A Multilevel Evaluation Method for Heuristics with an Application to the VRPTW

Jeroen Corstjens[1], Benoît Depaire[2], An Caris[1], and Kenneth Sörensen[3]

[1]UHasselt/Research Group Logistics, Agoralaan, Diepenbeek 3590, Belgium
[1]jeroen.corstjens@uhasselt.be
[2]UHasselt/Research Group Business Informatics, Agoralaan, Diepenbeek 3590, Belgium
[2]Universiteit Antwerpen/Department Engineering Management, Prinsstraat, Antwerpen 2000, Belgium

January 27, 2019

Abstract

The field of combinatorial optimisation has inspired the development of a large number of heuristic solution procedures. These methods are commonly assessed using a competitive evaluation methodology that may give an indication of which algorithm has a better performance. A next step in the experimental analysis is to uncover 'why' one algorithm performs better. Which elements are responsible for good or bad performance? How does the performance of elements vary across the design space? What is the influence of the specific problem instance that is being solved? We focus on gaining a better understanding of heuristic algorithm performance and demonstrate that the application of a proper statistical methodology can provide researchers insight into how performance is affected by the different algorithm parameters and components. As an example, we apply a multilevel statistical analysis to a large neighbourhood search algorithm for the vehicle routing problem with time windows.

Keywords: Algorithm Performance; Statistical Evaluation; Understanding; Vehicle Routing Problem with Time Windows; Large Neighborhood Search; Metaheuristics

1 Introduction

The field of combinatorial optimisation has inspired the development of a large number of heuristic algorithms which are able to produce good solutions in a reasonable time. They range from standard construction and improvement algorithms to powerful metaheuristic frameworks. The former two also constitute an important component in all metaheuristic frameworks (Bräysy and Gendreau, 2005).

The most common approach to evaluate heuristics for an optimisation problem is by studying their performance on a set of standard benchmark problems. This type of evaluation results in a competition between state-of-the-art methods in the literature. A newly developed heuristic algorithm is considered 'better' than existing ones if it outperforms them on the standard set of benchmark problems. This kind of experimental evaluation has been useful to get to know which algorithms work well on which applications. It is one type of knowledge acquisition. This research argues to go a step further and acquire another type of knowledge that focuses on understanding why heuristic algorithms or configurations of a single algorithm perform differently. An experimental evaluation focused on gaining insight into how things work and why they work well has been acknowledged as important, but has received little attention (Hooker, 1995; Cuervo et al., 2014; Sörensen, 2015).

This research aims to fill this gap by focusing on evaluating heuristic algorithms with the aim of understanding how the reported performance values are obtained rather than on developing the best performing algorithm. Therefore, we propose a general statistical evaluation framework to set up and analyse the results of an experimental study investigating the relationships between algorithm performance, algorithm parameters, and problem instance characteristics. The methodology has a statistical foundation since statistical tests offer a means of learning (Bartz-Beielstein, 2006) with the interest being to draw conclusions that are valid beyond the specific problem instances and parameter values chosen (Rardin and Uzsoy, 2001). We wish to identify how the algorithm parameters impact algorithm performance, positively or negatively, and how these effects vary across different parts of the problem space. The objective is to expose patterns in the performance data to establish which (combinations of) elements work well under which conditions. These patterns can then be further investigated by formulating falsifiable hypotheses in consecutive studies and the ultimate goal is the production of insights that increases our knowledge of heuristic algorithms. In this paper the contribution is threefold. First, the methodological framework is presented and discussed. Secondly, the framework is applied to produce falsifiable hypotheses that are to be validated in consecutive iterations of the experimental study (e.g., in Corstjens et al. (2019)). Thirdly, the framework is demonstrated to the domain of vehicle routing problems (VRP). These are an extensively studied class of combinatorial optimisation problems, with a wide spectrum of real-life applications. Understanding how a VRP influences heuristic algorithm behaviour is therefore relevant knowledge to acquire.

The paper is structured as follows: In Section 2 previous research efforts targeted at providing some explanation for performance results are reviewed. This is followed by a brief introduction to the hypothetico-deductive method in Section 3, which forms the foundation our evaluation methodology is built on. In Section 4 we introduce our methodology followed by an illustration in Section 5 on the large neighbourhood search (LNS) and vehicle routing problem with time windows (VRPTW) where we first elaborate on the problem instances generated (Section 5.2) and the heuristic algorithm used to solve these instances (Section 5.3). The results of the statistical analysis and the insights obtained are discussed in Section 5.5. A conclusion is finally given in Section 6.

2 Related Work

The surgence of automated algorithm configurators over the last decade, often inspired on concepts from machine learning, have gained a lot of popularity by introducing more formal procedures to determine parameter settings rather than the tedious and error-prone trial-and-error approach (Birattari, 2009). However, these configurators often provide no or limited insight on why identified elite parameter configurations perform better than other ones. Several importance analysis techniques have been proposed that try to explain which design choices are most important to performance. Nannen and Eiben (2007) propose a method based on information theory for parameter relevance estimation. Gunawan et al. (2011) suggest a preliminary phase to automatic algorithm configuration. They employ a factorial experimental design to first screen and rank parameters and then use response surface methodology to identify a good initial value range for the important parameters before applying automated tuning. These approaches are limited to a small number of parameters. Hutter et al. (2013) introduce a forward selection approach that repeatedly fits a regression model, each time adding the parameter or instance characteristic that results in the model with the lowest root mean squared error on the validation data. This repeatedly model learning can require a significant amount of computation time. Hutter et al. (2014) train a random forest model on an algorithm performance data set and then apply functional analysis of variance (fANOVA) (Hooker, 2007) on the prediction model to decompose the overall algorithm performance variance in additive components. Fawcett and Hoos (2015) present an automated technique that iteratively modifies parameter settings from a default to a target configuration in order to identify which parameter level changes induce the largest performance differences between the two algorithm configurations. It has the limitation of being bounded to two specific configurations.

These importance analyses prove their value when the aim is to perform a screening of parameters, but are not really suitable for confirmatory analyses. A technique like fANOVA (Hutter et al., 2014), for example, relies on a random forest prediction model. While random forests succeed very well in obtaining accurate predictions, it is not able to validate hypotheses. Hence, 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 (Cutler et al., 2007). In that respect, the work of Chiarandini and Goegebeur (2010) is the most related to the methodology proposed in this paper. The authors separate the effects of algorithm parameters and problem instance characteristics in a mixed effects or multilevel model, and are able to infer conclusions to the entire population of problem instances. We also rely on multilevel models, but go a step further by investigating how algorithm parameter effects alter throughout the problem space. Problem-parameter interactions are also investigated by Ries et al. (2012) in their instance-specific parameter tuning strategy. The authors consider a full factorial design with four problem instance characteristics and three algorithm parameters each having two levels (problem instance size has three levels). The experimental study performed in this paper considers more factors, some of which are continuous, resulting in a large number of levels. This makes a full factorial design impractical.

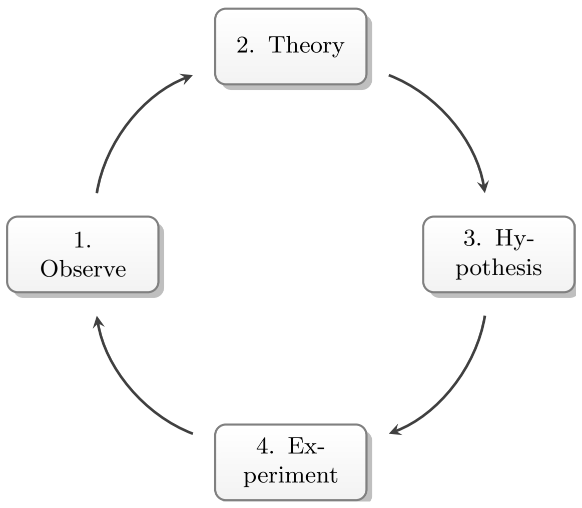

Figure 1: The hypothetico-deductive method.

