

Performance Evaluation of Vehicle Routing Heuristics

by

Louis Sikkes



Performance Evaluation of Vehicle Routing Heuristics

by

Louis Sikkes

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday October 28, 2019 at 10:00 AM.



Student number:	4298977	
Project duration:	February 1, 2019 – October 28, 2019	
Thesis committee:	Dr. Neil Yorke Smith	Associate Professor Algorithmics, TU Delft
	Dr. Annibale Panichella	Assistant Professor Software Engineering, TU Delft
	Joost Donkers	Software Engineer, ORTEC

An electronic version of this thesis is available at <http://repository.tudelft.nl/>.

Abstract

This thesis has researched the automation of performance evaluation of vehicle routing heuristics. The trade-off between solution quality, which is composed of multiple variables, and runtime make performance evaluation challenging. Therefore, it is often done by human experts. The research question of this thesis is: “How can we determine a performance measure that correctly represents the trade off between quality and runtime in vehicle routing heuristics?”.

A literature review revealed that much research was done on performance evaluation, but not on heuristics specifically. The performance profile, a cumulative distribution function, is said to reflect all major performance characteristics of a solver. This, combined with a clustering algorithm, is used in this thesis in a classifier to detect performance anomalies. The performance profile needs a performance measure, for which three options were introduced: the area under the chart, the quality at the same time and the maximum difference. Through experimentation, 18 measure configurations were tested and rated on their accuracy and apparent issues. Three of the measure configurations have promising results, with an accuracy of roughly 80%.

Preface

Before you lies my thesis "Performance Evaluation of Vehicle Routing Heuristics". This thesis has researched the automation of performance evaluation and specifically, ways to evaluate the quality versus runtime trade off. After little over 6 years, this thesis earns me the degree of master of science in computer science at the Delft University of Technology, where I also earned my bachelors degree. This thesis started with nothing but a subject and a completely new environment. Especially the first few months were difficult, where in a new environment and with a new subject I was posed with the challenge of formulating a research question. After 9 months of hard work, I am very happy with the end result.

I want to thank everyone who has supported me in the past months while writing my thesis. First, I want to thank my supervisors Neil Yorke-Smith of the Delft University of Technology and Joost Donkers of ORTEC for their advise during my thesis. Then, I want to thank everyone at ORTEC, who have provided me with the context for my project and a friendly and helpful environment for me to work in. Lastly, I want to thank my family for their support and encouragement during my years of study.

*Louis Sikkes
Delft, October 2019*

Contents

1	Introduction	1
1.1	Context description	1
1.1.1	Introduction to ORTEC	1
1.1.2	Vehicle routing problems	2
1.1.3	The performance testing procedure	3
1.2	Motivation	4
1.2.1	Issues with current procedure	4
1.2.2	Research questions	4
1.3	Methodology	5
1.3.1	Contributions	5
1.4	Overview of remaining chapters	5
2	Literature Survey	7
2.1	Literature survey strategy	7
2.1.1	Systematic approach to conducting literature survey	7
2.1.2	The do's and don'ts of experimental algorithm analysis	8
2.1.3	Good lab practice: what to do to ensure reproducible results	9
2.2	Performance analysis	9
2.2.1	How to measure performance	9
2.2.2	Performance analysis of optimization algorithms	9
2.2.3	Performance analysis in vehicle routing problems	11
2.2.4	Performance analysis outside optimization algorithms	12
2.2.5	Code profiling	13
2.3	Anomaly detection	14
2.3.1	Anomaly detection techniques.	14
2.3.2	Types of anomalies.	14
2.3.3	Applicable research areas	15
2.3.4	Anomaly detection using clustering	15
2.4	Summary	16
3	Classifier Design	17
3.1	Available data	17
3.1.1	Key Performance Indicators	17
3.1.2	Runtime measures	18
3.2	Performance profiles to create an overview	19
3.2.1	Motivation for using performance profiles	19
3.2.2	Implementation of performance profiles	19
3.3	Anomaly detection using clustering.	20
3.3.1	K-means implementation	20
3.4	Implementation	21
3.4.1	Step 1: Analysing test runs	21
3.4.2	Step 2: Analysing test cases	22
3.4.3	Step 3: Analysing the results	22
3.5	Summary	23
4	Performance Measures	25
4.1	Issues with runtime as performance measure.	25
4.2	Alternative performance measures	25
4.2.1	Quality comparison at same runtime	26
4.2.2	Area under the chart	26
4.2.3	Maximum difference in cost between solutions	27

4.3	Alternative measuring points	28
4.3.1	Determining the end time when runtimes are different	28
4.3.2	Alternative times for performance comparison	28
4.4	Summary	28
5	Experiments	31
5.1	Experiment design	31
5.1.1	Approach	31
5.1.2	Metrics for evaluating combinations	32
5.1.3	Cross-validation	33
5.2	Exploratory test case investigation	33
5.2.1	Conclusion.	35
5.3	Results	35
5.3.1	Comparison of measure variables	35
5.3.2	Comparison of test runs marked as anomalous	38
5.3.3	Issues with the classifier	38
5.3.4	Training data conclusions	38
5.3.5	Analysis of verification data results	39
5.4	Summary	40
6	Conclusion	41
6.1	Future work.	42
A	Evaluation sheet	45
B	Tool images	47
	Bibliography	51

Introduction

By 2050, the world's population is estimated to reach about 9 billion. Without innovation and optimization, three planets are needed in terms of available resources. ORTEC believes that for organizations, advanced analytics and optimization are key to survive, innovate and outperform. ORTEC is one of the largest suppliers of optimization software and analytics solutions of the world. They own a large variety of software products. The research of this thesis was done in collaboration with ORTEC and will focus specifically on their vehicle routing products.

This report will highlight the most important aspects of a thesis that has researched the automation of performance analysis in heuristics for vehicle routing problems. Performance analysis is often conducted through performance tests. These tests consist of common tasks for the software to execute that often require a hefty amount of resources. Unlike other types of tests, performance tests are not a simple pass or fail, but are more complicated in their evaluation. The trade off between solution quality, which is composed of multiple variables, and the runtime make this a hard task to do. Therefore, this evaluation is often done by human experts. As part of this thesis, a tool was developed that aids developers by detecting anomalies in the results of performance tests and provides additional insights, which can be helpful to resolve the issue at hand.

This introduction will first describe the problem at hand by providing the context to the research. This then leads into the issues with the current situation and a motivation of the research questions. Next, the methodology will outline how these research questions will be answered and what the contributions of this thesis are. Finally, an overview of the remainder of this document is given.

1.1. Context description

Before presenting the research questions and contributions made by this thesis, it is important to gain some knowledge about performance testing and the specific situation that will be researched during this thesis. This section will provide that information. Since this thesis was done in collaboration with ORTEC, the company will first be shortly introduced to provide the context of the research. Then, before describing the contributions of this thesis, the vehicle routing problem and the procedure of performance testing are explained.

1.1.1. Introduction to ORTEC

As stated earlier, this thesis has been done in collaboration with ORTEC. ORTEC is a company that specialises in optimisation software. Their biggest product solves vehicle routing problems at large customers all over the world. This product is being researched in this thesis. Even though the research is done within this specific context, abstractions are made so that the research can be applied to other situations as well, in order to increase its usefulness.

Customers expect orders to be delivered correctly and on time, regardless of last-minute changes, traffic jams or defects. The routing products of ORTEC optimize daily activities by dynamically planning deliveries, assets and loads. They also offer the possibility to plan for a specific season or a multi-year strategy. The routing software offers multiple solutions for varying scenario's so manufacturers can react to last-minute order changes and can take well-founded decisions regarding complex routes and deliveries. The results are perfectly executed logistical processes in every situation. The software developed at ORTEC is deployed for large companies like Coca Cola and Walmart.

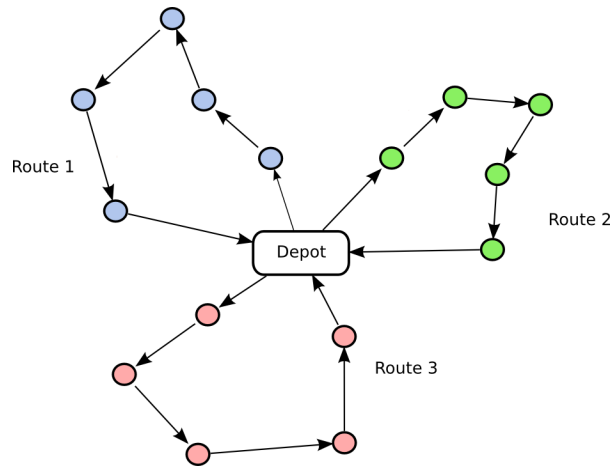


Figure 1.1: A simple example of the vehicle routing problem. Given are 1 depot with goods, 15 delivery locations and 3 available vehicles. The goal is to determine the most efficient route planning. Source: [38]

ORTEC is responsible for development and maintenance of their standard software solutions (products) for resource optimization. Their activities also include software testing, product support and UX/UI design. They own multiple products, but the focus of this project will be the vehicle routing optimizer. Other products include vehicle load optimization, warehouse logistical problems and workforce scheduling.

1.1.2. Vehicle routing problems

The vehicle routing problem (VRP) is an optimization problem with the goal of planning transport routes for a set of vehicles. Its goal is to deliver goods to the right location, with regards to a set of given constraints. The vehicle routing problem is a generalisation of the traveling salesman problem. Given a set of available vehicles, the current location and the destination of goods, the algorithm tries to create an efficient route plan for each vehicle. A visual example of the vehicle routing problem can be seen in Figure 1.1, which includes one depot from which 3 vehicles deliver goods to 15 locations. This problem is known to be NP-hard, which means that there is no method available to solve all problems in a reasonable amount of time, especially as cases grow large. Due to the practical applications of this problem, it has been researched a lot in the existing literature. There are many variations of the problem that include multiple depots, pickups of goods, time windows on pickups or deliveries and more, see Braekers et al. [11]. As a result it is impossible to generalise the research on these different situations.

Since the problem is NP-hard, most practical solutions to the vehicle routing problem make use of heuristics. Heuristics are algorithms that have shown to give good solutions in a small amount of time. These are different from exact methods, which calculate the optimal solution, rather than a good solution. Another category of optimization algorithms is approximation algorithms. These also calculate good solutions within a short amount of time. The difference between approximation algorithms and heuristics is that approximation algorithms have a mathematical boundary on solution quality that is proven. In contrast, the solution quality of heuristics is shown to be good in practice, based on past experiences, but has no mathematical prove behind it.

The primary objective of the software is to provide a high quality solution, a route planning. The reason this software is used is to reduce the cost for executing the route plan by means of optimizing several objectives. Since this has to be done on a daily basis and orders are often added throughout the day, this introduces a constraint on the amount of time available to obtain this solution. Since the problem instances are large and there is a restriction on the amount of computation time available, it is not feasible to calculate the optimal solution. The solution created by ORTEC combines multiple heuristics to calculate a route. Rather than the optimal solution, it creates a basic solution that satisfies the given constraints. Then, among many iterations, several heuristics aim to improve this solution. The algorithm that is run each iteration is semi-random. The routes are created to optimize KPIs (Key Performance Indicator's), for instance to minimize distance and cost and to maximize the amount of tasks planned. The solution that was implemented to solve the quality versus runtime trade off is that the software executes a set amount of optimization iterations. The optimization calls that plan the routes are complicated. Several algorithms are sequentially executed to improve the routes bit

by bit. One optimization call includes hundreds of trucks and thousands of stops. It takes a lot of computational power to calculate a solution.

Although each customer uses the same software, there are large differences in requirements for each customer. Some customers may have 10 minutes to calculate their daily routes, while others may have 30 minutes. One customer might do parcel delivery with small trucks, while another does restocking of warehouses with large trucks. There are many more different characteristics that introduce different challenges. It is hard to have a single piece of software be able to handle all of these different situations. The way that ORTEC handles this is by using settings to be able to tweak the software to the needs of each customer. These settings include the heuristics that are being run, but also the amount of iterations to come to a solution. There is a team of specialists within ORTEC to tweak these settings for each customer, with the help of several tools. Since this is outside the scope of this research, this will not further be discussed. The important aspect is that the results of the performance tests serve as a general basis. The test cases used in the performance tests serve to create confidence that the current software is of good quality. Then, when the software is deployed at a customer, the settings will be tweaked in order to optimize the results for that specific customer.

1.1.3. The performance testing procedure

As part of the testing procedure of ORTEC, 200 dedicated virtual machines run millions of automated tests every night. Among these tests are the performance tests. Performance tests measure the amount of resources required to perform certain predefined tasks. One problem that performance tests often face is that when the tests are ran on different machines, this introduces irregularities in the results of the tests. In order to keep the measurements of these resources as fair as possible, the performance tests at ORTEC are ran on a dedicated machine that only runs performance tests. Although this does not completely remove these irregularities, it highly reduces them to a small margin. Throughout the remainder of this thesis, it is assumed that these irregularities are negligible. Performance tests produce a set of resource measurements through benchmarking multiple scenarios. These measurements then have to be analysed to determine whether the software shows regression. The performance tests at ORTEC generate thousands of resource measurements on a weekly basis.

The situation under research makes use of Jenkins [28] to automate running the performance tests. There are roughly 20 regularly run test cases that present different scenarios in the routing software. These can range from parcel deliveries (which require a lot of small packages at multiple destinations) to restocking warehouses (where a truck needs to transport all its goods to one location) to everything in between. Jenkins runs these cases on a weekly basis on the most recent development version and stores the results. This setup has changed about a year before the start of this thesis from a similar setup with some minor differences. The reason for this change was the adoption of Jenkins. The test cases have stayed similar although their naming has changed slightly. The regularity of running these tests was not on a weekly basis, but rather when developers deemed it necessary. Although multiple test runs were conducted monthly, this resulted in a more skewed result set with a difference in which tests were run and on what time interval.

Performance analysis in other areas of software engineering, such as enterprise software or web applications, often has a clear objective; regression is defined as an increase in resource usage. The goal of performance analysis is then to observe the resource usage of common tasks over time and detect regression. In contrast, the primary goal of a heuristic is not just to minimize the amount of resources spent, but rather to improve the quality of the solution with regards to the KPIs. Resource usage is still an important aspect, but a secondary one rather than the primary objective. As a result, an increase in resource usage is not necessarily a regression, as long as the quality of the solution has improved accordingly.

The step of analysing the performance results, which is arguably the most important step, since it involves drawing conclusions, is currently done by a human expert. Although there is a tool to observe the results, by means of visualizing them, no system has yet been developed to process these results. Due to the complex nature of these optimization calls, small changes in the product can have a huge impact on performance. These performance changes can result from necessary changes, which can result in a decrease in performance but are necessary, or can be the result of a human error or illogical decision. In the second case, the changes that resulted in this change of performance have to be investigated. Small changes in the performance results are often a strong indicator of problematic performance issues. Conversely, small changes in the performance results can come from changes in the algorithms and do not necessarily mean that there is a performance issue. With continuous developments on the product and its algorithms in such a complex environment, improvements on the detection of performance anomalies can severely increase development efficiency.

1.2. Motivation

After introducing the context of this project in the previous section, this section will describe what will be researched during this project and provide argumentation why. The first subsection will describe the issues with the current procedure. The second subsection will present the research questions that followed from these issues.

1.2.1. Issues with current procedure

Anomalies can be defined as patterns in the data that do not behave as expected. Anomalies can be found by comparing the data points against each other and observing which ones are different from most. Note that different is a vague description and can be interpreted in several ways. Finding these anomalies, also called outliers, is called anomaly detection [16]. The goal of analysing the results of the performance tests is to detect anomalies, since these are indicators for regressions or bugs. Whenever an anomaly is detected, the next step is for a developer to investigate the potential issue. There is often limited information available about the sources of an anomaly and detecting the issue often requires a large amount of time. Usually, finding the source of the anomaly takes up a substantial amount of time, compared to actually fixing the issue.

As mentioned before, the analysis of the performance results is currently done by a human expert. This brings several issues that could be solved by automating this process. First of all, doing this analysis by hand on a regular basis requires time and expertise. Especially expertise is a tricky one, since its definition is not immediately clear and employees with the required knowledge could be unavailable or not present at all. To add to this, humans are not objective in their judgements. Experts often look at a certain set of measures, based on their previous experience. This can leave clear signs in other measurements completely ignored, resulting in issues left unnoticed.

Secondly, it is hard for a human to judge the overall performance of a build based on a set of results for several test cases. What should the conclusion be if one case shows regression and another shows improvements? Is there a threshold for deciding whether something is a regression or just a result of the hardware irregularity? There are no clear guidelines here, which make it hard to judge the overall performance of a test run and rather than rely on the data, experts often rely on past experience.

Finally, even within a test case it is hard to determine whether it has regressed or not. Heuristics aim to optimize several objectives while still remaining fast. Again, there are no guidelines as to what makes a regression a regression. What if the distance has reduced by 5% but the runtime has increased by 15%? And what if the runtime has instead decreased by 25%? Different experts would judge these situations differently based on their own experience, which makes it unreliable. There is no single measure available yet that represents the trade off between solution quality and runtime. The main focus of this thesis will be on finding a good measurement to compare two test runs.

Although fully automating the analysis of performance results would be ideal, it is not a feasible objective. A change in performance can be a logical result of for instance a bug fix, in which case no action should be taken. On the other hand it could be a unintended side issue that arises from a human error or illogical change, in which case the issue should be further investigated. The software responsible for the optimization is complex and non-deterministic. As a result, fully automating this process is deemed unfeasible. Instead the change in performance should be analyzed and when deemed necessary a notification should be made. A person can then further investigate the issue and determine the next steps.

To summarize, the current issues are:

- Human experts are non-deterministic and in practice often do not do analysis regularly.
- Given a set of measures from multiple test cases and multiple test runs, it is hard to judge the overall performance of a test run.
- It is unclear which of the many measures correctly represent a regression.

1.2.2. Research questions

During this thesis, a classifier was created to mark builds as anomalous. This classifier was inspired by the performance profile and initially used the runtime as a performance measure. After evaluation of this classifier, it was concluded that the runtime was not an appropriate measure to evaluate the performance of heuristics. In heuristics, there is a trade off between runtime and quality, which is not reflected when using only the runtime. This issue will be further discussed in Chapter 4. This has resulted in the following research

question:

How can we determine a performance measure that correctly represents the trade off between quality and runtime in vehicle routing heuristics?

