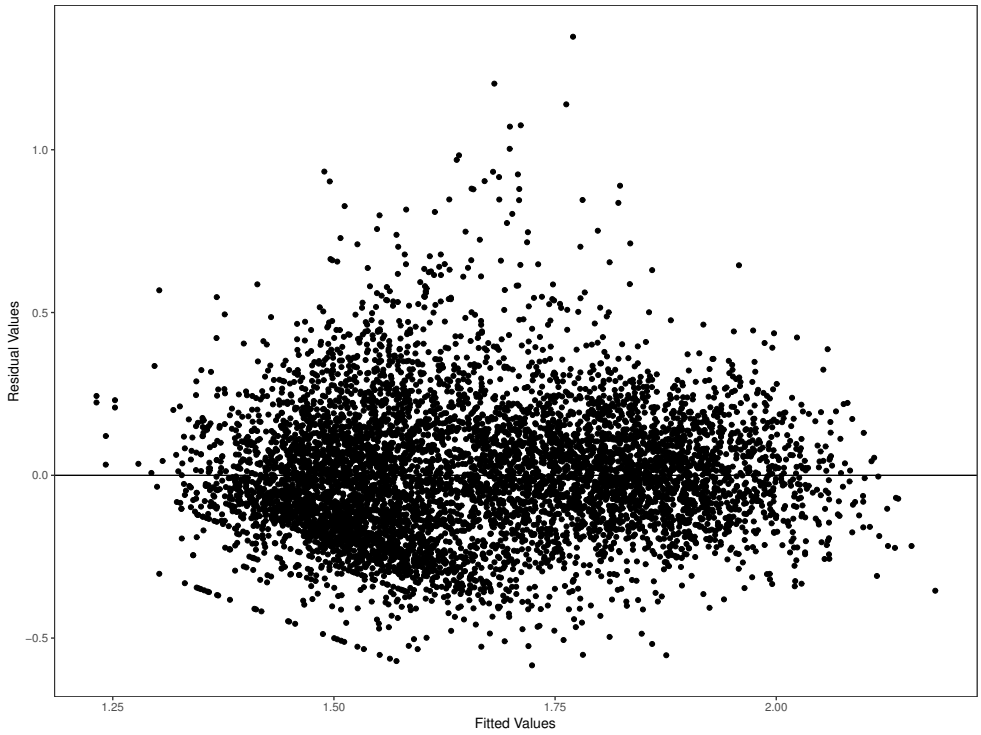


Figure C.2: Fitted versus residual values for regression Table 4.4 (Hypothesis 1).

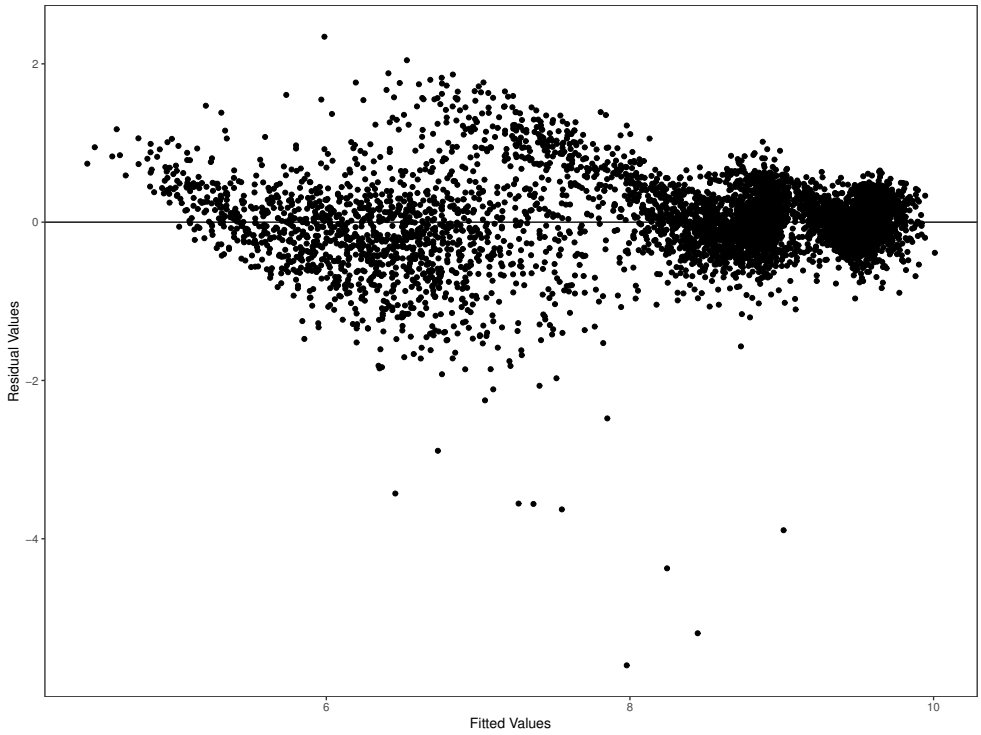


Figure C.3: Fitted versus residual values for regression Table 4.5 (Hypothesis 2).

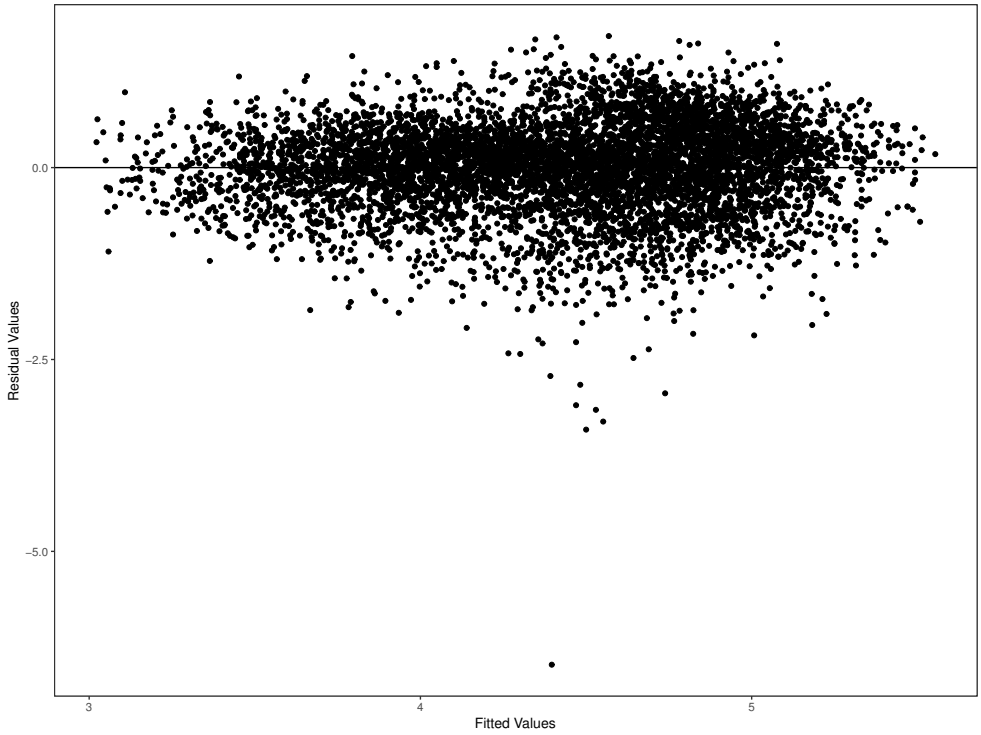


Figure C.4: Fitted versus residual values for regression Table B.2.

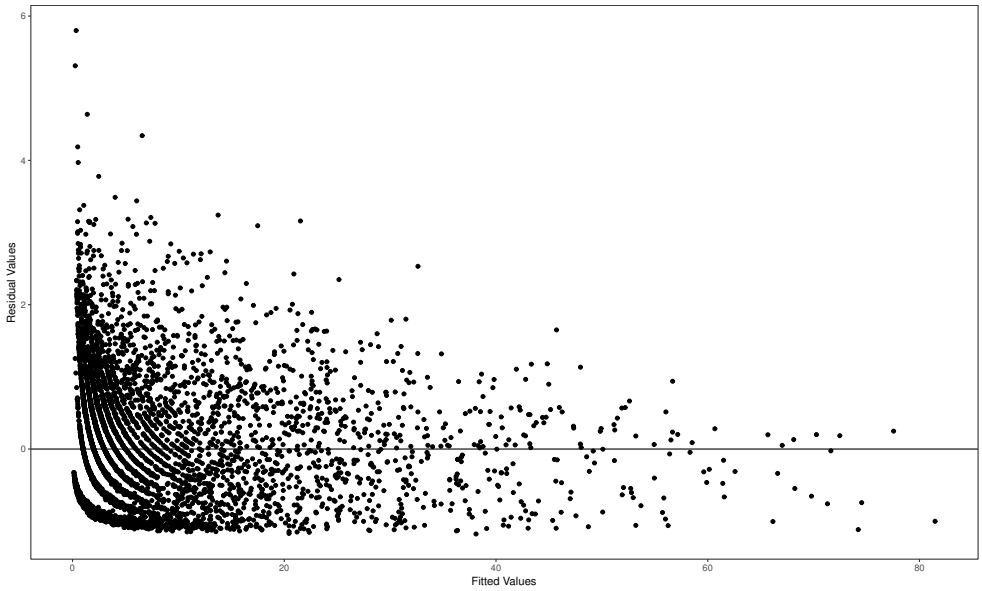


Figure C.5: Fitted versus residual values for regression Table 4.6 (Hypothesis 3).