The next section addresses the process we follow for gaining insight into algorithm performance.

3 Research methodology

The hypothetico-deductive method is the most common description of a set of steps scientists use to search for answers to research questions (Figure 1) (Dodig-Crnkovic, 2002). After running a controlled experiment on a heuristic algorithm, patterns in the data might be observed that spark new research questions. Theories are then formulated to answer these questions and hypotheses are deducted to test the theories. New controlled experiments are set up to validate the hypotheses after which a theory can be rejected or not. 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). In order to be clear about the kind of study we wish to perform, an analogy can be made 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 evaluation methodology we promote, 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 means to perform this iterative process 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). Since it is often impractical to obtain data for an entire population, a fraction of this population — referred to as a sample — is studied. Inferential statistics offer methods that enable practitioners to draw conclusions about an entire population or process based on sample data and express the amount of confidence that can be attributed to these conclusions (Mason et al., 2003; Moore et al., 2012). A key principle of experimental design is randomisation. This is a necessary condition for statistical methods which assume observations (or error terms) are independently distributed random variables. It also helps to "average out" effects of possible extraneous factors. Hence, it is important that the samples drawn are properly randomised (Montgomery, 2012).

The obtained insights and knowledge from the analysis can be used in the design, optimisation and comparison of heuristic methods. In the design phase insights can assist in making design choices. For example, Ribeiro et al. (2011) look into the need for effective stopping criteria for metaheuristics. They have discovered that the solutions produced by Greedy Randomized Adaptive Search Procedures (GRASP) can be approximated by a Normal distribution. The statistical properties of this distribution are used to derive effective probabilistic stopping rules. In addition, the analysis results 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 (Cuervo et al., 2014). 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, the insights can provide answers why two heuristic algorithms differ in performance in a comparative analysis.

In the following sections we propose a statistical evaluation framework (Section 4) that is model-based, works on an unbiased data set, takes into account the problem instance influence and enables the formulation and validation of hypotheses. The framework is then applied on an existing heuristic algorithm (Section 5). We will perform the first steps of the hypothetico-deductive process, i.e. looking for patterns in the performance data and start formulating hypotheses that might explain them.

4 A Multilevel Evaluation Methodology

The proposed methodology allows an algorithm designer to gain a thorough understanding of the relationship between algorithm performance, algorithm parameters, and problem instance characteristics. The algorithm parameters are commonly under control of the designer while the characteristics of a problem instance to be solved usually are not. In the proposed framework, however, we control both groups of factors. The framework is summarised in Figure 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. We interpret a parameter setting as a set of values and included operators. 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 (Section 4.1). The algorithm runs each scenario returning a desired performance measure. The scenarios with performance results are analysed by fitting a multilevel regression model (Section 4.2). From the regression output, we can investigate and interpret the relationships between performance, the algorithm parameters and problem instance characteristics.

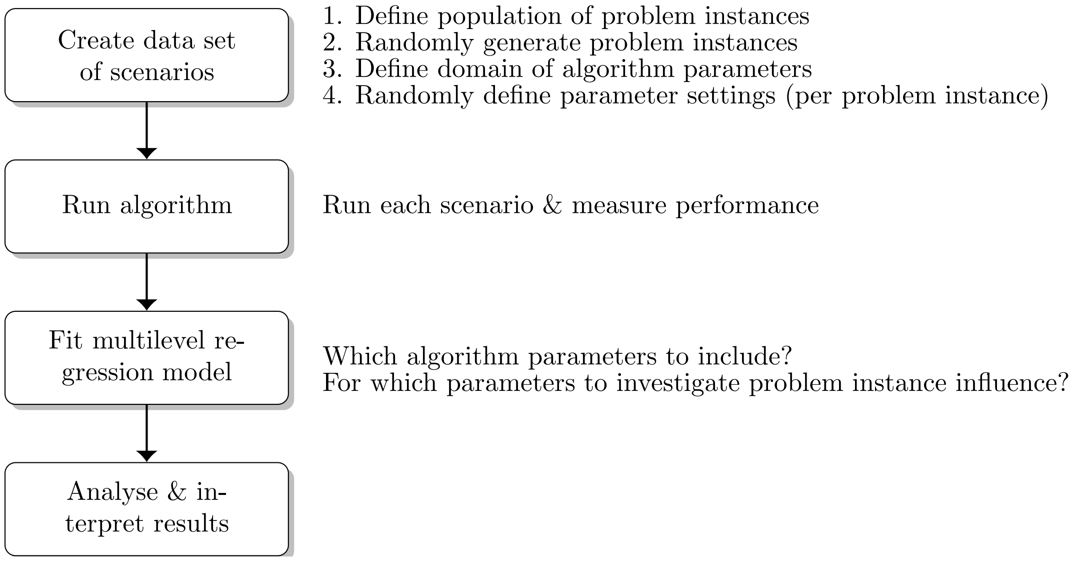
Create data set of scenarios
1. Define population of problem instances
2. Randomly generate problem instances
3. Define domain of algorithm parameters
4. Randomly define parameter settings (per problem instance)
Run algorithm
Run each scenario & measure performance
Fit multilevel regression model
Which algorithm parameters to include?
For which parameters to investigate problem instance influence?
Analyse & interpret results

Figure 2: Diagram of Multilevel Methodology.

4.1 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. The latter has to our knowledge not been analysed in previous research on the experimental analysis of heuristic algorithms. In order to effectively be able to study these relationships many different combinations of parameter settings and problem instances have to be analysed. A way of reducing the number of combinations and thereby computational effort 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

Table 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
...	...	...	...	...	...	...	...	...

problem instance influence. Therefore, the methodology relies on multilevel[1] models to efficiently study how effects vary by the group (or in this case problem instance) they belong to. For example, it could be possible that a heuristic operator only has a beneficial effect on performance when the problem size is large (say more than 300 customers) or one operator can cope with tight time windows better than other operators.

Multilevel models are regularly applied in social research where the need was expressed for statistical techniques that can account for information from both individuals and the groups these individuals belonged to (De Leeuw et al., 2008). The aim is 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). This traditional view of multilevel models being targeted at individuals nested within groups has changed to a view that consider these models as a flexible way to handle complex data, especially since more and more multilevel analysis software has become available (Hox et al., 2010).

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 5.2 and 5.3.

Table 1 gives an illustration of a multilevel experimental. For i going from one to four the values for the problem instance characteristics remain the same

[1]The term 'multilevel' implies a hierarchical or nested data structure having complex patterns of variability, particularly nested sources of variability. Ignoring the different sources of variability may lead to incorrect conclusions being drawn (Hox et al., 2010; Snijders, 2011)

while the values for the algorithm parameters change. In other words, four different parameter settings are tested on the same problem instance. The same goes for scenarios five to eight, but with a different problem instance. Each row in the table represents a unique combination of problem instance characteristics and algorithm parameter values.

4.2 Regression Model

The analysis of the multilevel design is performed by relying on parametric 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 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 4.1 demands a multilevel regression analysis since it violates the assumption of independent error terms made by traditional regression analysis. 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) 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. There is, however, no general rule to determine a minimum number of observations per group (Gelman and Hill, 2006).