This question will be answered through the following sub questions:

- Which performance measures are available?
- What are the theoretical pros and cons of each measure?
- Which performance measure(s) perform(s) best in a real-world scenario?
- What can be said about the quality of the best performance measures?

1.3. Methodology

In order to answer the aforementioned research questions, several steps were taken. This section will outline the used methodology briefly. For a more detailed description, refer to Chapters 3 and 4 for the theory and Chapter 5 for the experiments. Also a summary of the contributions made by this thesis will be given.

The first step that was taken was the creation of a classifier for the test runs. It is hard to judge a test run when comparing the results of different test cases. The proposed solution makes use of a technique used for evaluating exact optimization algorithm called performance profiles [18]. Although this technique was developed for exact methods, it is also useful in the context of this project. In order to detect anomalies in the performance profiles, a clustering algorithm was implemented. The combination of these two techniques is able to mark builds as anomalies based on the results of performance tests.

The performance profile is based on a performance measure. The original paper on performance profiles uses the runtime as a performance measure, since runtime is a useful comparison method for exact methods. However, heuristics function differently and therefore the runtime does not suffice. There is a trade off between runtime and quality, which is not reflected when evaluating based solely on the runtime. This led to the research question: "How can we determine a performance measure that correctly represents the trade off between quality and runtime in vehicle routing heuristics?" In order to answer this question, alternative measures are proposed and tested through experimentation.

The experimentation and context are within the data of ORTEC. The developed approach is built such that it is applicable to any performance test and conclusions drawn can be generalized to other data sets. Since the performance evaluation is based upon the performance tests, there will be a brief analysis of these tests. This analysis will focus on the usefulness and diversity of the test cases by investigating their changes over the last years. The analysis will establish confidence that these test cases are sufficient with regards to this thesis.

1.3.1. Contributions

The contributions that this thesis will make:

- Introduce a classifier to detect regressions in the continuous development process of vehicle routing heuristics, based on performance profiles [18]. Details will be described in Chapter 3.
- Introduce and evaluate three alternative performance measures, which represent the trade off between quality and runtime in vehicle routing heuristics. These measures are used in combination with the classifier from the previous point. Details will be described in Chapter 4.
- Test the aforementioned classifier and measures on real data of a large company. Details will be described in Chapter 5.

1.4. Overview of remaining chapters

The remainder of this report will be structured as follows. The next chapter will be a research on related literature, which discusses what others have done and where the possibilities for additions are. Chapter 3 will discuss how the classifier for detecting anomalous test runs works. Then, Chapter 4 will outline the issues that arose when implementing this classifier and propose the performance measures to be used in

the experiments. The experiments and their results will be described and discussed in Chapter 5. Finally, a conclusion will be given in Chapter 6. Most chapters contain a short summary at the end of the chapter. This summary is meant to recap the most important aspects of each chapter and guide the reader through the report.

2

Literature Survey

Before being able to make a useful contribution to a research field it is important to understand what others have researched. This can also shed some light on which aspects could be interesting to further investigate. From Chapter 1, it has become clear that there are two areas of interest to discover, they will be described in this chapter. First is the area of performance analysis of software systems. This is the main focus of this thesis so discovering what others have researched will be of interest. Secondly, effort will be spent on the detection of anomalies during this project. This is not something new and there has been a lot of research on this topic.

Since before this project I was relatively new to these specific research fields and doing research in general, it is important to have a systematical approach to this literature survey in order to facilitate completeness and reproducibility. The method described by Lavallée et al. [34] has been used to conduct the literature survey in a systematical way. This chapter will first describe the strategy adopted in this literature survey and then describe the research that has been done in the two areas of interest.

2.1. Literature survey strategy

In order to keep the quality of this thesis high, it is important to adopt some guidelines. These guidelines are meant to give all research papers a similar structure, which increases both their readability and usefulness. Therefore, this section will first describe the systematic approach that was used during the literature survey. Next, the do's and don'ts for experimental algorithm analysis are discussed. Finally there will be a word on good lab practice to ensure useful experiments. The next three sections reflect the core tasks during this thesis: writing a literature review, the methodology, which applies experimental analysis and finally the experiments themselves.

2.1.1. Systematic approach to conducting literature survey

According to Kitchenham et al. [31], the two key aspects of a systematic literature review are completeness and repeatability. Completeness refers to having researched all relevant literature in a certain domain. Repeatability implies that independent researchers, following the same procedure, should arrive at the same results. Lavallée et al. [34] add to this that novices to either doing research or a specific research domain often struggle with repeatability. Poor repeatability can result in serious doubts over the scientific value of the literature review. Therefore, they suggest an iterative systematic review (iSR) approach to doing a literature review, which is based on the theory of experiential learning. The following definition of learning best describes experiential learning: "Learning is the process whereby knowledge is created through the transformation of experience." [32]. The iSR approach will be used during the remainder of this literature survey.

Search strategy

In order to increase the scientific value of this project it is important to have a systematic approach to the literature survey. This chapter has been written throughout several months of the research project. Although most of this chapter was written at the start of the project, more information was added when new interesting topics or papers were discovered. The remainder of this section will describe this process.

Doing the literature research was among the first steps taken in this project. Since I was a novice to both the topic and the context, these first steps were of an exploratory nature. At the start of the thesis, there was

no specific goal of the research yet, only a subject: the performance of vehicle routing heuristics. As such, the first queries used to find papers were: "performance analysis", "performance anomaly" and "performance regression". In order to narrow the scope the following words were added to the queries: "heuristics", "optimization" and "vehicle routing problem". This resulted in an initial set of interesting papers. The next set of papers were ones that were found through this initial set. These were found firstly through references and secondly through terms often used in those papers. This led to three new queries to search papers: "code profiling", "anomaly detection" and "performance profiles". The final set of papers was obtained as a result of feedback from the university faculty and other students doing their master thesis, after presenting the midterm results of this thesis.

2.1.2. The do's and don'ts of experimental algorithm analysis

Heuristics are experimental by nature, their success does not have a mathematical proof but rather relies on experience that it performed well. Therefore, analysis on heuristics is done through experimental analysis, where observations are done through a set of experiments. This subsection will discuss a paper on guidelines when doing experimental algorithm analysis by Johnson [29]. He defines three approaches to algorithmic analysis: worst-case analysis, average-case analysis and experimental analysis. Although he notes that experimental analysis is the least exact and therefore least scientifically used, it has its uses. Since heuristics are not exact by nature, experimental analysis is the only option for evaluating them. Experimental analysis is conducted on real-world scenarios and applications and can lead to new issues and questions to be studied. However in this paper the focus will be on how theoretical analysis can improve experimental analysis.

To further illustrate the use case of experimental analysis, he makes a distinction between four types of papers that each represent a different reason to implement an algorithm. These four reasons are:

- To use an algorithm in a particular application. This typically leads to an **application paper** which specifically researches the context and use case of that application.
- To prove that an algorithm is better than others. This type of paper is called a **horse race paper**, where algorithms are benchmarked against other algorithms on specific cases and their superiority is shown.
- To better understand the strengths, weaknesses and operations of algorithmic ideas in practice. This motivates writing an **experimental analysis paper**.
- To analyse the average-case behaviour of algorithms in a specific distribution where probabilistic analysis is too hard. This leads to an **experimental average-case paper**.

Although a lot of points can be made for all of these paper types, the paper by Johnson specifically discusses the experimental analysis paper. It notes that papers often show defects and describe the pitfalls through ten basic principles one should consider when writing a paper. For further details on each of these principles, please refer to the paper itself. The principles are:

1. Perform newsworthy experiments
2. Tie your paper to the literature
3. Use instance testbeds that can support general conclusions
4. Use efficient and effective experimental designs
5. Use reasonably efficient implementations
6. Ensure reproducibility
7. Ensure comparability
8. Report the full story
9. Draw well-justified conclusions and look for explanations
10. Present your data in informative ways

2.1.3. Good lab practice: what to do to ensure reproducible results

Finally, doing experiments is an integral part to any thesis and doing them well can make a big difference. Therefore, a paper on good lab practices will be discussed, written by Kendall et al. [30]. The motivation behind this paper is that optimization research is said to have no standards regarding the reporting of algorithmic results. A comparison is made with Good Laboratory Practice, a set of standards which has been developed for non-clinical research in laboratories. These standards ensure uniformity, consistency, reliability, reproducibility, quality and integrity of the research. A similar set of rules would benefit any research area since these benefits form the basis of scientific research. The standards are presented as a set of 54 recommendations. Most of these are not new, although practice has shown that bad practices still exist. The full set of recommendations can be read in the original paper.

2.2. Performance analysis

The focus of this research will be on analysing performance results and therefore it is important to discover what has already been researched. After reading the literature, it has become clear that there are three types of performance analysis that are of interest to research, with two being closely related. First, there is research in the field of performance analysis of optimization solvers or mathematical software in general. This can be split in exact methods and heuristics. Although research on heuristics would be exactly the problem that is being researched in this thesis, more research has been done on exact methods. Finally, there is the detection of performance regressions of software systems in general. Most research in this area has been done on web applications and enterprise software. Although this is technically a different research area, it can also be reduced to the analysis of software systems through a set of metrics and both are therefore interesting to investigate. The first subsection will introduce empirical evaluation, which is a technique often used for performance evaluation. Later subsections will describe research on performance analysis in the optimization, non-optimization and vehicle routing fields, as well as a section on code profiling.

2.2.1. How to measure performance

This first paper, by Hutter et al. [26], addresses the issue of comparing different algorithm designs. During the development of heuristics, different design approaches can be used. One issue that always arises, no matter the taken approach, is that different algorithm designs have to be compared. As in other domains of interest, the existing theoretical techniques are not powerful enough to answer this question. As such, the comparison is often based on empirical evaluation.

Doing an empirical evaluation means that several algorithm designs are applied to a collection of specific instances and the solution quality and resource costs are compared. Since practical applications are always somewhat time constrained, it is infeasible to run every algorithm design on every instance available. Therefore the question arises: Which algorithm designs do I test and which instances do I use? Finally a decision on resource limitations has to be made. Heuristics can run for a long time and their performance may vary depending on their allowed time. A careful decision has to be made when to stop each algorithm.

This paper resolves these issues through answering eight questions. The first questions were of an exploratory nature. They mainly look at the difference between the algorithm designs and which instances could potentially be useful for ranking the algorithms. They do this by comparing plots of the hardness and CPU times. Next, they discussed the trade offs in ranking the algorithms on performance. The penalize average runtime (PAR) of a set of runs with cutoff time k is defined as the mean runtime, where unsuccessful runs are counted as $a * k$ with a constant $a \geq 1$. Next, the performance is evaluated at a captime of k_{max} , $k_{max}/10$ and $k_{max}/100$. Finally, the described approach is applied to six algorithm designs based on state-of-the-art algorithms.

2.2.2. Performance analysis of optimization algorithms

The comparison of optimization algorithms is a complicated task. It might not always immediately be clear which one is better, since better can be subject to change depending on the situation. Therefore, in order to obtain a fair and unbiased comparison there is a need for a systematic analysis. Recently, Beiranvand et al. [5] have outlined the best practices when benchmarking optimization algorithms. By gathering data from earlier reports and comparing methodologies they have extracted the following steps:

1. *Clarify the reason for benchmarking.* There can be several reasons to benchmark. A different reason can require a different approach. Whenever the reason for benchmarking is unclear some pitfalls can arise. Some examples of reasons to benchmark are: selecting the best algorithm for a real-world application,

showing the value of a novel algorithm or comparing a new version of optimization software with earlier releases.

2. *Select the test set.* There are various problem categories that require different test sets to experiment with. This choice will mainly depend on the reason for benchmarking that was described before.
3. *Perform the experiments.* Designing and performing experiments in such a way that they verify your methodology and are reproducible can be hard. Therefore it is important to outline the thought process as well as the different variables taken into account.
4. *Analyze and report the results.* This is possibly the most important aspect, it is where conclusions are drawn. There are different methods for reporting results that are relevant in different situations. Some examples are tabular methods, trajectory plots and ratio-based plots.

Furthermore it is noted that automated benchmarking is a promising concept and that research in this area is starting to rise. There are however some downsides with the current results so none of the developed tools are widely adopted yet.

One of the main issues of analyzing performance results is to get an overview of a solver in different scenario's. The performance results are often generated by running multiple solvers on multiple test cases. Analyzing these piece by piece is a time consuming job and judging the overall performance is hard, especially when done manually. Both in the field of performance analysis of optimization algorithmics and in performance regression detection techniques are developed to solve this issue. A developed technique in performance regression detection will be described later in Section 2.2.4.

Performance profiles

The golden standard of creating such an overview in the field of optimization algorithmics is called a performance profile, which was developed by Dolan and Moré [18]. They claim that a plot of the performance profile reveals all of the major performance characteristics. A performance profile is the cumulative distribution function such that the performance ratio of a solver is within a factor τ of the best known possible ratio. Performance profiles are introduced using the runtime as a performance measure and this has been widely adopted as the standard. It is however possible to use other performance measures, like Moré and Wild [37] have done in a later research. They used the amount of function evaluations as performance measure, which is of particular interest to fields that use function evaluations with heavy costs, like derivative-free optimization algorithms. The performance metric $t_{p,s}$ is defined as the runtime (or some other metric) obtained by solver s solving problem p . The performance ratio of a solver is then defined by dividing the performance metric of a specific solver by the best known:

$$r_{p,s} = \frac{t_{p,s}}{\min\{t_{p,s} : s \in S\}} \quad (2.1)$$

To get an overall assessment of the performance of a solver, the performance profile is introduced. A performance profile is defined as the percentage of performance ratio's that is within a factor τ of the best known solution for each test case. It is a cumulative distribution function of its performance ratio over all test cases, meaning that it is a value between 0 and 1 that is increasing the less strict the factor τ is. In the next equation n_p denotes the total number of test cases and P is the problem space that contains all test cases. The performance profile of solver s is defined as a function of τ :

$$\rho_s(\tau) = \frac{1}{n_p} \text{size}\{p \in P : r_{p,s} \leq \tau\} \quad (2.2)$$

This definition of performance profiles may sound a bit complicated without numbers, therefore an example is provided. Let's assume that after conducting a performance test the data as shown in Figure 2.1, on the left, is obtained. For each version of the software, there are 3 test cases that took a certain amount of seconds to complete. Version 1 of the software has taken 20 seconds to complete Case 1. Since the performance of the code is being optimized, the lower the number the better. From observing the results, Version 3 seems to be the best in 2 test cases while Version 2 is the best in the last case. There is not one version that clearly stands out as the best one. To create performance profiles from these test results, the first step is to calculate the performance ratio's of each of the results. This is done with respect to the best known, using Equation 2.1. These ratio's are displayed in the Figure 2.1 on the right.

	Case 1	Case 2	Case 3
Version 1	20	190	1000
Version 2	15	230	700
Version 3	10	180	950

	Case 1	Case 2	Case 3
Version 1	2,00	1,06	1,43
Version 2	1,50	1,28	1,00
Version 3	1,00	1,00	1,36

Figure 2.1: Data to exemplify the performance profiles. The tables contain the results of test cases Case 1, Case 2, Case 3 against Version 1, Version 2 and Version 3 of the software. The left table shows the actual runtimes in seconds. The right table has converted these to performance ratio's with respect to the best ones known.

To obtain the performance profiles, the percentage of runs above factor τ are counted, with τ increasing in steps of 20%. These factors can be chosen based on the ratio's, but in this case increments of 20% were chosen, since that keeps the example readable and clear. In an actual application these steps could be 1% or even less, depending on the minimum and maximum values and the required precision. Since a performance profile counts the fraction of tests that are at most x% worst than the best known, the faster its value goes to 1.00, the better. Doing this on the same data gives us the performance profiles that can be observed in Figure 2.2. By observing them, it can be observed that Version 1 is clearly slacking compared to the others, since its consistently the lowest line. Version 3 is clearly the best one since it is the best one in 2/3 cases and the last one is also quite good. In short, the performance profiles are capable of showing which solver is optimal in a situation where the raw data was insufficient to do so.

An example of where performance profiles are used in practice can be seen in PAVER [13], an environment for analyzing and verifying benchmarking data of mathematical programming problems. Their system generates several results, including performance profiles. Performance profiles have also been successfully applied to benchmark structural topology optimizers [40], in order to determine which type of solvers are efficient and reliable. Although performance profiles are widely adopted, recently there has been a study that shows the possibility of misinterpretation when benchmarking with performance profiles [23]. They warn for issues when comparing the 2nd and 3rd and so on best algorithms to each other, using an incomplete data set. Finally, Weise et al. [42] argue that end-of-run results alone do not give sufficient information to evaluate performance and therefore include an analysis of the algorithm's progress over time [42]. The measures that they use include cumulative distribution functions and the area under the curve.

2.2.3. Performance analysis in vehicle routing problems

Next, the existing literature on performance analysis in vehicle routing problems is discussed. Most types of analysis that were found were not that different from the ones earlier discussed. Most of the research in this area is focused on developing new methods to improve heuristics. These methods are then benchmarked on some data set and provide some basic analysis through a set of metrics.

	0%	20%	40%	60%	80%	100%
Version 1	0,00	0,33	0,33	0,67	0,67	1,00
Version 2	0,33	0,33	0,67	1,00	1,00	1,00
Version 3	0,67	0,67	1,00	1,00	1,00	1,00

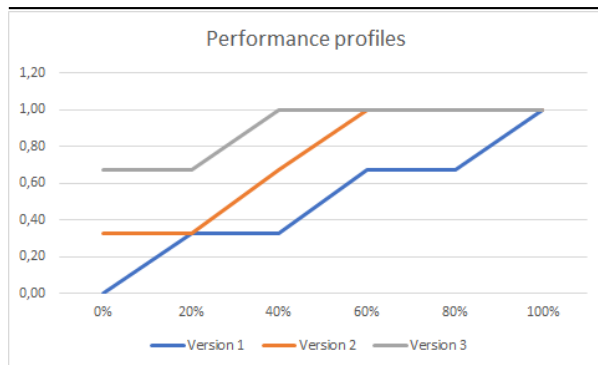


Figure 2.2: The data of Table 2.1 turned into performance profiles. These are distribution functions that count the amount of cases which have a performance ratio above a certain threshold (percentage). The image on the bottom shows a plot of the table on the top.

Before discussing specifics of performance analysis in vehicle routing problems, there are two things that should be noted. First off, most research that was found is based on scientific, generated data. Although the usefulness of this data lies in its artificial completeness (it covers a lot of hard cases), it is debatable whether this data is realistic. Secondly, most research focuses on solutions to vehicle routing problems rather than evaluating its performance. Since they often use benchmark data, they use the same measures that others have, for instance Carlsson et al. [15]. These are often simple measures such as total distance travelled, rather than something more complex that represents the trade off between runtime and quality. The reason for this is that the runtime is often limited to a set amount, which makes the quality the only interesting attribute.