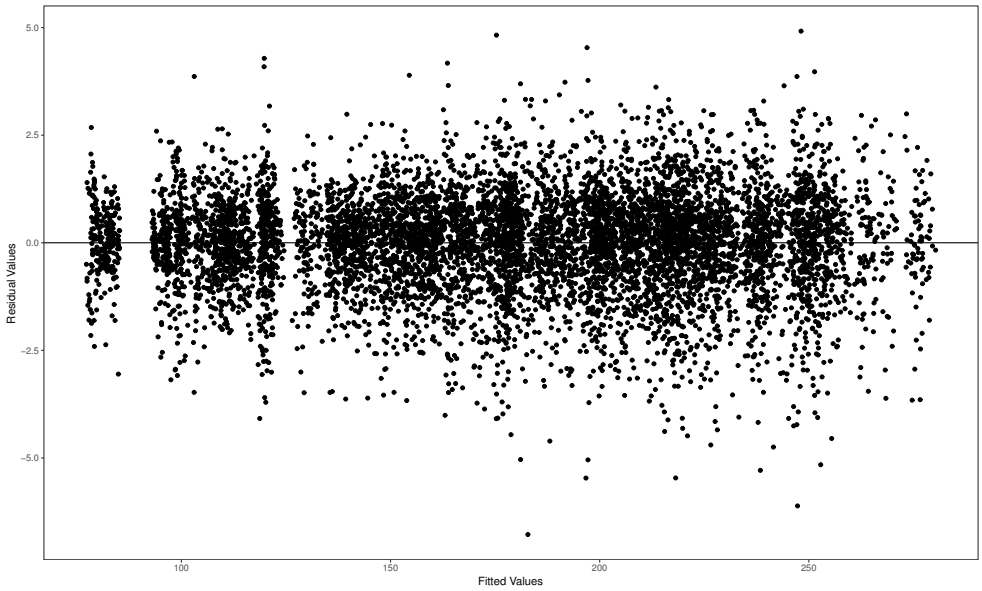


Figure C.6: Fitted versus residual values for regression Table 4.7.

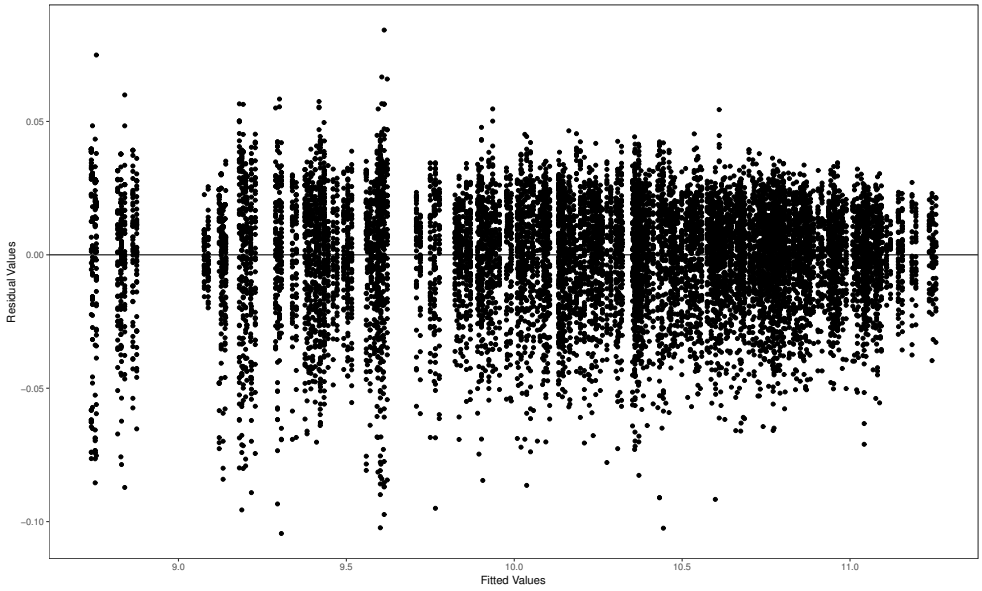


Figure C.7: Fitted versus residual values for regression Table 4.8 (Hypothesis 4).

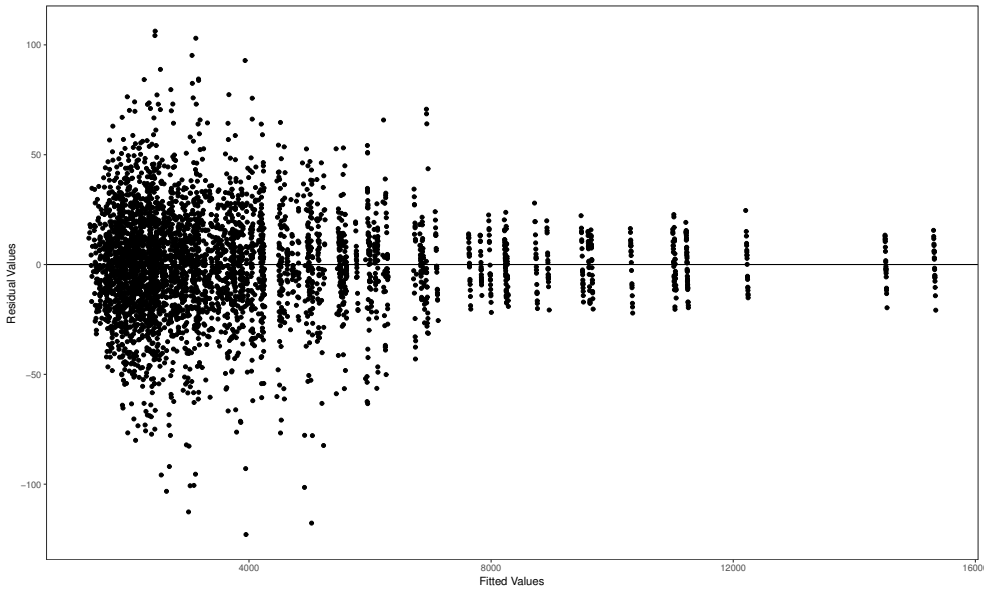


Figure C.8: Fitted versus residual values for regression Table B.3.

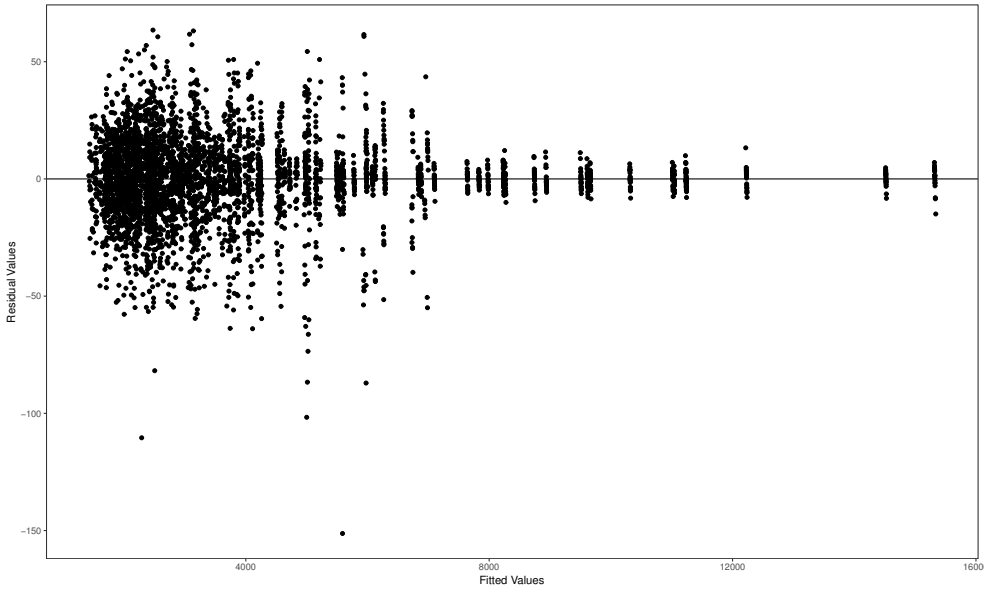


Figure C.9: Fitted versus residual values for regression Table B.4.