A multilevel regression model is basically a multilevel version of the well-known multiple regression model and can, therefore, also be applied to many research problems. In a multiple regression equation the intercept and coefficient terms are *fixed*, i.e. they do not vary from group to group — or, in our case, from problem instance to problem instance. In a multilevel regression model (some of) these terms are considered *random*, i.e. they are allowed to vary across problem instances such that the performance impact some heuristic operator has does not have to be the same when having to serve 50 customers or 300 customers (Hox et al., 2010). Like multiple regression, it 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 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 5.4 does consider non-linear effects and variable interactions.

$$Y_i = \alpha_{j[i]} + \sum_{k \in K} \beta_{kj[i]} Z_{ki} + \epsilon_i \tag{1}$$

$$\alpha_j = \mu_0^{\alpha} + \sum_{p \in P} \mu_p^{\alpha} X_{pj} + \eta_j^{\alpha} \tag{2}$$

$$\beta_{kj} = \mu_0^{\beta_k} + \sum_{p \in P} \mu_p^{\beta_k} X_{pj} + \eta_j^{\beta_k} \qquad \forall k \in K \tag{3}$$

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

Equation (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). Equations (2) and (3) represent the regression models at the problem instance level and measure the influence of the problem instance characteristics (X_p) on α and 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. Note that not all algorithm parameter coefficients (the β's) need to be modelled as a varying 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 (1) to (3) can also be written in a single regression equation by filling in (2) and (3) in equation (1).

$$Y_i = [\mu_0^{\alpha} + \sum_{p \in P} \mu_p^{\alpha} X_{pj[i]} + \eta_{j[i]}^{\alpha}] + \sum_{k \in K} [\mu_0^{\beta_k} + \sum_{p \in P} \mu_p^{\beta_k} X_{pj[i]} + \eta_{j[i]}^{\beta_k}] \times Z_{ki} + \epsilon_i \quad (4)$$

Or

$$\begin{aligned} Y_i =& [\mu_0^{\alpha} + \sum_{p \in P} \mu_p^{\alpha} X_{pj[i]}] + [\sum_{k \in K} \mu_0^{\beta_k} \times Z_{ki} + \sum_{k \in K} \sum_{p \in P} \mu_p^{\beta_k} X_{pj[i]} \times Z_{ki}] + \\ & [\epsilon_i + \eta_{j[i]}^{\alpha} + \sum_{k \in K} \eta_{j[i]}^{\beta_k} \times Z_{ki}] \end{aligned} \quad (5)$$

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.

5 Experimental Analysis

In this section the statistical evaluation framework is illustrated on a case study that is introduced in Section 5.1, followed by a description of how the problem instances used in the experiment are generated (Section 5.2) and a discussion of the heuristic algorithm of which we aim to gain a better understanding (Section 5.3).

5.1 Case

An analysis is performed on the results of a large neighbourhood search (LNS) algorithm run on a number of instances for the vehicle routing problem with time windows. The vehicle routing problem (VRP) in its basic form is the problem of finding a set of routes to serve a number of customers with the objective of minimising a total cost measure. In the vehicle routing problem with time windows (VRPTW), a number of customers have to be served at minimum cost without violating the customers' time-window constraints and the vehicle-capacity constraints. This variant is chosen since the importance of VRPTW in many distribution systems has spurred intensive research efforts for both heuristic and exact optimization approaches (Bräysy and Gendreau, 2005).

All experiments were 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 were available from the infrastructure of the Flemish Supercomputer Center (www.vscentrum.be).

5.2 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 concerns of overfitting and often unknown probability distributions of the characteristics of these instances. The Solomon instances represent a sample from some problem population, just like our sample of artificial problem instances. However, contrary to the Solomon instances, we know from what population our instances are drawn while this is not known for the Solomon benchmark since there is no information on the probability distributions used to generate these instances. Hence, we have no idea to what kind of problem population any conclusions coming out the analysis apply to and they are therefore limited to the set of benchmark instances (Banerjee and Chaudhury, 2010). Secondly, repeatedly using the same set of instances to evaluate the performance of various heuristic algorithms, very often with the focus on matching or surpassing the state-of-the-art performance that is reported for these instances, risks of obtaining optimistic evaluations. Benchmarks can thus become stale and not reflect how a heuristic algorithm truly performs (Bertsimas and Simchi-Levi, 1996; Birattari, 2009; Goodfellow et al., 2016). A problem instance generator can produce a new independent sample for every new analysis, thereby reducing the risk of overfitting. In addition, for a single analysis, an independent second problem instance set, the test set, can be used to produce an unbiased performance estimate.

The random generation of test instances enables proper statistical statements to be made about the experimental results. By applying a valid experimental design[2] and statistical analysis, inference from a sample to all possible problem instances producible 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.

[2]One-factor-at-a-time experimental designs are not considered as rigorous experimental designs when dealing with several factors. Factorial experiments that vary factors together, instead of one at a time, should be used (Montgomery, 2012).

Moreover, the question of which parameter values to test needs to be answered. We accounted for these risks in the selection of the value ranges which are discussed in the next paragraphs.

The characteristics for which the combined values constitute a single problem instance are listed in Table 2 and are 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 our scheme. Not all values are determined randomly, the coordinates of the depot and the vehicle capacity 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.

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 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 it is most probable 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 we assume a context in which a problem instance has to be solved within a certain time frame. 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.

The diversity of the sample of problem instances is assessed through summary statistics in Table 5 in Appendix. This was not only done for the problem instance characteristics listed in Table 2, 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 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 2: Problem Instance Characteristics

Characteristic	**Type**	**Value Ranges**
number of customers	Integer	U[25, 400]
capacity vehicle	Integer	150
(x,y)-coordinates	Integer	U[0,500] × U[0,500]
demand customers	Integer	U[10,50]
Service time	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)

5.3 Large Neighborhood Search

The heuristic algorithm under investigation is a Large Neighbourdhood Search (LNS) algorithm. LNS is a widely applied metaheuristic framework and proven to be very effective in solving various VRP variants (Gendreau and Potvin, 2010). Our implementation has two stages. First, starting from an initial solution, the number of vehicles is minimised 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 solution that could serve them all. In a second stage the focus is on minimising the total distance covered. At each iteration, the algorithm destroys and repairs the current solution by randomly selecting a destroy and repair operator from a set of destroy and repair operators. This process is repeated until some stopping criterion is met.

This implementation of the algorithm is based on the Adaptive Large Neighbourhood Search (ALNS) algorithm developed by Pisinger and Ropke (2007). Since it is recommended to start small when planning experiments (Lawson and Erjavec, 2016), the choice is made not to tackle the ALNS, but simplify it by removing the adaptive mechanism — and thereby all associated parameters — that updates the weights of the operators and instead assigning all operators an equal probability of being selected each iteration. 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 (here in terms of distance as in Pisinger and Ropke (2007)) and therefore easy to interchange. The q customers to remove is determined randomly each iteration and varies between 10% and 50% of the total number of customers. The set of repair heuristics we consider 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. Pisinger and Ropke (2007) use a certain number of iterations as a stopping criterion. We chose to set a maximum CPU time as stopping criterion to avoid very long computation times. The pseudocode is given in Algorithm 1.

Algorithm 1 Large Neighbourhood Search

```
Input: Problem instance $j$, Parameter setting $\theta$
Output: Best found solution $x^{best}$
    Initialization: initial solution $x$ constructed by regret-2 heuristic
    Stage 1: Vehicle Minimisation
 1: repeat
 2:   Remove one route from $x$
 3:   Schedule removed requests into remaining routes (as in Stage 2)
 4: until 20% of maximum run time met
    Stage 2: Minimisation of total distance covered
 5: repeat
 6:   select destroy and repair methods $d \in \Omega^-$ and $r \in \Omega^+$ using probabilities
      $\rho^-$ and $\rho^+$
 7:   $x^{temporary} = r(d(x))$
 8:   if $x^{temporary}$ is accepted then
 9:     $x = x^{temporary}$
10:   end if
11:   if $c(x^{temporary}) < c(x^{best})$ then
12:     $x^{best} = x^{temporary}$
13:   end if
14: until maximum run time met
```

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 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. According to Aarts et al. (2005) the values for the *cooling rate* are typically between 0.80 and 0.99 and we consider a step size of 0.0001 (i.e., 0.01%). 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: Algorithm Parameters

Parameter	**Type**	**Value ranges**
seed	Integer	U [1, 1000000]
determinism parameter	Integer	U [1, 30]
noise parameter	Discrete	U [0, 1]
cooling rate	Continuous	U[0.8000,0.9999]
start temperature control parameter	Discrete	U[0.01,0.10]
Destroy operators		
- Random removal	Dummy	True/False: ∼U[0,1]
- Worst removal	Dummy	True/False: ∼U[0,1]
- Related removal	Dummy	True/False: ∼U[0,1]
Repair operators		
- Basic greedy	Dummy	True/False: ∼U[0,1]
- Regret-2	Dummy	True/False: ∼U[0,1]

alone, or both operators are included. A similar logic was 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).

