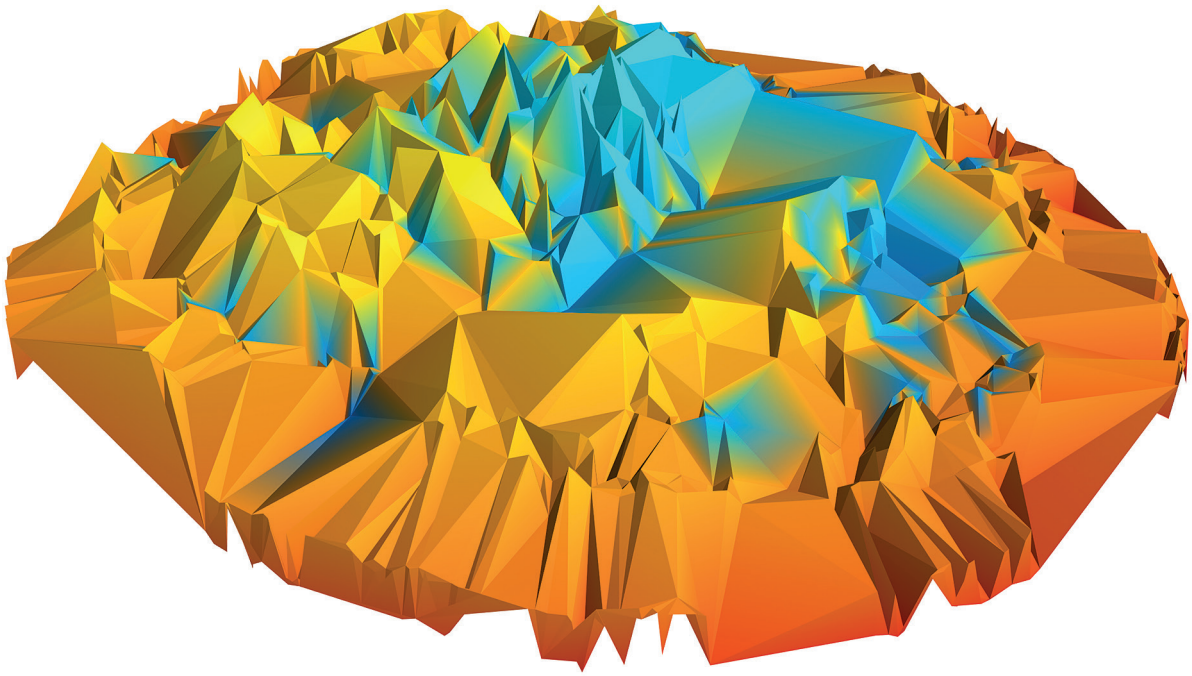
JYU DISSERTATIONS 113

Jussi Rasku

Toward Automatic Customization of Vehicle Routing Systems



UNIVERSITY OF JYVÄSKYLÄ
FACULTY OF INFORMATION TECHNOLOGY

JYU DISSERTATIONS 113

Jussi Rasku

Toward Automatic Customization of Vehicle Routing Systems

Esitetään Jyväskylän yliopiston informaatioteknologian tiedekunnan suostumuksella
julkisesti tarkastettavaksi yliopiston Agora-rakennuksen Lea Pulkkisen salissa
marraskuun 1. päivänä 2019 kello 12.

Academic dissertation to be publicly discussed, by permission of
the Faculty of Information Technology of the University of Jyväskylä,
in building Agora, Lea Pulkkinen hall, on November 1, 2019 at 12 o'clock noon.

JYVÄSKYLÄN YLIOPISTO
UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2019

Editors
Timo Männikkö
Faculty of Information Technology, University of Jyväskylä
Ville Korkiakangas
Open Science Centre, University of Jyväskylä

Permanent link to this publication: http://urn.fi/URN:ISBN:978-951-39-7826-6

ISBN 978-951-39-7826-6 (PDF)
URN:ISBN:978-951-39-7826-6
ISSN 2489-9003

ABSTRACT

Rasku, Jussi
Toward Automatic Customization of Vehicle Routing Systems
Jyväskylä: University of Jyväskylä, 2019, 98 p.(+included articles)
(JYU Dissertations
ISSN 2489-9003; 113)
ISBN 978-951-39-7826-6 (PDF)

This thesis was motivated by the desire to make the state-of-the-art vehicle routing problem models and algorithms more convenient for a non-expert to use. Currently, heavy customization is required whenever route optimization technology is adapted to solve new real-life routing problems. A critical part of this tailoring process involves choosing a suitable optimization algorithm and configuring its parameters; this requires developing a deep understanding of vehicle routing problems, their solution algorithms, and the software systems built around them. However, given that such information can be captured and represented as numerical feature values, machine learning can be used to find and exploit the patterns in the variation of algorithm performance.

This dissertation proposes a framework for automating the customization of different components and data transformations within a vehicle routing system. This is accompanied by a comprehensive set of empirical experiments that were conducted to verify the feasibility of the proposed approach. As such, this dissertation furthers our understanding of the vehicle routing problem instances, algorithms, and their search spaces. It also provides suggestions and evidence on how to effectively use the automatic algorithm configuration and algorithm selection techniques in an automated vehicle routing system customization context. The findings of this work indicate that meta-optimization is a promising approach that allows more convenient and effective use of existing tools and techniques for solving vehicle routing problems.

Overall, logistics plays a major role in modern society, which has made vehicle route optimization an important application of combinatorial optimization. The approach developed in this dissertation can reduce the friction in customization and deployment of optimization systems, thus allowing moving toward more economical, cost-effective, and environmentally friendly road transportation.

Keywords: vehicle routing problem, meta-optimization, automatic algorithm configuration, algorithm selection, feature selection

TIIVISTELMÄ (ABSTRACT IN FINNISH)

Rasku, Jussi
Kohti kuljetusten optimointijärjestelmien automatisoitua räätälöintiä
Jyväskylä: University of Jyväskylä, 2019, 98 p.(+artikkelit)
(JYU Dissertations
ISSN 2489-9003; 113)
ISBN 978-951-39-7826-6 (PDF)

Väitöskirjan taustalla oli halu tehdä huipputeknisistä ajoneuvojen reititysmalleista ja niiden ratkaisemiseen käytetyistä algoritmeista entistä helpompia käyttää. Nykytilanne vaatii merkittävän määrän asiantuntemusta vaativaa räätälöintiä aina, kun optimointitekniikkaa mukautetaan uusiin tosielämän reititystehtäviin. Kriittinen osa räätälöintiprosessia on sopivan optimointialgoritmin valitseminen ja sen parametrien konfigurointi, mutta tämä edellyttää ajoneuvojen reititystehtävien, niiden ratkaisualgoritmien ja niiden ympärille rakennettujen ohjelmistojärjestelmien syvällistä ymmärtämistä. Kuitenkin, tämä on automatisoitavissa. Koneoppimista voidaan käyttää löytämään ja hyödyntämään säännönmukaisuuksia algoritmien suorituskykyvaihtelussa, edellyttäen että niiden konfigurointiin liittyvät olennaiset seikat voidaan ilmaista numeerisina arvoina.

Tämä väitöskirja esittelee mallin, joka mahdollistaa kuljetusten optimointijärjestelmien räätälöinnin ja käyttöönoton osittaisen automatisoinnin. Mallin käyttökelpoisuuden osoittavat kattavat empiiriset kokeet, jotka suoritettiin ehdotetun lähestymistavan toteutettavuuden varmistamiseksi. Väitöskirja lisää ymmärrystämme ajoneuvojen reititystehtävistä, reititysalgoritmeista ja niiden hakuavaruuksista. Se kertoo myös, miten moderneja automaattisia algoritmien konfigurointi- ja valintatekniikoita voidaan käyttää tehostamaan kuljetusten optimointijärjestelmien räätälöintiä. Tulokset osoittavat, että metaoptimointi on lupaava lähestymistapa mahdollistamaan olemassa olevien työkalujen ja tekniikoiden helpomman ja tehokkaamman käytön ajoneuvojen reititystehtäviä ratkaistaessa.

Huomionarvoista on, että logistiikan merkittävä rooli nykyaikaisen yhteiskunnan tehokkaan toiminnan mahdollistajana on tehnyt ajoneuvojen reitityksestä tärkeän optimoinnin sovelluksen. Tässä väitöskirjassa kehitetty lähestymistapa on lupaava keino optimointijärjestelmien räätälöinnin ja käyttöönoton nopeuttamiseksi. Täten se osaltaan mahdollistaa entistä taloudellisemman, kustannustehokkaamman ja ympäristöystävällisemmän tieliikenneteen.

Avainsanat: ajoneuvojen reititystehtävä, metaoptimointi, automaattinen algoritmien konfigurointi, algoritmien valinta, piirteiden valinta

Author

Jussi Rasku
Faculty of Information Technology
University of Jyväskylä
Finland

Supervisors

Professor Tommi Kärkkäinen
Faculty of Information Technology
University of Jyväskylä
Finland

Priv.-Doz. Nysret Musliu
Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling
Institute of Logic and Computation, DBAI
Technische Universität Wien
Austria

Reviewers

Professor Martin Josef Geiger
Logistics Management Department
Helmut-Schmidt-University / University of the Federal Armed Forces Hamburg
Germany

Professor Kenneth Sörensen
ANT/OR – Operations Research Group
Department of Engineering Management
University of Antwerp
Belgium

Opponent

Associate Professor Luca Di Gaspero
Dipartimento di Matematica e Informatica
Università degli Studi di Udine
Italy

ACKNOWLEDGEMENTS

Looking back, the chain of events that led to this PhD dissertation has been fortuitous. I was introduced to the work carried out by the Research group on computational logistics when I was finishing my master's thesis. This piqued my interest and led me on this interesting path. It has allowed me to follow my curiosity and gain skills that will surely benefit me greatly in the future.

I'm deeply grateful to my supervisors, Professor Tommi Kärkkäinen and Privatdozent Nysret Musliu, who have helped me through the journey. Their unyielding positivity, insightful comments, and deep experience on the many topics along my chosen path have been instrumental in me finally reaching the destination. I'm privileged to have had the chance to work with you both. Also, I would like to express great appreciation to Professors Kenneth Sörensen and Martin Josef Geiger, who reviewed my dissertation, and to Associate Professor Luca Di Gaspero, who generously agreed to be my opponent at the defense.

I also want to use this opportunity to thank the many people who have helped and guided me through this endeavor. Without Professor Olli Bräysy and his efforts in founding the research group on computational logistics, I would not have had the chance to become familiarized with the topic in the first place. It was such a welcoming group of talented individuals and working with them through the years was a joy: Thank you Tuukka Puranen, Pekka Hotokka, Antoine Kalmbach, Jouko Nieminen, Joni Brigatti, Jukka Kemppainen, and Antti Hallamäki. Especially during the last few years, the help, understanding, and support from Tuukka and Jouko have been invaluable to me.

This work would not have been possible without the support of several institutions and organizations. Early in my PhD, our research group was funded by two Tekes projects, but the majority of the financial support for my work came from the COMAS Graduate School of the University of Jyväskylä. The last years of my PhD has been supported by grants from EPKY and the Dean of the Faculty of Information Technology. Furthermore, a Microsoft Azure4Research grant greatly accelerated the experimental part of my work. The role of Professor Pekka Neittaanmäki has been significant in organizing the funding for my PhD, and I greatly appreciate his support. Also, I'd like to express my gratitude towards University Consortium of Seinäjoki, which has offered me not only an office, but an academic community to work in. I'd like to thank all of my fellow researchers and staff there for the inspiration and interesting discussions.

Last but not least, I'm deeply grateful for the support from my family, relatives, and friends. Most importantly from you, Kirsi and Pihla, who have been with me thought the best and the worst. Although it has not always been easy, a large part of me making this accomplishment is due to your patience, support, and understanding. This is to you.