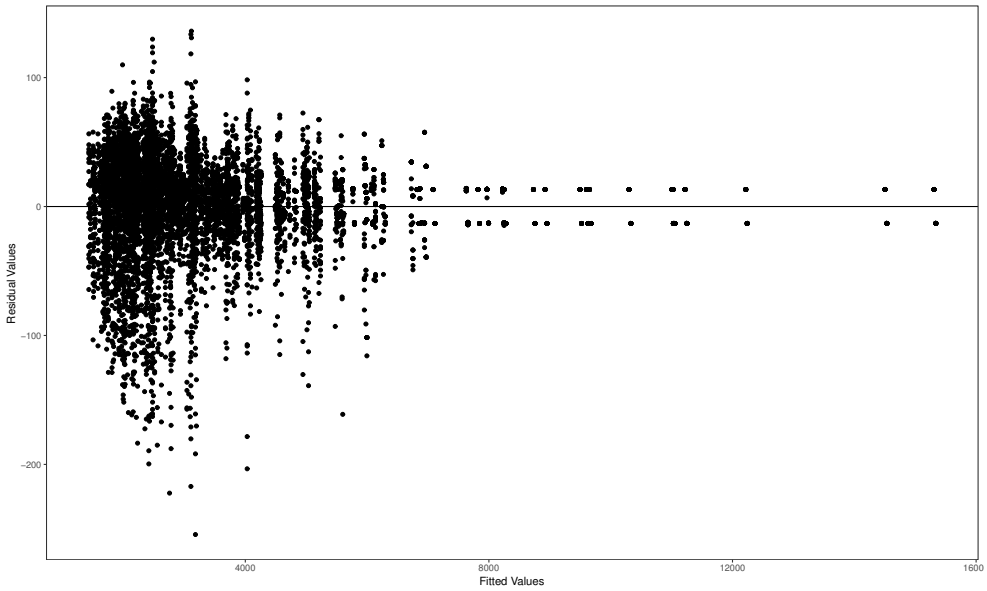


Figure C.10: Fitted versus residual values for regression Table 4.9.

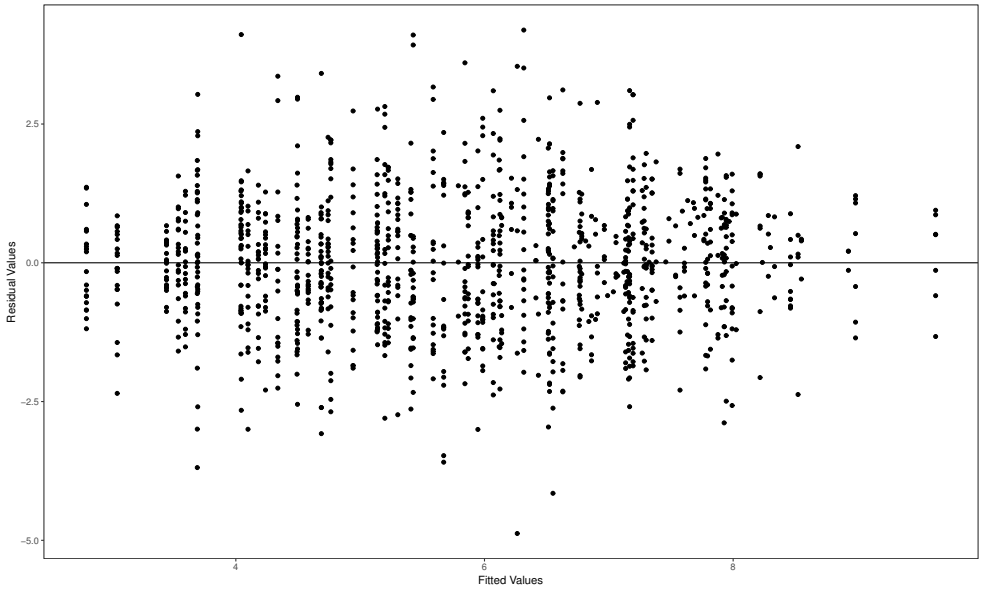


Figure C.11: Fitted versus residual values for regression Table 4.10.

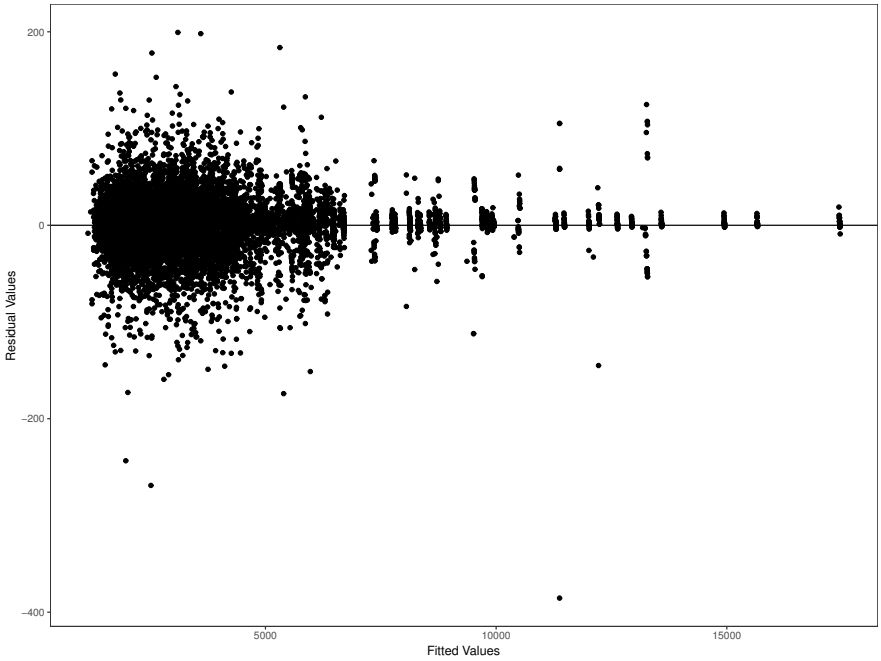


Figure C.12: Fitted versus residual values for regression Table 5.2.

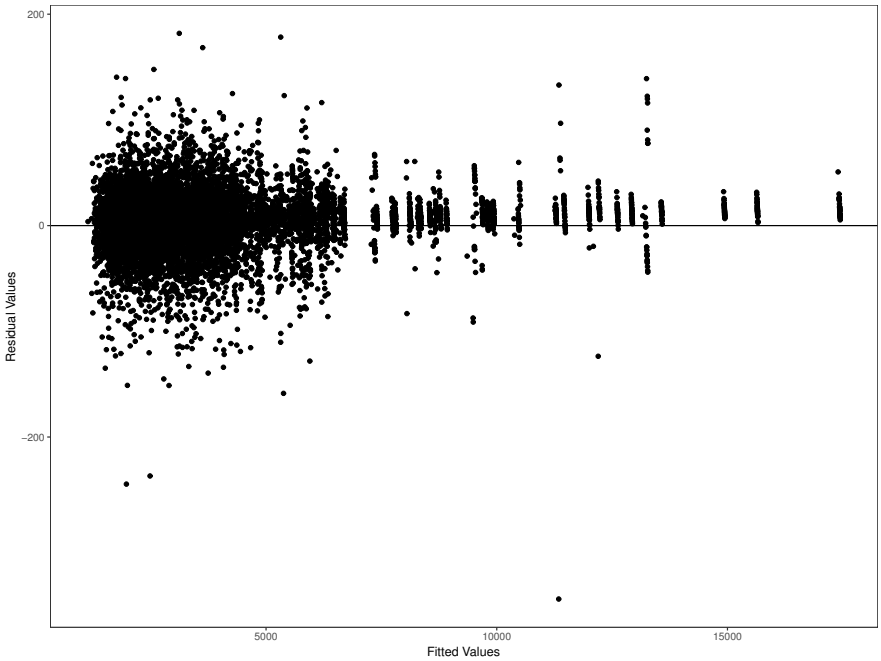


Figure C.13: Fitted versus residual values for regression Table B.6.