One option for comparing performance of vehicle routing problems across multiple optimization objectives is to use a hierarchical objective function. This is done by Berger et al. [8], in their research where they propose a hybrid genetic algorithm to solve vehicle routing with time windows. They omit the runtime in the evaluation of performance since they have put a time limit on each of the solvers. The hierarchical objective function looks primarily at the number of vehicles used and secondly at either the total distance or the total duration. This means that only when the number of vehicles is the same, it looks at the difference in the secondary objective.

Some researches have a less strict performance metric regarding runtime. Berger and Barkaoui [7] also work on a hybrid genetic algorithm for vehicle routing with time windows. They mostly focus on solution quality over runtime. Rather than concluding a best approach, they eliminate all approaches with a average deviation from best over 1%. They argue that the results of their experiments do not show any conclusive evidence to support a dominating heuristic over the others, even though the runtime differences are as large as average time (in minutes) ranging from 3,84 to 222,85.

Since the quality runtime trade off is present in most heuristics, not only vehicle routing problems, this final paper is one focused on heuristics in general, rather than vehicle routing heuristics specifically. The paper by Berthold [9] discusses the need of a new measure for evaluating the performance of primal heuristics. An important aspect here is finding and improving a feasible solution early in the process. They argue that classical performance measures, such as runtime, reflect the performance badly. Therefore the primal integral is introduced, which is tested on five state-of-the-art MIP solvers. The primal integral is based on the primal gap, which is defined as the gap between the current solution and the optimal solution. When this is taken as a function over time, the integral can be calculated from this function, resulting in the primal integral.

2.2.4. Performance analysis outside optimization algorithms

This section discusses the research that has been conducted in performance analysis in computer science fields other than optimization algorithms. Although these studies were not conducted in the field of optimization algorithms, they still compare the performance of different pieces of software. The techniques that are used in these studies have some subtle differences but may still be applicable to the problem that is researched during this project. Therefore investigating what has been studied in this area is deemed interesting.

Most research on performance analysis has been done on detecting performance regressions or anomalies. There are several techniques that can be applied to a wide variety of measures. Therefore the goal of this section is to give examples of what has been done. Although each study uses different techniques, their general approach is often the same. In contrast to many other software bugs, which trigger direct failures or unexpected function returns, performance anomalies often manifest in the form of increased resource usage. The performance tests or load tests require the simulation of a lot of users sending requests to the system. These simulations require a hefty amount of resources and are therefore often not executed as often as they should be. Ideally these tests would be ran on every new version of software, even during early stages of development. In practice this is almost never done and issues are usually detected in the production environment, where their impact is much larger, according to Langner and Andrzejak [33]. After gathering enough performance data, this data is used to create an expectation of the behaviour of the performance of a system. This is often done using statistical or machine learning approaches. The expected behaviour that is created is often called a model, profile or signature. Whenever a new version of the system is benchmarked, it is compared to the expected behaviour to detect regressions. Most of these studies are aimed at either web based applications or enterprise software.

General approach to performance analysis

Shang et al. [41] formulate the problem with performance analysis in the following way. During a performance test thousands of counters are collected. Some examples of these counters are CPU usage, memory usage and response time. The process of comparing thousands of counters across multiple versions of soft-

ware is not only time consuming but also error-prone. This problem only becomes bigger when multiple scenarios generate even more results. Therefore experts often build a model from only a handful of these counters. This selection is based on experience or gut feeling and therefore often not objective relative to the data that was gathered. Furthermore, new forms of regressions, that show different behaviour than in the past, often go unnoticed because of this. In their research, Shang et al. [41] first group the performance counters into clusters. From each of the clusters they then select the most representative measure to represent the performance of the system. They then use statistical tests to build a regression model of the expected performance behaviour. This model is then applied to a new version of the system to detect performance regressions. One of the main advantages of this approach is that analysis of a system does not require an in-depth experience since the algorithm determines which measures to use based on the data.

Use cases of performance analysis

It is also possible to determine a lower and an upper threshold for a series of measure and calculate the violation rate of a piece of software, as done by Nguyen et al. [39]. These so called control charts originate from manufacturing processes, where they are widely used. If the violation rate exceeds a predetermined rate, a warning is triggered. Another commonly used technique is to train a classifier which can classify a piece of software as either having expected or unexpected behaviour. Li et al. [36] have conducted a benchmarking study in software defect prediction using 17 classifiers on 27 datasets. They observe that using a complex model rather than a simpler approach does not necessarily improve the results. Furthermore, even when using different measures, they cannot find a single best classifier but are able to determine a set of models that achieve good result. Other techniques that are used to detect performance regressions include Application Performance Management tools [3], transaction profiles [21], resource profiles [12] and performance signatures [17].

Finally it is noted that practical applications of performance regression detection techniques are often hindered by the assumption that all testing environments are homogeneous. This assumption is often false in practice. Foo et al. [20] propose a solution that consists of ensemble models to compose individual models of the expected behaviour. It makes use of association rule derivation, after which the distance with a new test run's data is computed to detect regressions.

2.2.5. Code profiling

Finally, code profiling is a software analysis technique that reports the resources being spent by different parts of the software. This form of analysis is dynamic rather than static, meaning that it requires the code to run in order to gather information. The goal of this analysis is often to give developers insight as to where to focus their efforts in optimizing their software, as a means to improve software quality. A tool that conducts the profiling of source code is called a profiler. After the profiler has reported where the resources are being used, developers can use this information to detect hot spots and performance bottlenecks. The resources that are spent are, according to Bergel et al. [6], often presented as numerical measurements such as the number of method invocations, the number of object creations or the time spent in a method or piece of code. Reporting such information about resources being spent causes some overhead. There are two methods of profiling: sampled profiling and exact profiling. Sampled profiling gathers data every x seconds whereas exact profiling gathers all possible data from every call that is being made. The exact method provides the most complete data but also causes the most overhead. This is a trade off that has to be made and the best method is different depending on the use case and context. In his research, Walter [10] reduces the overhead of the profiler by exploiting instruction counting.

One area where profiling is often used is for big data applications. These applications are very demanding in terms of resources and therefore require an in-depth analysis of where these resources are being spent. In their paper, Enes et al. [19], have developed BDWatchdog, a solution for code profiling of big data applications. A major reason for this platform is the fact that more and more modern technologies no longer run on isolated instances but rather on software containers, for instance Docker [27]. Because of this, their analysis monitors the resources of processes as opposed to machines, which is often used by others.

Since the goal of a profiler is to provide insights, it is important to visualize all the gathered data in a correct way to the developers, as understated by Byma and Larus [14], while building Memoro for heap profiling. The tools provided by a profiler collect large amounts of raw data which often contains patterns that are hard to observe. Memoro solves this by calculating scores for specific patterns in order to help this process. In combination with other visualization techniques, these scores provide useful insights in the runtime behaviour of the code. Finally, the visualizer is divided in two views. The 'global view' provides the developer

with an overview of the behaviour of the program as a whole. The 'detailed view' provides a more detailed specification of a specific point in the program.

2.3. Anomaly detection

This section will focus on the research that has been done in anomaly detection. Anomaly detection can be defined as an analysis where one or more data points are discovered as being different with respect to the entire data set. The detection of anomalies becomes increasingly difficult as the complexity of a data set increases. An often used synonym for an anomaly is an outlier. Anomalies in data often reveal a significant amount of information and are used in a variety of different fields. This section will first describe the different techniques to detect anomalies. Then it will differentiate between the types of anomalies and some research fields in which anomaly detection is applicable are discussed. Finally an algorithm for anomaly detection is discussed.

Even though a lot research has been done in this field, with many different techniques considered, the approach that is used is often a set of steps that are the same. In short, from a monitored environment, a selection of parameters is made which are then used to create a data set. This data set is then used to create a model that represents the expected behaviour of the system, based on previously observed behaviour. The creating of this model often includes a training stage, where the model is iteratively modified. Any data that is not conform to this model is then marked as unexpected or anomalous data [1].

2.3.1. Anomaly detection techniques

Within anomaly detection, there are different types of techniques that are applicable to different types of data in different scenario's. Which type of technique is useful in which situation will now be discussed. The main types of techniques that recur in most researches, according to Ahmed et al. [2] and Agrawal and Agrawal [1], are classification and clustering. Classification is a type of algorithm that classifies a data set in one of x classes. In the case of anomaly detection there would be two classes: normal behaviour and anomalous behaviour. A classification algorithm is trained using data for which the class is known before hand. The training of a classifier requires a data set that contains information about whether it represents normal or anomalous behaviour. This type of data, where the class is known beforehand, is called labelled data. After training on enough data, the assumption is that the classifier is able to classify unknown data into one of its classes. Clustering on the other hand works with unlabelled data. For this type of algorithm, there is data available, but it is unknown whether the data exerts normal or anomalous behaviour. This method relies on the assumption that the majority of the data is normal behaviour. The data is clustered into groups based on a similarity function. After clustering, the biggest group of data is assumed to be normal behaviour and smaller groups or singular points are marked as anomalous. This distinction is also called supervised learning, which uses labelled data. The opposite is unsupervised learning, which uses unlabelled data. Generally, supervised learning is more reliable in terms of results, but it requires labelled data. This labelled data is often not available or costly to acquire, which is often why unsupervised learning is used. Goldstein and Seiichi [22] have researched unsupervised detection algorithms and add to this that clustering techniques have a lower computation time compared to classification approaches.

Classification and clustering are both approaches that originate from computer science research. There are different approaches that are often used, originating from different research fields. Two other often used techniques are noted by, amongst others, Ahmed et al. [2] and Hubert et al. [25]. The first describe statistical techniques, which originate from mathematics. Statistical techniques build a stochastic model that represents the expected behaviour. Data that is not generated by the assumed model is marked as anomalous. Specific statistical techniques include Gaussian model-, regression model- and histogram based techniques. The other type of technique originates from information theory. These techniques analyse the content of a data set using different measures like entropy and information gain or cost. The key assumption is that anomalies in data induce irregularities in the information content of the data set.

2.3.2. Types of anomalies

The definition of an anomaly is broad. Unexpected behaviour can be significantly different based on the context of the data. Therefore it is important to know what types of anomalies can occur. The research of Chandola et al. [16] distinguishes 3 types of anomalies, that require different techniques of detection. A similar distinction is made by Hubert et al. [25], but they differentiate between a short or long interval of anomalous data. The 3 types detected by Chandola are:

- *Point anomalies.* Single points of data that exert anomalous behaviour with respect to the other data. These anomalies are the simplest and most obvious to spot. An example is the number 105 in the following set of numbers {1, 2, 1, 3, **105**, 2, 4}.
- *Contextual anomalies.* These anomalies are only considered anomalous in their specific context. In another context their behaviour might not be marked as anomalous. An example is the second 1 in the following sequence of numbers {1, 2, 3, 4, **1**, 5, 6, 7}. All numbers are increasing, with exception of the second 1. Note that the number 1 would not be anomalous if the ordering, i.e. the context, was different.
- *Collective anomalies.* A collection of data points that may not be anomalous by themselves, but as a group exert anomalous behaviour with respect to the other data. An example is the sequence of 20's in the sequence {10, 20, 10, 20, **20, 20**, 20, 10, 20, 10, 20 } when the expected behaviour is a 10 followed by a 20 followed by a 10, etc.

2.3.3. Applicable research areas

Since the definition of an anomaly is so broad, anomaly detection is useful in several different areas of research. This section describes some of the uses of anomaly detection. The following examples should give an indication of possible uses of anomaly detection algorithms:

- *Intrusion detection* is concerned with computers or networks of computers that are intruded by people with malicious intents. This intrusion is often done by sending malicious packets to the computers. Once a computer is intruded, intruders have access to (sensitive) data and the sooner an intrusion is detected, the sooner actions can be taken. Anomaly detection is used to detect these anomalous packages and neutralize the dangers they bring.
- *Fraud detection* is concerned with different methods of fraud, like credit card fraud, insurance claim fraud or insider trading. The behaviour of frauds is often anomalous with respect to historical data or data from others. An example could be a number of large purchases on a credit card, which could indicate that it was stolen.
- *Medical and public health* is concerned with working with patient records or instrumentation- or recording errors. By comparing the records among different patients, anomalies could be detected. Detecting disease outbreaks in an area has also been researched.
- *Industrial damage detection* is concerned with the detection of faulty mechanical units or structural defects. Due to the continuous usage of these units, they suffer damage. These damaged materials require early detection to prevent further escalation and losses.
- *Image processing* is concerned with the detection of abnormal regions within an image, but also motion detection in a series of images. Anomaly detection techniques can help reduce the amount of data required to represent an image or video.

2.3.4. Anomaly detection using clustering

One method to do anomaly detection is to use clustering. Clustering is the task of grouping data into clusters that have similar behaviour. Clustering is an unsupervised technique that has the advantage of being understandable for humans due to the ease of visualizing clusters. The specific clustering algorithm that was used, k-means clustering, has been around for half a century and is often used for its simplicity and computational speed. It has to be noted though that, as stated by Chandola et al. [16], anomaly detection comes as a by-product of clustering and it is therefore not optimized for this goal. Its primary goal is to cluster data based on similarities. There are two possible assumptions underlying anomaly detection using clustering. These assumptions are: "Normal data instances lie close to their closest cluster centroid, while anomalies are far away from their closest cluster centroid." and "Normal data instances belong to large and dense clusters while anomalies either belong to small or sparse clusters.", both cited from Chandola et al. [16]. Even though clustering is not designed for anomaly detection, these points argue why it is a good method to detect anomalies.

The k-means clustering algorithm is quite simple to understand, yet effective in its goal. It consists of several steps that continue until convergence or a maximum number of iterations. A visualized example

can be seen in Figure 3.3, which contains a red and a yellow cluster. The initial division on the top left is not that great, but after 3 more iterations of the k-means clustering algorithm the bottom left shows a large improvement. The steps taken in this algorithm are:

1. Define / determine the number of clusters K
2. Initialize K centers. There are different ways to do that that might affect the outcome of the algorithm. The initial centers are often chosen (semi-)randomly but there are multiple options available.
3. For all points in the data set: calculate their distance to the center, based on some distance metric, and assign to the closest center.
4. Recalculate the center based on the current clusters by taking the one with the smallest total distance to the other members of the cluster.
5. Go back to step 3 until the centers converge or the maximum number of iterations has been reached.

2.4. Summary

This chapter has discussed the existing literature related to this thesis project. The first section focused on guidelines to increase the quality of this thesis. Next was a section on performance analysis. Although a lot of research has been done on performance analysis, not so much was done on performance of heuristics specifically. The performance profile mentioned in this section has inspired the classifier that is developed for this thesis. Finally there was a section on anomaly detection techniques, types and areas where it's used. This will be useful for finding anomalies in the performance profiles as part of the classifier.

Next, Chapter 3 will explain the methodology behind the classifier that is built, which was inspired by the performance profile, described in Section 2.2. This performance profile needs a performance measure to compare tests against each other. The existing measures are insufficient for evaluating heuristics, so Chapter 4 will explain the alternative measures that could be useful.

3

Classifier Design

One of the issues that was described in Section 1.2 was that it is hard to make a judgement, given a set of measures over a set of test cases for multiple test runs. This chapter will discuss the approach that was taken to solve this issue. The first section will describe and discuss the available data. The next section will focus on how this data is used to create an overview of a test run. Thirdly, the clustering algorithm, used to find anomalies in this overview is explained. The final section will discuss how this approach is used on the ORTEC data. The approach described in this chapter will be used during the experiments, as will be described later in Chapter 5.

3.1. Available data

In order judge the performance of different test runs automatically, there is a need for data. This section will present the available data and its usefulness in evaluating the performance. The data available is generated by running performance tests on a regular basis, currently weekly. The available data spans almost three years, from 2017 until 2019. Given the current development version, a predefined set of test cases is ran on this build and the results are stored for comparison. The next subsections will discuss the measurements obtained from running the tests.

3.1.1. Key Performance Indicators

The Key Performance Indicators or KPI's measure the quality of the generated solution. The KPI's themselves are values that are being optimized, be it maximized or minimized, during the execution of the algorithmics. These are essentially the most important results that customers are interested in, as they represent the quality of the generated solution. The better optimized these values are, the more efficient the solution is. During the development cycle these values should not change much. If they do change this is often the result of a bug or large development project. Changes in these values are often an indicator of bugs, even when they show improvements. One of the difficulties of detecting regressions is determining whether changes in the KPI's are the result of an intentional change or of a performance bug. The following values are being optimized in each test case:

- **Distance travelled** - The objective of this measure is straight forward. A decrease in distance travelled results in less total time spend, which leaves more time for additional tasks. It also leads to less gasoline used, which is good for the environment but also reduces the costs for executing this route planning. The distance is a minimization objective.
- **Costs** - This measure is related to the distance travelled, but not the same. The cost function is a user defined function which often has the distance as (one of) its input variables. The costs represent the relative value of executing this route planning. The costs are a minimization objective.
- **Amount of tasks planned** - The amount of tasks planned is a variable rather than a predefined amount. This is because most of the time there are more tasks that can be planned than tasks that can be executed. The remainder of the tasks, the ones that are not executed, have to be planned at a later moment. The amount of tasks planned is a maximization objective.

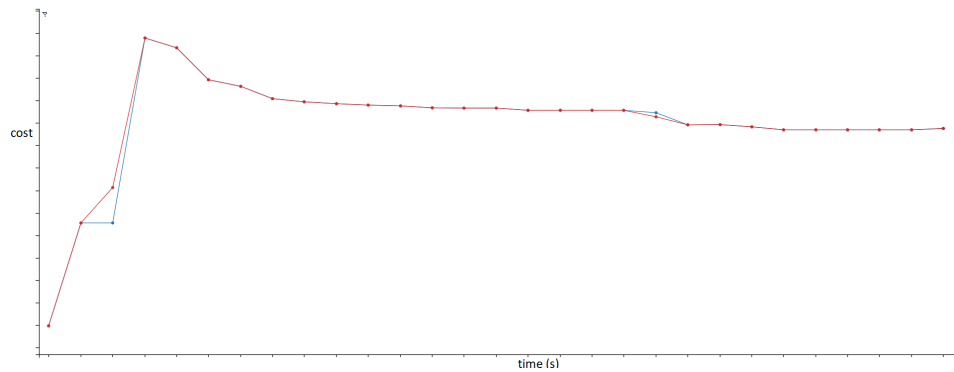


Figure 3.1: An example of the cost function over time. This image shows two different builds with minor performance differences. The chart can be split in two phases: The first phase is where the cost increases due to a solution being created. The second phase is the optimization phase which shows a decrease in the costs.