Seinäjoki, September 2019
Jussi Rasku

LIST OF FIGURES

FIGURE 1 The main topics discussed in this dissertation. 18
FIGURE 2 A CVRP problem instance and its optimal solution. 24
FIGURE 3 Trends in VRP research. 25
FIGURE 4 Modules of a VRS. 36
FIGURE 5 Clustering with three clusters and two inputs x_1 and x_2 45
FIGURE 6 Three-label classification with two inputs x_1 and x_2 47
FIGURE 7 Algorithm configuration workflow. 52
FIGURE 8 Model for algorithm selection. 54
FIGURE 9 The included papers in relation to data flow through VRS 79

CONTENTS

ABSTRACT
TIIVISTELMÄ (ABSTRACT IN FINNISH)
ACKNOWLEDGEMENTS
LIST OF FIGURES
CONTENTS
LIST OF INCLUDED ARTICLES

1 INTRODUCTION 13
1.1 Background and motivation 14
1.2 Positioning the study 16
1.3 Research problem and objective 19
1.4 Main contributions 21
1.5 Structure of the dissertation 22

2 VEHICLE ROUTING PROBLEM 23
2.1 Variants 25
2.2 Solution methods 26
2.2.1 Exact methods 27
2.2.2 Classical heuristics 28
2.2.3 Metaheuristics 30

3 VEHICLE ROUTING SYSTEMS 33
3.1 Functionality and typical modules 35
3.2 Customization 39

4 MACHINE LEARNING 43
4.1 Clustering 44
4.2 Dimension reduction 46
4.3 Classification 47
4.4 Feature selection 47

5 META-OPTIMIZATION 50
5.1 Automatic algorithm configuration 51
5.2 Algorithm selection 54
5.3 Meta-features 56

6 SUMMARY OF THE PUBLICATIONS 59
6.1 Paper PI: Automatic Customization Framework for Efficient Vehicle Routing System Deployment 59
6.2 Paper PII: Solution Space Visualization as a Tool for Vehicle Routing Algorithm Development 61
6.3 Paper PIII: Feature Extractors for Describing Vehicle Routing Problem Instances 62

6.4 Paper PIV : Meta-Survey and Implementations of Classical Capacitated Vehicle Routing Heuristics with Reproduced Results 64
6.5 Paper PV : Feature and Algorithm Selection for Capacitated Vehicle Routing Problems 67
6.6 Paper PVI: On Automatic Algorithm Configuration of Vehicle Routing Problem Solvers 69
6.7 Author's contributions 72

7 DISCUSSION 75
7.1 Discussion of the results 75
7.2 Conclusions and future research directions 80

YHTEENVETO (SUMMARY IN FINNISH) 83

REFERENCES 84

INCLUDED ARTICLES

LIST OF INCLUDED ARTICLES

PI Jussi Rasku, Tuukka Puranen, Antoine Kalmbach, Tommi Kärkkäinen. Automatic Customization Framework for Efficient Vehicle Routing System Deployment. *Proceedings of ECCOMAS Thematic Conference : CM3 - Computational Multi Physics, Multi Scales and Multi Big Data in Transport Modeling, Simulation and Optimization*, Computational Methods and Models for Transport - New Challenges for the Greening of Transport Systems, Springer, 2016.

PII Jussi Rasku, Tommi Kärkkäinen, Pekka Hotokka. Solution Space Visualization as a Tool for Vehicle Routing Algorithm Development. *Proceedings of the Finnish Operations Research Society 40th Anniversary Workshop (FORS40)*, 2013.

PIII Jussi Rasku, Tommi Kärkkäinen, Nysret Musliu. Feature Extractors for Describing Vehicle Routing Problem Instances. *Proceedings of 5th Student Conference on Operational Research (SCOR 2016)*, OpenAccess Series in Informatics (OASIcs), 2016.

PIV Jussi Rasku, Tommi Kärkkäinen, Nysret Musliu. Meta-Survey and Implementations of Classical Capacitated Vehicle Routing Heuristics with Reproduced Results. *Manuscript*, 2019.

PV Jussi Rasku, Nysret Musliu, Tommi Kärkkäinen. Feature and Algorithm Selection for Capacitated Vehicle Routing Problems. *In Proceedings of European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2019)*, 2019.

PVI Jussi Rasku, Nysret Musliu, Tommi Kärkkäinen. On Automatic Algorithm Configuration of Vehicle Routing Problem Solvers. *Journal on Vehicle Routing Algorithms*, 2019.

1 INTRODUCTION

This dissertation seeks to increase the level of self-adaptivity in vehicle routing systems by applying meta-optimization. Vehicle routing systems are specialized decision support systems capable of solving the vehicle routing problem, and the proposed approach allows for their automatic configuration and customization. Designing models and algorithms for the vehicle routing problem is usually carried out in the research fields of management science and operations research. These problems originate from the transportation industry and the research has always had a practical, applied focus (Toth and Vigo, 2014). The research on solving these problems has advanced concurrently with increasing computational resources, and computation has grown to be an integral part of operations research. This has allowed the complexity of problems, models, and solutions to increase. These developments, and the practical importance of the applications, have led to a great number of proposed VRP model variants and the algorithms needed to solve them (Golden et al., 2008; Eksioglu et al., 2009; Toth and Vigo, 2014). Thus, the sheer volume of research on the topic can be overwhelming. Instead of proposing yet another new VRP variant or algorithm, the overarching theme of this work is finding ways to more effectively utilize the existing knowledge on how to *model* and *solve* these problems.

The importance of software engineering should also be recognized here. Despite the strong connection between operations research and computer science, software engineering topics have received relatively little attention in the operations research literature. This is surprising because managing the complexity and variability of the models, problems, and algorithms is a serious challenge in building, customizing, and deploying these software solutions (Drexl, 2012; de la Banda et al., 2014). In fact, some of the major barriers for adopting route optimization technology are related to the significant and expensive customization effort required in deploying these systems (Partyka and Hall, 2014; Rincon-Garcia et al., 2018). In addition to challenges in integration and migration from the current systems (Neittaanmäki and Puranen, 2015), modeling the problem and tuning the optimization components is laborious and requires expensive, hard-to-find expertise (Sörensen et al., 2008; de la Banda et al., 2014). This can create practical

obstacles that prevent the latest innovative research from being disseminated for wider use. This dissertation addresses some of these challenges by introducing a framework and a workflow for automating parts of the vehicle routing system customization and deployment.

1.1 Background and motivation

The origins of the modern operations research can be traced back to the logistics of World War II (Fortun and Schweber, 1993). Since then, the term logistics has expanded to cover not only the management of storing and moving physical items, but also intangible resources such as time and data. By doing so, logistics has irreversibly shaped the formation of the modern world. Meanwhile, the logistics sector has become a huge industry. It is a key enabler in the efficient operation of virtually all economic sectors (Bräysy and Hasle, 2014; Ecorys et al., 2015). According to Schwemmer (2015), the logistics market size of EU-28 together with Norway and Switzerland was 1050 billion euros in 2016, out of which transportation generates 45% of the share (473 billion euros). Overall, transportation contributed to around 5% of the EU-30 region gross value added in 2016 (European Commission, 2018). Also, the scale of operations is growing rapidly: from 1995 to 2005, there was a 38% increase in the number of vehicles used for road-based goods transportation in the EU-27 (Huggins et al., 2009).

Because of its impact, cost effective logistics is crucial to the productivity and competitiveness of the service, industry, and public sectors (Bräysy and Hasle, 2014). This is especially true in sparsely populated Finland, where the transportation distances are long and where a large portion of the logistics is done using roads. In 2017, Finland had the seventh largest road freight transport per capita of the EU-28 countries at 5,081 tonne-kilometers per person (Eurostat, 2019). According to the logistics report of Solakivi et al. (2017), logistics operations accounted for around 13.9% of the turnover of Finnish manufacturing and trade in 2015. Hence, the cost of logistics is around 37 billion euros, which is around 11.2% of the Finnish GDP. Here, the proportion of transportation costs of the total logistics costs is 38%.

Technological innovation seems to be the driving force behind the digitalization of the transportation sector, at least in Finland (Leviäkangas, 2016). Advances with the internet of things (IoT), computation, smartphone technology, and electric and autonomous vehicles are opening up new possibilities for fleet monitoring and control (Bräysy and Hasle, 2014; Speranza, 2018). Furthermore, we are currently living in the age of increasing online retail, global supply chains, and home deliveries, which is making the logistics even more dynamic, hectic, and difficult to manage (Crainic et al., 2009; Hoff et al., 2010; Speranza, 2018). There is also climate change, which has recently become a powerful driver of effective logistics (Hoff et al., 2010). To make the situation even more challenging for transportation companies, the market is characterized by low profitability

(Rincon-Garcia et al., 2018) and tightening competition (Partyka and Hall, 2014). The requirements for efficiency are not limited to transportation business: route planning is done in diverse sectors (Drexl, 2012). For example, there are many public sector operations involving transportation of goods and people that have a significant potential for cost savings (Bräysy et al., 2009).

Hence, there are very strong incentives to improve the utilization of existing transportation resources on all sectors of the economy (Ecorys et al., 2015). Meanwhile, transportation operators already struggle under high demands and have trouble obtaining efficiency, customer service, timeliness, reactivity, and cost savings (Hoff et al., 2010). To answer these challenges, the operators need to constantly innovate and improve the management and planning of their logistic systems (Labadie et al., 2016). Unfortunately, the ever-increasing transportation volumes and the accruing complexity because of the growing number of operational rules and limitations has made many distribution planning tasks unreasonably difficult for a human planner to do (Gacias et al., 2012, p. 805). Hence, it is not surprising that there is a growing interest toward deploying computerized vehicle route optimization systems (Carić et al., 2008).

The advantages of automated route planning are well-known: it can considerably reduce the planning time, total travel distance, and overall resource usage by making the routes more economical. In fact, industry experience indicates that the total travel distance can typically be reduced by 5–30% compared with manual planning (Hasle and Kloster, 2007). In addition, there are a number of other associated benefits: the plans can be illustrated visually; the operational statistics can be calculated and stored; and integration, interoperability, and communication between other systems becomes possible. On the planning side, the task becomes quicker and less error-prone to do, routes can be designed to be more robust to disruptive changes, and the optimization can be used to balance between cost, service quality, or other key performance indicators (Drexl, 2012).

However, there remain many challenges in constructing such systems (Partyka and Hall, 2014), perhaps the most critical one being that building them is expensive and time-consuming (Maturana et al., 2004). The situation can make the customization and deployment of these systems prohibitively costly, especially for small- and medium-sized enterprises. Furthermore, no two transportation companies or sectors are alike, and consequentially, the family of different routing problems has become heterogeneous (see, e.g., Vidal et al., 2013b; Bräysy and Hasle, 2014). To address these challenges, academics and system providers need to use a generic approach to address the heterogeneity of the modeling needs. Simultaneously, expectations are increasing: the optimization technology is expected to be user-friendly, faster, more scalable and reliable, and have a strong and expressive modeling layer (Bräysy and Hasle, 2014). The last requirement means that the mathematical models are expected to consider a growing number of properties and features of real-life logistics, which leads to so-called "rich" models (Lahyani et al., 2015). Only these rich models can consider all the relevant rules, parameters, and variables of the specific transportation problem they are targeting. These expectations tend to increase the complexity of the route opti-

mization solutions. This accumulated complexity places a significant cognitive burden on the practitioners of operations research, and therefore, the models and algorithms are becoming difficult for even an expert to manage (de la Banda et al., 2014). The situation calls for the development of new tools and techniques that can help to unlock the benefits offered by advances in the underlying technology.

1.2 Positioning the study

The field of operations research seeks to find advanced analytical and mathematical methods that can be applied to support decision making in science, industry, and society. This work, which is positioned at the intersection between operations research, machine learning, and software engineering, concentrates on algorithms and software systems targeting the distribution function of logistics. This function generally considers the transportation of goods, people, and services from supply points to demand points. The transportation fleet may consist of trucks, vans, cars, or even bicycles. To complete the operational objectives they are required to do a set of tasks at a number of locations.

In the field of operations research, the *combinatorial optimization problem* (COP) for planning these transportation tasks is known as the *vehicle routing problem* (VRP). It involves determining which tasks should be assigned to each vehicle and in which order the tasks should be performed to minimize the objective function (e.g., the total cost of operations). The problem definition includes constraints that make sure that vehicles leave from and return to a central depot and that each customer is served only once by a single vehicle. In addition, the assignments must not violate side constraints such as the capacity of the vehicles in the classic capacitated vehicle routing problem (CVRP) or the service time windows in the vehicle routing problem with time windows (VRPTW). Lately, more complex "rich" models (Lahyani et al., 2015; Vidal et al., 2014) have been proposed. These rich vehicle routing problems can have a large number of customers and complex constraints which makes them notoriously hard to solve because of the combinatorial nature of the problem.

VRP optimization is an integral part of decision support systems (DSS) for transportation planning, where they are applied to increase the efficiency of the distribution function. In practice, such *vehicle routing systems* (VRS) are used to model and solve real-word route optimization problems. The capability to effectively solve these problems is central for staying competitive in transportation logistics (Bräysy and Hasle, 2014). The significant and practical importance of the topic has led operations researchers to propose a wide variety of optimization techniques to model and provide solutions to these problems (Speranza, 2018). Hence, VRP research has always had a strong applied connection, and a significant part of the literature involves reporting successful applications of the optimization technology in diverse sectors of the economy (e.g., Watson-Gandy and Foulds 1981, p. 76; Part III of Golden et al. 2008 and Toth and Vigo 2014).