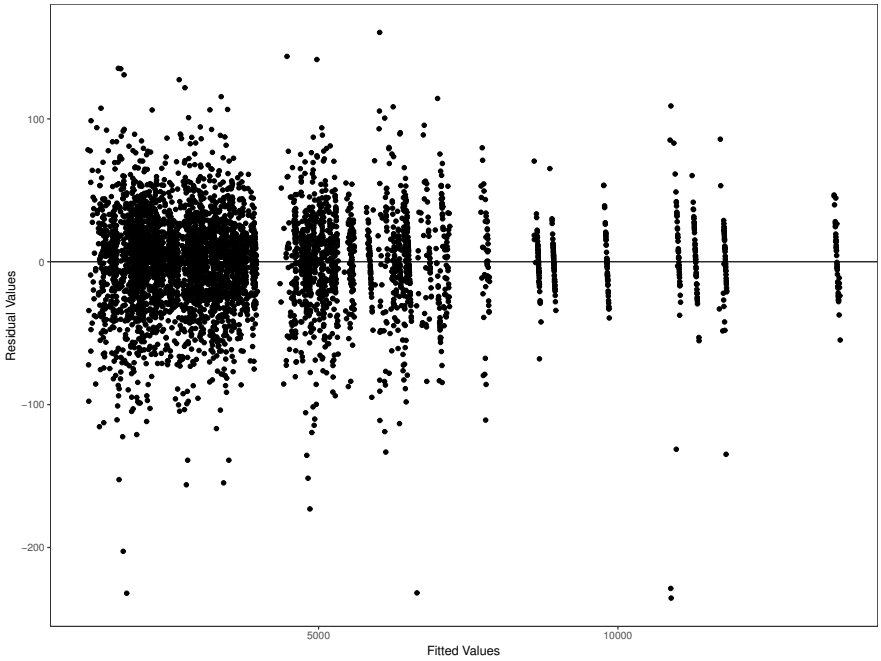


Figure C.14: Fitted versus residual values for regression Table B.7.

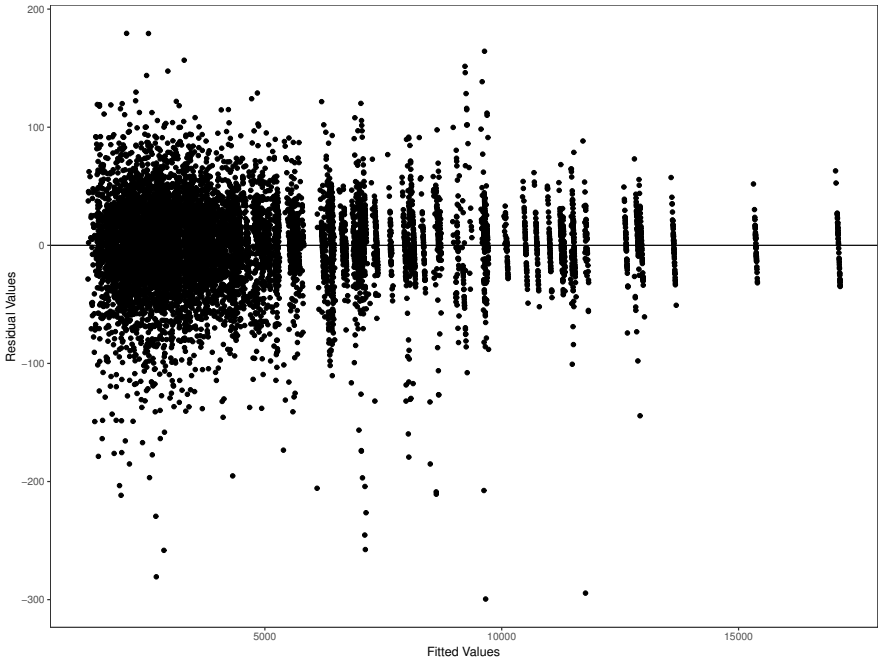


Figure C.15: Fitted versus residual values for regression Table B.8.

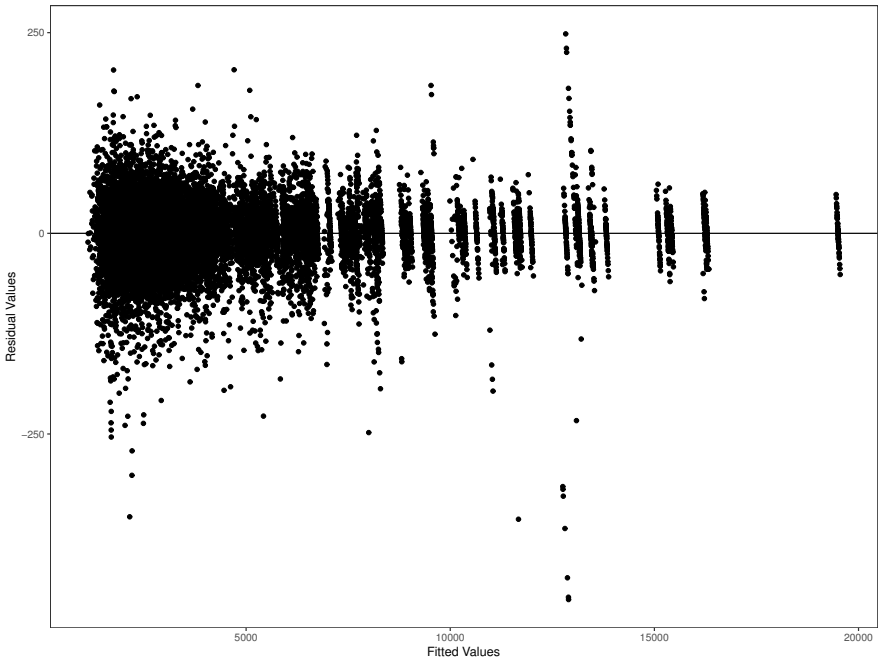


Figure C.16: Fitted versus residual values for regression Table B.9.

Bibliography

Aarts, E., Korst, J., Michiels, W., 2005. Simulated Annealing. In: Burke, E. K., Kendall, G. (Eds.), Search Methodologies. Springer US, pp. 187–210.

Abdoli, B., MirHassani, S. A., Hooshmand, F., 2017. Model and algorithm for bi-fuel vehicle routing problem to reduce ghg emissions. Environmental Science and Pollution Research 24 (27), 21610–21624.

Adenso-Díaz, B., Laguna, M., Feb. 2006. Fine-Tuning of Algorithms Using Fractional Experimental Designs and Local Search. Operations Research 54 (1), 99–114.

Ahuja, R. K., Orlin, J. B., 1996. Use of representative operation counts in computational testing of algorithms. INFORMS Journal on Computing 8 (3), 318–330.

Amini, M. M., Racer, M., Jul. 1994. A Rigorous Computational Comparison of Alternative Solution Methods for the Generalized Assignment Problem. Management Science 40 (7), 868–890.

Ansótegui, C., Sellmann, M., Tierney, K., 2009. A Gender-Based Genetic Algorithm for the Automatic Configuration of Algorithms. In: Gent, I. P. (Ed.), Principles and Practice of Constraint Programming - CP 2009. No. 5732 in Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 142–157.

Assis, L. P., Maravilha, A. L., Vivas, A., Campelo, F., Ramírez, J. A., 2013. Multiobjective vehicle routing problem with fixed delivery and optional collections. Optimization Letters 7 (7), 1419–1431.

Audet, C., Orban, D., 2006. Finding optimal algorithmic parameters using derivative-free optimization. SIAM Journal on Optimization 17 (3), 642–664.

Balaprakash, P., Birattari, M., Stutzle, T., 2007. Improvement strategies for the f-race algorithm: Sampling design and iterative refinement. Hybrid Metaheuristics 4771, 108–122.

Banerjee, A., Chaudhury, S., 2010. Statistics without tears: Populations and samples. Industrial psychiatry journal 19 (1), 60.

Barr, R. S., Golden, B. L., Kelly, J. P., Resende, M. G. C., Jr, W. R. S., Sep. 1995. Designing and reporting on computational experiments with heuristic methods. Journal of Heuristics 1 (1), 9–32.

Bartz-Beielstein, T., Lasarczyk, C., Preuss, M., 2010. The sequential parameter optimization toolbox. Experimental methods for the analysis of optimization algorithms, 337–360.

Bartz-Beielstein, T., Lasarczyk, C. W., Preuß, M., 2005. Sequential parameter optimization. In: Evolutionary Computation, 2005. The 2005 IEEE Congress on. Vol. 1. IEEE, pp. 773–780.

Bartz-Beielstein, T., Parsopoulos, K. E., Vrahatis, M. N., 2004. Design and analysis of optimization algorithms using computational statistics. Applied Numerical Analysis & Computational Mathematics 1 (2), 413–433.

Bartz-Beielstein, T., Preuß, M., 2014. Experimental analysis of optimization algorithms: Tuning and beyond. In: Theory and Principled Methods for the Design of Metaheuristics. Springer, pp. 205–245.

Bates, D., Mächler, M., Bolker, B., Walker, S., 2015. Fitting linear mixed-effects models using lme4. Journal of Statistical Software 67 (1), 1–48.