- **Amount of trucks used** - Although the amount of available trucks is a constant, the amount of trucks that is actually used is variable. Less vehicles can lead to a decrease in the amount of costs, but a single vehicle cannot efficiently deliver all packages.

These variables often interact with each other and a combination of them is required to come to a good solution. There is often a trade off that has to be made between increasing one KPI while reducing another, for instance: the distance travelled can be reduced, but this results in additional trucks used. However, if a trip is already passing a specific destination on its route, adding that task which destination would already be visited anyway results in (almost) no increase in distance and costs.

As will be explained later in Chapter 4, the final results are not always a proper representation of the performance of a test run. Therefore it could be interesting to look at how the KPI's change during the execution of the algorithms, in addition to the end results. Figure 3.1 shows a real example of the cost function over time for a single test case, for two different test runs. The vertical axis shows the current costs and the horizontal axis shows the time. The image shows two test runs that have minor differences. The chart can be split in two phases: The first phase is where the cost increases due to a solution being created. The second phase is the optimization phase which shows a decrease in the costs. As illustration, to evaluate the performance in the old procedure, a human expert has to analyze 4 of these graphs per test case (for each of the KPI's), for 10+ test cases, for multiple test runs.

3.1.2. Runtime measures

In contrast to the KPI's, which are meant to measure the quality of a solution, the runtime measures reflect the amount of resources or computational requirements required to create this solution. The first and main measure available in this category is the amount of time which it takes to come to the given solution. Something that has to be considered is that the runtime can vary, based on the system that it is run on. Therefore, comparing the results obtained from multiple machines is a daunting challenge that has been the subject of several researches. In the data that is used in this thesis it is however assumed that all the data is obtained from running the performance tests on the same hardware. This is the case for the ORTEC data, which is generated from a single machine that is dedicated to only running the performance tests to maximize the fairness of the runtime results.

Since the solver makes use of heuristics, there is a trade off between runtime and solution quality. One of the two does not tell the whole story, therefore a combination of these measures has to be observed. What if the runtime has decreased with 5% but the solution quality has increased with 5%? Is that a good or a bad thing?

In order to analyze the reasons for a performance anomaly, there is additional information available regarding the runtime. The code base contains a set of marks, which report the time at which they were encountered. These marks can be combined to measure how much time has been spent in specific parts of the system. These results are logged with the user-defined name of the part, together with the amount of time that was spent in it. Since each of these parts is called multiple times, a summary can be generated by summing the times of each part with the same name. This summary could also include the amount of times that this part was ran, which could possibly explain the increase in time spent in a part. Although this data is

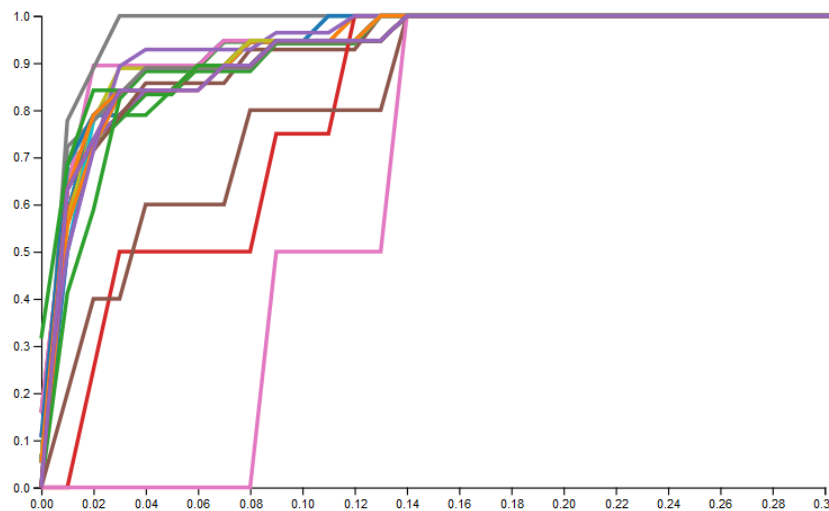


Figure 3.2: The performance profiles on the real data of ORTEC for the first months of 2019, using area under the chart. See Section 4.2 for more on this measure. The horizontal axis shows the factor of performance ratio's and the vertical axis shows the percentage of test cases within this factor. Each line represents a different test run. It can be observed that many test runs have 70% of their tests cases be at most 1% worse than the best known solution. The image shows 3 clear anomalous test runs.

not used in the end result of this thesis, its uses have been investigated and it could be useful when trying to improve the current system.

3.2. Performance profiles to create an overview

Looking at the results of all test cases and getting an overview can be overwhelming. Not only that, but the interpretation is speculative and can lead to disagreement. In order to create an overview of all test cases, the performance profiles that were explained in Section 2.2.2 will be used. This section will first describe the motivation for using the performance profiles. Then it will outline how the performance profiles are used. Finally, the issues are discussed.

3.2.1. Motivation for using performance profiles

During the research no other significant methods for creating an overview have been found. Some researchers have used the average or total cumulative runtimes to create an overview of the overall performance. There are some issues with these techniques which is why the performance profiles were chosen. The first reason is that with the average or cumulative total, large cases tend to dominate the performance metrics in favour of smaller instances. Imagine two test cases with a runtime of 10 seconds and of 1000 seconds. The average would be 505 and the cumulative total would be 1010. There could be a small change in the large case which could lead it to 995 seconds. The average and cumulative would then be 503 and 1005 respectively. There is however no difference with the case that the small case would have changed from 10 to 5, which would be a 50% change. As a result, cases with large numbers tend to dominate the performance metrics. The performance profile counters this by calculating the relative change of a metric using its performance ratio. Another issue with the average or cumulative is that, since it is a single number, it is hard to discriminate between changes in a single case. One case could increase with the same amount that another decreases. The average or cumulative values would not change at all, although a significant difference could indicate an issue. The performance profiles make use of a distribution function, which reflects this property better.

3.2.2. Implementation of performance profiles

In order to create the performance profiles a performance measure has to be chosen. Initially the runtime was chosen, since this was used by others. During the project it became clear that the runtime was not an appropriate measure to evaluate the performance of vehicle routing heuristics. Chapter 4 outlines why the runtime is not an appropriate measure and which measures could be useful in evaluating the performance of these heuristics. However, the initial version was built using the runtime and since it does not affect the explanation, this section uses the version with the runtime as performance measure.

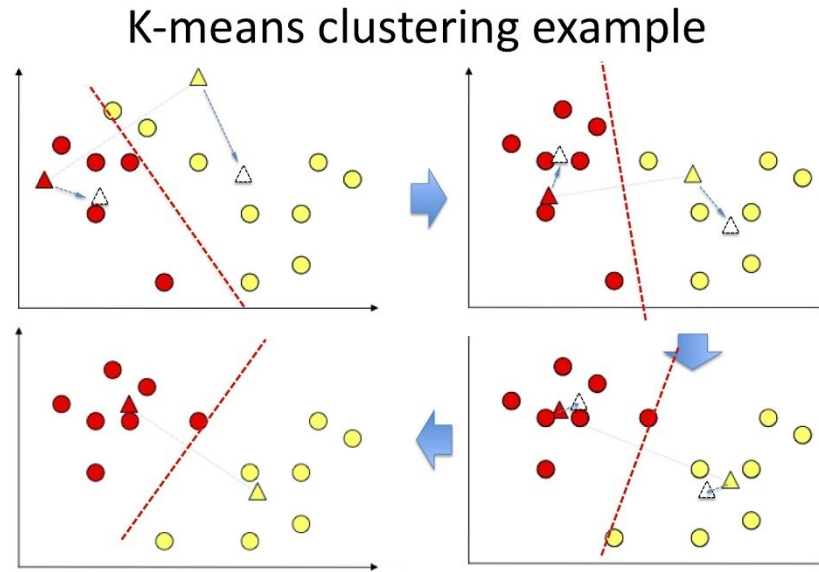


Figure 3.3: A visual example of k-means clustering, which contains a red and a yellow cluster. The initial division on the top left is not that great, but after 3 more iterations of the k-means clustering algorithm the bottom left shows a large improvement. Source: [35]

At first, the performance profiles had to be created. The first step is determining per test case what the best known solution is. Then, using these solutions, the raw runtime data are converted to performance ratio's, with respect to the best known solution. Finally, these performance ratio's are used to create the performance profiles. For a more detailed explanation on the performance profiles, see Section 2.2.2. During the implementation, the data for the first months of 2019 was used, since using all data made the computation time too long to test with. The results for the 2019 data can be seen in Figure 3.2. The horizontal axis shows the factor of performance ratio's and the vertical axis shows the percentage of test cases within this factor.

3.3. Anomaly detection using clustering

The problem of anomaly detection is one occurring in many different fields of research. Given a set of data points, find the ones that do not behave as expected. Section 2.3 has explained this problem and the existing techniques within the literature. This section will describe the requirements for this specific use case and how it was implemented.

The first thing to note is that there is no labelled data available. Although data of performance test of previous years are available, there is no knowing which of these are anomalous or not and labelling the data is both costly and time expensive. Therefore, the technique should be unsupervised of nature. Secondly, one of the goals of this thesis is to explain the reason for an anomaly. Therefore it would be preferred if the technique that is chosen reflects this. It is a pro if there is some form of visualization or other form of understanding that humans can have to the technique.

3.3.1. K-means implementation

The technique that was chosen to detect anomalies is k-means clustering, which was described in Section 2.3. Although the implementation of this algorithm is straight forward, there are two design choices that have to be made. The first choice that has to be made is how to create the initial centers. The literature discusses different techniques, varying from choosing k at random to more advanced techniques. The one chosen is described by Arthur and Vassilvitskii [4] in their paper on k-means++. Their technique chooses the first center at random and then proceeds to choose the remaining centers with a probability that is based on the distance to the closest center. The larger this distance, the larger the chance this point will be a center. This technique was chosen since it is widely used for its simplicity and speed, which are appealing in practice. Although this technique was chosen and implemented, it is not actually used in the current implementation. Since the amount of clusters is set to 1, since there was 1 clear baseline for each performance profile, the initial cluster



Figure 3.4: An overview of the performance tests that were ran during the first months of 2019. The performance profiles and clustering techniques were used to mark 2 test runs as anomalous. The next step is to further investigate these test runs regarding information on the specific test cases that cause this anomaly.

center is simply chosen at random.

The second and last choice was determining a distance function. Again the choice has been made for a widely used technique, the Euclidian distance. For two vectors (or lines) this is defined as sum of the root of the squared difference between two points, or, with q_i and p_i values at point i and n the length of the lines:

$$\sqrt{\sum_{i=1}^n (q_i - p_i)^2}, \quad (3.1)$$

3.4. Implementation

Recall that the goal of this research project is to investigate the performance to find anomalies and investigate the reasons for these anomalies. This section will give a description of the approach that was used to reach this goal. Although the final tool looks different to the images in this section, it functions the same and therefore the images used in this section will suffice.

Before explaining the approach it is important to know what the data looks like. The easiest way to understand it is to envision a matrix that contains test runs as rows and test cases as columns. Each entry of the matrix is made up of the results of a single run that belongs to one test case run on one changelist. It should be noted that the matrix is not completely filled, meaning that not each test case has been run during every test run. The approach that was taken analyses the data in different steps, where each step goes into more details to uncover the reason for the anomaly.

3.4.1. Step 1: Analysing test runs

The first step is to compare test runs against each other. One way to do this would be to compare the results of each test case of each test run against the other test cases. However, this poses some problems which is why a different approach is taken. First of all, not every test case is ran on every test run and therefore it is not possible to compare each of the results since they are not complete. Secondly, this approach is error-prone and takes a lot of time.

A solution was found in the literature. The golden standard for benchmarking optimization software are performance profiles, by Dolan and Moré [18]. These have been explained before in Section 3.2. The claim that the paper by Dolan and Moré makes is that a plot of the performance profiles reveals all of the major

Product version:	transport:development
Date:	05 February 2019
Changelist:	# 1161254
Total runs:	23
Detected anomalies:	3
Previous changelist:	# 1157870

# 20190206012450757	CVRS_distribution_1	+	Details
# 20190205231940355	CVRS_distribution_2	+	Details
# 20190205225913048	CVRS_distribution_3	+	Details
# 20190205232158437	CVRS_distribution_4	+	Details
# 20190206032605896	HomeDeliveryEnglishSupermarket_Case1400	+	Details
No processing time information found			
# 20190205212727768	LongTripsWithLegislation_Case1000	+	Details
No processing time information found			
# 20190205193336622	LongTripsWithLegislation_Case2000	+	Details
No processing time information found			
# 20190206008083721	LongTripsWithLegislation_Case500	+	Details
No processing time information found			
# 20190206024214497	LotOfPackagesInTrip_Case1000	+	Details
No processing time information found			
# 20190205221541448	LotOfPackagesInTrip_Case2000	+	Details
No processing time information found			
# 20190206044213064	LotOfPackagesInTrip_Case500	+	Details
No processing time information found			
# 20190206024145498	RefillSupermarket_Case1000	+	Details
No processing time information found			
# 20190206051136039	RefillSupermarket_Case1000_2	+	Details
No processing time information found			
# 20190205200845100	RefillSupermarket_Case2000	+	Details
No processing time information found			

Figure 3.5: The first step of the anomaly explanation procedure. The initial step has marked this build as anomalous. This image shows an overview of all the test cases that were ran on this build, along with some of its results. The clustering algorithm was used to determine which test cases showed anomalous runtime behaviour.

performance characteristics.

As such, the first step taken to analyse changelists is to create a performance profile for each changelist and compare them to each other. After one or multiple test runs have been marked as anomalous they can be further investigated. This comparison will be done using k-means clustering as an anomaly detection technique. The results can be seen in Figure 3.4.

3.4.2. Step 2: Analysing test cases

The first step has marked a test run as anomalous or regular. This step used the performance profiles to judge the overall performance of each test run. Now that it is known whether the build is anomalous or not, it would be interesting to find out why it is anomalous. One way of doing this would be to compare the individual test cases of a run to other runs. The same clustering algorithm is used to detect anomalies based on the individual running times of each test case. It is possible that even though a test run is marked as anomalous, none of the test cases are anomalous by itself, which is a great benefit. In this case, each case could be a little worse, which is still an anomaly. The results can be seen in Figure 3.5.

3.4.3. Step 3: Analysing the results

To understand the last step of the approach, recall that the intent of the approach is to inform developers of anomalies and to point them in the right directions to solving this anomaly. Step 2 has left us with an anomalous test case for an anomalous test run. The last step is to convey information to the developer which can be used to solve the anomaly.

The way to do this is to visualize the results of a test case and compare them to other non-anomalous test cases in such a way that the developer can deduce which part of the software needs fixing. The results can be seen in Figure 3.6. This figure shows all the information available on a single test case of a single test run. On the top is the run information, which includes the test case and date it was run. The KPI end values show the final results of the optimizer. Then, the algorithm processing times show how much time the optimizer has spent using each of the heuristics that was used during optimization. Finally, these algorithm processing times are compared to those of other test runs, which can be used to spot differences.



Figure 3.6: The final step of the anomaly detection procedure, where the results of a specific test case on a specific build are shown. The bar charts show the runtimes of other builds on specific parts of the algorithmics. The bar chart on the bottom shows a single larger bar, which could indicate an issue in that algorithm.

3.5. Summary

Chapter 3 has explained the basic structure of the classifier. It is inspired by the performance profile, which originates from evaluating exact optimization methods. This is a cumulative distribution function such that the performance ratio of a solver is within a factor τ of the best known possible ratio. Through a clustering algorithm, the anomalous builds are then detected from these performance profiles. The initial implementation used the runtime as a performance measure, but this does not suffice for heuristics, since it does not account for the solution quality. Therefore, Chapter 4 will explain which alternatives will be tested through the experiments, which will be described in Chapter 5.

4

Performance Measures

Chapter 3 has described the classifier that is used to determine which test runs show anomalous behaviour. This approach is based on a performance measure, for which the runtime was initially used. During this thesis, the conclusion was drawn that runtime was not an appropriate measure to evaluate the performance of vehicle routing heuristics. This chapter will provide arguments for this statement. Furthermore, it will state which characteristics the measure should have. Finally, the alternative measures will be outlined. Chapter 5 will explain how the experiments will be used to show the usefulness of each of these measures and present their results.

4.1. Issues with runtime as performance measure

The initial classifier that was created to detect anomalous test runs made use of runtime as the performance measure. This does not appropriately evaluate the performance of a heuristic. To recap, a heuristic is a technique for solving optimization problems that is used when exact methods are too slow. Heuristics employ methods that have shown to be useful in practice. In contrast to approximation algorithms, there are no guarantees on their solution quality. As a result, there is a trade off between runtime and quality, that should be reflected when evaluating the performance. Letting a heuristic run for longer often increases the quality at the cost of it requiring more time.

The problem with using runtime as the performance measure is that it does not reflect the full picture of this trade off. Imagine that the runtime has shown some regression. In one scenario, the solution quality could have improved a lot, in which case the overall performance has improved so this should not be marked as an anomaly. If in another scenario the solution quality has stayed the same, the overall performance has decreased and it should be marked as an anomaly. As said before, the runtime alone does not reflect the full picture.

To further illustrate this example, take a look at Figure 4.1. This image shows the results of two builds, the green and red lines, on the same test case. The chart shows the cost function on the y-axis and the time on the x-axis. The chart can be divided in two phases: The first phase, where the value of the cost function is increasing, is where the algorithm starts with an empty solution and starts planning tasks to create a solution. The second phase starts when the value of the cost function starts decreasing since the heuristics start to optimize the given solution iteratively. From the chart it can be observed that the green line ends before the red line. However, the quality of the green line is worse than that of the red line, since the value of the cost function is higher. Again, if only the runtime would be considered, the wrong conclusions would be drawn. The remainder of this chapter will outline other measures and motivate their usefulness.