The practical applicability of VRSs depends heavily on the suitability of the built-in algorithms on the problem that they are intended to solve. Fortunately, several decades of intensive research have produced a great number of mathematical programming, approximation, and heuristic approaches for solving them (see, e.g., Toth and Vigo, 2014). Many of these technologies have also been integrated into commercial software packages (Speranza, 2018). Still, applying these more theoretical contributions to practice is not always straightforward. A tailored algorithm tends to produce the best results, and, in the industry and academia alike, models and solution methods are hand-tuned by experts for each new problem to maximize the algorithm's performance (Coy et al., 2001). However, creating such a specialized solution for every new real-world routing case is prohibitively expensive (Sörensen et al., 2008; Rincon-Garcia et al., 2018). Hence, Sörensen et al. (2008) recognized that there exists a strong need for self-adaptive algorithms and automatic configuration, an opportunity also recognized earlier by Desrochers et al. (1999). Furthermore, Hoff et al. (2010) noted that because of technological progress, there is more information available to the decision maker and to the VRS vendor. This is because the amount of historical data, including customer and usage patterns, driving times, speeds, and other tracking data, are growing (Speranza, 2018). Hence, Hoff et al. identified a need to develop DSSs that could utilize such data to provide better and more realistic solutions. Similarly, Calleja et al. (2019) noted that there has been an explosion of data, and this has created a strong need to turn these data into actionable insight.

Machine learning is a tool for automatically extracting previously unknown patterns and potentially useful knowledge from large databases (Olafsson et al., 2008). Although combining machine learning and VRP research is not a new idea (see, e.g., Potvin et al., 1990), emerging technologies such as artificial intelligence and big data analytics have opened new, promising areas of research in operations research (Calleja et al., 2019). Recent increases in computational capacity, decreases in the cost of data storage, and advances in machine learning algorithms have opened new possibilities for applying data-driven techniques in solving VRPs.

Simultaneously, the number of VRP algorithms proposed over the past few decades has grown rapidly (Braekers et al., 2016). However, different optimization algorithms show different performance characteristics on different classes of the problem instances. This variation is because of their differing operating principles. The theoretical foundations for this phenomena can be found from the "No Free Lunch Theorem for Optimization" (Wolpert and Macready, 1997), which states that no single algorithm is superior to all others for all problems. Hence, it is important to consider the theorem when generalizing results and acknowledge that it applies also to self-adaptive algorithms (De Jong, 2007). However, the theorem also implies that, in practice, it is possible to fine-tune or find a good algorithm for a given set of problem instances.

Analogously to the machine learning topics meta-learning (Brazdil et al., 2009) and hyperparameter optimization (Bergstra et al., 2011), *meta-optimization* has received growing attention from the researchers working on combinatorial

optimization problems (Kotthoff, 2014). Meta-optimization involves using an optimization or machine learning model to optimize the behavior of a *target* optimization method (Battiti, 1989). Similar ideas have been independently proposed several times in several research fields, but recently, a concentrated effort on topics such as automatic algorithm configuration, algorithm selection, and hyper-heuristics (Calvet et al., 2017) have provided a significant amount of information on how to apply this approach effectively. Hence, while the poor generalization ability of the early VRP models and heuristics has traditionally been tackled using hybridization (Vidal et al., 2014) or with orthogonal solution methods (Arnold and Sörensen, 2019a), the use of automatic algorithm selection, algorithm configuration, and hyper-heuristics have steadily gained popularity (e.g., Nygard et al., 1990; Coy et al., 2001; Pellegrini and Birattari, 2007; Garrido and Riff, 2010).

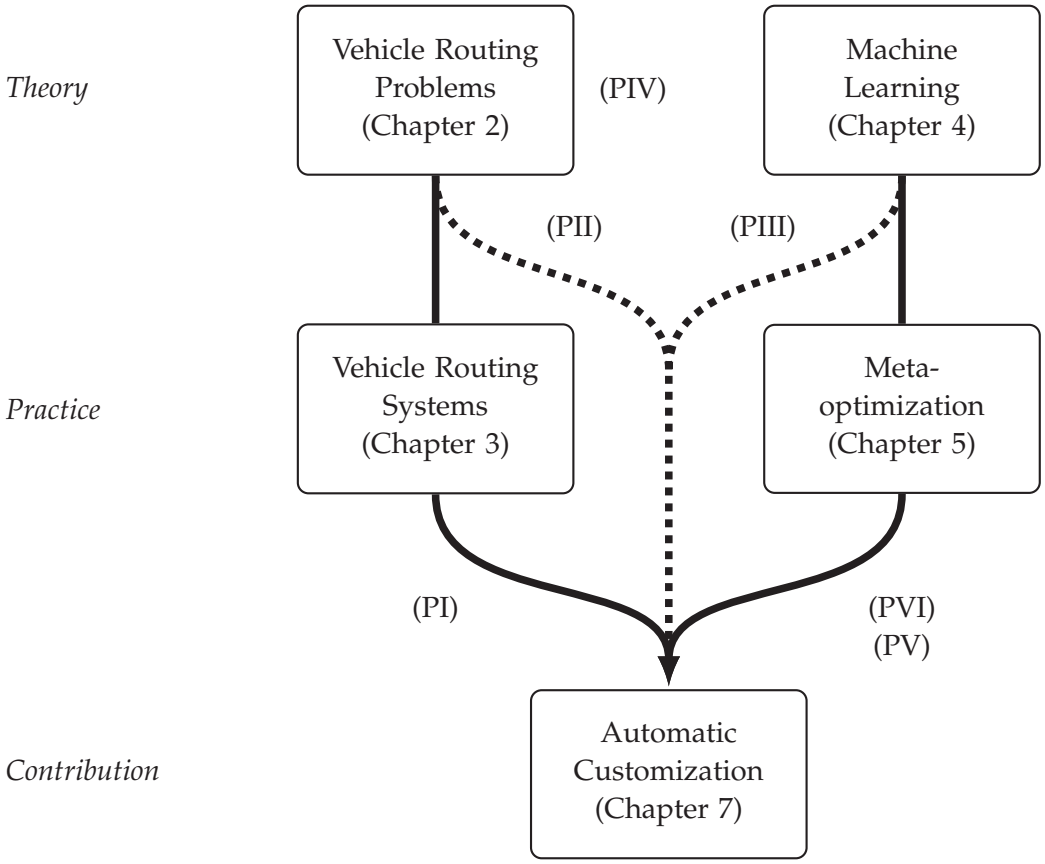


FIGURE 1 The main topics discussed in this dissertation.

To summarize, this dissertation builds on the fundamental research on vehicle routing problems and machine learning. In practice, the existing literature on modeling and solving vehicle routing problems has been used to create vehicle routing systems that target real-world transportation and distribution tasks, whereas machine learning techniques have provided theoretical foundations and tools for many meta-optimization techniques. Figure 1 illustrates how these topics are related. The premise of this dissertation is that combining the theory and practice allows for the modeling and solving of increasingly complex real-world problems.

1.3 Research problem and objective

The use of meta-optimization has not yet been comprehensively explored in the context of vehicle routing systems. In this dissertation, we study how the customization of vehicle routing systems can be accelerated via the use of machine learning and meta-optimization (Figure 1). Our main motivation was to make the state-of-the-art vehicle routing tools easier to use and deploy, for both experts and non-experts. To narrow down the scope even further, in this work, we concentrate on answering research questions related to usability and data-integration concerns and to the limited generalization ability, robustness, speed, and adaptivity of the existing solution methods. This requires us to do the following:

1. recognize the relevant components and transformations within a vehicle routing system susceptible to customization,
2. propose new tools and techniques for observing the behavior of VRP heuristics that can be used in developing new algorithms,
3. provide a numerical description of vehicle routing problem instances for machine learning, and
4. verify the applicability of existing automatic algorithm configuration and algorithm selection techniques in this domain.

There are multiple expected advantages with this approach to the VRS vendors and their customers. Currently, the tailoring and use of vehicle routing systems requires hard-to-gain expertise (Bräysy and Hasle, 2014), which is emphasized by the fact that novel methods are often more complex to use—even for expert practitioners (de la Banda et al., 2014). Hence, de la Banda et al. recognized the need for the simplification of combinatorial optimization technology. While their focus was on constraint programming solvers, the topic is also important in related domains. Birattari (2009) argued that time-consuming algorithm configuration can, and should, be substituted by an automated and replicable process, and Neittaanmäki and Puranen (2015) explicitly recognized the possibilities of automatic configuration and performance tuning of vehicle routing systems. With these techniques, the customization and deployment of a vehicle routing system is expected to become less labor intensive, without needing to sacrifice the algorithm performance. Clearly, this is beneficial also to the end user because the automatic adaptation of the algorithm parameters makes the software easier to use. Given that some of the required tailoring effort can be automated with the proposed methods, the time of the expensive consultants (Sörensen et al., 2008) can be better spent in helping the customer to define the objectives and constraints of their problem. This is important as this is a task that the customer often struggles with without outside help (Drexl, 2012). Also, the approach has the potential to address the aspects of reusability, customizability, and flexibility of vehicle routing systems. These topics have received surprisingly little attention despite their practical relevance (Derigs and Vogel, 2014a).

A system with a similar goal was envisioned by Desrochers et al. (1999), who made the observation that there already was a vast amount of information on how to solve VRPs in the literature, but modeling the problem and selecting a suitable algorithm for it required a significant amount of knowledge and expertise. They proposed a vocabulary for representing this knowledge and an inference scheme for manipulating the information. However, they did not offer a comprehensive solution, and building the knowledge base they proposed still required a large amount of manual work from the expert. Hence, this dissertation aims to complement their work and address this issue by proposing data-driven methods that can automatically adapt a vehicle routing system to the specific real-world problems with only minor involvement from the expert.

This dissertation also builds on the earlier work on vehicle routing systems by Puranen (2011, 2012), who proposed a metamodel for several VRP variants and a blueprint for a model-driven VRP DSS software product line architecture that could be customized to generate specialized software products for different end users. Hence, when the applications of the research presented here are considered, one should assume a similar modeling system that can express multiple important VRP variants, and a solver with a selection of generic and flexible heuristic algorithms that can be used to solve them. It should also be noted that Neittaanmäki and Puranen (2015) explicitly called for advances in automating the configuration of vehicle routing systems and techniques that could infer the necessary model features and constraints from the data. Hence, the aim of this dissertation is to realize a part of this vision.

In academia, the rigorous design of computational experiments is a requirement of high-quality empirical science (Barr et al., 1995). To facilitate reproducibility, authors should explicitly list and define the parameters of the tested algorithms, and the parameter values should be set systematically (Golden et al., 1998). Unfortunately, this is still not a standard procedure when proposing and testing VRP algorithms. Sörensen (2015) pointed out that "the scientific value of most papers on metaheuristics would increase considerably if the step of parameter-tuning is be done in a transparent way." They continued by noting that automatic algorithm configuration methods and guidelines to perform it are readily available, and the algorithm developers should begin to use this solid experimental methodology when testing their methods. Hence, the meta-optimization topics explored in this dissertation should also interest academics. Meta-optimization removes the effect the expertise of the researcher may have on the experimental results (Eggensperger et al., 2019): If instead of relying on the tedious and subjective manual configuration work all algorithms are tuned by an independent system, the true potential of the different methods is measured rather than the algorithm tuning ability of the researcher (Ansótegui et al., 2009).

The obsession to create novel algorithms that produce marginal improvements on a very specific circumstances has been criticized, for example, by Hooker (1995). Such competitive testing only reveals which algorithm has better performance in that specific experimental setting but rarely goes into details why this happens. However, scholars should be interested in which algorithmic elements

are responsible for the performance of a specific solution method and how it is correlated with the characteristics of the problem instances (Corstjens et al., 2019). Meta-optimization exploits and can reveal the patterns between the problem instance, heuristic algorithm, and optimization search spaces. The characterization of these spaces is central to the successful application of the meta-optimization approach; hence, finding suitable techniques for extracting this information is needed.

1.4 Main contributions

This dissertation provides a holistic survey, a conceptual framework, and proofs of concept of the suitability of meta-optimization applied to vehicle routing systems (Figure 1; Paper PI). The core of this dissertation consists of an extensive set of comparative experiments on automatic configuration (PVI) and algorithm selection (PV) techniques used to configure and select heuristics targeting the vehicle routing problem. The empirical results of these experiments support our hypothesis that an automated, more systematic approach for customizing routing systems can be achieved through the application of machine learning and meta-optimization. We have shown how such automation can improve the performance of the existing algorithms while simultaneously allowing algorithm developers to avoid the onerous task of manually fine-tuning the solvers. The proposed meta-optimization approach also addresses the usability concerns of modern metaheuristics, because it can reduce the number of parameters that the user needs to set. This can be used to make decision support systems easier to use, but also simultaneously improve the robustness, simplicity, and flexibility of the algorithms powering them. Also, configuring, predicting, and using the most suitable algorithm seems to be a promising approach to reduce the total computing time while still maintaining a good solution quality. Desaulniers et al. (2014, p. 151) recognized the importance of such advances, especially when targeting interactive and dynamic planning.