Battiti, R., Tecchiolli, G., 1994. The reactive tabu search. ORSA journal on computing 6 (2), 126–140.

Bertsimas, D. J., Simchi-Levi, D., Apr. 1996. A new generation of vehicle routing research: Robust Algorithms, Addressing Uncertainty. Operations Research 44 (2), 286.

Birattari, M., 2002. A racing algorithm for configuring metaheuristics. In: Proceedings of the Genetic and Evolutionary Computation Conference. Morgan Kaufmann Publishers, pp. 11–18.

Birattari, M., 2009. Tuning Metaheuristics. Vol. 197 of Studies in Computational Intelligence. Springer Berlin Heidelberg, Berlin, Heidelberg.

Birattari, M., Dorigo, M., 2007. How to assess and report the performance of a stochastic algorithm on a benchmark problem: Mean or best result on a number of runs? Optimization Letters 1 (3), 309–311.

Box, G. E. P., Hunter, J. S., Hunter, W. G., May 2005. Statistics for Experimenters: Design, Innovation, and Discovery. Wiley-Interscience.

Brambor, T., Clark, W. R., Golder, M., 2006. Understanding interaction models: Improving empirical analyses. Political Analysis 14 (1), 63–82.

Bräysy, O., Gendreau, M., Feb. 2005. Vehicle Routing Problem with Time Windows, Part I: Route Construction and Local Search Algorithms. Transportation Science 39 (1), 104–118.

Breiman, L., 2001. Random forests. Machine Learning 45 (1), 5–32.

Brus, D., De Gruijter, J., 1997. Random sampling or geostatistical modelling? choosing between design-based and model-based sampling strategies for soil (with discussion). Geoderma 80 (1), 1–44.

Bürkner, P.-C., 2017. brms: An R package for bayesian multilevel models using Stan. Journal of Statistical Software 80 (1), 1–28.

Busing, F., 1993. Distribution characteristics of variance estimates in two-level models. Preprint PRM, 93–04.

Cameron, A. C., Trivedi, P. K., 2013. Regression Analysis of Count Data. Vol. 53. Cambridge university press.

Chiarandini, M., Goegebeur, Y., 2010. Mixed Models for the Analysis of Optimization Algorithms. In: Bartz-Beielstein, T., Chiarandini, M., Paquete, L., Preuss, M. (Eds.), Experimental Methods for the Analysis of Optimization Algorithms. Springer Berlin Heidelberg, pp. 225–264.

Colorni, A., Dorigo, M., Maniezzo, V., et al., 1991. Distributed optimization by ant colonies. In: Proceedings of the first European conference on artificial life. Vol. 142. Paris, France, pp. 134–142.

Cordeau, J.-F., Gendreau, M., Laporte, G., Potvin, J.-Y., Semet, F., 2002. A guide to vehicle routing heuristics. Journal of the Operational Research society, 512–522.

Cordeau, J.-F., Laporte, G., Savelsbergh, M. W., Vigo, D., 2007. Vehicle routing. In: Barnhart, C., Laporte, G. (Eds.), Transportation, Handbooks in Operations Research and Management Science. Vol. 14. Elsevier, Amsterdam, Ch. 6, pp. 367–428.

Coy, S. P., Golden, B. L., Runger, G. C., Wasil, E. A., Jan. 2001. Using Experimental Design to Find Effective Parameter Settings for Heuristics. Journal of Heuristics 7 (1), 77–97.

Dang, N., Cáceres, L. P., De Causmaecker, P., Stützle, T., 2017. Configuring irace using surrogate configuration benchmarks. In: Proceedings of the Genetic and Evolutionary Computation Conference. ACM, pp. 243–250.

Dantzig, G. B., Ramser, J. H., 1959. The truck dispatching problem. Management science 6 (1), 80–91.

Dean, A. M., Voss, D., et al., 1999. Design and Analysis of Experiments. Vol. 1. Springer.

Dietterich, T., 1995. Overfitting and undercomputing in machine learning. ACM computing surveys (CSUR) 27 (3), 326–327.

Dobslaw, F., 2010. Recent development in automatic parameter tuning for metaheuristics. In: Proceedings of the 19th Annual Conference of Doctoral Students-WDS 2010.

Dodig-Crnkovic, G., 2002. Scientific methods in computer science. In: Proceedings of the Conference for the Promotion of Research in IT at New Universities and at University Colleges in Sweden, Skövde, Suecia. pp. 126–130.

Dongarra, J. J., 1993. Performance of various computers using standard linear equations software. University of Tennessee, Computer Science Department.

Fawcett, C., Hoos, H. H., 2015. Analysing differences between algorithm configurations through ablation. Journal of Heuristics 22 (4), 431–458.

Frees, E. W., 2009. Regression modeling with actuarial and financial applications. Cambridge University Press.

Gehring, H., Homberger, J., 1999. A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. In: Proceedings of EUROGEN99. Vol. 2. Citeseer, pp. 57–64.

Gelman, A., Hill, J., Dec. 2006. Data Analysis Using Regression and Multilevel/Hierarchical Models. Cambridge University Press.

Gendreau, M., Tarantilis, C. D., 2010. Solving Large-Scale Vehicle Routing Problems with Time Windows: The State-of-the-Art. Cirrelt Montreal.

Gibco, 2016. Cell Culture Basics. Invitrogen Life Technologies.
URL `www.thermofisher.com/cellculturebasics`

Glover, F., 1989. Tabu searchpart i. ORSA Journal on computing 1 (3), 190–206.

Golden, B. L., Assad, A. A., Wasil, E. A., Baker, E., Oct. 1986. Experimentation in optimization. European Journal of Operational Research 27 (1), 1–16.

Hansen, N., Ostermeier, A., 2001. Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation 9 (2), 159–195.

Hemmati, A., Hvattum, L. M., 2017. Evaluating the importance of randomization in adaptive large neighborhood search. International Transactions in Operational Research 24 (5), 929–942.

Hooker, G., 2012. Generalized functional anova diagnostics for high-dimensional functions of dependent variables. Journal of Computational and Graphical Statistics.

Hooker, J. N., 1994. Needed: an empirical science of algorithms. Operations Research 42 (2), 201–212.

Hooker, J. N., Sep. 1995. Testing heuristics: We have it all wrong. Journal of Heuristics 1 (1), 33–42.

Hoos, H. H., 2011. Automated Algorithm Configuration and Parameter Tuning. In: Hamadi, Y., Monfroy, E., Saubion, F. (Eds.), Autonomous Search. Springer Berlin Heidelberg, pp. 37–71.

Hoos, H. H., 2012. Programming by optimization. Communications of the ACM 55 (2), 70–80.

Hox, J. J., Moerbeek, M., Schoot, R. v. d., Sep. 2010. Multilevel Analysis: Techniques and Applications, 2nd Edition. Routledge.

Hutter, F., Hamadi, Y., Hoos, H. H., Leyton-Brown, K., 2006. Performance prediction and automated tuning of randomized and parametric algorithms. In: International Conference on Principles and Practice of Constraint Programming. Springer, pp. 213–228.

Hutter, F., Hoos, H., Leyton-Brown, K., 2014. An Efficient Approach for Assessing Hyperparameter Importance. In: International Conference on Machine Learning. pp. 754–762.

Hutter, F., Hoos, H. H., Leyton-Brown, K., 2011. Sequential Model-Based Optimization for General Algorithm Configuration. In: Coello, C. A. C. (Ed.), Learning and Intelligent Optimization. No. 6683 in Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 507–523.

Hutter, F., Hoos, H. H., Leyton-Brown, K., 2013. Identifying Key Algorithm Parameters and Instance Features Using Forward Selection. In: Nicosia, G., Pardalos, P. (Eds.), Learning and Intelligent Optimization. No. 7997 in Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 364–381.

Hutter, F., Hoos, H. H., Leyton-Brown, K., Stützle, T., 2009. ParamILS: an automatic algorithm configuration framework. Journal of Artificial Intelligence Research 36 (1), 267–306.

Jacqmin-Gadda, H., Sibillot, S., Proust, C., Molina, J.-M., Thibaut, R., Jun. 2007. Robustness of the linear mixed model to misspecified error distribution. Computational Statistics & Data Analysis 51 (10), 5142–5154.

Johnson, D. S., 2002. A theoretician's guide to the experimental analysis of algorithms. Data structures, near neighbor searches, and methodology: fifth and sixth DIMACS implementation challenges 59, 215–250.

Jones, Z. M., Linder, F., 2015. Exploratory data analysis using random forests. In: Proceedings of the 73rd annual MPSA conference.

Kadioglu, S., Malitsky, Y., Sellmann, M., Tierney, K., 2010. Isac-instance-specific algorithm configuration. In: ECAI. Vol. 215. pp. 751–756.