4.2. Alternative performance measures

It has been concluded that the runtime is not an appropriate measure to evaluate the performance of vehicle routing heuristics. It does not reflect the quality runtime trade off that is the most desired characteristic of this measure. This section will discuss which measures would be appropriate to evaluate the performance. It will first explain them and finally outline the theoretical pros and cons of each measure. Chapter 5 will discuss how these measures will be tested and how the measures perform in practice. Most of the measures

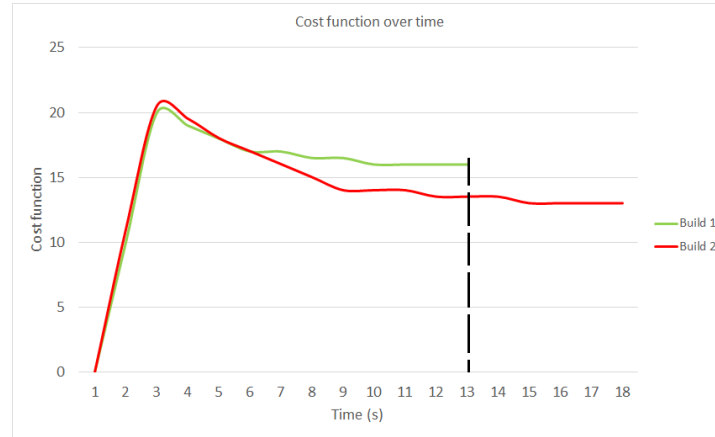


Figure 4.1: This image shows the results of two test runs, the green and red lines, on the same test case. The chart shows the cost function on the y-axis and the time on the x-axis. The chart can be divided in two phases: The first phase, where the value of the cost function is increasing, is where the algorithm starts with an empty solution and starts planning tasks to create a solution. The second phase starts when the value of the cost function starts decreasing since the heuristics start to optimize the given solution iteratively. The red line would be the better build in this situation, since its solution quality is better if it is compared at the same point in time.

that will be described rely on a point in time to measure them at. A discussion on the options will be given in Section 4.3. A summary of the pros and cons of the measures can be seen in Table 4.1.

4.2.1. Quality comparison at same runtime

The first measure compares the quality of both solutions at the same point in time. A major issue in comparing the performance is that comparing two solutions at different points in time is difficult. Generally, the longer the algorithm runs, the better the solution becomes. As an example, observe Figure 4.1. There are two solution paths that lead to different solutions. Although the green build takes less time to reach its solution, its quality is also worse. This newly proposed measure would evaluate the quality of both builds at the same point in time, which is visualized with the dotted line. At this point in time, the red build is clearly better than the green solution. This solution is a realistic one, since the limit on computation time allowed can be altered. This would result in the situation just described.

The pro of this measure is that it is able to compare the solution quality of two builds at a specific point in time. It is a concrete measure, where the only point of debate would be at which point in time to compare the quality. The con of this measure is that the quality is only reflected at a single point in time. For instance, observe Figure 4.2. The builds in this example result in the same end solution. However, the yellow build has a better temporary solution. If the quality would only be compared at the end of the run, it would seem like these builds have the same performance, although they clearly don't.

4.2.2. Area under the chart

Whereas the previous measure focused on comparing a single point in time, the area under the chart gives an overview of the solution quality over its whole runtime. The area under the chart can be calculated using an integral between points. Although the test cases reflect some of the real world scenario's, there are still different characteristics between customers. One customer may have 10 minutes available to compute a solution, while another may have 30 minutes available. Not all of these situations can be reflected in the test cases, since there are too many variables. As a result, the intermediate solutions should also be somewhat evaluated. An example of the usefulness of the area under the chart can be seen in Figure 4.2. The yellow build achieves the same end result as the blue one, while its intermediate solutions are better. Therefore, the yellow build is the preferred one in this scenario, which is reflected when using the area under the chart. A point of debate for this measure is the points in time to use for calculating the area under the chart. Although this example has two builds with the same runtime, this is often not the case. Using different points in time could give a twisted result, since a longer runtime inevitably increases the area under the chart. Therefore the decision has been made to use the same point in time when evaluating the performance.

The pro of this measure is that it gives an overview of the solution quality during the computation process. It is able to give a preference to builds with a better intermediate solution. The performance of both examples in Figures 4.1 and 4.2 could be appropriately evaluated using this measure. The con of this measure is that it

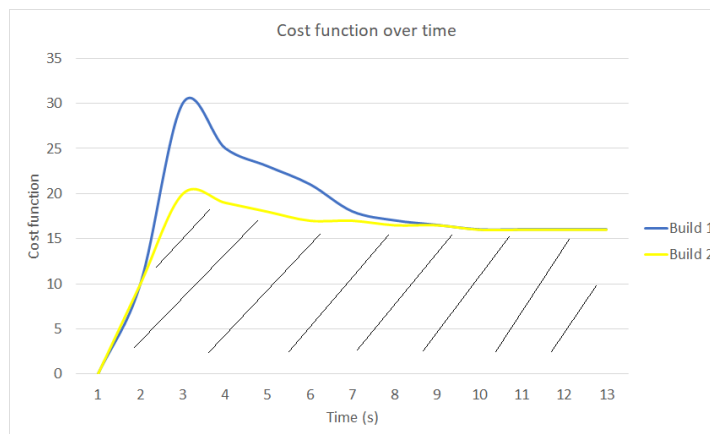


Figure 4.2: An example of the usefulness of the area under the chart as a measure of evaluating the performance. Contrary to comparing at a single point in time, the area under the chart gives an overview of the solution quality over the whole runtime. The yellow build achieves the same end result as the blue one, while its intermediate solutions are better. Therefore the yellow build is the preferred one in this scenario, which is reflected when using the area under the chart.

might prefer a better intermediate solution, at the cost of ignoring the quality of the end solution. It is debatable whether this behaviour is desired, since it can be argued that the end solution is the most important.

4.2.3. Maximum difference in cost between solutions

The final measure that will be discussed will be the maximum difference in costs between two builds. The previous solutions considered the solution quality at the end of the run and over the duration of the run. Although these were good at pointing out a clear winner, such as in Figures 4.1 and 4.2, there may be situations where there is no clear winner, as in Figure 4.3. In this figure, the orange build has a better early solution, while the purple build has the better final solution. It could be argued that the final solution is the most important and therefore the purple build should be preferred build in this scenario. However, there are some other interesting measures that could be used to make a performance comparison between these two builds. This would be to look at the maximum distance between the builds, both vertically and horizontally. Large distances between two builds would be a good indicator for large changes, which are often an indication of anomalies or at least worth inspecting anyway. Using this as a method for evaluating the performance could be a good option. The maximum vertical distance between two builds would reflect the maximum difference in solution quality. The larger the gap, the worse the performance. Overall, the more consistent build is often preferable, therefore in Figure 4.3, the purple line would be the preferred one, since the gap in solution quality is quite large.

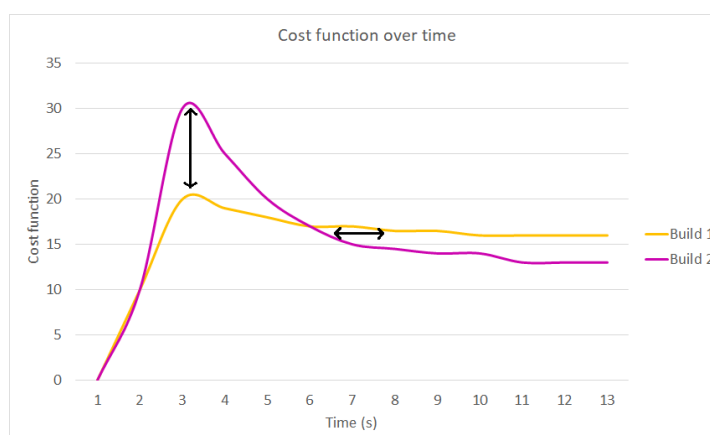


Figure 4.3: An example that shows the usefulness of the maximum distance between two solutions. The previously discussed examples show a clear better build, while there is more discussion possible in this one. Both the vertical and horizontal difference could be useful when discussing the better build in this situation.

	Subsection	Pros	Cons
Quality	4.2.1	Concrete and realistic comparison of final results.	Only reflects a single point in time.
Area	4.2.2	Assesses the quality of the whole computation process.	Might prefer intermediate solutions over end solutions.
Difference	4.2.3	Can spot large changes which often reflect issues.	Not so useful when changes are small.

Table 4.1: A summary of the theoretical pros and cons of each performance measure. For details on the measures, read their respective subsection.

The pro of this measure is its ability to evaluate the performance in case of large differences, such as in Figure 4.3. In these scenario's, the two earlier described measures are indecisive, although the charts show clear differences between the builds. The con of this measure is when there are small differences between the builds. Although Figure 4.1 has a clear winner, the distances between the lines are so small that the maximum distance would poorly reflect this.

4.3. Alternative measuring points

The first two measures, being the comparison at the same runtime and the area under the chart, both require a point or area at which to calculate its value. If the runtime of both builds is the same, this is not an issue, since the end time can be used. However, in practice the runtime is often not the same. This section will discuss the possibilities and discuss an alternative for these points.

4.3.1. Determining the end time when runtimes are different

The first debate is the definition of the end time when the runtime of two builds is different. There are two viable options which will be discussed. The first option is to use the end time of the shortest build as the end time, which can be seen in Fig 4.1 at the dotted line. In the situation of this example this would arguably be the best option. It is certain what the solution quality at this time is for both solutions. The issue arises when the green line would instead stop a lot earlier due to for instance a bug. At that point in time the solution quality could be different, resulting in a wrong evaluation of the performance. The solution for this would be to ignore the runtime of both builds and look at the solution quality obtained at the end of the end of the run. This would be equivalent to evaluating the performance at the end time of the longest run, while extending the line of the shortest run by a straight horizontal line. However, the downside of this method is that if the shorter line would be allowed more runtime, for instance by fixing the bug that caused it to stop earlier, it will generally further optimize its solution further and it might be the better build of the two. This would be the case if an improvement was made, but also a small bug was inserted. Both of these methods will be experimented with. During later chapters the term LONG will refer to the end time of the longest build, while SHORT will refer to the end time of the shortest build.

4.3.2. Alternative times for performance comparison

The first measure, to compare the quality at the same point in time, has been described by comparing the quality at the end time of the builds. Although this is a good option that has its uses, there are alternatives that would better reflect the intermediate quality of each of the builds. Hutter et al. [26] made use of the capttime, which is the time a heuristic was allowed to run. In their research they have observed the solution quality of their heuristics at captimes of k_{max} , $k_{max}/10$ and $k_{max}/100$. This would reflect the intermediate qualities, as well as the end solution. Therefore, during the experiments the solution quality at 10%, 50% and 100% will be evaluated. The change in numbers was made since this includes one point in the construction phase of the algorithm, and two times in the optimization phase. During later chapters the term START will refer to the quality at 10%, MIDDLE will refer to 50% and END will refer to 100%.

4.4. Summary

The classifier that was described in Chapter 3 requires a performance measure to compare test results against each other. The original paper on performance profiles used the runtime as a performance measure for exact

optimization methods. However, the goal of this thesis is to evaluate heuristics. When evaluating heuristics it is important to also consider the quality of the obtained solution, which is not reflected by the runtime. Since the runtime did not suffice for evaluating heuristics, this chapter introduced three alternatives. The first alternative is to evaluate the quality of the obtained solutions, at the same point in time. This measure focuses on the end result, since that is arguably the most important. The second option, the area under the chart, gives a better overview of the complete run. Finally, the maximum difference should be useful when there are large changes between two cost functions, that might not be reflected in the other two measures. For each of the three measures, there are several evaluation options available. This chapter introduced the LONG and SHORT terms for which end time to use, when runtimes are different. Then it introduced the START, MIDDLE and END terms as alternatives times for performance comparison. This results in 3 measures * 2 end times * 3 evaluation points = 18 measure configurations total. Next, Chapter 5 will explain how these 18 measure configurations will be used in the experiments and present the results.

5

Experiments

The previous chapters have provided a theoretical description of the methodology of this thesis. First, Chapter 3 has described the methodology behind the classifier that will be used. Next, Chapter 4 has described three possible measures to compare the tests results against each other. Including the variables evaluation point and end point, there are 18 measure configurations. This chapter will describe how the experiments will test which of these 18 measures is best able to detect anomalies in the performance test results and finally present the results.

Secondly, this chapter will include a brief exploratory test case investigation. The goal of the classifier is to detect anomalies. This exploratory investigation will observe the diversity of the test cases and their ability to spot anomalies. These two characteristics make the test cases useful in detecting anomalous test runs.

5.1. Experiment design

The goal of the experiments is to apply the theory that was described in the previous chapters to a real world scenario. In short, the approach used in the experiments is to test the different combinations that have been described in Chapter 4 and see how they perform in practice. Each combination will be evaluated individually to evaluate its strengths, weaknesses and possible issues. Once that is done, there will be a comparison between all of the combinations. The goal of this comparison is to find out which one(s) perform best in general and to determine whether there are specific situations in which the measures are good or bad. For each combination, the judgement made will be evaluated by analysing test runs. This will result in a positive or negative judgement for the analysed test runs. Based on this, the combinations will be ranked and an overall judgement on them will be made.

The remainder of this section will discuss the most important aspects of the experimental procedure. First, the used approach will be further specified. Then, the evaluation of each combination will be discussed. Finally, it will be explained how cross-validation has been applied.

5.1.1. Approach

In order to evaluate each measure configuration in the same fair way, it is important to adopt a systematic approach to doing this evaluation. The systematic approach adopted in this thesis can be described in a couple of steps. This will be done during this subsection.

As mentioned, each measure configuration will be evaluated individually at first. The goal of this evaluation is to observe how well this configuration works in practice and to see if there are any apparent issues with it. To do this, each test run that is marked as anomalous is inspected. The majority of the test run will be marked as non-anomalous. Of these non-anomalous test runs, 10% will be inspected at random. After inspecting a single test run with multiple test cases, it will be judged as a good or a bad marking. These judgements will then be used to calculate the amount of false positives and false negatives per measure configuration. Comparing the amount of false positives and false negatives across the different configurations gives a good indication as to which configurations perform well and which do not.

For the analysis of each test run, a tool has been developed to help in this process. Images of this tool can be observed in Figure 5.1 or Appendix B, which contains more detailed images. This tool consist of three columns. The left column shows the test runs that have been analysed. Each test run that is marked as

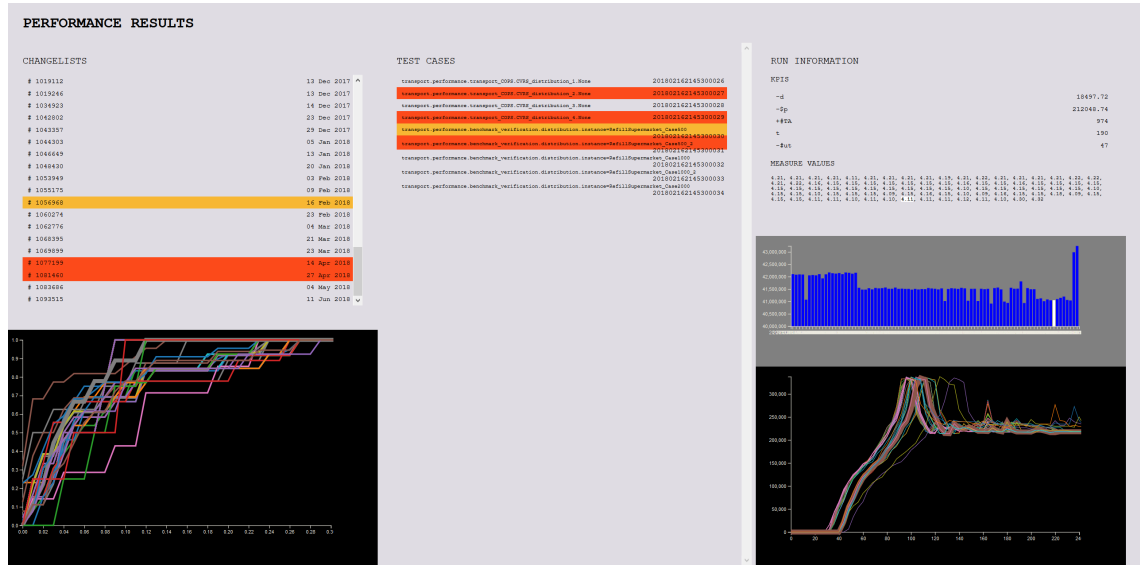


Figure 5.1: The tool that was used to analyse the test runs and test cases of a single combination and evaluate its usefulness. It is split in three columns. The left column shows all the analysed test runs. The middle column shows the test cases for the currently selected test run. Finally, the right column shows the information for the currently selected test case and compares the measure values and cost functions to other runs of the same test case.

anomalous has been given a red color. The middle column shows the test cases for the currently selected test run. Again, the anomalous ones are marked as red. Finally, the right column shows the information for the selected test case. This column includes the measure value in the bar chart, which also contains the results of other runs of this test case. The bottom line chart plots the cost function, also of the other runs of this test case.

Secondly, an evaluation sheet was made to help with the evaluation of the measure configuration on several test runs, see Appendix A for an example. Each anomalous test case will be inspected, as well as 3 randomly selected non-anomalous test cases. Since the marking of anomalous or non-anomalous is made based on the measure value, this one is first observed. The next step is to look at the line chart of the cost function to determine whether this test run is actually anomalous or not, and a severity of 1-5 will be given. The combination of these two observations, for multiple test cases of a test run, gives enough information to draw a conclusion about the judgement made for this test run, either good or bad. Finally, any apparent issues will also be noted.

5.1.2. Metrics for evaluating combinations

As part of the systematic procedure of evaluation, it is important to adopt a set of metrics that will be used to evaluate the performance of each of the combinations. There are several options to evaluate the performance of binary classifiers. The most obvious choice is to calculate the amount of times the classifier had it right or wrong. This leads to the first four metrics: false positives, false negatives, true positives and true negatives. In the context of this thesis, a positive result is a test run that is marked as non-anomalous, while a negative value is a test run that is marked as anomalous. As such, a true positive is a test run that is correctly marked as non-anomalous. Conversely, a false positive is a test run that is marked as non-anomalous, while it is actually anomalous. The reverse goes for true and false negatives. Since the goal of a classifier is to minimize the amount of times it is wrong, the average of the false positives and false negatives is also adopted as a metric.

Although the average of wrong classifications is a good metric, it does not consider the ratio false positives to false negatives. A metric that better reflects this ratio is the F_1 score. The F_1 score considers the recall and the precision of the classifier and is a measure for its accuracy. The precision tells us how many of the selected items are relevant. The recall tells us how many of the relevant items that are in the set we have selected. The equations for precision (p), recall (r) and F_1 can be found in Equations 5.1, 5.2 and 5.3. The terms TP, FP and FN are used for True Positives, False Positives and False Negatives respectively. An application of the F_1 score can be found in this paper by Goutte and Gaussier [24].