Our LNS implementation is available at https://github.com/corstjens/lns along with the problem instance sample used for the analysis in Section 5.5 and the Python script used to generate these problem instances.

5.4 Data set and model formulation

The data set serving as an input for the algorithm has 4000 scenarios, consisting of 200 randomly generated problem instances and 20 randomly created parameter settings per problem instance. The performance measure recorded is the total distance travelled by the vehicles. The analysis performed investigates how the different algorithm parameters and problem instance characteristics influence this total distance measure. Since some of these problem instance characteristics (time window width, 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 is fitted with varying (i.e., random) effects for all algorithm parameters and components, while interaction-coefficients are fixed and do not vary per problem instance. It is run using the brms package version 2.4.0 (Bürkner, 2017) in R version 3.4.2 (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[3] inference using Stan, a probabilistic programming language

[3]In Bayesian regression parameters are estimated using Bayes theorem *posterior* $\propto$ *likelihood* $\times$ *prior* to obtain a posterior distribution of a parameter based on the information available in the data (*likelihood*) and the practitioner's beliefs (*prior*) about this parameter. The regression analysis in this paper considers weakly informative priors (Bürkner, 2017) since we have no preliminary beliefs regarding these parameters, making the resulting posterior distribution primarily dependent on the information in the data. More information on Bayesian regression can be found in, for example, Wakefield (2013)

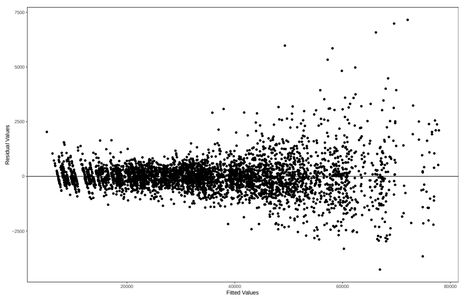

Figure 3: Fitted versus residual values for original model.

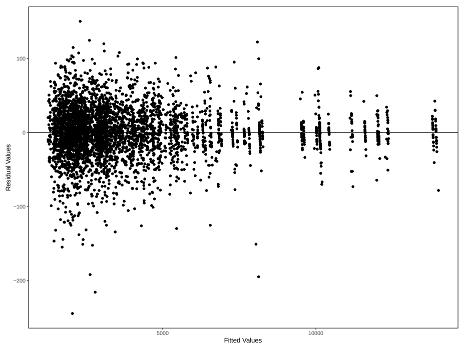

Figure 4: Fitted versus residual values for transformed model.

for statistical inference written in C++.

A first linear[4] regression model showed not to satisfy all assumptions. The residuals (cf. Figure 3) reveal the presence of heteroscedasticity. 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 (cf. equations (6) to (8)). We empirically found the reciprocal transformation of the response variable together with the cube root of the problem instance characteristic *Customers* to succeed in resolving the heteroscedasticity (cf. Figure 4).

$$\begin{aligned}
\frac{1}{Y_i} &= \alpha_{j[i]} + \beta_{1j[i]} Greedy_i + \beta_{2j[i]} Regret2_i + ... + \beta_{33j[i]} Noise_i + \epsilon_i && (6)\\
\alpha_j &= \mu_0^{\alpha} + \mu_1^{\alpha} Customers_j^{\frac{1}{3}} + ... + \mu_5^{\alpha} Runtime_j + \eta_j^{\alpha} && (7)\\
\beta_{kj} &= \mu_0^{\beta_k} + \mu_1^{\beta_k} Customers_j^{\frac{1}{3}} + ... + \mu_5^{\beta_k} Runtime_j + \eta_j^{\beta_k} \qquad \forall k \in K && (8)
\end{aligned}$$

5.5 Analysis of Results

The output[5] of the regression analysis indicates significant effects for all repair and most destroy operator combinations, the interaction of the determinism

[4]Linear is interpreted here as linear in the parameters and not as linear in the variables.
[5]The R script used for the multilevel regression analysis is provided in Appendix 6.

Table 4: Significant Effects

Variable	Estimate	Est.Error	95% CI	
Intercept[a]	$3,810.45{*}{*}$	119.69	$[3,571.73;$	$4,043.58]$
Greedy	$-157.27{*}{*}$	5.97	$[-168.90;$	$-145.48]$
Regret2	$13.43{*}{*}$	4.27	$[5.09;$	$21.81]$
Random	$21.70{*}{*}$	4.21	$[13.43;$	$29.92]$
Related	$-50.38{*}{*}$	4.68	$[-59.48;$	$-41.20]$
RandomWorst	$12.11{*}{*}$	4.40	$[3.50;$	$20.75]$
WorstRelated	$-17.30{*}{*}$	4.28	$[-25.74;$	$-8.92]$
Noise_parameter	$-11.30{*}$	4.07	$[-19.27;$	$-3.34]$
$\text{Customers}^{\frac{1}{3}}$	$-423.48{*}{*}$	27.58	$[-477.26;$	$-369.47]$
Avg_time_window_width	$37.21{*}$	17.81	$[1.53;$	$71.87]$
$\text{Customers}^{\frac{1}{3}} \times$ Runtime	$8.17{*}$	3.56	$[1.25;$	$15.21]$
Related$\times$Determinism_parameter	$-1.34{*}{*}$	0.30	$[-1.94;$	$-0.75]$
Greedy$\times$Noise_parameter	$-40.71{*}{*}$	5.88	$[-52.43;$	$-29.21]$
Greedy$\times$Random	$-61.59{*}{*}$	6.16	$[-73.71;$	$-49.64]$
Greedy$\times$Worst	$-79.79{*}{*}$	6.41	$[-92.27;$	$-67.28]$
Greedy$\times$Related	$66.71{*}{*}$	6.58	$[53.86;$	$79.48]$
Greedy$\times$RandomWorst	$-75.62{*}{*}$	6.38	$[-88.11;$	$-63.23]$
Greedy$\times\text{Customers}^{\frac{1}{3}}$	$14.19{*}{*}$	1.03	$[-16.23;$	$-12.16]$
Greedy$\times$Avg_service_time	$3.43{*}{*}$	1.06	$[1.38;$	$5.51]$
Greedy$\times$Avg_time_window_width	$-2.54{*}{*}$	0.67	$[-3.86;$	$-1.23]$
Greedy$\times$Runtime	$1.67{*}$	0.73	$[0.23;$	$3.11]$
Worst$\times\text{Customers}^{\frac{1}{3}}$	$1.77{*}{*}$	0.63	$[0.53;$	$3.01]$
Related$\times\text{Customers}^{\frac{1}{3}}$	$-6.04{*}{*}$	0.71	$[-7.43;$	$-4.66]$
Related$\times$Runtime	$-1.03{*}$	0.50	$[0.05;$	$2.03]$
RandomRelated$\times\text{Customers}^{\frac{1}{3}}$	$-1.38{*}$	0.59	$[-2.55;$	$-0.22]$

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

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

parameter with one individual destroy operator, the noise parameter, and the interaction of the noise parameter with one individual repair operator. The operator effects are also significantly moderated by certain problem instance characteristics. For all other algorithm parameters included in the model, no significant effects are found. Table 4 lists for all significant effects the estimated performance impact on $\frac{1}{Y}$ (column 'Estimate'), the uncertainty regarding this estimate (column 'Est.Error') and 95% confidence interval (columns 'l-95% CI' and 'u-95% CI'). The uncertainty regarding the coefficient estimate is used to calculate the 95% confidence interval which is interpreted as that we are 95% confident that the true performance impact lies within this value range. If this range does not include zero, the effect is indicated as significantly different from zero (for a significance level at 5%) and not due to chance. A complete summary of the regression analysis is given in Table 6 in Appendix 6.