Karakatič, S., Podgorelec, V., 2015. A survey of genetic algorithms for solving multi depot vehicle routing problem. Applied Soft Computing 27, 519–532.

Kilby, P., Prosser, P., Shaw, P., 2000. A comparison of traditional and constraint-based heuristic methods on vehicle routing problems with side constraints. Constraints 5 (4), 389–414.

Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P., 1983. Optimization by simulated annealing. science 220 (4598), 671–680.

Lawson, J., Erjavec, J., 2016. Basic Experimental Strategies and Data Analysis for Science and Engineering. CRC Press.

Leyton-Brown, K., Nudelman, E., Andrew, G., McFadden, J., Shoham, Y., 2003. A portfolio approach to algorithm selection. In: IJCAI. Vol. 3. pp. 1542–1543.

Lin, B. W., Rardin, R. L., Dec. 1979. Controlled Experimental Design for Statistical Comparison of Integer Programming Algorithms. Management Science 25 (12), 1258–1271.

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

Maas, C. J., Hox, J. J., 2005. Sufficient sample sizes for multilevel modeling. Methodology 1 (3), 86–92.

Malitsky, Y., Sellmann, M., 2010. Stochastic offline programming. International Journal on Artificial Intelligence Tools 19 (04), 351–371.

McGeoch, C. C., Feb. 1996. Feature ArticleToward an Experimental Method for Algorithm Simulation. INFORMS Journal on Computing 8 (1), 1–15.

McGeoch, C. C., Moret, B. M., 1999. How to present a paper on experimental work with algorithms. ACM SIGACT News 30 (LCBB-ARTICLE-1999-001), 85–90.

McNabb, M. E., Weir, J. D., Hill, R. R., Hall, S. N., Apr. 2015. Testing local search move operators on the vehicle routing problem with split deliveries and time windows. Computers & Operations Research 56, 93–109.

Montgomery, D., 2012. Design and Analysis of Experiments, 8th Edition. John Wiley & Sons, Incorporated.

Montgomery, D. C., Runger, G. C., 2010. Applied statistics and probability for engineers. John Wiley & Sons.

Moret, B. M., Shapiro, H. D., 2001. Algorithms and experiments: The new (and old) methodology. Journal of Universal Computer Science 7 (5), 434–446.

Nannen, V., Eiben, A. E., 2007. Relevance estimation and value calibration of evolutionary algorithm parameters. In: IJCAI. Vol. 7. pp. 975–980.

Osborne, J. W., 2014. Best practices in logistic regression. Sage Publications.

Palhazi Cuervo, D., Goos, P., Srensen, K., Arriz, E., Sep. 2014. An iterated local search algorithm for the vehicle routing problem with backhauls. European Journal of Operational Research 237 (2), 454–464.

Pisinger, D., Ropke, S., Aug. 2007. A general heuristic for vehicle routing problems. Computers & Operations Research 34 (8), 2403–2435.

Potvin, J.-Y., Rousseau, J.-M., 1993. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. European Journal of Operational Research 66 (3), 331–340.

R Core Team, 2016. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria.
URL `https://www.R-project.org/`

Rahimi-Vahed, A., Crainic, T. G., Gendreau, M., Rei, W., Apr. 2013. A path relinking algorithm for a multi-depot periodic vehicle routing problem. Journal of Heuristics 19 (3), 497–524.

Rardin, R. L., Uzsoy, R., May 2001. Experimental Evaluation of Heuristic Optimization Algorithms: A Tutorial. Journal of Heuristics 7 (3), 261–304.

Rasku, J., Kärkkäinen, T., Musliu, N., 2016. Feature Extractors for Describing Vehicle Routing Problem Instances. In: Hardy, B., Qazi, A., Ravizza, S. (Eds.), 5th Student Conference on Operational Research (SCOR 2016). Vol. 50 of OpenAccess Series in Informatics (OASIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, pp. 7:1–7:13.
URL `http://drops.dagstuhl.de/opus/volltexte/2016/6519`

Rasku, J., Musliu, N., Kärkkäinen, T., 2014. Automating the parameter selection in vrp: An off-line parameter tuning tool comparison. In: Modeling, Simulation and Optimization for Science and Technology. Springer, pp. 191–209.

Rice, J. R., 1976. The algorithm selection problem. Advances in computers 15, 65–118.

Ridge, E., Curry, E., Jan. 2007. A Roadmap of Nature-inspired Systems Research and Development. Multiagent Grid Syst. 3 (1), 3–8.

Ridge, E., Kudenko, D., 2006. Sequential experiment designs for screening and tuning parameters of stochastic heuristics. In: Workshop on Empirical Methods for the Analysis of Algorithms at the Ninth International Conference on Parallel Problem Solving from Nature (PPSN). Citeseer, pp. 27–34.

Ridge, E., Kudenko, D., 2007. Screening the Parameters Affecting Heuristic Performance. In: In Proceedings of the Genetic and Evolutionary Computation Conference. ACM.

Rodríguez, A., Ruiz, R., Sep. 2012. A study on the effect of the asymmetry on real capacitated vehicle routing problems. Computers & Operations Research 39 (9), 2142–2151.

Röpke, S., 2016. 10 years of Adaptive Large Neighborhood Search (ALNS), VeRoLog 2016: annual workshop of the EURO working group on Vehicle Routing and Logistics optimization (VeRoLog).
URL `https://verolog2016.sciencesconf.org/resource/page/id/4`

Ropke, S., Pisinger, D., 2006. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation science 40 (4), 455–472.

Russell, R. A., May 1995. Hybrid Heuristics for the Vehicle Routing Problem with Time Windows. Transportation Science 29 (2), 156–166.

Saremi, A., Elmekkawy, T. Y., Wang, G. G., 2007. Tuning the Parameters of a Memetic Algorithm to Solve Vehicle Routing Problem with Backhauls Using Design of Experiments. International Journal of Operations Research 4 (4), 206–219.

Shaw, P., 1998. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In: Maher, M., Puget, J.-F. (Eds.), Principles and Practice of Constraint Programming CP98. No. 1520 in Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 417–431.

Shukla, N., Choudhary, A., Prakash, P., Fernandes, K., Tiwari, M., 2013. Algorithm portfolios for logistics optimization considering stochastic demands and mobility allowance. International Journal of Production Economics 141 (1), 146–166.

Sifaleras, A., 2014. Classification of network optimization software packages. In: Khosrow-Pour, M. (Ed.), Encyclopedia of Information Science and Technology, 3rd Edition. IGI Global, Ch. 695, pp. 7054–7062.

Silberholz, J., Golden, B., 2010. Comparison of metaheuristics. In: Handbook of metaheuristics. Springer, pp. 625–640.

Silva, A. L., Ramírez, J. A., Campelo, F., 2013. A statistical study of discrete differential evolution approaches for the capacitated vehicle routing problem. In: Proceedings of the 15th annual conference companion on Genetic and evolutionary computation. ACM, pp. 77–78.

Simmons, J. P., Nelson, L. D., Simonsohn, U., 2011. False-positive psychology: Undisclosed flexibility in data collection and analysis allows presenting anything as significant. Psychological Science 22 (11), 1359–1366, pMID: 22006061.
URL `https://doi.org/10.1177/0956797611417632`

Smith-Miles, K., Baatar, D., Wreford, B., Lewis, R., 2014. Towards objective measures of algorithm performance across instance space. Computers & Operations Research 45, 12–24.

Smith-Miles, K., Bowly, S., Nov. 2015. Generating new test instances by evolving in instance space. Computers & Operations Research 63, 102–113.

Smith-Miles, K., Lopes, L., 2012. Measuring instance difficulty for combinatorial optimization problems. Computers & Operations Research 39 (5), 875–889.

Solomon, M. M., 1987. Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations research 35 (2), 254–265.

Sörensen, K., 2015. Metaheuristics - the metaphor exposed. International Transactions in Operational Research 22 (1), 3–18.

Sörensen, K., Schittekat, P., Dec. 2013. Statistical analysis of distance-based path relinking for the capacitated vehicle routing problem. Computers & Operations Research 40 (12), 3197–3205.

Sörensen, K., Sevaux, M., Glover, F., 2018. A history of metaheuristics. Handbook of heuristics, 1–18.

Stock, J., Watson, M. W., 2011. Introduction to Econometrics. Prentice Hall, New York.

Sullivan, G. M., Feinn, R., 2012. Using effect size — or why the p value is not enough. Journal of graduate medical education 4 (3), 279–282.

Talarico, L., Sörensen, K., Springael, J., Jul. 2015. Metaheuristics for the risk-constrained cash-in-transit vehicle routing problem. European Journal of Operational Research 244 (2), 457–470.