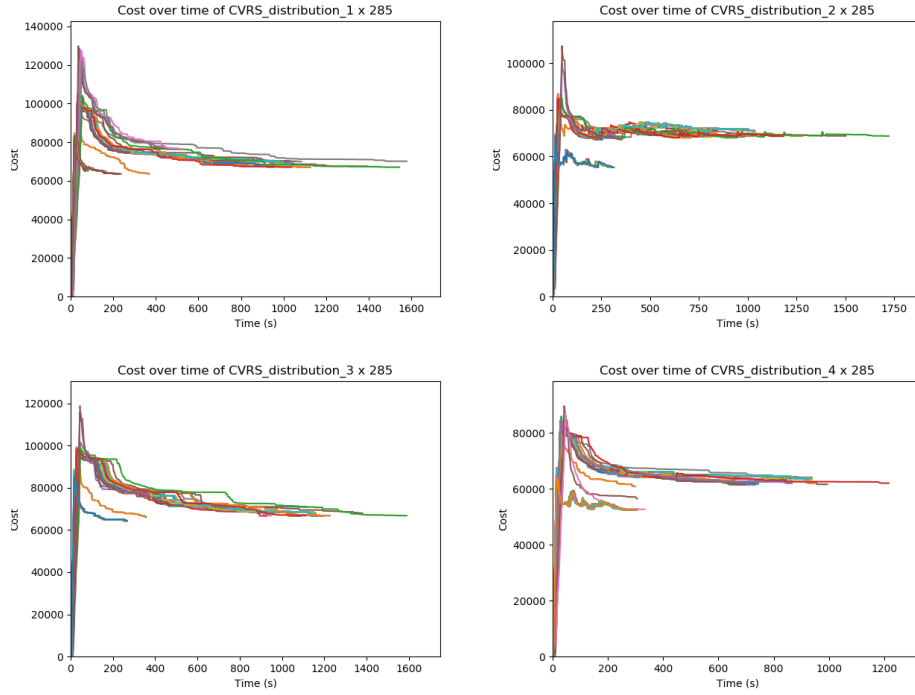


Figure 5.2: Cost functions over runtime of the CVRS distribution cases for data of 2017, 2018 and 2019. Four different cases, each ran 285 times throughout the years.

$$p = \frac{TP}{TP + FP} \quad (5.1)$$

$$r = \frac{TP}{TP + FN} \quad (5.2)$$

$$F_1 = \frac{2 * p * r}{p + r} \quad (5.3)$$

5.1.3. Cross-validation

The data available to run the experiments covers a period of 3 years, ranging from 2017 to 2019. During these 3 years, the performance tests are run semi-regularly, on average about once a week.

An often used technique when experimenting is called cross-validation. It is a technique that assesses how well a developed model behaves in general. This is done by splitting the data in two parts. The first set of the data is used to train the model. This set is called the training or known data set. After the model has been trained on the first set, its working is validated on the second set. This second set is called the validation or unknown data set. The goal of this technique is to test the functioning of the model on data that it was not trained with. Since training is considered harder than validating, the training data set is often larger than the validation data set.

Cross-validation has been adopted in this thesis in order to increase confidence in the developed methodology. As stated before, the available data originated from three years; 2017, 2018 and 2019. For the experiments, the data has been split in two. The training set consist of all data originating from 2017 and 2018. This data will be used to train the model. In this case, that means that all 18 combinations will be ran and analysed on this data set. The best combination(s) will then be validated on the validation data set, which consist of the data originating from 2019.

5.2. Exploratory test case investigation

The working of the methodology developed during this thesis relies on the ability of the test cases to detect anomalies. A good test case has a few characteristics. Although analysing the usefulness of a set of test cases

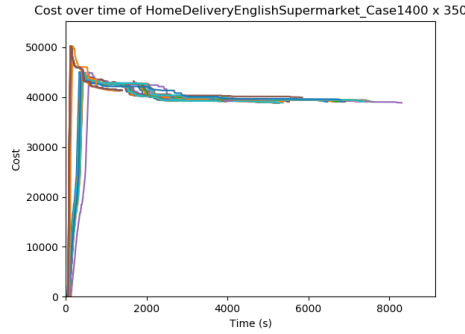


Figure 5.3: Cost functions over runtime of the Home Delivery English Supermarket case for data of 2017, 2018 and 2019. One cases, ran about 350 times throughout the years.

could be a thesis by itself, it is still useful to do a small exploratory investigation. The goal of this investigation is to verify that the test cases are able to find anomalies and to show their diversity. All of this will be done using the available data.

Throughout the years of the available data, some test cases have been used regularly, while others were used more infrequent. The regular ones are of interest to investigate and can be divided into four categories, based on their names. These four categories contain 13 test cases which appear regularly throughout the data of 2017, 2018 and 2019. To asses the usefulness of each of the test cases, their cost functions have been plotted. A good test set would contain different characteristics for these cost functions that also show clear differences within the case itself, which would indicate anomalies. The following subsections highlight the important aspects of each category.

CVRS distribution

The first set of cases are the CVRS distribution cases of which there are four. CVRS is ORTEC's route optimizer and since the test cases are just numbered 1 until 4, this probably means that they are different routes. The plots can be found in Figure 5.2. The characteristics from these four test cases look similar; each of them is ran about 285 times, the cost function has values in the range of 100.000 more or less and the runtime of each case is about 1.500 seconds. The shape of each of the plots is similar, which can be explained by them being in the same category. Finally, each case has a baseline, a line which most of the cases follow. All the cases have some lines that are significantly different from this baseline, which would indicate anomalies. To conclude, these test cases look useful. The four cases have a clear baseline and some deviations from this, which means the cases are able to detect anomalies.

HomeDeliveryEnglishSupermarket

This second category only contains a single test case. It has the suffix "*_Case1400*", which indicates 1400 deliveries to do. The plot can be found in Figure 5.3. The characteristics are quite different from the ones in the previous section; its ran 350 times, the cost peaks around 50.000 and then slowly converges to 40.000 and the time required is up to 8.000 seconds. Even though these characteristics are different, there are no significant differences regarding the cost function within this case itself. There are differences in the amount of runtime taken, but this could simply be changed by limiting the amount of allowed runtime given. The ability of this case to detect anomalies is therefore questionable. As such, even though this test cases' characteristics are different from the first category, is it not as useful in detecting anomalies.

LongTripsWithLegislation

The third category contains three test cases. Their suffixes are "500", "1000" and "2000", which again, indicate the amount of deliveries to do. Their plots can be found in Figure 5.4. Unlike the CVRS distribution categories, the cost and runtime of these test cases are quite different, which can be explained by the fact that there are varying amounts of deliveries to do. These cases are each ran about 330 times. The things that makes these test cases different is their shape. Where other cases started at $t = 0$, these cases require a startup period, which clearly varies. This unique characteristics, along with the fact that they also show clear differences, make this test cases a strong addition to the test suite.

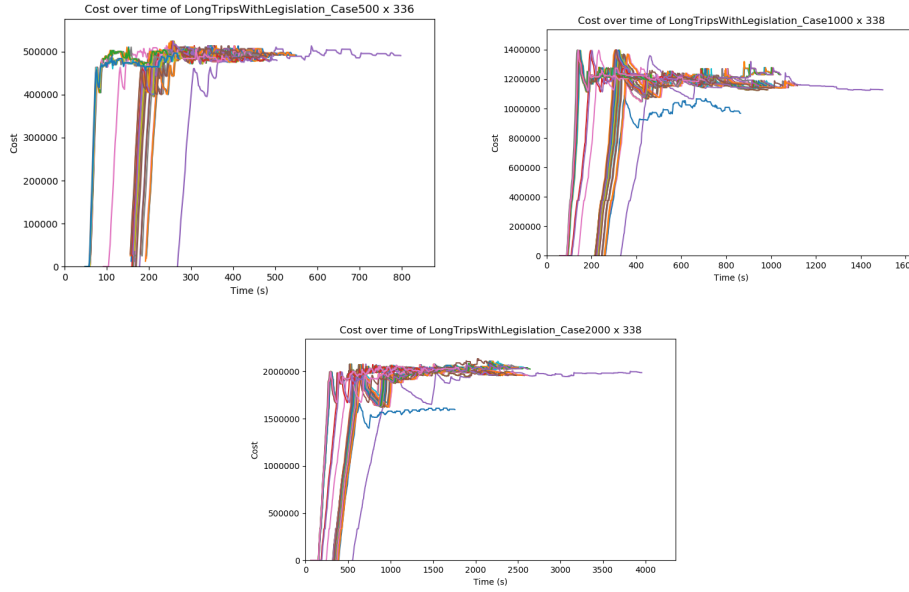


Figure 5.4: Cost functions over runtime of the Long Trips With Legislation cases for data of 2017, 2018 and 2019. Three different cases, each ran about 350 times throughout the years.

RefillSupermarket

The final category contains five test cases. Their suffixes are "500", "500_2", "1000", "1000_2" and "2000", which again, indicate the amount of deliveries to do. The plots can be found in Figure 5.5. These cases are ran about 330 times each and have notable different costs and runtimes, as in the previous category. The shape of the plots is similar to those of the CVRS distribution category, but there is also a difference in starting time. The thing that makes this category unique, is its shape. Other categories have a steady converging line, whereas this category does not. There are notable spikes where the cost function increases again, which is likely due to another optimization being more important. Along with the fact that there are differences within the individual cases make this a strong addition to the test suite.

5.2.1. Conclusion

The goal of this exploratory test case investigation was to verify the ability of the test cases to find anomalies and to show their diversity. Four categories were inspected, which are run a significant amount of times. Three out of the four categories show the ability to detect anomalies, while all categories have clearly different categories. Therefore it can be concluded that this test suite is well suited for the purposes of this thesis, which is detecting anomalies in different scenario's.

5.3. Results

This section will present the results obtained by running the experiments described in the previous sections. The reasoning for each measure variable used has been described in Chapter 4. The experiments have been conducted in two phases, as prescribed by the cross-validation technique: a training phase and a validation phase. The first subsections of this section will discuss the results obtained by the training phase. The final subsection discusses the observations obtained from the verification phase.

5.3.1. Comparison of measure variables

The first set of results that will be discussed are obtained during the training phase. As described in Section 5.1, all 18 combinations have been run on data of the years 2017 and 2018. The results can be observed in Table 5.1. The metrics shown in this table have also been described in Section 5.1. The yellow marked measure configurations are the best performing configurations, based on the metrics average and F1-score. The remainder of this section will discuss observations made from these results.

First, it can be noted that the measures "area under the chart", or area from now on, and "quality at the same time", or quality from now on, both have well performing configurations. Several measure configura-

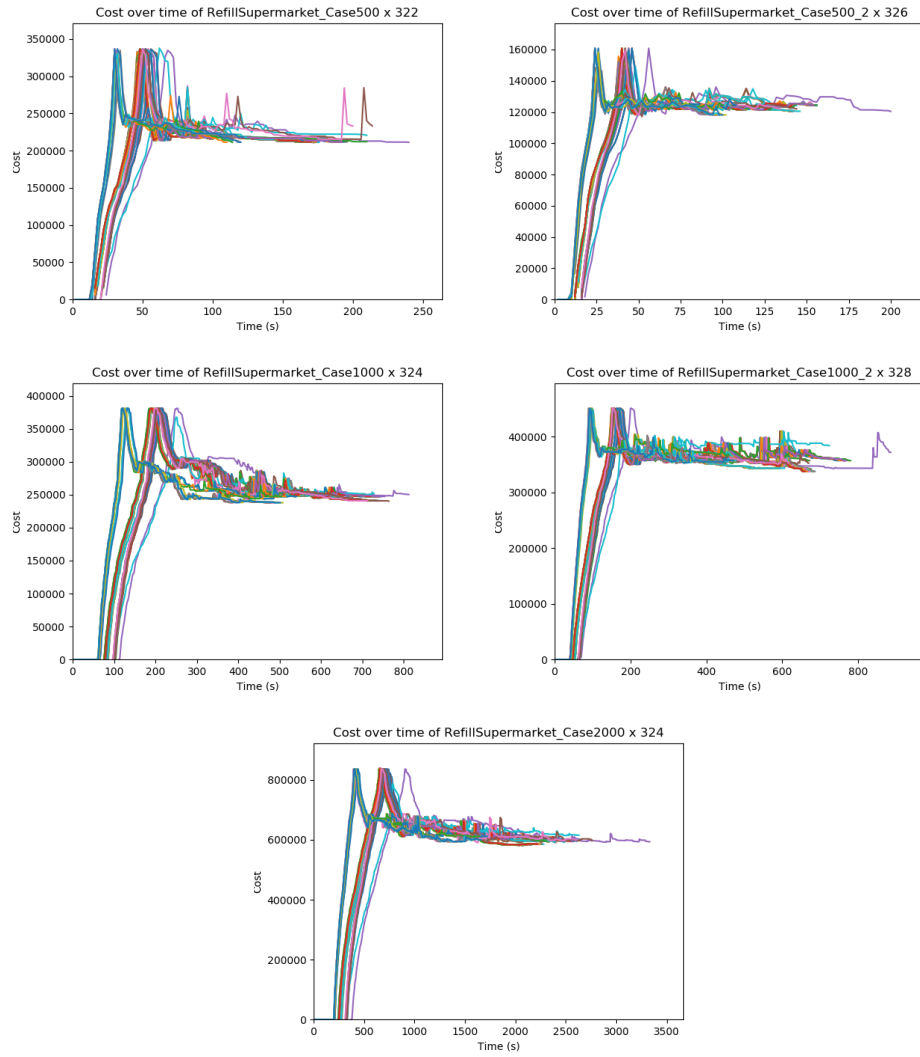


Figure 5.5: Cost functions over runtime of the Refill Supermarket cases for data of 2017, 2018 and 2019. Four different cases, each ran about 330 times throughout the years.

Name	FP	FN	TP	TN	Avg.	P	R	F1
AREA_LONG_START	0,17	0,60	0,83	0,40	0,38	0,83	0,58	0,68
AREA_LONG_MIDDLE	0,17	0,29	0,83	0,71	0,23	0,83	0,74	0,79
AREA_LONG_END	0,00	0,29	1,00	0,71	0,14	1,00	0,78	0,88
AREA_SHORT_START	0,78	0,71	0,22	0,29	0,75	0,22	0,24	0,23
AREA_SHORT_MIDDLE	0,33	0,63	0,67	0,38	0,48	0,67	0,52	0,58
AREA_SHORT_END	0,17	0,63	0,83	0,38	0,40	0,83	0,57	0,68
QUALITY_LONG_START	0,50	0,67	0,50	0,33	0,58	0,50	0,43	0,46
QUALITY_LONG_MIDDLE	0,00	0,50	1,00	0,50	0,25	1,00	0,67	0,80
QUALITY_LONG_END	0,17	0,25	0,83	0,75	0,21	0,83	0,77	0,80
QUALITY_SHORT_START	0,86	0,88	0,14	0,13	0,87	0,14	0,14	0,14
QUALITY_SHORT_MIDDLE	0,17	0,71	0,83	0,29	0,44	0,83	0,54	0,65
QUALITY_SHORT_END	0,17	0,25	0,83	0,75	0,21	0,83	0,77	0,80
DIFF_LONG_START	0,50	0,50	0,50	0,50	0,50	0,50	0,50	0,50
DIFF_LONG_MIDDLE	0,17	0,50	0,83	0,50	0,33	0,83	0,63	0,71
DIFF_LONG_END	0,33	0,78	0,67	0,22	0,56	0,67	0,46	0,55
DIFF_SHORT_START	0,75	1,00	0,25	0,00	0,88	0,25	0,20	0,22
DIFF_SHORT_MIDDLE	0,17	0,75	0,83	0,25	0,46	0,83	0,53	0,65
DIFF_SHORT_END	0,75	0,33	0,25	0,67	0,54	0,25	0,43	0,32

Table 5.1: Results of the training phase that show the metrics for all measure configurations. The acronyms are the true/false positives/negatives. The average metric is based on the FP and FN metrics. P is for precision and R is for recall. The yellow marked rows are the best performing ones, according to the average and F1-score metrics. The F1-score should be maximized, since it represents accuracy. The average should be minimized, since it represents wrong classifications.

tions that use these measures score well, judging by the metrics average and F1-score. These configurations have a F1-score of around 80%, with one even at 88%. This means that they are accurate roughly 80% of the time. Although a higher score would be better, the current implementation is an initial version and improvements made could improve this score even more. The amount of false positives and negatives, that constitute the average value, are relatively low for these configurations. The percentages higher than 0 represent 1 or sometimes 2 failures in a sample size of ~8 analysed test runs. In short, the experiments show that the classifier can point out anomalies with reasonable certainty. The "maximum difference", difference from now on, metric is notably worse in all its measure configurations.

In the two well performing measures, there seems to be a recurrent pattern when looking at the relation between quality in the START, MIDDLE and END evaluation points. The START point seems to be the worst one, with MIDDLE being better than it and END outperforming both. Similarly, the SHORT option seems to be worse than the LONG option in most cases. Both of these observations can be summarized as the more information is available, the more accurate the prediction becomes. Although this sounds straight forward, there is a case to be made for the other configuration options that only consider a subsection of all available data. There could be anomalies that are most prominent in the first half of the test case run. These issues would potentially not appear when considering all available information. The argumentation to only using a subsection of the available data was that different anomalies would be found that would otherwise not be found. These measure configurations were not expected to do well in all situations, but rather complement the other ones. To find out whether this was the case, further investigation is done in Section 5.3.2. The SHORT variable was used to promote equal comparison between test runs with largely different running times. It is inevitable that the longer the runtime, the better the solution. Therefore, comparing the solution at the end of the shortest run seemed fair. Although the SHORT variable is worse than the LONG one in most situations, the "QUALITY_x_END" configurations have the same results for LONG and SHORT.