All problem variables are centred around their mean value (before fitting the regression model) such that the intercept term 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)[6]. In this case, the expected cost is predicted

[6] Including all destroy variables or all repair variables would lead to perfect multicollinearity.

to be 26 243.62[7]. 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.

5.5.1 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 4 suggest to use random removal as sole destroy operator, since it has the largest positive performance impact (21.70) 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 (13.43) while using greedy repair alone would lead to a deterioration of performance (-157.27). 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 (21.70) compared to a solution that is destroyed by the related removal operator (-50.38). 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 4: the performance impact of switching to random removal becomes negative when accounting for the interaction with greedy repair ($21.70 - 61.59 = -39.89$), while the impact of related removal turns positive ($-50.38 + 66.71 = 16.33$).

Figure 5 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 27 675.61. The plot also shows the effect of switching from using both regret-2 and greedy 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 ($-157.27+$ interaction term). The configuration with both repair operators expects its best performance when combined with random removal (21.70), while the highest average cost measure is predicted with related removal (-50.38). The configuration with only greedy repair performs best with related removal ($-50.38 + 66.71$), and obtains the highest total cost value with worst removal ($-5.60 - 79.79$). 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 6), an improvement in the performance measure is observed for all destroy operator combinations (13.43). 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

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. Therefore, one variable of each needs to be left out and serve as a reference value which is represented in the regression intercept.

[7]The Intercept value in Table 4 is backtransformed to the original scale through division by 100 000 000 and taking the inverse of the resulting value.

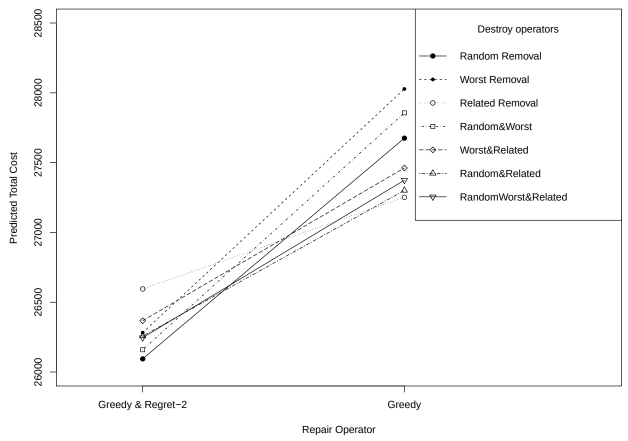

Figure 5: Total cost plot switching from Greedy & Regret-2 to Greedy.

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. We now analyse the influence these problem instance characteristics have on the performance impact of the repair operators. It is observed that it becomes more detrimental to use only greedy as repair operator as the problem size increases. This is derived from the estimates of Table 4. The impact of switching to greedy repair alone (when combined with random, worst and related removal) for an average instance is given by the estimate for *Greedy* (-157.27), the influence of the problem size is given by the estimate for the interaction term *Greedy* $\times$ *Customers*$^{\frac{1}{3}}$ (-14.19). Filling in the equation $-157.27 - 14.19\Delta Customers^{\frac{1}{3}}$ shows the impact estimate increasing 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., -61.59 is added for the combination with random removal and the equation becomes $-157.27 - 61.59 - 14.19\Delta Customers^{\frac{1}{3}}$. For the combination with related removal ($-157.27 + 66.71 - 14.19\Delta Customers^{\frac{1}{3}}$), the marginal effect of *Greedy* becomes insignificant when serving less than 73 customers, while it remains significantly negative for all other destroy operator combinations.

Further, the effect of *Greedy* is also influenced by the average service time per customer, and the average time window width. The more constraining these problem instance characteristics become, the smaller the performance differences between using greedy alone and the other two repair configurations. This is deducted from the positive influence of the average service time (3.43) and the negative influence of the average time window width (-2.54) on the effect

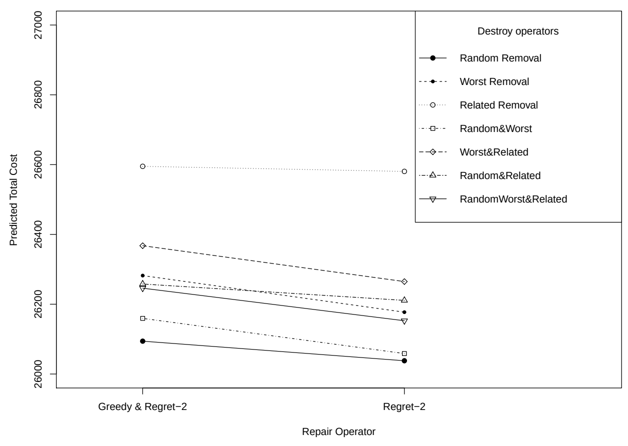

Figure 6: Total cost plot switching from Greedy & Regret-2 to Regret-2.

of *Greedy*. Finally, the effect of *Greedy* is positively influenced by the maximum run time given (1.67), 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. This is a logical deduction as you would expect performance to converge as run time increases.

In a similar way, we investigate 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 analysis suggests that worst removal performs relatively better on larger problem instances than on smaller problem instances while it is the other way around with related removal. This follows from the positive influence (1.77) of problem size on the effect of *Worst* and the negative influence (-6.04 and -1.38) on the effect of respectively *Related* and *RandomRelated*. Recall that the effect estimates in Table 4 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., -50.38), 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., $-50.38 + 66.71 = 16.33$), meaning additional customers will diminish this positive impact and at about 218 customers the estimated impact can no longer be distinguished from the configuration with all destroy operators. It even becomes significantly negative at about 319 customers. The effects of *Worst* and *RandomRelated* were not significant for an average problem instance, but become significant when accounting for the problem size influence. The effect of *Worst* is significantly negative up to 207 customers when combined with regret-2 (alone or together with greedy), but is always significantly negative with greedy as sole repair operator. The effect of *RandomRelated* is significantly positive with *Greedy* up to 216 customers and always insignificant when combined with regret-2 (alone or together with greedy).

The only other significant problem influence observed is on the impact of *Related*, which is positively moderated by the average run time (1.03). The impact for the combination with *Greedy* becomes more positive for additional run time, but is insignificant for run times up to 10 minutes. This is probably due to the fact that a related removal operator is slower than an operator like random removal and can therefore perform less iterations in the same amount of time. For the combination with either *Regret2* or *GreedyRegret2*, the impact remains indistinguishable from the configuration with all destroy operators.

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 related removal should strictly focus on removing customers that are easy to interchange. On the other hand, if related removal is 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 negative effect estimates for the determinism and noise parameter in Table 4. The determinism parameter has a negative effect on the performance impact of *Related* (-1.34). Its effect on *Worst* is insignificant, so there is not enough evidence in the data to make a valid statement regarding this effect even though its 95% confidence interval hints that it will be probably also be negative. The noise parameter negatively influences the impact of *Regret2* and *GreedyRegret2* (-11.30) and *Greedy* ($-11.30 - 40.71 = -52.01$). The importance of randomisation to ALNS performance is further investigated by Hemmati and Hvattum (2017) who propose deterministic alternatives and found they perform mostly similar to the randomised variants.