Our research also contributes to the development of VRP modeling, visualization, and optimization tools. The contributed results, characterizations, and software increase our understanding of the interactions between the vehicle routing problem instances, the algorithms used to solve them, and their parameters. Hence, our work on VRP algorithm performance analysis (PIV), problem instance characterization (PIII) and search space visualization (PII) can be used to recognize the different strengths and weaknesses of different heuristics.

The aforementioned contributions can be reflected against the needs and issues recognized in the literature on vehicle routing systems. Sörensen et al. (2008) recognized a strong need to make commercial vehicle routing software more self-adaptive and easier to customize and use. Here, the meta-optimization techniques offer practical solutions on how to address these challenges: We explore a number of techniques that can make the tailoring of vehicle routing al-

gorithms in an industrial software engineering context more scalable and user-friendly. Furthermore, our experiments show that the approach can improve the overall performance of a typical vehicle routing system that contains a portfolio of alternative algorithms. Especially when compared with the option of using a single sophisticated solution method.

The presented research also contributes to the recent important discussion on the experimental study of VRP algorithms (Geiger, 2018; Sörensen et al., 2019; Corstjens et al., 2019), and designing solid computational experiments for testing VRP heuristics is a strong theme throughout this work. We strongly agree with Eiben and Smit (2011) that the automatic configuration of parametrized algorithms is needed to conduct solid experimental comparisons because it removes the effect of varied manual configuration effort and expertise. We also acknowledge that the topics of reproducibility, replicability, and openness of vehicle routing algorithm research are very important (Sörensen et al., 2019). This dissertation contributes to this topic by presenting replications of the results of many well-known VRP heuristics, by providing analysis of their performance and time complexity characteristics, and by publishing an open source software library implementing them (PIV).

Taken together, our research shows how combining the research on vehicle routing problems, vehicle routing systems, machine learning, and meta-optimization (see Figure 1) can lead to useful contributions. We believe that these contributions allow moving towards more adaptive and easier-to-use vehicle routing models, algorithms, and software.

1.5 Structure of the dissertation

This dissertation is organized as follows: This chapter (Chapter 1) has outlined the context, aims, and major contributions of the work. Chapters 2 and 4 are theoretical in nature and give an introduction to their respective research fields, whereas more practical applications in the context of customization and deployment of vehicle route optimization are given in Chapters 3 and 5.

To be more specific, Chapter 2 gives both informal and formal descriptions of the vehicle routing problem, its variants, and solution methods, whereas Chapter 3 shifts the focus to the vehicle routing systems. These systems are used in planning and optimizing practical transportation cases, and the chapter includes a discussion on the varied functionality, typical modules, and customization of such systems. Similarly, although Chapter 4 introduces the topic of machine learning, these techniques are mostly used in meta-optimization tasks, which are discussed in Chapter 5. Finally, Chapter 6 provides detailed summaries of the original publications included in this dissertation and Chapter 7 gives the context for these contributions. This final chapter finishes with a discussion on the limitations and future research topics.

2 VEHICLE ROUTING PROBLEM

The vehicle routing problem (VRP) is a well-known $\mathcal{NP}$-hard problem in combinatorial optimization (Lenstra and Kan, 1981) and an extension of the traveling salesman problem (TSP). The NP-hard property makes it computationally difficult to solve. It was originally introduced in 1959 by Dantzig and Ramser (1959) and because of its practical relevance, it has been intensively researched ever since. In fact, it has become one of the most widely studied topics in the field of operations research (Braekers et al., 2016). Thus, this chapter gives only an overview of the most central topics. The reader is referred to Toth and Vigo (2014) for a more comprehensive handling of the topic.

The classic capacitated vehicle routing problem (CVRP) can be stated as follows: the task is to plan optimal *routes* for a set of *vehicles* leaving from a *depot*. On their routes, the vehicles serve a number of *clients* by fulfilling their *orders*. Each client must be *served* exactly once by exactly one vehicle, that is, the orders cannot be split. Each vehicle leaves the depot and returns there when completing its route. The *capacities* of the vehicles together with the *demands* of the customers are known, and the total demand of a route must not exceed the vehicle's capacity.

A useful definition can be given using graph-based formalism. Let $G = (V, E)$ be a graph, where the set of vertices $V = 0, \ldots, n$ consists of the depot at vertex 0 and the n customers at vertices $1, \ldots, n$. Each edge $(i, j), i \neq j$ in the set of edges $E = V \times V$ has an associated non-negative weight d_{ij}. The weight corresponds to the minimum cost of traveling from node i to j. Hence, $d_{ij} = 0 \; \forall \; i = j$. The graph is assumed to be fully connected, and thus, all the weights together can be represented in a *distance matrix* D. Additionally, if $d_{ij} = d_{ji} \; \forall \; i, j$, the problem is said to be symmetric. Now, each route $r^k \in R$ can be defined as a path of customers $\langle r^i_1, r^i_2, ..., r^i_m \rangle$, where $r^i_1 \in V \setminus \{0\}$ is the first customer and $r^i_m \in V \setminus \{0\}$ is the last customer of the route r_i. Here, R is the set of all possible routes, that is, all Hamiltonian paths for the subgraphs of G that start from the vertex 0 but with the depot node removed from the path. This notation allows for formally specifying the side constraints. In CVRP, each customer has a demand q_i. Now, assuming the vehicles are identical and have a capacity Q, then the capacity constraint $\sum_{i \in r^k} q_i \leq Q$ specifies that the route must not exceed this limit. Other

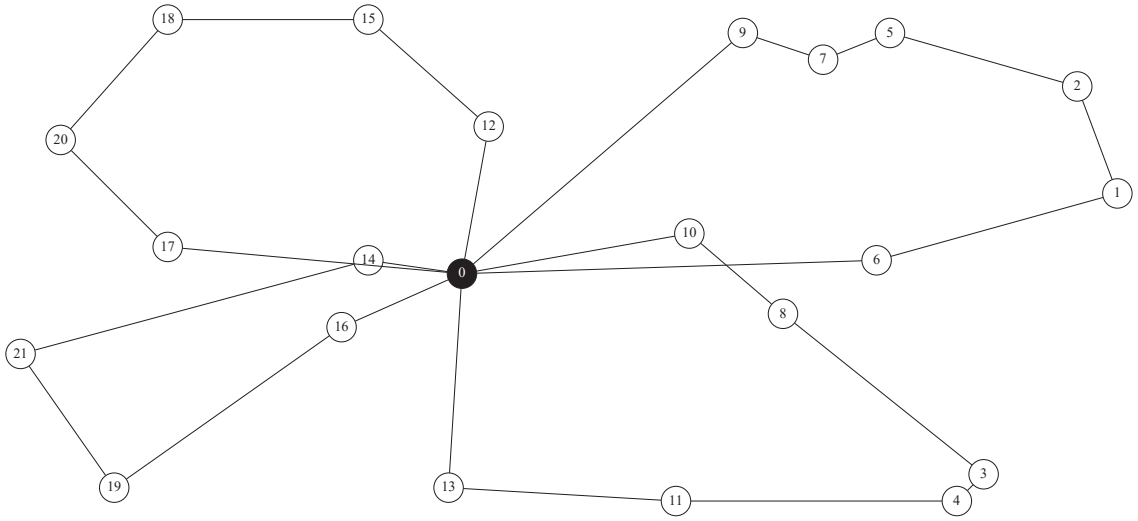


FIGURE 2 A CVRP problem instance and its optimal solution.

VRP variants (Toth and Vigo, 2014, Part II) introduce additional side constraints.

Assuming that all of the side constraints are satisfied, $S \subset R$ is a solution to the problem and if and only if all customers are served, as guaranteed by the condition $\bigcup S = V \setminus \{0\}$, and the condition $\sum_{r^k \in S} |r^k| = |V \setminus \{0\}|$ making sure that no customer is served twice. If all constraints are satisfied, then the solution is said to be *feasible*. Furthermore, if $\mathbb{S}$ is a set of all such feasible solutions and if $f(S)$ gives the objective function value of solution S that we are minimizing, then S' is the optimal solution if $f(S') \leq f(S)\ \forall\ S \in \mathbb{S}$. One such problem and optimal solution is illustrated in Figure 2

The graph-based formalism presented above gives a useful but simplistic view of the breadth of modeling development and in the shifts in the focus that has happened throughout the 60 years of VRP research (see Figure 3). The early models were *idealized*, but quite quickly, the practical applications of VRP introduced new side constraints. These extensions ultimately led to the so-called *"rich"* VRP models (surveyed in Lahyani et al., 2015), which could capture several complex characteristics of real-life routing and scheduling problems. The large number of variants and their combinations created interest toward *unified* modeling frameworks that could be used to model a large selection of different VRP features (Desaulniers et al., 1998; Ropke and Pisinger, 2006a; Irnich, 2008; Puranen, 2012; Vidal et al., 2014; Kritzinger et al., 2017). Recently, Puranen (2011) observed that the research on models was progressing toward generic models that could be *composed* as needed.

Modeling, however, is only part of a solution. Finding solutions involves solving combinatorially difficult problems using exact, heuristic, metaheuristic, or hybridized search methods (Toth and Vigo, 2014). These problems tend to have several local optima (Czech, 2008) where the optimization algorithms can prematurely converge to. Also, the constraints may make it hard to find a single feasible solution (Beck et al., 2003), and increasing the problem size leads to extremely large solution search spaces because of the combinatorial explosion. These properties make these problems challenging and, thus, interesting. Hence,

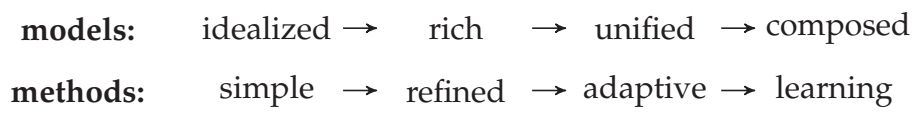


FIGURE 3 Trends in VRP research. Adapted from Puranen (2011, p. 80).

novel search algorithms still constantly emerge and are being evaluated (see, e.g., Arnold and Sörensen, 2019a). Much like the trends in modeling, the proposed solution approaches have evolved. The first algorithms were *simple* one-stage constructive heuristics. These were soon followed by a generation of more *refined* exact solvers that relied on advanced mixed integer solving techniques such as cutting plane generation, branch-and-bound, and column generation. Heuristics became multi-phased and drew inspiration from solving other related combinatorial optimization problems. This development continued and led to *adaptive* metaheuristic algorithms based on artificial intelligence ideas (e.g., Ropke and Pisinger, 2006b; Battiti et al., 2010; Vidal et al., 2013a). Recently, the rise of hybrid algorithms and the use of meta-optimization techniques such as hyper-heuristics has led to algorithms capable of *learning* from the earlier solving attempts. These are typically able to identify the particularities of specific problem instances and adjust the heuristic components based on the earlier and current search progress (see, e.g., Garrido and Riff, 2010).

In VRP research, the prowess of a new algorithm is typically demonstrated through an experimental procedure where improvements over existing results on a set of known benchmark instances are demonstrated. For this purpose, each popular VRP variant tends to have its own set of such well-known benchmark problems. However, there is inherent variation in the performance between the solution methods for these problem instances. Understanding the strengths and weaknesses of different algorithms is important, and this has increased the interest in describing and understanding the idiosyncrasies of the problem instances and their search spaces (Kubiak, 2007; Czech, 2008; Pitzer et al., 2012; Ventresca et al., 2013; Steinhaus, 2015; Arnold and Sörensen, 2019a).

2.1 Variants

Dantzig and Ramser (1959) originally called the problem *the truck dispatching problem*, which, as the name suggests, indicates that the initiative to model and solve the problem arose from practical origins. In fact, according to Braekers et al. (2016), the practical vehicle routing task that Dantzig and Ramser were solving was the distribution of oil to a number of gas stations. Since its first formal definition in 1959, the problem has been extended, studied, and applied from numerous different perspectives. And despite its practical origin, the topic has also received a great amount of purely academic interest. Thus, it has slightly deviated from the practice, and there has been criticism that the models are sometimes too idealized to have a practical value (Hoff et al., 2010; Bräysy and Hasle, 2014).

Still, because modeling and solving VRPs has a broad range of possible applications, research also has led to number of extensions and variations to the basic problem definition that more closely capture several real-world constraints and objectives. The early extensions (Bodin et al., 1983) included multiple depots (MDVRP), service time windows (VRPTW), a heterogeneous fleet (HFVRP), and stochastic demands (VRPSD). Later, the ability to split deliveries, periodic planning, backhauls (VRPB), open routes, working time regulation rules, special loading constraints, service compatibles, alternative service locations, synchronization, transshipments, and many others were proposed. For a survey and synthesis on the topic, please see Vidal et al. (2013b). Related popular families of problems are the pickup and delivery problem (PDP) (Parragh et al., 2008) and the dial-a-ride problem (DARP) (Cordeau and Laporte, 2003), where there does not need to be a central location for the transported items or people and where the pickup and delivery locations can be chosen arbitrarily.