Part of the reason why the "maximum difference" performed badly has to do with the method of comparison introduced by the performance profile. The performance metric used by the original paper that introduced the performance profile is the runtime. This metric has a specific characteristic that the area and quality measures also have, but the difference does not. The range of values of the runtime, area and quality is relatively small, percentage wise, compared to the difference measure. A measure that is twice as bad, or 100% worse, as the best result obtained is extremely bad. Imagine a test of which the quality is suddenly twice as bad, that should never happen. While performing the experiments, most if not all values were found to be

within 30% of the best one. This is different for the difference measure, where the range is much larger. It is not unusual for test runs to be similar to the previous one, which results in a small difference measure. Let's assume a cost function with a maximum value of 100.000 and imagine the best found difference to be 100. As a result, a test run with a difference of 500, which is still pretty good, is already 5 times or 500% worse. These values are not unusual, but result in a much larger range of values. This makes the comparison of test runs much harder and in the current implementation worse, as can be observed in the results. Some potential improvements for the difference measure are discussed in Section 6.1.

5.3.2. Comparison of test runs marked as anomalous

To investigate whether the different measure configuration find different anomalies, a further analysis focused on which specific test runs were marked as anomalous, rather than just looking at how many were correct or incorrect. If two measure configurations were to mark completely different test runs as anomalous, both correctly, they might be useful in finding different types of anomalies. Although there were some differences across the different configurations, most of these different anomalies were incorrectly marked as anomalies. It was only rarely the case that these different anomalies were actually correct. A potential improvement could be to make use of multiple measure configurations and a decision rule to make a judgement on anomalies. This will be discussed in Section 6.1.

5.3.3. Issues with the classifier

There was one issue that was noted for all measure configurations. Rather than being a strong or weak point of a specific measure configuration, this issue represents a situation in which the classifier itself is performing bad. Throughout the data, the amount of test cases that each test run contains varies. There were some test runs that only contained 4 test cases, while some other situations had 15. In order to understand why this is an issue, the structure of the classifier has to be inspected. The classifier makes use of the performance profile to determine which test runs are anomalous. The performance profile is a cumulative distribution function of which the y-axis values only increases as the x-axis values increase. The y-axis value represents a fraction of the test suite. This means that if there are only 4 test cases, it will increase in steps of at least 0.25, since that is equivalent to a single test case. However, if there are 15 test cases, the smallest step is 1/15 which is roughly 0.07, much smaller. Anomalies are then detected by using a clustering algorithm, which calculates the distance between these lines to cluster them. The two situations above are inevitably going to be far away from each other, simply because their minimum step size is so different. This means that even though both cases could be non-anomalous, the clustering will see one of them as far away and mark it as anomalous, regardless of whether it is actually anomalous.

This issue is further enhanced by the already apparent difference that a differing amount of test cases would bring. Imagine a situation in which 4 out of 20 test cases would be anomalous, while the others are not. That might not be a big issue, since there are still a lot of cases that are non-anomalous. However, if in the same situation these 4 anomalous test cases would be the only cases that were run, the same situation would look severely worse. These two issues make it such that, with the current configuration, the classifier does not handle a varying amount of test cases well.

A different issue that appeared in all measure configuration were test cases that were only ran a single time. Since the performance profile compares each test case to the best known test case, it would think that this single test case is the best one and see this test run as a good run, even though it is not compared with anything. This skews the results of the other test cases present in this test run, potentially misinterpreting them.

5.3.4. Training data conclusions

As can be seen in the results, there are 5 measure configurations that perform best according to the measures. These ones are marked yellow in Figure 5.1. These configurations have an average of below 25% and a F1-score of 79% or higher. The metrics show 4 configurations that perform equally well, with the *AREA_LONG_END* being clearly the best one in both the average and F1-score metrics. Of the 5 well performing configurations, there are 2 *MIDDLE* evaluation points. The corresponding *END* evaluation points of the same variable options perform better, which is a pattern that was discussed before. This difference in performance is only further enhanced by the fact that the *SHORT* and *MIDDLE* only use a part of the available data in the hopes of finding anomalies in that specific part. Therefore, even if they would obtain similar results to the *END* variable, they would still be worse in general, since they ignore half of the data available and as a consequence any anomalies present in that part of the data. Finally, the best methods all seemed to

Name	FP	FN	TP	TN	Avg.	P	R	F1
AREA_LONG_END	0,25	0,00	0,75	1,00	0,13	0,75	1,00	0,86
QUALITY_LONG_END	0,25	0,14	0,75	0,86	0,20	0,75	0,84	0,79
QUALITY_SHORT_END	0,00	0,38	1,00	0,63	0,19	1,00	0,73	0,84

Table 5.2: Results of the verification data that show the metrics for the three best performing measure configurations. The acronyms are the true/false positives/negatives. The average metric considers the FP and FN metrics. P is for precision and R is for recall. The F1-score should be maximized, since it represents accuracy. The average should be minimized, since it represents wrong classifications. All configurations perform well with a over 80% accuracy and a low amount of false classifications.

mark similar test runs as anomalous, which further increases the confidence in their usability. As such, the 3 measure configurations that are deemed best are the *AREA_LONG_END*, *QUALITY_LONG_END* and *QUALITY_SHORT_END*. Since the other measure configurations have already shown to be worse, it is no use running them again on the verification data. Therefore, the best three will be ran on the verification data to confirm that they also work well on unseen data.

5.3.5. Analysis of verification data results

The experiments on the training data resulted in three measure configurations that performed best. These three configurations have then been run on the verification data to verify that they also work on unseen data. The results can be observed in Table 5.2 and will be discussed in this section.

The goal of running the same experiments again on a different data set are meant to increase confidence in the developed system. It counters over-fitting on a single data set and confirms that the methodologies work in general, rather than on a specific data set. From the results, it can be observed that all three measure configurations perform similar on the verification data as on the training data. The average values were slightly over 20% on the training data. In the verification data they have similar values, even slightly below the 20%. Secondly, the F1-scores are also similar with an accuracy of roughly 80%. Similar to the training data, the *AREA_LONG_END* is again clearly the best configuration. Both the average and the F1-score are better than the other two configurations. Second is the *QUALITY_SHORT_END*. Although the amount of false negatives is quite high (38%), its accuracy is still good since it has no false positives. Finally, the *QUALITY_LONG_END* performs well with an accuracy of 79%.

To confirm that the results that were obtained were correct, an ORTEC developer, who is an expert on algorithmics, was asked to give a second opinion on the results. Since analysing all the results would take too much time, only a part was discussed. If the expert agreed with that part, this would be a good indication for the quality of the other results. The results that were discussed were the ones of the *AREA_LONG_END* configuration. First off, he was enthusiastic and interested in the applications of the thesis. He agreed that the analysis of the performance of routing heuristics is a difficult task and this thesis would help a lot to that end. Through analysing the results, no strange judgements were found. The first interesting observation was on the area, when the starting time of the algorithm is variable, as in Figure 5.6. The earlier the algorithm starts, the better it is in general. However, the area under the chart becomes smaller if it starts later. Since the area is the performance measure here, the classifier thinks that the bad ones are actually doing well. The expert agrees with the intuition behind the area measure, but in this case its not doing what it should. It should be noted that since all values are compared relatively to each other, the right ones are still marked as anomalous though so this is not such a big issue, rather a minor one. The second interesting remark made was to see how the anomalies that were found are reflected in the source code. It would be interesting to see if the developers agree that there was an anomaly at those test runs. This would take a considerable amount of time to check, especially since the test run of a period have to be combined since the tests are ran each week and not on each change.

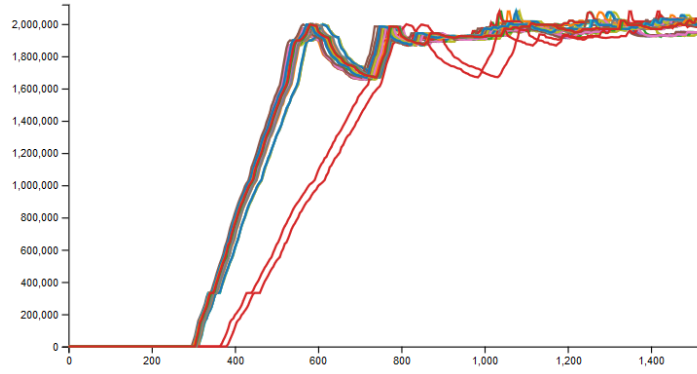


Figure 5.6: Cost function of an example case where the starting time of the algorithms are variable. The intuition behind the area under the chart is that the smaller the area, the better. In this situation, the runs with the later startup time have a smaller area while they are actually worse.

5.4. Summary

This chapter has described how the theory of the previous chapters has been used in the experiments. All 18 measure configurations have been tested on their ability to detect anomalies. Each one has been evaluated through a systematic approach that results in the metrics average (of false classifications) and the F_1 score. The experiments were conducted in two phases, according to the cross-validation technique. Three measure configurations performed best with an accuracy of roughly 80% on both data sets. The shortcomings of the difference measure were mostly due to the large range of its values. The START and MIDDLE options were mostly worse because they ignored part of the available data and produced too many false positives as a result.

The next and final chapter of this thesis report will shortly summarize this report, draw conclusions and present the future work.

6

Conclusion

This thesis has described research on the automation of the performance evaluation of vehicle routing heuristics. Testing the performance of a heuristic is hard to do. There is a trade off between the quality of the solution and the time spend to obtain that solution. What makes it even harder is that the solution quality consist of several measurements. As such, the performance evaluation is often done by a human expert. Humans are not only non-deterministic in their evaluation, they also do not do it as regularly as a computer would. Furthermore, it is hard to judge the performance of a test run by comparing the results of multiple tests across multiple measures to each other. The research question of this thesis is: "How can we determine a performance measure that correctly represents the trade off between quality and runtime in vehicle routing heuristics?".

The literature survey revealed that a substantial amount of research has been done on performance evaluation, but not so much on performance evaluation of heuristics. The performance evaluation of heuristics was often simple in that it focused on a single measure, rather than evaluating multiple. The survey lead to the performance profiles, a method that is used to evaluate the performance of exact optimization algorithms. The performance profile is used in the classifier developed during this thesis.

This thesis has made two contributions and tested both on real data of a large company. The first contribution is a classifier that is able to detect regressions in the continuous development process of vehicle routing heuristics software. This classifier is based on the performance profiles, which were found in the literature. A performance profile is the cumulative distribution function such that the performance ratio of a solver is within a factor τ of the best known possible ratio. To create this function, a performance measure is needed, which will be described in the next paragraph. The original paper on performance profiles claimed that they reflected all major performance characteristics of a solver. Therefore, anomalies in the performance profiles indicate anomalous performance behaviour. The detection of these anomalies is done with a clustering algorithm.

As a second contribution, this thesis has researched performance measures to use in the earlier described classifier. The three possible measures are the area under the chart (area), the quality at the same time (quality) and the maximum difference in cost (difference). Theoretically, the quality is concrete and realistic but only reflects a single point in time. The area assesses the average quality over time but not specifically the end result. Finally the difference should work well when there are large, temporary differences. Two more variables were introduced. First, the evaluation point, for which the options t_{max} , $t_{max}/2$ and $t_{max}/10$ were chosen. Secondly, the end times of the longest and the shortest builds will be used for evaluation. This resulted in 18 measure configurations to be tested through experimentation.

The experimentation has tested all 18 possible measure configurations on the data provided by ORTEC. The measure configurations were rated by their accuracy to classify test runs as anomalous or regular. The best performing ones were the AREA_LONG_END, QUALITY_LONG_END and QUALITY_SHORT_END, with an accuracy of roughly 80%. These configurations did well because of their expected ability to detect anomalies and because they considered most of the data, in contrast to other configurations. The START and MIDDLE configurations were useful in some situations, but at the cost of introducing additional false classifications. The wide range of values for the difference measure was an issue with the current implementation, but

could possibly be fixed by somehow scaling it.

To recap, the research question is: "How can we determine a performance measure that correctly represents the trade off between quality and runtime in vehicle routing heuristics?". This thesis has described the process of selecting the appropriate measures for evaluating the performance of vehicle routing heuristics. This lead to three best performing measure configurations with an accuracy of roughly 80%. Although these measure configurations are best for the ORTEC setup, different measure configurations could be best for a different setup. The same methodology and evaluation approach described in this thesis could be applied to find which are best or to test other alternative measures to evaluate the performance.

6.1. Future work

As Chapter 5 has described, there were some issues with the current implementation of the classifier. Since there was a limited amount of time available for this project the following items were not explored. This section will focus on the possible future work regarding the research of this thesis.

Classifier in different setups

First, it would be interesting to investigate how the research of this thesis holds up in a different setup. Three measure configurations have shown an accuracy of roughly 80% on the data provided by ORTEC. This investigation could research whether the system works as well in the setup of a different system or with a different heuristic type than vehicle routing.

Improving the difference measure

The first recommendation is to improve the difference measure. The idea is clear and has potential, but in its current implementation it is lackluster. The biggest issue is the large range of values that this measure can attain, which makes the comparison hard. Perhaps if the range of values were to be reduced by scaling it to the maximum attained cost function.

Combination of measure configurations

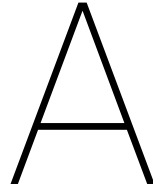
The second recommendations is to combine the results of different measure configuration into a single judgement. Each measure configuration produces a binary value for each test run that indicates whether it is anomalous or regular. It would require a decision rule to combine the results of multiple configurations in a single judgement. A simple example would be to take the best three configurations and mark a test run as anomalous when 2 out of the 3 mark it as anomalous. Taking it even further, by annotating all test runs in the current data set a supervised learning method could be used. This method could try out different combinations and determine which one would be the best one. This supervised method would only work on that data set, since it requires annotation of anomalous test runs.

Improving robustness against varying amount of test cases

The final recommendation concerns improving the robustness of the classifier in situations where the amount of test cases varies. The current implementation has the issue that two test runs with a different amount of test cases result in a large distance between their performance profiles, regardless of the actual results of those test cases. This is not desired behaviour, but there is an argument to be made that it is not that bad. Comparing two test runs on only a small set of cases is error-prone. In general, more test cases means more reliable results. Considering this, simply flagging a build as anomalous because of this might actually be a good thing. Alternatively, instead of calculating the distance between the performance profiles, a threshold could be used for when they reach 0.9 or 0.99 for instance. The simplest option might be to ensure all cases have roughly the same amount of tests. This could be ensured by simply ignoring all test runs with a significantly different amount of cases.

Finally, the people at ORTEC were pleased with the end results of this thesis and plan to implement it in their real world system. It has given them a useful tool to evaluate the performance of their algorithms, which had to be done manually before. The results were promising and the tool that was developed can be used to gain insights in why test runs are marked as anomalous. To conclude, this thesis has shown a method to automate the performance analysis of vehicle routing algorithms. The three best measure configurations showed a accuracy of roughly 80%. Combined with the fact that the evaluation is now automated, instead of

done manually, this is a big improvement to the performance evaluation process.

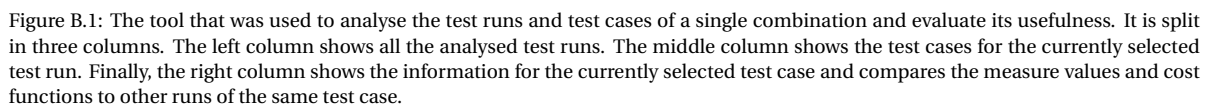


Evaluation sheet

	A	B	C	D	E	F
1	Changelist	966745				
2	Date	14-4-2017				
3						
4	# non-anomalous	8	47,06%			
5	# anomalous	9	52,94%			
6	# total	17				
7						
8		Case name	Anomalous (measure)	Anomalous (KPI chart)	Severity of anomaly (0 -5)	Comment
9	Anomalous case 1	CVRS_distribution_2	Yes a bit	No	0	
10	Anomalous case 2	CVRS_distribution_3	Yes a bit	Yes a bit	2	
11	Anomalous case 3	RefillSupermarket_Case500	Yes	Yes	4	
12	Anomalous case 4	RefillSupermarket_Case1000_2	Yes	Yes	5	
13	Anomalous case 5	RefillSupermarket_Case2000	Yes	Yes	5	
14	Anomalous case 6	LongTripsWithLegislation_Case1000	Yes	Yes	5	
15	Anomalous case 7	LongTripsWithLegislation_Case2000	Yes	Yes	5	
16	Anomalous case 8	CVRS_distribution_2_converted	No	No	0	Only 3 runs
17	Anomalous case 9	HomeDeliveryEnglishSupermarket_Case1400	Yes	Yes	4	
18	Anomalous case 10					
19	Anomalous case 11					
20	Anomalous case 12					
21	Anomalous case 13					
22	Anomalous case 14					
23	Anomalous case 15					
24						
25	Regular case 1	CVRS_distribution_1	No	No	0	
26	Regular case 2	CVRS_distribution_1_converted	No	No	0	Only 3 runs
27	Regular case 3					
28						
29	Issues	Cases with only 3 runs				
30						
31	Conclusion	Definetly an anomaly				
32						

Figure A.1: Example of the evaluation form used during the experiments. This sheet inspects a single test run. All anomalous test cases are inspected and some non-anomalous cases are checked. Based on the aggregation of whether the judgement was agreeable, this test run is marked as a true/false positive/negative.