Summarising the discussion on the effect of the operators, we can conclude 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. The results also showed that diversification 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. These conclusions are confirmed in Corstjens et al. (2018) when applying a functional analysis of variance.

5.5.2 Validity Check

In order to verify whether the previous findings are not confined to that particular data set, the regression model is fitted on a second, independent data set and the obtained estimates compared with the ones in Tables 4 and 6. The effects significant in our training analysis are also significant in our test analysis, except for six terms: *Avg_time_window_width*, $Customers^{\frac{1}{3}} \times$ *Runtime*, *RandomWorst* ,*Greedy* $\times$ *Avg_service_time*, *Greedy* $\times$ *Runtime* and *Worst* $\times$ $Customers^{\frac{1}{3}}$. Since the focus is on investigating problem influences on algorithm element effects, the uncertainty regarding the terms *Avg_time_window_width* and $Customers^{\frac{1}{3}} \times$ *Runtime* is not relevant for the discussion in this paper. The test estimates of *RandomWorst*, *Greedy* $\times$ *Avg_service_time* and *Greedy* $\times$ *Runtime* are at the border of significance and are of the same direction and size as their training estimates. The test estimate of *Worst* $\times$ $Customers^{\frac{1}{3}}$ is of the same direction as the training estimate and has about the same standard error, but the size of the test estimate is only half of the training estimate. A larger sample size that reduces estimate uncertainty would probably make these effects similarly significant as was found in the training analysis. Hence, the analysis discussed in this paper does not contain any spurious significant results.

5.5.3 New questions

We 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 our 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 thought experiment about random and related removal is performed. We reason 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 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 scenarios using random removal more flexibility in repairing a solution. From the previous, we can formulate two hypotheses 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.

The operator pattern is analysed in detail in Corstjens et al. (2019). 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. So, when a removed customer cannot be reinserted in one of the existing routes, its insertion (in a new route) should not be considered less important, but rather as one that is urgent. All details on the experimental analyses performed to reach this conclusion are provided in Corstjens et al. (2019)

Similarly, we can look for reasons why related removal has more trouble with larger instances than smaller instances. Further, we also observed 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 we observe no performance difference, 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?

6 Conclusions

This paper proposes a statistical methodology 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. We see it as 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 we 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 depend 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.

Finally, 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. Further, regression model complexity grows with the number of variables added. We look into possibilities of limiting the set of included variables through some kind of preliminary importance analysis. In addition, the current methodology considers single-objective optimisation, but optimisation problems like vehicle routing typically consider multiple objectives. Our aim, therefore, is to incorporate multi-objective optimisation in the current framework, formulating a multivariate regression model.

Acknowledgments

This work is supported by the National Bank of Belgium and the Interuniversity Attraction Poles Programme initiated by the Belgian Science Policy Office (research project COMEX, Combinatorial Optimization: Metaheuristics & Exact Methods). The computational resources and services used in this work were provided by the VSC (Flemish Supercomputer Center), funded by the Research Foundation - Flanders (FWO) and the Flemish Government department EWI.

References

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

Adenso-Díaz, B., Laguna, M., 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., 1994. A Rigorous Computational Comparison of Alternative Solution Methods for the Generalized Assignment Problem. *Management Science* 40, 7, 868–890.

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.

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., 1995. Designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics* 1, 1, 9–32.

Bartz-Beielstein, T., 2006. *Experimental Research in Evolutionary Computation: The New Experimentalism.* Springer Science & Business Media.

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.

Bertsimas, D.J., Simchi-Levi, D., 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*, *Studies in Computational Intelligence.* Vol. 197. Springer Berlin Heidelberg.

Box, G.E.P., Hunter, J.S., Hunter, W.G., 2005. *Statistics for experimenters: design, innovation, and discovery.* Wiley-Interscience.

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

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.

Calvet, L., Juan, A.A., Serrat, C., Ries, J., 2016. A statistical learning based approach for parameter fine-tuning of metaheuristics. *SORT-Statistics and Operations Research Transactions* 1, 1, 201–224.

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

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

Corstjens, J., Caris, A., Depaire, B., 2019. Explaining heuristic performance differences for vehicle routing problems with time windows. In Kotsireas, I. and Pardalos, P. (eds), *Learning and Intelligent Optimization. LION 12 2018, Lecture Notes in Computer Science*. Vol. 11353. Springer, pp. 159–174.

Corstjens, J., Dang, N., Depaire, B., Caris, A., De Causmaecker, P., 2018. A combined approach for analysing heuristic algorithms. *Journal of Heuristics*

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

Cuervo, D.P., Goos, P., Sörensen, K., Arráiz, E., 2014. An iterated local search algorithm for the vehicle routing problem with backhauls. *European Journal of Operational Research* 237, 2, 454–464.

Cutler, D.R., Edwards Jr, T.C., Beard, K.H., Cutler, A., Hess, K.T., Gibson, J., Lawler, J.J., 2007. Random forests for classification in ecology. *Ecology* 88, 11, 2783–2792.

De Leeuw, J., Meijer, E., Goldstein, H., 2008. *Handbook of multilevel analysis*. Springer.

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.

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

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

Gendreau, M., Potvin, J.Y., 2010. *Handbook of metaheuristics*, Vol. 2. Springer.

Gibco, 2016. *Cell Culture Basics*. Invitrogen Life Technologies.

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

Goodfellow, I., Bengio, Y., Courville, A., Bengio, Y., 2016. *Deep learning*, Vol. 1. MIT press Cambridge.

Gunawan, A., Lau, H.C., et al., 2011. Fine-tuning algorithm parameters using the design of experiments approach. In *International Conference on Learning and Intelligent Optimization*, Springer, pp. 278–292.

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., 2007. Generalized functional anova diagnostics for high-dimensional functions of dependent variables. *Journal of Computational and Graphical Statistics* 16, 3, 709–732.

Hooker, J.N., 1994. Needed: An empirical science of algorithms. *Operations Research* 42, 2, 201–212.

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

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

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 *International Conference on Learning and Intelligent Optimization*, Springer, 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. and Pardalos, P. (eds), *Learning and Intelligent Optimization.* Springer Berlin Heidelberg, number 7997 In Lecture Notes in Computer Science, pp. 364–381.

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

Jones, Z., Linder, F., 2015. Exploratory data analysis using random forests. In *Prepared for the 73rd annual MPSA conference.*

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

Kendall, G., Bai, R., Błazewicz, J., De Causmaecker, P., Gendreau, M., John, R., Li, J., McCollum, B., Pesch, E., Qu, R., et al., 2016. Good laboratory practice for optimization research. *Journal of the Operational Research Society* 67, 4, 676–689.

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

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

Mason, R.L., Gunst, R.F., Hess, J.L., 2003. *Statistical design and analysis of experiments: with applications to engineering and science*, Vol. 474. John Wiley & Sons.

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

McNabb, M.E., Weir, J.D., Hill, R.R., Hall, S.N., 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.

Moore, D.S., Craig, B.A., McCabe, G.P., 2012. *Introduction to the Practice of Statistics.* WH Freeman.

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

Palhazi Cuervo, D., Goos, P., Srensen, K., Arriz, E., 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., 2007. A general heuristic for vehicle routing problems. *Computers & Operations Research* 34, 8, 2403–2435.

Pongcharoen, P., Chainate, W., Thapatsuwan, P., 2007. Exploration of genetic parameters and operators through travelling salesman problem. *Science Asia* 33, 2, 215–222.

R Core Team, 2016. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria.

Rahimi-Vahed, A., Crainic, T.G., Gendreau, M., Rei, W., 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., 2001. Experimental Evaluation of Heuristic Optimization Algorithms: A Tutorial. *Journal of Heuristics* 7, 3, 261–304.