The European Commission, 2017. Statistical Pocketbook - EU Transport in Figures. The European Commission.
URL `https://ec.europa.eu/transport/`

Toth, P., Vigo, D., 2003. The granular tabu search and its application to the vehicle-routing problem. Informs Journal on computing 15 (4), 333–346.

Tuzun, D., Magent, M. A., Burke, L. I., 1997. Selection of vehicle routing heuristic using neural networks. International Transactions in Operational Research 4 (3), 211–221.

Tyasnurita, R., Özcan, E., John, R., 2017. Learning heuristic selection using a time delay neural network for open vehicle routing. In: Evolutionary Computation (CEC), 2017 IEEE Congress on. IEEE, pp. 1474–1481.

Van der Leeden, R., Busing, F., 1994. First iteration versus igls/rigls estimates in two-level models: A monte carlo study with ml3 (technical report prm 94-03). Department of Psychometrics and Research Methodology, University of Leiden, Leiden, the Netherlands.

van Rijn, J. N., Hutter, F., 2018. Hyperparameter importance across datasets. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. ACM, pp. 2367–2376.

Wolpert, D. H., Macready, W. G., 1997. No free lunch theorems for optimization. IEEE transactions on evolutionary computation 1 (1), 67–82.

Xu, J., Kelly, J. P., 1996. A network flow-based tabu search heuristic for the vehicle routing problem. Transportation Science 30 (4), 379–393.

Xu, L., Hoos, H., Leyton-Brown, K., 2010. Hydra: Automatically configuring algorithms for portfolio-based selection. In: Association for the Advancement of Artificial Intelligence (AAAI). Vol. 10. pp. 210–216.

Publications and Conference Participation

Journal Publications

Corstjens, J., Depaire, B., Caris, A., Sörensen, K., in review. A multilevel evaluation method for heuristics with an application to the vrptw. Manuscript submitted for publication.

Corstjens, J., Dang, N., Depaire, B., Caris, A., De Causmaecker, P. 2018. A combined approach for analysing heuristic algorithms. Journal of Heuristics (forthcoming). doi: 10.1007/s10732-018-9388-7.

Publications in Conference Proceedings

Corstjens, J., Caris, A., Depaire, B., 2018. Explaining heuristic performance differences for vehicle routing problems with time windows. In: Kotsireas, I., Pardalos, P. (Eds.) Learning and Intelligent Optimization, Lecture Notes in Computer Science (forthcoming). Springer Berlin Heidelberg.

Other Conference Participation (Abstract)

Corstjens, J., Caris, A., Depaire, B., 2015. Evaluating and Optimizing Metaheuristic Algorithms:A step ahead in VRP research. In: 29th Annual Conference of the Belgian Operations Research Society, Antwerp, Belgium, 5-6 February.

Corstjens, J., Caris, A., Depaire, B., 2015. Evaluation and optimization of

metaheuristic algorithms for the vehicle routing problem with time windows. In: EURO2015 - 27th European Conference on Operational Research, Glasgow, United Kingdom, 12-15 July.

Corstjens, J., Depaire, B., Caris, A., Sörensen, K., 2016. A Multilevel Methodology for Analysing Metaheuristic Algorithms for the VRPTW. In: EU/ME 2016 Workshop on Design and Analysis of Metaheuristics.

Corstjens, J., Caris, A., Depaire, B., 2016. Experimental Analysis of Metaheuristic Algorithms for the VRPTW. In: The 30th Conference of the Belgian Operational Research Society.

Corstjens, J., Depaire, B., Caris, A., Sörensen, K., 2016 Analysing metaheuristic algorithms for the vehicle routing problem with time windows. In: VeRoLog 2016, Nantes, France, 6-8 June.

Corstjens, J., Caris, A., Depaire, B., Sörensen, K., 2017. A statistical methodology for analysing heuristic algorithms. In: Belin, Jeroen (Ed.). 31st Belgian Conference on Operations Research - abstracts,p. 26-27.

Corstjens, J., Caris, A., Depaire, B., 2017. Understanding heuristic algorithm behaviour through iterative experimentation. In: 5th COSEAL Workshop on automated configuration and selection of algorithms, Brussels, Belgium, 11-12 September.

Corstjens, J., Caris, A., Depaire, B., 2018. An analysis on the destroy and repair process in large neighbourhood search applied on the vehicle routing problem with time windows. In: The 32nd Conference of the Belgian Operational Research Society, Liège, Belgium, 1-2 February.

Corstjens, J., Caris, A., Depaire, B., 2018. Explaining metaheuristic performance through iterative experimentation. In: 19th EU/ME Workshop on Metaheuristics for Industry 2018, Geneva, Switzerland, 22-23 March.

Samenvatting

Combinatorische optimalisatieproblemen, waarbij gezocht wordt naar de beste oplossing uit een eindig aantal oplossingen, behandelen vaak problemen met een grote economische relevantie, zoals de planning van ritten voor de distributie van goederen en het invullen van dienstregelingen. Dit heeft geleid tot de ontwikkeling van een groot aantal procedures die in staat zijn een oplossing voor deze problemen te genereren. Hierbij wordt een onderscheid gemaakt tussen exacte en heuristische methoden. Veel combinatorische optimalisatieproblemen zijn moeilijk op te lossen en worden daarom als NP-moeilijk aangeduid. Hiermee wordt bedoeld dat er geen snel (d.i. polynomiaal) exact algoritme bestaat dat elke instantie van het probleem in een redelijke tijd optimaal kan oplossen. Exacte methoden garanderen dat de optimale oplossing gevonden wordt en kunnen bijgevolg een enorme hoeveelheid rekentijd vereisen om een dergelijke oplossing te waarborgen. Deze algoritmes zijn daarom enkel toepasbaar op kleine probleeminstanties. Het merendeel van de ontwikkelde oplossingsmethoden zijn heuristisch van aard en in staat goede oplossingen te genereren binnen een aanvaardbare rekentijd, zonder te garanderen dat deze oplossing ook de optimale oplossing is.

Hoe effectief of efficiënt een heuristiek is, wordt gewoonlijk beoordeeld in een empirische studie waarbij de performantie van een heuristiek toegepast op een optimalisatieprobleem geëvalueerd wordt door instanties van een of meerdere probleembenchmarks op te lossen en de vergelijking te maken met de resultaten die bestaande methoden behaald hebben op deze benchmarks. Deze evaluatiebenadering legt de nadruk op het competitief zijn, op het ontwikkelen van een algoritme dat 'beter' kan presteren dan reeds bestaande algoritmes. 'Beter' betekent het behalen van een verbeterde oplossingskwaliteit, vereisen van minder rekentijd voor eenzelfde oplossingskwaliteit, of een gunstigere afweging van beide prestatiemaatstaven. Wat een dergelijke competitieve evaluatiebenadering niet toelaat, is het verklaren waarom

een nieuw voorgestelde heuristiek beter presteert. Kan het toegeschreven worden aan een ingenieus ontwikkelde operator werkzaam binnen het algoritme? Of misschien ligt het simpelweg aan een meer efficiënte implementatie van een bestaande methode? Dragen alle componenten significant bij tot de behaalde prestatiewaarde, of zouden bepaalde elementen weggelaten kunnen worden om zo de efficiëntie te verhogen? Dit zijn allemaal vragen die vaak onbeantwoord blijven wanneer een heuristiek gepresenteerd wordt in een onderzoekspublicatie. Hoewel enige competitie tussen onderzoekers zeker kan aanzetten tot innovatieve ontdekkingen, is het erkend dat 'ware innovatie' gefundeerd is op het begrijpen van hoe een heuristische oplossingsmethode zich gedraagt, niet op bewijs van concurrentievermogen. Het is het ultieme doel van wetenschap om te begrijpen, niet om te winnen van anderen.

Een competitieve evaluatie is nuttig indien het doel is om de snelst en best mogelijke procedure te ontwikkelen voor een specifieke probleemcontext. Indien het doel is te begrijpen hoe een prestatiewaarde behaald werd, te ontdekken welke elementen in de heuristiek een bijdrage leveren en conclusies af te leiden die niet enkel gelden voor de specifiek onderzochte probleeminstanties, dan is een statistische evaluatie vereist. De doelstelling van dit doctoraatsonderzoek is dan ook het promoten van een werkwijze voor het experimenteren met heuristieken waarbij de focus ligt op het verkrijgen van begrip en inzicht in hoe een heuristiek tot een prestatiewaarde komt. De verworven kennis kan dan gebruikt worden bij het ontwerpen en optimaliseren van algoritmes alsook bij het vergelijken van verschillende algoritmes. Om deze doelstelling te vervullen wordt een evaluatiemethodologie voorgesteld op basis van de concepten uit *Design of Experiments* en het gebruik ervan gedemonstreerd in een aantal experimentele studies. De methodologie is toepasbaar op een breed scala aan optimalisatieproblemen en algoritmes, maar in deze thesis ligt de focus op het analyseren van de performantie van een *large neighbourhood search* algoritme toegepast op instanties van het rittenplanningsprobleem met tijdvensters. Dit is een belangrijk probleem in menig distributiesysteem en heeft daarom heel wat onderzoekswerk naar zowel exacte als heuristische methoden geïnspireerd. Maar relatief weinig onderzoeksinspanningen zijn geleverd waarbij statistische technieken toegepast worden bij de analyse van deze problemen of waarbij men erop uit is te begrijpen hoe het probleem van invloed is op de performantie van een algoritme.