Combining several of these extensions is often required when modeling the complex characteristics of real-life VRPs. These problems are known as "'rich" vehicle routing problems (Lahyani et al., 2015). Furthermore, many real-world VRPs with inherent uncertainty and variability of system conditions can be classified by their dynamic and stochastic dimensions (Pillac et al., 2013). In dynamic problems, new information can appear during the planned operations, and the routes must be adjusted accordingly. In stochastic VRPs, some aspects such as travel time, demand, or cost are not known exactly, but their probability distributions are given. Both of these dimensions increase the complexity of the model formulations and algorithms to solve them. It is also possible to increase the scope of planning by considering related questions of the supply chain together with the route optimization. An example could be *fleet size and mix*, where decisions include deciding fleet composition and the extent of subcontracting. Please refer to Toth and Vigo (2014, Section 1.3) for a more thorough list of VRP variants.

2.2 Solution methods

For the past 60 years, there has been steady progress in the development of VRP solution methods (see, e.g., Cordeau et al., 2002; Vidal et al., 2013b; Toth and Vigo, 2014). During this time, the advances made on exact methods and approximation algorithms, including classical constructive heuristics, local search, and metaheuristics, have been significant. This section provides an overview of the different solution approaches to these problems.

Also, the strengths and limitations of different approaches are explored on a general level. Useful properties for estimating the usefulness of VRP algorithms was provided by Cordeau et al. (2002), and because there is a large variation in algorithm *accuracy*, *speed*, *consistency* or *robustness*, and *simplicity*, each of these should be considered carefully when the suitability of an algorithm is assessed.

2.2.1 Exact methods

The exact optimization algorithms guarantee that if a solution is found, it will be an optimal solution. That is, when measured with the *objective function*, there can be other solutions that are equally good but not a better one. For any nontrivial VRP, the number of possible solutions is too large for a naive exhaustive search. Therefore, numerous relaxation, decomposition, and preprocessing approaches have been proposed.

One of the oldest techniques for solving such combinatorial problems is the branch-and-bound, where the leafs of a search tree contain only a single solution. The tree is branched by, for example, fixing the decision variables. On each iteration, there is a possibility that the algorithm can use bounds to determine that some group of solutions (a branch) many not contain an optimal solution. An upper bound is usually calculated using heuristics, or it can be set to be the objective function value of the best feasible solution found during the search. Similarly, a solution to a relaxed form of the problem can be used as the lower bound. By traversing the tree while simultaneously tightening the bounds, the algorithm will, given enough time, find the optimal solution.

The decision variables are usually constrained to have integer or binary values. Thus, the techniques used in mixed integer programming (MIP) can be used to accelerate the search. These include, among others, branch-and-cut, branch-cut-and-price, column generation, route enumeration, and variable labeling (see, e.g., Lysgaard et al., 2004; Pecin et al., 2017); these techniques are used to rule out large groups of solutions very efficiently, or to speed up the search in some other way. Most involve finding solutions to linear programming (LP) relaxations of the original VRP, where related acceleration techniques can be used.

Although the vehicle and commodity flow formulations (see, e.g., Laporte, 2009) are popular for exact VRP algorithms, an alternative formulation is based on the weighted set covering. This formulation is more relevant to the heuristic approach of this dissertation, and can be given as follows: Each binary decision variable y_r chooses if a feasible route r is included in the solution. The related binary coefficients a_{ir} equal 1 if and only if the order $i, i \in V \setminus \{0\}$ is served by the route r, and it will be 0 otherwise. The cost of using the route r is c_r, and it can be determined by optimizing the order of the customers on the route leaving from and ending at the depot with a TSP algorithm. Using these variables and coefficients, the problem can be written as a binary linear programming model:

$$\min \quad \sum_{r \in R} c_r y_r \tag{1a}$$

$$\text{s.t.} \quad \sum_{r \in R} a_{ir} y_r = 1, \qquad \forall\, i = 1, \ldots, n \tag{1b}$$

$$\sum_{r \in R} y_r \leq K, \tag{1c}$$

$$y_r = 0 \text{ or } 1, \qquad \forall\, r \in R \tag{1d}$$

Here, objective 1a is the total cost of the selected routes, constraint 1b ensures that each order is served exactly once, and constraint 1c specifies that at most K vehicles are used. Because the basic objects of the formulation are feasible routes, the size of the set of feasible routes R grows unreasonably large, even for small VRPs. Despite its limitations, it is a simple formulation, and it has significant practical importance: Instead of a full enumeration of all feasible routes, a heuristic algorithm can be used to generate a promising subset of feasible routes (Taillard, 1999). This means that solving the weighted set covering problem (SCP) can be done as a post-optimization step (Taillard, 1999).

The largest VRP problem instances that currently can be solved using exact methods have only a few hundred points (Pecin et al., 2017), which limits their practical applicability. This limitation is due to the combinatorial explosion of the search space and the related increase in the decision variable and constraint count. Furthermore, extending these models is nontrivial and is often a subject of a research project of its own. And, although the exact methods can guarantee the optimality of the obtained solution, a good enough solution that can be found relatively quickly will usually suffice. Therefore, vehicle routing problems arising from the industry, including the larger and more complex ones, are typically solved using heuristics or metaheuristics.

2.2.2 Classical heuristics

"Classical" VRP heuristics refer to the relatively simple heuristics developed between 1960 and 1990 (Laporte and Semet, 2002). Cordeau et al. (2007) further divided the classical VRP heuristics into *constructive*, *two-phase*, and *improvement* heuristics. Here, a heuristic is a rule-of-thumb algorithm that can be applied to produce a solution to a problem—or to improve an existing one. They are often used when one needs to produce good solutions in a reasonable amount of time. However, they cannot guarantee the optimality of the solution, or even a lower bound for its quality. Also, although they are relatively simple to implement, they are sometimes hard to use effectively because it is hard to know which heuristic should be applied and when. This is because of how heuristic algorithms are created: they are developed for specific use; thus, they are highly specialized, and their performance is problem and situation dependent.

The earliest of the classical VRP heuristics belong to the class of *constructive heuristics*. They are capable of starting from an empty solution and usually can build feasible solutions with surprisingly good quality. Such constructive heuristics typically work by inserting unassigned clients to routes until all customers have been assigned. Such an insertion strategy builds routes either sequentially one route at the time or in parallel by considering all of the routes together. Note that this terminology should not be confused with parallel computing.

Perhaps the best-known parallel constructive algorithm is the savings heuristic from Clarke and Wright (1964). It starts from a solution where each customer is served using a separate route and savings values are calculated for each potential merging of a pair of routes. The algorithm then applies the merges in

the decreasing savings value order. Each merge is checked for validity to avoid making the solution infeasible. Because of its popularity, many heuristics that extend this basic scheme have been proposed (see, e.g., Paessens, 1988). Alternative constructive algorithms have relied, for example, on the greedy insertion of nearest neighbors or otherwise promising candidates (Fisher, 1995). All such heuristics are simple to implement and, in most cases, easy to extend to solve the more complex VRP variants, which explains their long-lasting popularity in the literature (Cordeau et al., 2002). However, their accuracy and robustness are not usually that good (Laporte and Semet, 2002), and they should be complemented with modern metaheuristics in most practical applications.

Two-phase heuristics tackle VRP instances by solving two separate subproblems: 1) *clustering*, that is, assigning the customer visits to routes or vehicles, and 2) *routing*, that is, sequencing the visits in an optimal order. The order of these two phases varies between the different heuristics and two-phase heuristics can be further divided into *cluster-first, route-second*, and *route-first, cluster-second* heuristics (Laporte, 2007). The Petal heuristic from Foster and Ryan (1976) is a good example of an early two-phase heuristic. First, a set of feasible candidate routes is built by radially sweeping over the customers. These routes form the collection of sets in a weighted set covering problem (SCP). Solving the SCP is followed by an inter-route improvement procedure that can produce new routes, which are then used during the subsequent iterations that alternate between the SCP and the improvement procedure. The algorithm terminates when no improving routes are found. Although the Petal heuristic is quite flexible (Cordeau et al., 2002), many of the two-phase heuristics tend to be fairly fragile and suitable only for some specific classes of VRP problems. It should be noted that some two-phase heuristics, such as the Petal algorithm and the generalized assignment problem (GAP) heuristic of Fisher and Jaikumar (1981), also belong to a class of algorithms known as *matheuristics*. These algorithms are characterized by interoperation between heuristics and mathematical programming techniques. For a survey of VRP matheuristics, refer to Archetti and Speranza (2014).

Many classical heuristics have employed simple improvement heuristics. Improvement heuristics work by taking an existing solution and making small changes that improve the objective function value of the solution. The early heuristics were adapted to VRP from TSP literature (see, e.g., Savelsbergh, 1985). The framework for improvement heuristics is provided by the concept of *local search* (LS). LS has become a key element in modern metaheuristics because of its flexibility and robustness. To formally describe LS, the notation of Laporte (2007) is used: When trying to improve a solution $s \in S$, where S is the set of all feasible solutions, LS can employ different heuristic operations that each define a neighborhood $N(s) \subset S$. To make a move in the local search, a new solution $s' \in N(s)$ is selected. Given that $f : S \to \mathbb{R}$ tells the objective value of a solution, and if the simple strategy of always taking the best improving move is used, the s' is the one for which $f(s') \leq f(s) \; \forall \; s \in N(s)$. Applying this simple strategy iteratively will ultimately lead the LS to a local optima. While doing so, the search creates a *trace* in the search space consisting of the candidate solutions.

Well-known local search heuristics for VRP are shared with TSP and include 2-opt, 3-opt, relocate, and exchange, but there are also inter-route variants such as 2-opt*, 3-opt*, one-point move, and two-point swap (see, e.g., Laporte and Semet, 2002; Bräysy and Gendreau, 2005; Groër et al., 2010). In the past 30 years, scholars have shifted their focus from these simple heuristics to the use of metaheuristics, which, in turn, has allowed significant advances in solving real-world VRPs.

2.2.3 Metaheuristics

As stated earlier, the disadvantage of exact methods is that they are limited to solving only relatively small problems. Similarly, classical heuristics are not very robust and are unable to consistently produce high-quality solutions because the local optimum they stop at is rarely the global optimum. Thus, methods that are capable of wider exploration of the search space have been under intensive research. These methods are known as metaheuristics and have become, in practice, the standard way of solving VRPs. The literature on metaheuristics is too extensive to be discussed here in depth, and thus, only those metaheuristics that are relevant to this dissertation are mentioned. Please refer to the book from Labadie et al. (2016) for a comprehensive discussion on the topic.

The simple early heuristics tended to be problem dependent. The meta-prefix in metaheuristics refers to the fact that they are a more generic approach (Olafsson et al., 2008). Indeed, metaheuristics are based on high-level and abstract principles (Birattari, 2009) and on rely these strategies to guide the search. This usually involves the orchestration of several lower-level application-specific (local search) heuristics. Metaheuristics also typically include a mechanism for escaping a local optima attractor (Labadie et al., 2016), allowing for a more thorough search of the solution space. This is achieved through the use of sophisticated mechanisms such as memory structures, perturbation moves, route combination, and so forth. However, such flexibility comes at a cost. The implementations of such metaheuristics expose many design choices through parameters that need to be configured for the algorithm to work effectively (Birattari, 2009).

Following Battiti et al. (2010), some general strategies of metaheuristics can be recognized: there are methods that 1) optimize the problem a subproblem at the time, 2) manipulate the objective, 3) allow non-improving moves, or 4) prohibit some local moves. For a more in-depth handling of the search diversification methods in the context of VRP see, for example, Laporte (2007) or Toth and Vigo (2014, Chapter 4, Section 4.4). Popular VRP metaheuristics include, among others, simulated annealing (SA), tabu search (TS), ant colony optimization (ACO), genetic algorithms (GA), and iterated local search (ILS):

SA: Simulated annealing (Kirkpatrick et al., 1983) is a metaheuristic that randomly allows worsening moves to escape a local optima. The probability of allowing these moves depends on the simulated temperature of the system. When the search is intensified, the temperature of the system is lowered. In contrast, if it is detected that no improvements cannot be made, the system is reheated to make further exploration of the search space possible.