Ribeiro, C.C., Rosseti, I., Souza, R.C., 2011. Effective probabilistic stopping rules for randomized metaheuristics: Grasp implementations. In *International Conference on Learning and Intelligent Optimization*, Springer, pp. 146–160.

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

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. Analyzing heuristic performance with response surface models: prediction, optimization and robustness. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, ACM, pp. 150–157.

Ries, J., Beullens, P., Salt, D., 2012. Instance-specific multi-objective parameter tuning based on fuzzy logic. *European Journal of Operational Research* 218, 2, 305–315.

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.

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

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. and Puget, J.F. (eds), *Principles and Practice of Constraint Programming CP98*. Springer Berlin Heidelberg, number 1520 In Lecture Notes in Computer Science, pp. 417–431.

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.

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., 2015. Generating new test instances by evolving in instance space. *Computers & Operations Research* 63, 102–113.

Snijders, T.A., 2011. Multilevel analysis. In *International encyclopedia of statistical science*. Springer, pp. 879–882.

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., 2013. Statistical analysis of distance-based path relinking for the capacitated vehicle routing problem. *Computers & Operations Research* 40, 12, 3197–3205.

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

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.

Wakefield, J., 2013. *Bayesian and frequentist regression methods.* Springer Science & Business Media.

Xu, J., Chiu, S.Y., Glover, F., 1998. Fine-tuning a tabu search algorithm with statistical tests. *International Transactions in Operational Research* 5, 3, 233–244.

Appendix A

Table 5: 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

```r
library(brms)

Data <- read.table ("./Summary_LNS_experiment.csv", header=T, sep=",")

#variable transformations
Data$Customers <- scale(Data$Customers, center=TRUE, (*@scale@*)=FALSE)
Data$Customers_cbrt <- sign(Data$Customers)*abs(Data$Customers)**(1/3)
Data$Avg_servtime <- scale(Data$Avg_servtime, center=TRUE, (*@scale@*)=FALSE)
Data$Avg_tw_width <- scale(Data$Avg_tw_width, center=TRUE, (*@scale@*)=FALSE)
Data$Avg_demand <- scale(Data$Avg_demand, center=TRUE, (*@scale@*)=FALSE)
Data$Runtime <- scale(Data$maxruntime/60, center=TRUE, (*@scale@*)=FALSE)
Data$Cooling_rate <- scale(Data$cooling_rate, center=TRUE, (*@scale@*)=FALSE)
Data$Start_temp_ctrl_param <- scale(Data$Start_temp_ctrl_param,center=TRUE, (*@scale@*)=FALSE)
Data$Noise_param <- scale(Data$Noise_param, center=TRUE, (*@scale@*)=FALSE)
Data$Determinism_param <- scale(Data$Determinism_param, center=TRUE, (*@scale@*)=FALSE)

Cost_inv <- (1/Data$total_cost)*100000000

# In the model formulated below ':' are used to formulate interaction terms, the random
# effects are defined as a sum between brackets at the end of the model with '1'
# indicating the random intercept and the grouping factor defined after '|'.
# The default value for max_treedepth is 10 and should be raised when model output
# suggests to increase it. There are 4 Markov chains used (default), 8000 iterations
# per chains of which 2000 are warmup.
M1 <- brm(Cost_inv ~ Greedy + (*@Regret2@*) + Random + Worst + Related + RandomWorst +
          WorstRelated + RandomRelated + Cooling_rate + Start_temp_ctrl_param +
          Cooling_rate:Start_temp_ctrl_param + Noise_param + Determinism_param +
          Determinism_param:Random + Determinism_param:Worst + Determinism_param:Related +
          Determinism_param:RandomWorst + Determinism_param:WorstRelated +
          Determinism_param:RandomRelated +
          Noise_param:Greedy + Noise_param:(*@Regret2@*) +
          Greedy:Random + Greedy:Worst +Greedy:Related + Greedy:RandomWorst +
          Greedy:WorstRelated + Greedy:RandomRelated + (*@Regret2@*):Random + (*@Regret2@*):Worst +
          (*@Regret2@*):Related + (*@Regret2@*):RandomWorst + (*@Regret2@*):WorstRelated +
          (*@Regret2@*):RandomRelated +
          Customers_cbrt + Avg_servtime + Avg_tw_width + Avg_demand + Runtime +
          Customer_number_cbrt:Runtime +
          Greedy:Customers_cbrt + Greedy:Avg_servtime + Greedy:Avg_tw_width +
          Greedy:Avg_demand + Greedy:Runtime +
          (*@Regret2@*):Customers_cbrt + (*@Regret2@*):Avg_servtime + (*@Regret2@*):Avg_tw_width +
          (*@Regret2@*):Avg_demand + (*@Regret2@*):Runtime +
          Random:Customers_cbrt + Random:Avg_servtime + Random:Avg_tw_width +
          Random:Avg_demand + Random:Runtime +
          Worst:Customers_cbrt + Worst:Avg_servtime + Worst:Avg_tw_width +
          Worst:Avg_demand + Worst:Runtime +
          Related:Customers_cbrt + Related:Avg_servtime + Related:Avg_tw_width +
          Related:Avg_demand + Related:Runtime +
          RandomWorst:Customers_cbrt + RandomWorst:Avg_servtime +
          RandomWorst:Avg_tw_width + RandomWorst:Avg_demand + RandomWorst:Runtime +
          WorstRelated:Customers_cbrt + WorstRelated:Avg_servtime +
          WorstRelated:Avg_tw_width + WorstRelated:Avg_demand + WorstRelated:Runtime +
          RandomRelated:Customers_cbrt + RandomRelated:Avg_servtime +
          RandomRelated:Avg_tw_width  + RandomRelated:Avg_demand + RandomRelated:Runtime +
          Cooling_rate:Customers_cbrt + Cooling_rate:Avg_servtime +
          Cooling_rate:Avg_tw_width + Cooling_rate:Avg_demand + Cooling_rate:Runtime +
          Start_temp_ctrl_param:Customers_cbrt + Start_temp_ctrl_param:Avg_servtime +
          Start_temp_ctrl_param:Avg_tw_width + Start_temp_ctrl_param:Avg_demand +
          Start_temp_ctrl_param:Runtime +
          Determinism_param:Customers_cbrt + Determinism_param:Avg_servtime +
          Determinism_param:Avg_tw_width + Determinism_param:Avg_demand +
          Determinism_param:Runtime +
          Noise_param:Customers_cbrt + Noise_param:Avg_servtime +
          Noise_param:Avg_tw_width + Noise_param:Avg_demand + Noise_param:Runtime +
         (1 + Greedy + Regret2 + Random + Worst + Related + RandomWorst + WorstRelated +
          RandomRelated + Cooling_rate + Start_temp_ctrl_param + Determinism_param +
          Noise_param|problem_instance), data= Data, control = list((*@max@*)_treedepth = 12),
          chains = 4,warmup = 2000, iter = 8000, cores=4)

#Generate regression table
summary(M1)
```