Tool images



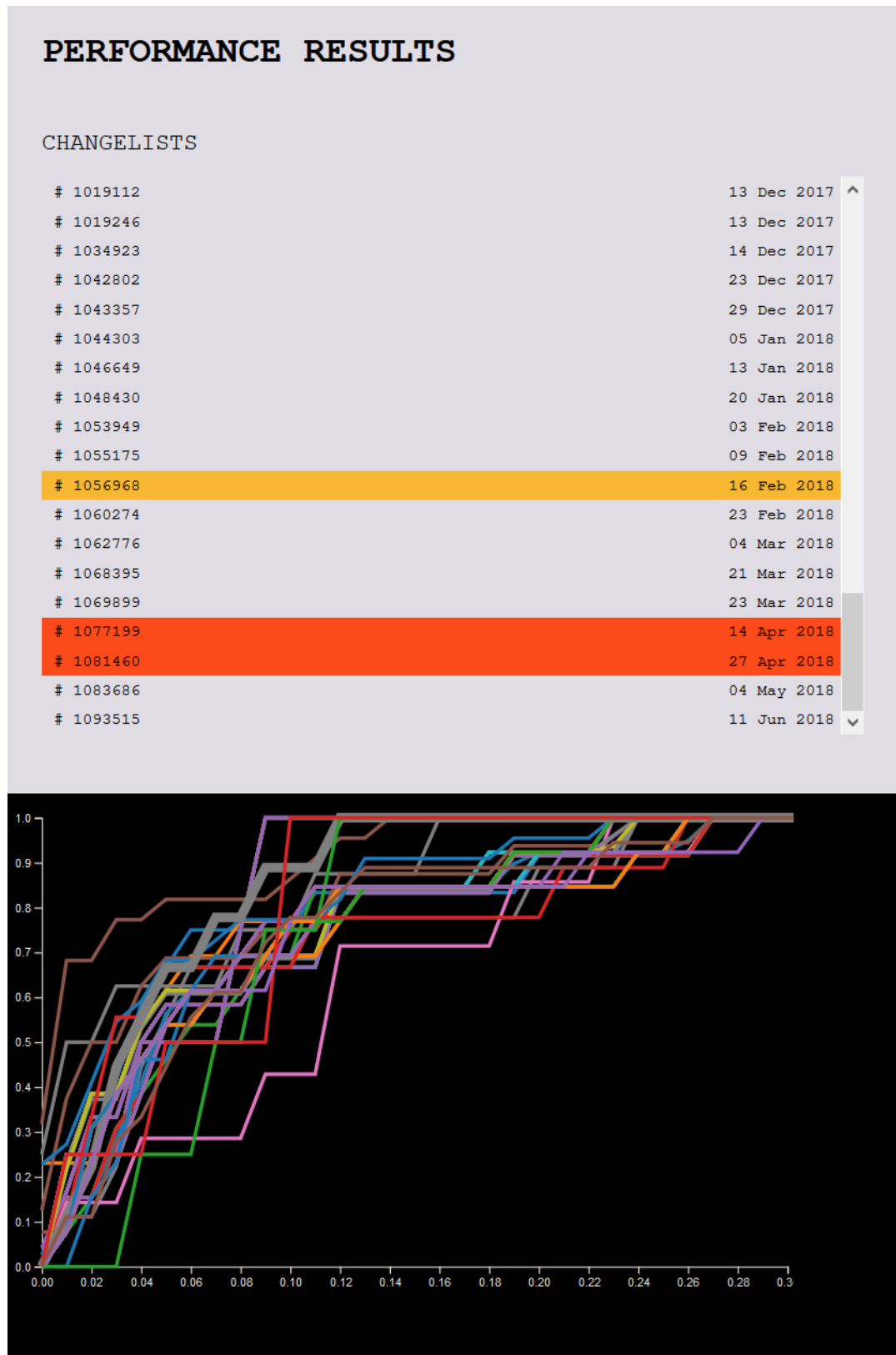
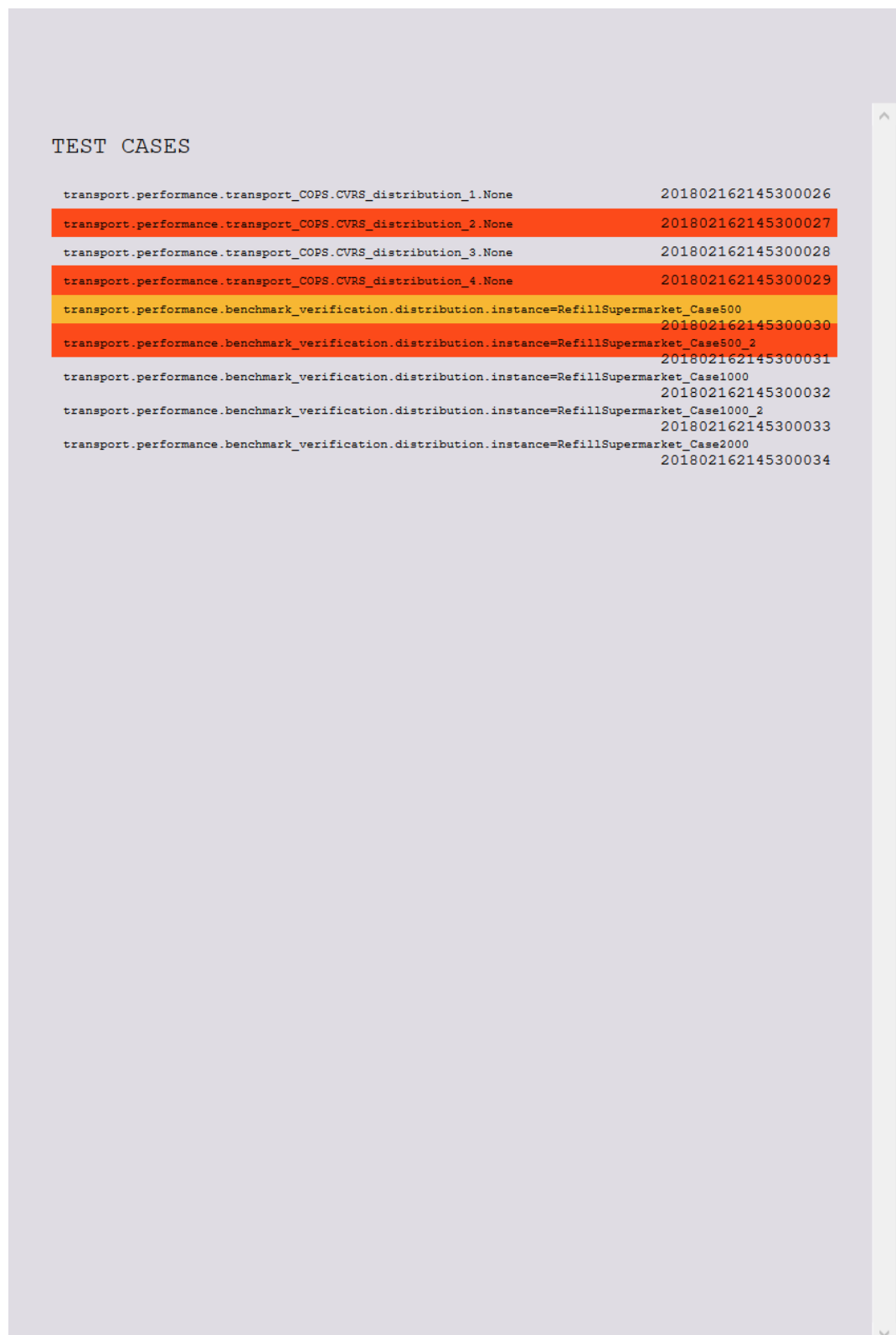


Figure B.2: Image of the tool, zoomed in on the changelist section. On the top are all the analysed test runs. The red ones are anomalous, the yellow one is currently selected. On the bottom is a plot of all performance profiles of these test runs.



transport.performance.transport_COPS.CVRS_distribution_1.None	201802162145300026
transport.performance.transport_COPS.CVRS_distribution_2.None	201802162145300027
transport.performance.transport_COPS.CVRS_distribution_3.None	201802162145300028
transport.performance.transport_COPS.CVRS_distribution_4.None	201802162145300029
transport.performance.benchmark_verification.distribution.instance=RefillSupermarket_Case500	201802162145300030
transport.performance.benchmark_verification.distribution.instance=RefillSupermarket_Case500_2	201802162145300031
transport.performance.benchmark_verification.distribution.instance=RefillSupermarket_Case1000	201802162145300032
transport.performance.benchmark_verification.distribution.instance=RefillSupermarket_Case1000_2	201802162145300033
transport.performance.benchmark_verification.distribution.instance=RefillSupermarket_Case2000	201802162145300034

Figure B.3: Image of the tool, zoomed in on the test cases section. These are all the test cases for the currently selected test run. The red ones are anomalous, the yellow one is currently selected. On the left is the name of the test case and on the right the date.

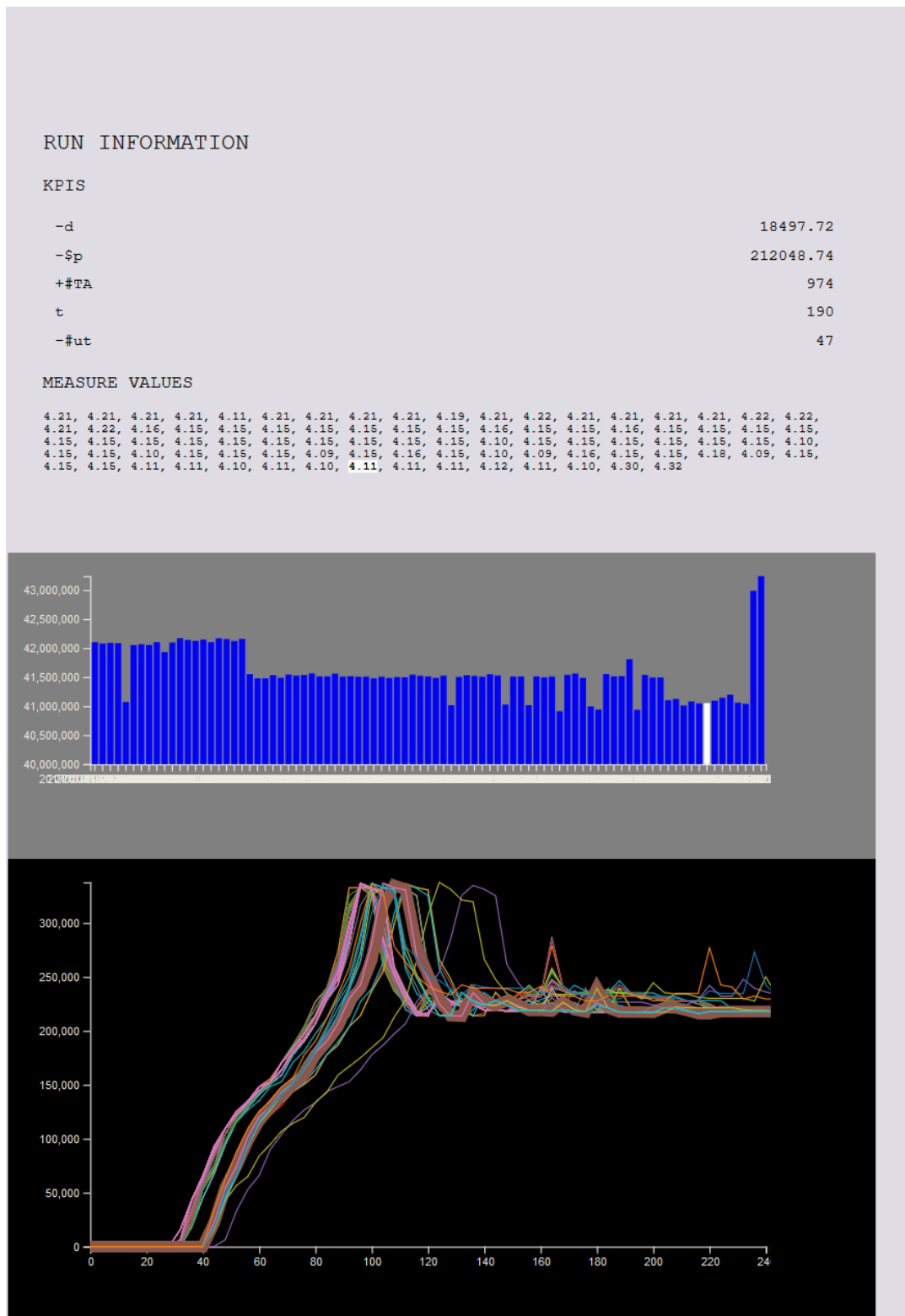


Figure B.4: Image of the tool, zoomed in on the run information section. This is the available information for the currently selected test case of the currently selected test run. The measure values are plotted to compare them to other test runs. On the bottom is a plot of the cost functions of this test case.

Bibliography

- [1] S. Agrawal and J. Agrawal. Survey on anomaly detection using data mining techniques. In *Procedia Computer Science*, 60, 2015.
- [2] M. Ahmed, A.N. Mahmood, and J. Hu. A survey of network anomaly detection techniques. In *Journal of Network and Computer Applications*, 60, 2016.
- [3] T.M. Ahmed, C.P. Bezemer, T.H. Chen, A.E. Hassan, and W. Shang. Studying the effectiveness of application performance management (apm) tools for detecting performance regressions for web applications: an experience report. In *Proceedings of the 13th International Conference on Mining Software Repositories*, 2016.
- [4] D. Arthur and S. Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, 2007.
- [5] V. Beiranvand, W. Hare, and Y. Lucet. Best practices for comparing optimization algorithms. In *Optimization and Engineering*, 18(4), 2017.
- [6] A. Bergel, F. Banados, R. Robbes, and D. Röthlisberger. Spy: A flexible code profiling framework. In *Computer Languages, Systems & Structures*, 38(1), 2012.
- [7] J. Berger and M. Barkaoui. A hybrid genetic algorithm for the capacitated vehicle routing problem. In *Genetic and evolutionary computation conference*, 2003.
- [8] J. Berger and M. Barkaoui. A parallel hybrid genetic algorithm for the vehicle routing problem with time windows. In *Computers & operations research*, 31(12), 2004.
- [9] T. Berthold. Measuring the impact of primal heuristics. In *Operations Research Letters*, 41(6), 2013.
- [10] W. Binder. Portable and accurate sampling profiling for java. In *Software: Practice and Experience*, 36(6), 2006.
- [11] K. Braekers, K. Ramaekers, and I. Van Nieuwenhuysse. The vehicle routing problem: State of the art classification and review. In *Computers & Industrial Engineering*, 49, 2016.
- [12] A. Brunnert and H. Krcmar. Detecting performance change in enterprise application versions using resource profiles. In *Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools*, 2014.
- [13] M.R. Bussieck, S.P. Dirkse, and S. Vigerske. Paver 2.0: An open source environment for automated performance analysis of benchmarking data. In *Journal of Global Optimization*, 59(2-3), 2014.
- [14] S. Byma and J.R. Larus. Detailed heap profiling. In *Proceedings of the 2018 ACM SIGPLAN International Symposium on Memory Management*, 2018.
- [15] J. Carlsson, D. Ge, A. Subramaniam, A. Wu, and Y. Ye. Solving min-max multi-depot vehicle routing problem. In *Lectures on global optimization*, 55, 2009.
- [16] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. In *ACM computing surveys (CSUR)*, 41(3), 2009.
- [17] L. Cherkasova, K. Ozonat, N. Mi, J. Symons, and E. Smirni. Anomaly? application change? or workload change? towards automated detection of application performance anomaly and change. In *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*, 2008.
- [18] E.D. Dolan and J.J. Moré. Benchmarking optimization software with performance profiles. In *Mathematical programming*, 91(2), 2002.

- [19] J. Enes, R.R. Expósito, and J. Touriño. Bdwatchdog: Real-time monitoring and profiling of big data applications and frameworks. In *Future Generation Computer Systems*, 87, 2018.
- [20] K.C. Foo, Z.M.J. Jiang, B. Adams, A.E Hassan, Y. Zou, and P. Flora. An industrial case study on the automated detection of performance regressions in heterogeneous environments. In *Proceedings of the 37th International Conference on Software Engineering*, 2, 2015.
- [21] S. Ghaith, M. Wang, P. Perry, Z.M. Jiang, P. O’Sullivan, and J. Murphy. Anomaly detection in performance regression testing by transaction profile estimation. In *Software Testing, Verification and Reliability*, 26(1), 2016.
- [22] M. Goldstein and S. Uchida. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. In *PloS one*, 11(4), 2016.
- [23] N. Gould and J. Scott. A note on performance profiles for benchmarking software. In *ACM Transactions on Mathematical Software (TOMS)*, 43(2), 2016.
- [24] C. Goutte and E. Gaussier. A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. In *European Conference on Information Retrieval*, 2005.
- [25] M. Hubert, P.J. Rousseeuw, and P. Segaeert. Multivariate functional outlier detection. In *Statistical Methods & Applications*, 24(2), 2015.
- [26] F. Hutter, H.H. Hoos, and K. Leyton-Brown. Tradeoffs in the empirical evaluation of competing algorithm designs. In *Annals of Mathematics and Artificial Intelligence*, 60, volume 60, 2010.
- [27] Docker Inc. Docker. <https://www.docker.com/>, 2019. Accessed: 24-05-2019.
- [28] Jenkins. Jenkins. <https://jenkins.io/>, 2019. Accessed: 19-06-2019.
- [29] D.S. Johnson. A theoretician’s guide to the experimental analysis of algorithms. In *Data structures, near neighbor searches, and methodology: fifth and sixth DIMACS implementation challenges*, 59, 2002.
- [30] G. Kendall, R. Bai, J. Błazewicz, P. De Causmaecker, M. Gendreau, R. John, J. Li, B. McCollum, E. Pesch, R. Qu, et al. Good laboratory practice for optimization research. In *Journal of the Operational Research Society*, 67(4), 2016.
- [31] B. Kitchenham, P. Brereton, and D. Budgen. Mapping study completeness and reliability-a case study. In *16th International Conference on Evaluation & Assessment in Software Engineering (EASE 2012)*, 2012.
- [32] D.A. Kolb. Experiential learning: Experience as the source of learning and development. In *FT press*, 2014.
- [33] F. Langner and A. Andrzejak. Detecting software aging in a cloud computing framework by comparing development versions. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, 2013.
- [34] M. Lavallee, P.N. Robillard, and R. Mirsalari. Performing systematic literature reviews with novices: An iterative approach. In *IEEE Transactions on Education*, 57(3), 2014.
- [35] V. Lavrenko. K-means clustering: how it works. https://www.youtube.com/watch?v=_aWzGGNrcic, 2014. Accessed: 24-09-2019.
- [36] L. Li, S. Lessmann, and B. Baesens. Evaluating software defect prediction performance: An updated benchmarking study. In *arXiv preprint arXiv:1901.01726*, 2019.
- [37] J.J. Moré and S.M. Wild. Benchmarking derivative-free optimization algorithms. In *SIAM Journal on Optimization*, 20(1), 2009.
- [38] NEO: Networking and Emerging Optimization. Dynamic vehicle routing problem (dvrp). <http://neo.lcc.uma.es/dynamic/vrp.html>, 2009. Accessed: 24-09-2019.

- [39] T.H.D. Nguyen, B. Adams, Z.M. Jiang, A.E. Hassan, M. Nasser, and P. Flora. Automated detection of performance regressions using statistical process control techniques. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, 2012.
- [40] S. Rojas-Labanda and M. Stolpe. Benchmarking optimization solvers for structural topology optimization. In *Structural and Multidisciplinary Optimization*, 52(3), 2015.
- [41] W. Shang, A.E. Hassan, M. Nasser, and P. Flora. Automated detection of performance regressions using regression models on clustered performance counters. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, 2015.
- [42] T. Weise, R. Chiong, J. Lassig, K. Tang, S. Tsutsui, W. Chen, Z. Michalewicz, and X. Yao. Benchmarking optimization algorithms: An open source framework for the traveling salesman problem. In *IEEE Computational Intelligence Magazine*, 9(3), 2014.