TS: In tabu search (Glover and Laguna, 1998), some solutions of a neighborhood can be marked as tabu (forbidden). This may force the local search to do non-improving moves and allows climbing out of the local optima basin of attraction. The mechanism also prevents the search from cycling. In addition to selecting the local search neighborhoods, the *forbid* and *free* strategies that manage the tabu list, the aspiration strategy governing these, the selection strategy for choosing the next non tabu move, and the stopping criterion need to be selected and configured. Gendreau et al. (1994) and Osman (1993) were among the first to apply TS in solving VRPs, and Osman (1993) included a comparison between TS and SA.

RTR: Similarly to SA, the record-to-record travel metaheuristic (Dueck, 1993) uses a deterministic rule to accept a worsening move; If r is the best-known solution (the record), then an alternative solution $s' \in N(s)$ is selected if $f(s') < f(r) + d$, where d is the allowed deviation from the record proportional to the objective value of r. A RTR variant targeting HFVRP (Li et al., 2007) was shown to be competitive against the other contemporary algorithms.

ACO: Nature-inspired computational methods have always been popular in the literature on metaheuristics, and there are numerous studies where they have been successfully applied to solving VRPs (Potvin, 2009). In ant colony optimization (Dorigo and Di Caro, 1999), the behavior of ants is mimicked by modeling the pheromone trail they leave behind and follow while exploring their surroundings. ACO have been adapted in VRP, for example, by Rizzoli et al. (2007).

GA: Genetic algorithms (Holland, 1975) can be used to evolve a population of solutions. GA-based approaches usually encode VRP solutions into chromosomes through a list of node indexes (Potvin, 2009), and the fitness of a solution is measured by the objective function value. Through the crossover, mutation, and selection of these chromosomes, fitter generations are evolved. For a review, refer to Potvin (2009).

ILS: Iterated local search is a generic heuristic that applies one or several local search operators until a local optimum has been reached. Then, the solution is perturbed with a carefully selected operation to obtain a new initial solution for the local search. This process is iterated until a stopping criteria has been fulfilled. Please refer to the book chapter from Lourenço et al. (2003) for a comprehensive handling of ILS with VRP references.

The recent trend in metaheuristic research has been toward hybrids. *Hybrids* combine various ideas from different algorithms. The motivation behind hybridization is to build more robust and capable algorithms by exploiting the strengths of different search strategies. Hence, many recently proposed state-of-the-art algorithms are, unsurprisingly, hybrids. An example of such an algorithm is the state-of-the-art unified hybrid genetic search metaheuristic from Vidal et al. (2014). The

algorithm targets multi-attribute VRPs and has demonstrated remarkably good performance on a wide range of VRP variants. Note that hybrids where a local search is combined with an evolutionary computation are sometimes called memetic algorithms (Neri and Cotta, 2012). Also, the hybridization of metaheuristics with machine learning to create solution methods capable of learning from earlier solution attempts is an emerging research topic in operations research (Calvet et al., 2017).

3 VEHICLE ROUTING SYSTEMS

Software systems play a crucial role in planning real-world transportation and delivery operations (Bräysy and Hasle, 2014). This dissertation is mainly concerned with the functionality that allows the optimization of the delivery and transportation routes. Such route optimization can be built as part of more general purpose enterprise resource planning (ERP) systems, that can deeply integrate into the business and operational processes of a company. Typically, these systems contain a wide range of features and applications: from product, inventory, and sales management; through timesheet and employee records; and to customer, purchase order, billing, delivery management, and tracking. The more specialized systems for managing the transportation and distribution functions are called transportation management systems (TMS) or advanced fleet management systems (AFMS) (Crainic et al., 2009, Section 4).

Vehicle route optimization may be available as a module for a TMS or AFMS, but it can also be sold as separate stand-alone software (Drexl, 2012). To differentiate it from the aforementioned more comprehensive management solutions, we call such a stand-alone application a *vehicle routing system* (VRS). Concentrating only on the planning component and the related support functions prevents the topic from straying too far from the vehicle routing problem. The responsibilities of a VRS are described by Drexl (2012) as follows: A VRS is a computer program that can read in a transportation problem with vehicles, drivers, depots, customers, and their requests with locations that define the specific problem scenario. From there, a VRS allows manual, interactive, or fully automated optimization of the routes. The key feature of the system is the computational algorithms that can build a complete routing *plan* from the input data.

It is useful to reiterate some of the advantages of the optimization technology in the context of VRSs: the solutions are often superior to those made by human planners, and producing them reduces planning time and excludes the chance of human error (Drexl, 2012). This is because software can address and balance among many constraints and competing requirements, such as robustness to change, carbon emissions, level of service, and operational costs in combinations, which would create a too complex planning task for a human planner.

Still, the main reason for introducing VRS is often to help use the transportation resources of the company more efficiently and reduce the operating costs (UK Department for Transport, 2005). The typical cost savings from introducing a VRS can range from 5% up to 20%. The savings come from operational efficiencies in reducing the journey length and time, improving the utilization of transportation resources, and improving the reliability of the schedules. This alone does not necessarily lead to happier customers, but through management and service enhancements, the approach can support better processes for order management, communications, and planning (UK Department for Transport, 2005). Irnich et al. (2014) recognized other benefits, such as standardization and tighter integration of the planning processes with the existing software systems. This makes planning less time-consuming and more cost-efficient compared with manual planning. Furthermore, the benefits are not limited to the operational level, and Drexl (2012) also emphasized the macroeconomic benefits, such as improved traffic flow and lowered transportation emissions.

Besides optimization at the operational level, that is, the ongoing, daily, or weekly routes and schedules, these tools can be used in strategic planning via modeling of different scenarios involving, for example, employing subcontractors, planning for forecasted developments, evaluation of alternative strategies, comparing the effects of various resource procurement or allocation options, or testing the effect of service level changes (UK Department for Transport, 2005). Such strategic and tactical planning allows organizations to better allocate their resources and estimate their future resource needs. A VRS can also be used to estimate the cost figures of the transportation offer calculations on which the contracts are based. These cost estimations can provide immediate value whether the user of the VRS is providing the transportation service or subcontracting it.

If we consider the impressive volume of research on VRPs, the advantages of route optimization, and the progress of computational technology, it is no wonder that a growing number of software companies have emerged to provide commercial vehicle routing systems (Sörensen et al., 2008; Hoff et al., 2010; Bräysy and Hasle, 2014). According to Baker (2002), in 1997, there were already 133 software packages available for routing and scheduling applications. Since then, OR/MS Today has published biennial surveys of VRS software vendors, with the latest being done in 2018 (Partyka and Hall, 2012, 2014; Horner, 2018). These surveys keep the vendors and their potential customers up-to-date on the latest developments. For example, in recent years, there has been a clear trend toward web-based, software-as-a service (SaaS) solutions.

The early expectation in the 1980s was that organizations would adopt these systems with enthusiasm, but coming into 1990s, almost all planning was still done manually (Waters, 1992). Given the prevalence of modern optimization technology and its many benefits, the results of the survey made by the Department of Transport of the United Kingdom in 2010 (as summarized by Rincon-Garcia et al., 2018) still reported that the adaptation rate was surprisingly low among the surveyed transportation companies. The study found that the main reasons for not adopting a VRS were that the existing systems were not suitable,

were too expensive, or were too complex. Hence, the difficulties in VRS adaptation are mostly because of the complex operating environment these systems face (Partyka and Hall, 2012). This is also the reason there is a need for advances that can increase the self-adaptivity of such systems (Sörensen et al., 2008) and automate some of the systems' expensive customization work (Neittaanmäki and Puranen, 2015).

3.1 Functionality and typical modules

In this section, we discuss the features of a typical VRS, outline the software modules that the system consists of, and describe how they interact. However, it is important to remember that the specific functionality offered by a VRS will vary according to its intended use. The features required when optimizing routes, for example, for transporting people, have different modeling requirements than when planning for parcel deliveries, transportation of perishables like food, or hazardous liquids. At minimum, a software system that is used for vehicle route optimization requires the depots, the customer visits (stops), and the types, operating costs, and capacities of the vehicles (the fleet) along with their locations (street address or coordinates) to be given as inputs (Baker, 2002). In addition, sometimes, the driver qualifications, shift scheduling, and related drivers' working-hour regulations are an important part of the planning, and they need to be optimized along with the routes. The exact nature of the transportation, be it perishable goods, parcels, products, or people, may induce some special rules. For example, the demand of a customer can be described using a single number or using various dimensions (e.g., weight and volume). Furthermore, the customer orders may be last-mile deliveries, with or without backhauls, or a mix of pickups and deliveries during the day or with a longer planning horizon. There may be single or multiple allowed time windows for the visit. Also, compatibility rules between the customers, drivers, vehicles, and products are typical in real-world transportation planning scenarios. Furthermore, it is possible that the exact demand of the customer is not known, and the task can be to maintain a certain level of a product over a longer time span. It is also possible for the orders to be removed or added dynamically during the planning horizon. These dynamic features make the problems harder to solve because this increases the interdependencies between the decisions (Pillac et al., 2013). These requirements illustrate the multifaceted nature of these problems.

The modular structure of a typical VRS is illustrated in Figure 4. One can usually recognize modules (a) intended for interfacing with the problem instance data storage, (b) a geographical information system (GIS) module for handling addresses, coordinates, and travel distance and time calculations, (c) a conceptual/domain model (Hasle and Kloster, 2007; Puranen, 2011) for building an abstract model of the problem using the concepts of the problem domain, (d) automatic planning (solver) module that offers the optimization algorithms and re-

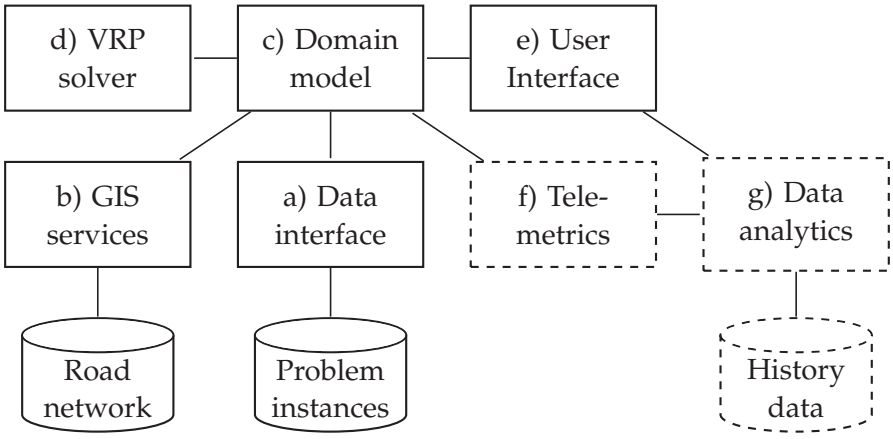


FIGURE 4 Modules of a VRS.

lated mathematical modeling services, and (e) a user interface component for visualization and interaction with the software. In addition, there may be a telematics module (f) for receiving real-time data from the field operations and data collection and statistics module (g) for the calculation of key performance indicators and generation of other relevant reports. A similar modular structure was presented by Drexl (2012), Puranen (2011), and Bräysy and Hasle (2014, p. 354).

An important requirement for a VRS is the ability to integrate it with other systems to read in the depots, vehicles, orders, locations, and other necessary information (Sörensen et al., 2008). In this case the data interface module is customized to support an external data source. In fact, Baker (2002) noted that a major portion of the functionality in a commercial VRS is devoted to file manipulation and geocoding.

However, the most central module to this dissertation is the VRP solver component, which resides at the core of every VRS. According to Sörensen et al. (2008), the ability to model many types of problem characteristics is one of the most important requirements of a commercial VRS. There needs be a way to specify the objective and operational constraints, be it through a domain-specific language (DSL) or by configuring the software some other way. Hence, the planning/optimization component also defines the modeling capabilities, and the *richness* of the underlying VRP model influences the applicability of the entire system (Hasle and Kloster, 2007). If, for example, there is a time limit for how long a product can be transported but this is not part of the optimization model, then the resulting plans cannot be executed as is, and manual changes may be required to produce actionable plans. Hence, the modeling capabilities are very important to the end users, even if they are not always able to recognize it. The necessary *attributes* to the optimization model used in typical transportation applications are often numerous. These include, but are not limited to, multiple capacities (weight, volume, etc.), maximum travel time, multiple depots, heterogeneous fleet, time windows, pickup and delivery, the stochasticity of the demand and travel times, periodicity, and so forth. Also, the planning task may include many qualification and compatibility rules between drivers, vehicles, or-

ders, depots, and even the transported products. Complementing a VRP model with such attributes leads to *multi-attribute vehicle routing problems* or *rich* models (Lahyani et al., 2015). Related to these model attributes, Drexl (2012, pp. 57–58) and Bräysy and Hasle (2014, pp. 357–364) provided a list and descriptions of the modeling features found in a typical commercial system. Also, Hasle and Kloster (2007); Puranen (2011); and Derigs and Vogel (2014b) described considerations regarding the implementation of such a general model.