Dit doctoraatsonderzoek beschouwt het als de volgende stap in experimenteel onderzoek naar rittenplanningsproblemen om een beter begrip en inzicht te verkrijgen in de effecten die parameters en componenten hebben op de performantie

van een heuristisch algoritme. De voorgestelde methodologie is in staat aan te duiden welke elementen een significante impact hebben op de oplossingskwaliteit die een algoritme behaalt en hoe de probleemkenmerken deze effecten beïnvloeden. Multilevel experimentele ontwerpen worden gebruikt om efficiënt te analyseren hoe effecten variëren naargelang de probleeminstantie die opgelost moet worden. Zo kunnen verschillende aanbevelingen voor verschillende delen van de probleemruimte afgeleid worden. Verder laat het onderzoekers ook toe vaststellingen te doen voor een volledige populatie van probleeminstanties in plaats van een beperkt aantal benchmarkinstanties.

De methodologie omvat een iteratief proces waarbij gegevens eerst geobserveerd worden om vervolgens vragen die uit deze observaties voortkomen te beantwoorden in vervolgstudies, die op hun beurt weer tot observaties kunnen leiden waarbij vragen gesteld worden. Het vertrekpunt is dus het uitvoeren van een verkennende analyse om bloot te leggen hoe de elementen van het algoritme gecorreleerd zijn met de prestatiewaarde evenals de correlaties van de probleemkenmerken met de algoritmeparameters en -componenten. Vervolgens zal een verklarende analyse, waarbij hypotheses geformuleerd en getoetst worden, uitzoeken hoe geobserveerde correlaties tot stand komen. De verkennende analyse is in eerste instantie volledig gebaseerd op een multilevel regressieanalyse en vervolgens enkel op de elementen en probleemkenmerken die het belangrijkst zijn om een goede prestatie van het algoritme te behalen. Deze focus op de belangrijkste effecten wordt verkregen door een *functional analysis of variance* (fANOVA) uit te voeren voordat het multilevel regressiemodel wordt opgesteld. De rangschikking van effecten die fANOVA als output aanlevert zal leiden tot een meer beknopt regressiemodel met minder variabelen. De regressieanalyse voorziet een meer gedetailleerde analyse van de effecten die algoritme-elementen hebben en laat ook toe om verklarende analyses uit te voeren.

De toepassing van de methodologie wordt geïllustreerd aan de hand van een gevalstudie waarbij de performantie van een *large neighbourhood search* algoritme op het rittenplanningsprobleem met tijdvensters wordt geanalyseerd. Een verkennende studie geeft aan dat alle *destroy* en *repair* operatoren opnemen in het algoritme niet noodzakelijk tot de beste resultaten leidt. Enkel regret-2 als *repair* operator gebruiken wordt gesuggereerd als de gemiddeld beste keuze. De *destroy* operator die hierbij het best presteert is *random removal*. Deze bevindingen worden bevestigd door een fANOVA die niet gebonden is aan de assumpties waaraan regressiemodellen dienen te voldoen en dus de analyse meer robust maakt. De verkennende analyse leidde

ook tot nieuwe vragen, zoals waarom klanten willekeurig verwijderen beter werkt dan een geografische cluster van klanten te verwijderen indien deze klanten opnieuw in de oplossing worden ingevoegd, gebruikmakende van een moeilijkheidscriterium (d.i. de regretwaarde). Verklaringen worden daarom gezocht voor het geobserveerde prestatieverschil tussen deze twee *destroy* operatoren in het geval zij gebruikt worden met een bepaald type *repair* operator. Uit de verklarende analyse blijkt dat geografische clusters van klanten verwijderen het aantal invoegingsalternatieven waaruit gekozen kan worden tijdens de repair-fase reduceert. Verschillende klanten hebben zelfs geen enkele haalbare (d.i. 'feasible') invoegingsoptie in een van de bestaande routes en worden daarom beschouwd als geïsoleerde gevallen (bij het begin van de repair-fase). De plaatsing van deze klanten uitstellen blijkt nefast te zijn voor de oplossingskwaliteit. Daarom wordt getest of een betere oplossing verkregen kan worden indien deze geïsoleerde klanten een hogere prioriteit krijgen door toe te laten dat zij ingevoegd worden in een individuele route die vanuit het depot rechtstreeks naar de betreffende klant rijdt en vervolgens terugkeert naar het depot. Een dergelijke optie werd voorheen gezien als een laatste alternatief. Doordat deze individuele routes eerder gecreëerd kunnen worden tijdens het herstelproces worden ook goede invoegingsalternatieven toegevoegd voor andere verwijderde klanten, resulterend in een posifief effect op de oplossingskwaliteit. Een *regret* operator zal dus een betere inschatting maken van hoe moeilijk de plaatsing van een klant is en dus een betere prioritering indien iedere individuele klant bestaande routes in de nabijheid heeft waarin deze ingevoegd kan worden. Door middel van een dergelijke gedetailleerde analyse van een *destroy* en *repair* iteratie werd het merendeel van het performantieverschil tussen beide *destroy* scenario's verklaard.

De verkennende analyse kan niet enkel gebruikt worden als vertrekpunt voor volgende verklarende analyses die trachten geobserveerde correlaties te verklaren, maar kan ook dienen om goedpresterende parameterwaarden en componenten te kiezen, voor zowel een gemiddelde probleeminstantie als voor iedere individuele instantie. De methodologie wordt hier dus toegepast om het algoritme te tunen zodanig dat de beste prestatie verkregen wordt. Beslissingsregels worden afgeleid voor iedere parameter en component en zijn geformuleerd op basis van de (significante) probleemkenmerken om zo een algoritmeconfiguratie te bekomen, gepersonaliseerd op iedere probleeminstantie. Het is aangetoond dat er vaak niet een enkele configuratie bestaat die het best presteert voor iedere individuele probleeminstantie, maar dat de prestatie van een heuristische methode varieert over de set van instanties die opgelost dienen te worden. De overtuiging is dat de methodologie die in deze thesis wordt

voorgesteld die prestatievariatie kan exploiteren.

Eerst wordt een enkele configuratie uit de analyse afgeleid die voorspeld wordt het best te presteren voor een gemiddelde probleeminstantie, gelijkaardig aan de configuratie verkregen uit bestaande automatische algoritmeconfiguratoren zoals *irace*. Hierbij wordt de invloed van probleemkenmerken op de effecten van parameters en componenten buiten beschouwing gelaten. Deze configuratie presteert gelijkaardig aan die verkregen uit *irace* en beter en beter naargelang het regressiemodel op meer data getraind wordt. Vervolgens wordt nagegaan hoe de instantiespecifieke configuraties presteren. Deze blijken geen voordeel te bieden ten opzichte van een enkele configuratie voor een gemiddelde instantie. Dit is waarschijnlijk te wijten aan het feit dat de gegenereerde probleeminstanties vrij homogeen zijn en dus geen mogelijkheid bieden om prestatievariatie te exploiteren.

De kernboodschap van dit doctoraatsproefschrift is dat wetenschappelijk onderzoek doen eerder gaat over begrijpen dan over het winnen van een competitie. Een grondige analyse uitvoeren op experimentele resultaten kan waardevolle kennis opleveren die niet beperkt is tot de specifieke experimentele context, maar die nuttig kan zijn voor elk gerelateerd onderzoek. Wanneer men geconfronteerd wordt met een rittenplanningsprobleem, kan de beslissing over welke oplossingsstrategieën toe te passen of hoe ze te optimaliseren beargumenteerd worden op basis van eerder uitgevoerde analyses. Deze analyses dienen rekenschap te geven aan zowel het algoritme als de probleeminstanties die opgelost worden, zodanig dat hun wisselwerking onderzocht kan worden. Deze thesis spoort aan om onderzoek uit te voeren dat niet gebonden is aan de specificiteiten van een experiment, maar dat veralgemeenbaar is naar een breder geheel van gelijkaardige contexten. Dit proefschrift biedt de onderzoeksgemeenschap een plan van aanpak om dit uit te voeren en om te leren over zowel het probleem als de oplossingsmethode.