Table 6: Summary Table Multilevel Regression Analysis

Variable	Estimate	Est.Error	95% CI
Intercept	3,810.45**	119.69	[3,571.73; 4,043.58]
Greedy	−157.27**	5.97	[−168.90; −145.48]
Regret2	13.43**	4.27	[5.09; 21.81]
Random	21.70**	4.21	[13.43; 29.92]
Worst	−5.60	4.53	[−14.49; 3.39]
Related	−50.38**	4.68	[−59.48; −41.20]
RandomWorst	12.11**	4.40	[3.50; 20.75]
WorstRelated	−17.30**	4.28	[−25.74; −8.92]
RandomRelated	−1.90	4.37	[−10.49; 6.62]
Cooling_rate	−18.55	12.58	[−43.24; 6.40]
Start_temperature_control_parameter	−22.02	25.90	[−72.77; 28.42]
Noise_parameter	−11.30**	4.07	[−19.27; −3.34]
Determinism_parameter	0.17	0.22	[−0.26; 0.60]
Customers$^{\frac{1}{3}}$	−423.48**	27.58	[−477.26; −369.47]
Avg_service_time	−18.19	28.31	[−72.15; 37.90]
Avg_time_window_width	37.21*	17.81	[1.53; 71.87]
Avg_demand	69.79	114.49	[−149.39; 301.35]
Runtime	−14.74	20.34	[−54.44; 25.27]
Cooling_rate × Start_temperature_control_parameter	−392.47	439.16	[−1,253.45; 475.22]
Random × Determinism_parameter	−0.23	0.29	[−0.81; 0.34]
Worst × Determinism_parameter	−0.41	0.30	[−1.00; 0.18]
Related × Determinism_parameter	−1.34**	0.30	[−1.94; −0.75]
RandomWorst × Determinism_parameter	−0.03	0.30	[−0.61; 0.57]
WorstRelated × Determinism_parameter	−0.07	0.29	[−0.66; 0.51]
RandomRelated × Determinism_parameter	−0.20	0.30	[−0.78; 0.37]
Greedy × Noise_parameter	−40.71**	5.88	[−52.43; −29.21]
Regret2 × Noise_parameter	−5.56	5.60	[−16.47; 5.35]
Greedy × Random	−61.59**	6.16	[−73.71; −49.64]
Greedy × Worst	−79.79**	6.41	[−92.27; −67.28]
Greedy × Related	66.71**	6.58	[53.86; 79.48]
Greedy × RandomWorst	−75.62**	6.38	[−88.11; −63.23]
Greedy × WorstRelated	5.73	6.04	[−6.07; 17.61]
Greedy × RandomRelated	11.50	6.31	[−0.87; 23.88]
Regret2 × Random	−5.64	5.96	[−17.30; 6.12]
Regret2 × Worst	1.75	6.14	[−10.26; 13.79]
Regret2 × Related	−11.49	6.27	[−23.89; 0.99]
Regret2 × RandomWorst	1.43	6.18	[−10.52; 13.64]
Regret2 × WorstRelated	0.83	5.98	[−11.01; 12.49]
Regret2 × RandomRelated	−6.40	5.96	[−18.04; 5.31]
Customers$^{\frac{1}{3}}$ × Runtime	8.17*	3.56	[1.25; 15.21]
Greedy× Customers$^{\frac{1}{3}}$	−14.19**	1.03	[−16.23; −12.16]
Greedy× Avg_service_time	3.43**	1.06	[1.38; 5.51]
Greedy× Avg_time_window_width	−2.54**	0.67	[−3.86; −1.23]
Greedy1× Avg_demand	2.65	4.29	[−5.65; 11.09]
Greedy1× Runtime	1.67*	0.73	[0.23; 3.11]
Regret21× Customers$^{\frac{1}{3}}$	0.63	0.38	[−0.13; 1.38]
Regret2× Avg_service_time	0.18	0.38	[−0.57; 0.94]
Regret2× Avg_time_window_width	0.01	0.25	[−0.48; 0.49]
Regret2× Avg_demand	−0.21	1.62	[−3.40; 2.98]
Regret2× Runtime	−0.51	0.27	[−1.03; 0.02]
Random× Customers$^{\frac{1}{3}}$	−0.35	0.59	[−1.51; 0.81]
Random× Avg_service_time	0.55	0.60	[−0.62; 1.75]
Random× Avg_time_window_width	0.08	0.39	[−0.70; 0.85]
Random× Avg_demand	−1.64	2.53	[−6.52; 3.35]
Random× Runtime	0.13	0.42	[−0.69; 0.96]
Worst× Customers$^{\frac{1}{3}}$	1.77**	0.63	[0.53; 3.01]
Worst× Avg_service_time	−0.01	0.64	[−1.27; 1.24]
Worst× Avg_time_window_width	−0.22	0.42	[−1.05; 0.62]
Worst× Avg_demand	0.11	2.71	[−5.18; 5.37]
Worst× Runtime	0.21	0.46	[−0.68; 1.10]
Related× Customers$^{\frac{1}{3}}$	−6.05**	0.71	[−7.43; −4.66]
Related× Avg_service_time	0.34	0.71	[−1.06; 1.73]

Related× Avg_time_window_width	−0.48	0.47	[−1.39;	0.43]
Related× Avg_demand	−2.47	2.91	[−8.18;	3.30]
Related× Runtime	1.03*	0.50	[0.05;	2.03]
RandomWorst× Customers$^{\frac{1}{3}}$	0.29	0.61	[−0.88;	1.49]
RandomWorst× Avg_service_time	0.44	0.61	[−0.76;	1.63]
RandomWorst× Avg_time_window_width	−0.08	[0.40;	−0.87]	0.71
RandomWorst× Avg_demand	0.26	2.58	[−4.79;	5.32]
RandomWorst× Runtime	−0.01	0.43	[−0.85;	0.82]
WorstRelated× Customers$^{\frac{1}{3}}$	−1.07	0.58	[−2.21;	0.06]
WorstRelated× Avg_service_time	0.39	0.58	[−0.75;	1.54]
WorstRelated× Avg_time_window_width	0.09	0.39	[−0.67;	0.85]
WorstRelated1× Avg_demand	0.58	2.40	[−4.08;	5.29]
WorstRelated1× Runtime	0.52	0.40	[−0.27;	1.31]
RandomRelated× Customers$^{\frac{1}{3}}$	−1.38**	0.59	[−2.55;	−0.22]
RandomRelated× Avg_service_time	0.07	0.60	[−1.11;	1.25]
RandomRelated× Avg_time_window_width	−0.15	0.40	[−0.93;	0.63]
RandomRelated× Avg_demand	−2.16	2.45	[−6.95;	2.64]
RandomRelated× Runtime	0.05	0.41	[−0.76;	0.87]
Cooling_rate× Customers$^{\frac{1}{3}}$	−2.52	2.89	[−8.15;	3.15]
Cooling_rate× Avg_service_time	1.66	3.01	[−4.30;	7.56]
Cooling_rate× Avg_time_window_width	1.16	1.92	[−2.60;	4.94]
Cooling_rate× Avg_demand	−0.46	11.99	[−24.04;	23.04]
Cooling_rate× Runtime	1.84	2.04	[−2.19;	5.86]
Start_temperature_control_parameter × Customers$^{\frac{1}{3}}$	−8.63	5.95	[−20.21;	3.11]
Start_temperature_control_parameter × Avg_service_time	−3.58	6.07	[−15.41;	8.19]
Start_temperature_control_parameter × Avg_time_window_width	−4.21	3.97	[−12.00;	3.50]
Start_temperature_control_parameter × Avg_demand	14.57	25.17	[−34.40;	64.03]
Start_temperature_control_parameter × Runtime	−1.43	4.29	[−9.85;	6.97]
Determinism_parameter× Customers$^{\frac{1}{3}}$	0.01	0.02	[−0.03;	0.05]
Determinism_parameter× Avg_service_time	0.01	0.02	[−0.03;	0.06]
Determinism_parameter× Avg_time_window_width	−0.003	0.01	[−0.03;	0.03]
Determinism_parameter× Avg_demand	0.02	0.10	[−0.17;	0.21]
Determinism_parameter× Runtime	−0.02	0.02	[−0.05;	0.02]
Noise_parameter× Customers$^{\frac{1}{3}}$	−0.47	0.60	[−1.64;	0.72]
Noise_parameter× Avg_service_time	−0.09	0.61	[−1.29;	1.10]
Noise_parameter× Avg_time_window_width	−0.44	0.40	[−1.22;	0.34]
Noise_parameter× Avg_demand	−0.89	2.49	[−5.80;	3.91]
Noise_parameter× Runtime	0.12	0.42	[−0.71;	0.95]

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