As mentioned in Section 2.2.1, VRPs can be modeled and solved as mixed integer programming problems. Usually, this involves describing the individual problems in a modeling language for linear programming. Although this approach can and is used in some special applications (Drexl, 2012, p. 59), even the state-of-the-art exact solution methods are limited to VRP instances of less than a few hundred customers (Uchoa et al., 2017). Hence, the commercial VRSs use metaheuristics that usually are some variant of a variable neighborhood search (VNS) (Sörensen et al., 2008). Such solver may offer a few initialization heuristics and multiple local search procedures to improve the initial solution. VNS also has the benefit that it is relatively simple to implement, which helps to keep the complexity of the software low. Also, this approach can usually provide feasible solutions within a few minutes, which makes the quick re-optimization of the plan possible in cases where the operational situation changes during the day (Drexl, 2012).

Most commercial systems claim to be capable of solving problems up to 10,000 customers with ease (Drexl, 2012). However, because they are built around a rich model and a unified algorithmic approach, they may be less computationally efficient than some of their more specialized academic counterparts (Hasle and Kloster, 2007). This illustrates the differences in the focus between academic research and industry: researchers are more concerned with the modeling and solving efficiency, whereas in commercial routing systems facing real-world transportation tasks, the usability, flexibility, and scalability are usually more important (Bräysy and Hasle, 2014). Related to this, the impressive performance of some academic state-of-the-art metaheuristics may fail to generalize unless the parameters are manually tuned specifically for the specific problems being solved (Sörensen, 2015). Requiring such a specific calibration effort is not desirable with a commercial solution (Rincon-Garcia et al., 2018).

The requirements for interactivity vary from one application to another. Sometimes it might be desirable to require the minimal amount of interaction with the routing system—to the extent that the plan is generated completely without any user intervention (Baker, 2002, p. 359). However, in most cases, a commercial solution must have a graphical user interface (GUI) (Sörensen et al., 2008; Drexl, 2012). The GUI is used to validate the solution or to tweak some aspects of it. Gacias et al. (2012) concentrated on this functionality and studied the human factors and ergonomics of using a vehicle routing system. Instead of a fully automated system, they advocated an approach where through user-interaction tools, the human planners can build solutions that adhere to informal constraints and that allow for relaxing or violating some of the existing constraints. The role

of algorithms in their proposed DSS was to assist the human planner to solve the problem, and the authors argued that this allowed producing solutions that are more suitable for real-world use. Also Drexl (2012) discussed the usability concerns related to easy-to-use GUIs, data interfaces, and documentation. According to him, one of the most central features of a VRS GUI is the ability to display the depot, customer, and vehicle locations and planned routes on the map. Some of this functionality is supported by the GIS module, which also provides services to the solver module.

The solver module requires a distance matrix to be given as an input. However, the matrix can also be represented by a fully connected graph between the depot(s) and the customer locations, where each edge (i.e., the cell of the matrix) represents the travel cost (length or time) from one node to another. The matrix is used to quickly retrieve the distance between any location pair. Populating the distance matrix involves a GIS module that can efficiently do many-to-many shortest path calculations, which is a feature that exists only in a few GIS products (Tarantilis and Kiranoudis, 2002). The shortest path computation requires converting the geographical map data to a graph format, where the nodes and edges represent the roads, junctions, terminals, and delivery locations of the transportation network. A number of preprocessing techniques can be used (e.g., Geisberger et al., 2008) to speed up the calculation. In most applications, it is very important to carefully consider the accuracy of the data and the level of detail provided by the GIS (Hasle and Kloster, 2007) because inaccurate travel distances may make the final solution non viable. An advanced GIS module may also take one-way streets; speed limits; forbidden areas; speed limits; turning, height, weight, and other related road usage restrictions; and traffic congestion into account. Furthermore, it is usual to provide separate distance matrices for the travel time, distance, and each vehicle type in the fleet. The other typical map-related features are the support for geocoding and reverse geocoding to transform street addresses into GPS coordinates and back.

Modern VRS tend to offer comprehensive reporting capabilities. The optimization results can be viewed on the screen or printed as maps, delivery schedules, or load manifests. Alternatively, a VRS may interact with an existing ERP system, which then has the ability to put the optimized plans into operational use. In addition to such plan deployment options, a VRS may have advanced reporting functionality and the ability to produce cost reports, summaries, and plan actualization reports and comparisons. Furthermore, the data can usually be exported in compatible file formats for further analysis in an external application (UK Department for Transport, 2005). Besides such *usability* concerns, Drexl (2012) lists other non-functional features such as *versatility*, *generality*, and *maintainability*. The first two are strongly related to the modeling and solving capabilities of the solver component, whereas maintainability is linked to the flexibility, expandability, and customization of such systems. These non-functional requirements were also discussed in-depth by Puranen (2011), where he considered the *implementability* of a VRS. The most relevant aspect to this dissertation, though, is the *customizability* of vehicle routing systems, which is discussed next.

3.2 Customization

Many VRP modeling and solution techniques originating from academic research can be inflexible and require serious effort to adapt them to new problems and operational contexts (Carić et al., 2008). This can be an issue for the VRS vendors. Drexl (2012) noted, "It is not an option to develop and implement a specialized algorithm for each new customer. Therefore, algorithms used in CVRS must necessarily be generic and easily extendable to new problem features." Because of this, a solver targeted for industrial use usually contains many different algorithms or algorithmic components, meaning that they expose a large set of different parameters (Becker et al., 2005). The same applies to the multi-attribute VRP models used in real-world transportation planning, where the modeling requirements of each individual application determine which features of the mathematical model and which algorithms are needed.

The varying requirements are not limited to the optimization component. According to Cordeau et al. (2002), most of the development effort is devoted to implementing data management and user interfaces. To make such customization tasks more manageable, several software engineering techniques have been proposed. Some, such as the integrated VRP algorithm design environment from Potvin et al. (1994), concentrate on the modeling and solution capabilities and are more intended to be used by an operations researcher. However, the focus of this section is customization in the commercial context; hence, only papers targeting real-world transportation are surveyed below.

The machine-learning-based self-adjusting VRS proposed by Kadaba et al. (1991) was ahead of its time. They stated that their goal was to "assist researchers and decision makers in applying mathematical models to the problem description." and proposed an approach that could adapt to the problem type at hand by utilizing information managed by a frame-based knowledge representation scheme. They used neural networks for feature extraction, to recognize the problem type, select a mathematical model that fits a problem instance, and suggest the seed point locations and control rules for VRP heuristics. In addition to these learning and self-adaptive modules, there is a control algorithm that orchestrates their execution. The information is encoded into the control rules and the weights of the numerous neural networks. The authors presented experimental results, showing that the automatic tuning of parameters is beneficial and that their system provides better results than many of the second-generation VRP algorithms. It remains unclear which VRP variants were considered besides the basic CVRP. Still, the system shows great self-adaptivity and seems to present a flexible and expandable, if complex, architecture for realizing modular and data-driven VRSs. Furthermore, the neural network parts of the system can learn from the data, which is, a scalable approach for customization. Related to this, Kadaba et al. (1991) noted that the good results were because of the careful analysis of important problem instance features. The results of our Paper PV supports and complements their observations.

Desrochers et al. (1999) proposed a system that would allow for the management of VRP models and that could also suggest suitable algorithms for solving the stored models; they stated that the system is meant to be used by a consultant with a basic understanding of mathematical programming. Their approach was to formalize and store the existing knowledge on VRP modeling and solving in multiple knowledge bases. The inference made by the system was symbolic and the stored knowledge was based on manual annotation. For example, the models were stored in a formulation knowledge base in the AMPL modeling language, whereas the model and algorithm suitability was configured by specifying the confidence factors of a semantic network residing in another knowledge base. The customization of the system was done by describing the problem variant using a special formal language (Desrochers et al., 1999, Appendix A), and, if necessary, by manually extending the knowledge bases to support new variants. This also included encoding the information on how to effectively solve them. Unfortunately, only the concept was given in detail. Thus, it is difficult to see how some of the concepts would work in practice. Although one can see how the knowledge of an expert could be encoded into the knowledge bases, building them manually would still require significant effort. Still, the VRP variant classification language is an interesting approach for formalizing the VRP taxonomy in a computer-friendly format.

The challenges in satisfying the varying requirements and needs of different users was recognized by Du and Wu (2001). In addition, they acknowledged the challenges of ever-shifting expectations. They proposed an object-oriented component assembly process focusing on software reuse to manage the complexity in a rapidly changing environment. In their approach, customization was done by storing application functions, class definitions, and object instances in an object-oriented database that acted as persistent storage of the software component variants. The application function implementations could be iteratively revised and reused in other applications. The approach allowed Du and Wu to develop applications for five VRP variants. However, it is difficult to see how the object-oriented database driven development process makes the customization of VRSs easier compared with a more traditional object-oriented programming (OOP) approach where the database is used as "plain" data storage and where the source code is stored in a version control system.

The VRS architecture advocated by Tarantilis and Kiranoudis (2002) was based on distributed components and OOP. Thus, their approach was similar to that of modern microservices (Newman, 2015) and so were the advantages: they claimed that the architecture allowed software component reuse and clearer modularization. The main VRS components (Figure 4) can easily be recognized. However, instead of being composed into a monolithic application, the modules act as separate services and communicate through a SQL database or HTTP. This includes the user interface, which is the only module the user can access directly. Tarantilis and Kiranoudis reported that their VRS has been used successfully in several transportation tasks, but all applications are rather similar (and can be modeled as CVRP).

A more generic view to implementing VRSs can be taken by considering VRS as a type of optimization-based decision support system, as was done in (Maturana et al., 2004). They surveyed the ways building such systems had been approached in the literature and proposed a DSS generator to support the design and construction of optimization-based DSS applications. They claimed that their generator, which combined the latest advances in operations research and software engineering, allowed them to develop DSSs at a fraction of the time and cost compared with a traditional approach. Their approach was centered around MIP models that are defined with structured modeling (Geoffrion, 2013) ideas and a related SML modeling language. Each application had a custom module that could instantiate a MIP model using the data stored in a database. The populated model was then given for the exact solver. The database schema and the related triggers were central in customizing a new application, and it was possible to specify them with a configuration tool. Although the generator-based system allowed reusing many of the components of a DSS, the customization relied entirely on the modeling and specification efforts of an expert. In addition, the data transformations are written manually in SQL, and Maturana et al. only considered an MIP solver, not heuristic solvers. Interestingly, they still recognized the importance of the solver parameters to the solution times, but failed to provide a solution other than exposing the parameters to the end user and trial-and-error to configure them.

Carić et al. (2008) presented an integrated modeling and optimization framework for VRPs based on a modular design that had a focus on code reusability. The algorithms were scripted using a custom DSL that provided the abstractions relevant to vehicle routing. The DSL, together with tools for algorithm testing, tuning, and visualization, allowed for fast experimentation with existing VRP heuristics. The authors also presented rudimentary self-adaptation capabilities of the framework in the form of algorithm performance prediction. The flexibility of the VRP model was not demonstrated, and extendability of the approach beyond VRPTW remained unclear.

The customization solution of Puranen (2011) was built around a model-driven software production line architecture and a unified PDPTW metamodel. The proposed optimization system was shown to be flexible by adjusting it to support many different VRP model attributes. The modeling approach relied on defining model transformations that targeted a generic PDPTW model, much like Ropke and Pisinger (2006a). The customized applications were derived manually by developing the differing elements on the application layer. Thanks to the flexibility of the metamodel, this did not require making changes on the algorithm level. Thus, the approach seems to be effective in managing the complexity of the VRP domain and a useful tool for balancing between generality and specificity. The variability is managed systematically by explicitly defining the *variation points* (Pohl et al., 2005). The customization is done by defining the transformations that specify the variability objects associated with these points. The approach creates a good foundation for a data-driven automatic product derivation framework. In fact, Puranen expected the approach to allow automatic data

interpretation in model building, automatic algorithm configuration, and the use of hyper-heuristics (Neittaanmäki and Puranen, 2015). Hence, the concepts of software product lines and variability management were central in the work of Puranen (for more, see Pohl et al., 2005).

Derigs and Vogel (2014a) suggested that their heuristic framework for rich VRPs was user-friendly and enabled flexible customization of problem-specific solvers. The base implementation of the framework only supported the reading and solving of CVRP instances. From there, customization was done on an adaptation layer by overriding the existing simpler functionality and by extending the capabilities of the system through OOP inheritance. The algorithm framework separated the local search move evaluation and move application, allowing the reuse of the higher level functionality. Although the framework seems to allow for some code reuse, they admitted that a lot of experience and creativity was required from the developer in the form of manual design, programming, and fine-tuning to derive a new application.

We acknowledge that there has been earlier studies that have sought to automate the fine-tuning of algorithm performance (Nygard et al., 1990; Kadaba et al., 1991; Becker et al., 2005). Also, the idea of using machine learning to automate software engineering tasks is not new. Zhang and Tsai (2005) dealt with topics related to using learning algorithms in software development and maintenance tasks; indeed, they saw the software reuse topics as a fertile ground for applying case-based reasoning. However, we are not aware of a pre-existing, holistic, data-driven, and machine learning-based approach for VRS customization and believe that these topics can be combined in a novel way.

To summarize this subsection, we have surveyed the methods for deriving and specifying the optimization model in a vehicle routing system. To make this task more manageable, the aforementioned architectures typically used a clear separation between the conceptual data layer, the modeling layer, and the solver layer. In this dissertation, we concentrated our customization effort to the data transformations that happen between these layers.

4 MACHINE LEARNING

This dissertation proposes leveraging *machine learning* (ML) in automating the customization of the vehicle routing systems. The purpose of this section is to give the reader a brief overview of ML topics and techniques used in the included papers. A more comprehensive operations research perspective can be found, for example, from the survey by Olafsson et al. (2008).

Machine learning can be defined (Mitchell, 1997) to be the art of seeking an answer to the question: how do you build computer programs that improve their performance at some task through experience? Hence, with ML techniques we can build software that can *learn* and discover patterns in data. The task of learning involves fitting a model to observations, and the resulting models can be used to predict outcomes, support decisions, or guide the search for solutions in optimization algorithms.

When applying machine learning, it is crucial to understand what the algorithm is actually trying to learn. This includes having some generic idea about the correct answers the final model should give. ML is often able to gradually find better solution variations to a specific problem, but it can rarely produce novel, unexpected solutions. This is also the reason why the selection of a correct model and input data are very important. Clearly, the data should contain information about the phenomena we are trying to model: Relevant information in the form of *features* has to be available for an algorithm to learn useful patterns. One should also note that there is a fine balance between what features are available and which subset of them is useful to the learning task at hand (Guyon and Elisseeff, 2006).

As ML all is data-driven, it is essential that the available data sets are good, unbiased, and representative. Furthermore, the data set should be large enough so that the algorithm is able to produce a useful generalization instead of getting an incomplete picture of the phenomena. Gathering the necessary data can be difficult and time-consuming, but if the data is incomplete, barely sufficient, or biased the quality of the final model will be poor (Kotsiantis et al., 2006; Domingos, 2012). Based on the available dataset, and the goal of applying ML, it is useful to differentiate between ML task types:

Unsupervised learning: These techniques are designed to work on unlabeled datasets, where the correct responses (labels) are not known, but we still need to extract meaning or patterns from the data. Thus, they can be used in exploratory data analysis or to build predictive models on datasets we know little about. For example, *clustering* involves grouping inputs that are similar to one another. It can be used to produce labels for unlabeled data based on the hidden structure of the dataset. Another unsupervised technique is *dimension reduction*, which involves finding a data mapping to a space that has a lower dimension than the original dataset. Usually, dimension reduction techniques are used for visualization or as a preprocessing step for other supervised or unsupervised learning techniques

Supervised learning: This approach is used when the correct responses are a part of the dataset, and we want to build a model that is able to predict these responses from the inputs. A good supervised learning algorithm is able to respond correctly to previously unseen inputs. *Classification* and *regression* are typical supervised ML tasks.

Reinforcement learning: This ML technique allows finding model parameters that maximize a certain reward through a process of trial and error (Bishop, 2006). Thus, the technique is not used for learning the patterns in the data, but to find a model how a given system operates.

The data used in ML is typically split into three sets: *training*, *testing*, and *validation* data. The training data set is used when the algorithm learns the model. The testing data set is used to measure how well the trained model works on previously unseen data, that is, how well its performance *generalizes* (Marsland, 2009, pp. 66–67). It is also possible that the model *overfits*. When this happens, the model learns the patterns in the training data too well, and the model's ability to generalize has gotten weaker, not better. The validation dataset is optional; it can be used for model selection. The exact method used for splitting the data depends on the application. If there are only a small number of data samples available, it is recommended to do a k-fold cross-validation. This involves partitioning the data set into k parts and training the model k times, each time holding out different folds for testing (Marsland, 2009, pp. 67–68). When sampling the data or generating the folds, it is important to keep the samples representative, for example, through stratified sampling. A special case is the leave-one-out cross-validation, where the test set contains only one sample and the rest of the samples are used to teach the model. This is repeated for all samples in the data set.

4.1 Clustering

Sometimes the data set does not have pre-existing labels that indicate the classes of the data points. Still, we would often like to know if different categories can be

recognized from the data. Unsupervised clustering algorithms can create models that group the data in a way that "natural" groups of similar objects, or *clusters*, are formed. See Figure 5 for an illustrative example.

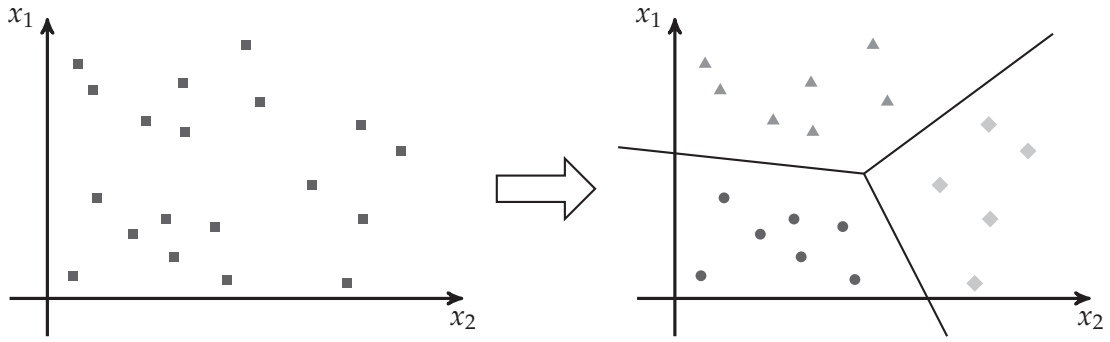


FIGURE 5 Clustering with three clusters and two inputs x_1 and x_2.

Informally, the clusters are formed of data points with small inter-point distances compared with distances to points in other clusters (Bishop, 2006). Thus, choosing the right *distance measure* is very important as it directly influences the formation of the clusters. Even if the canonical ℓ^2 norm is used, a care should be taken to scale the data dimensions so that they are commensurate. Furthermore, as the quality of the resulting clusters is more or less subjective, the choice of the clustering algorithm is important. An in-depth discussion of different clustering algorithms and their taxonomy is beyond the scope of this overview, but the interested reader is referred to the surveys by Xu and Wunsch (2005) and Xu and Tian (2015). However, two popular clustering algorithms, *k*-means and DBSCAN, that were used in the included papers are briefly introduced.

The *k*-means algorithm is perhaps the best-known of the partitional clustering methods. In *k*-means, the partitions are defined by cluster centroids $\boldsymbol{\mu}_i$, and the number of these cluster centroids, k, needs to be known a priori. The algorithm works by iteratively updating the positions of these centroids so that, ultimately, the partitions minimize the within-cluster sum of squares (Bishop, 2006, p. 424). The algorithm essentially partitions the space to Voronoi cells which means that the resulting clusters are always convex, limiting its applicability.

Another popular clustering algorithm is DBSCAN (Ester et al., 1996). Unlike *k*-means, where the clusters always form around the cluster centers, DBSCAN clusters can be of arbitrary shape. If a data point is at least minPts points inside a radius of ϵ it is marked a *core point*. If a data point is not a core point but is within ϵ from a closest core point, it is an *edge point* belonging to the same cluster as the core point. A point that is neither core nor edge point is an *outlier*. DBSCAN is a rather robust algorithm, and with carefully set values for the minPts and ϵ parameters, it can produce aesthetically pleasing clusters.

There are multiple examples where clustering techniques are used in vehicle routing. It is, for example, usual to tackle very-large-scale problem instances with a divide-and-conquer strategy. Sáez et al. (2008) adopted a fuzzy *k*-means algorithm for systematic zoning in solving dynamic PDP problems, and, similarly, Dondo and Cerdá (2007) used a preprocessing phase that clusters the orders

in a multi-depot heterogeneous fleet VRP with time windows (MDHFVRPTW). Hence, clustering can be used to divide a large VRP instance to a more manageable subproblems. An estimate for the number of clusters in a routing problem can be made, for example, by calculating rough lower and upper bounds for routes needed to satisfy all of the model constraints. Gacias et al. (2012) considered both spatial and temporal clustering to suggest constraints that should be relaxed to find a feasible solution.

4.2 Dimension reduction

Many popular ML algorithms do not perform well with high-dimensional data. This may manifest as poor predictive performance or as high computational requirements of the algorithm. The effect is known as *"the curse of dimensionality"* of high-dimensional datasets (Bellman and Dreyfus, 1962), which means that the volume of the feature space grows rapidly with increasing dimensionality. As a result, the dataset becomes too sparse for learning. Additionally, it is hard to devise a relevant distance metric for a high-dimensional space (Domingos, 2012).

Dimension reduction is used to address this issue. The goal in these methods is to find a mapping to a lower dimensional space that minimizes the topographical distortion. In other words, dimension reduction techniques try to find a low-dimensional representation for the high-dimensional vectors that preserves the structure and the information of the dataset. This task can be achieved through feature selection (which is discussed later) or through a number of different data transformations. Two such basic data transformation techniques are principal component analysis (PCA) and linear discriminant analysis (LDA). Of these, PCA finds a linear projection that minimizes the squared distance between the data points and their projections and while doing so also maximizes the variance of the projected data (Bishop, 2006). Due to its solid statistical properties, it is a popular method in data analysis and visualization.

Multi-dimensional scaling (MDS) methods (Cox and Cox, 2000) do not require the coordinates of the data points to be known. It relies on the information of the dissimilarity matrix $\Delta^{N\times N}$, which contains (dis)similarity scores between all pairings of data points. MDS methods generate output embeddings for the data points in a lower dimensional space $y_i \in \mathcal{R}^L$ so that the structure implied by the matrix Δ is closely preserved. A popular MDS algorithm is SMACOF, which does stress minimization using majorization (Borg and Groenen, 2005). The algorithm iteratively minimizes the stress function σ, which is usually a weighted sum of squared residuals. In the domain of vehicle routing, MDS has been used, for example, by Feng et al. (2014) to transform arc routing problems into vehicle routing problems, and by Ventresca et al. (2013) to visualize genetic algorithm operator pairings.

4.3 Classification

Classification involves predicting a label (category) for an input, which is usually given in a form of a feature vector. A useful concept in classification tasks is that of a *decision boundary*, which is a hypersurface that divides the input space into two subspaces. By learning the parameters defining these surfaces, the model can determine the *class label* for any input by assigning it according to the subspace of the data point. The model is called a *classifier*, and the number of decision boundaries that need to be learned depends on the number of class labels.

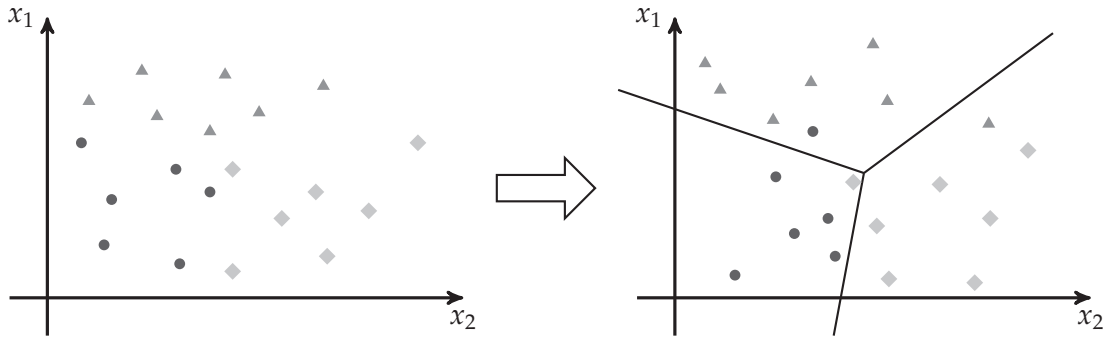


FIGURE 6 Three-label classification with two inputs x_1 and x_2.

Figure 6 illustrates a three-label classification case. As one can see, the *classification accuracy* in this example is not perfect as some of the test data will be classified incorrectly with the given decision boundaries. Improving the accuracy would be possible with a better fit of the model or by using more complex decision boundaries.

Popular classification algorithms such as *k*-nearest neighbor (k-NN), decision trees (e.g., CART), support vector machines (SVM), random forests (RF), and artificial neural networks (ANN) are interchangeable to some extent (see, e.g., Kotsiantis et al., 2007; Marsland, 2009). However, their suitability for a specific learning task varies greatly, and it is recommended practice to experimentally verify which one should be used with a given application problem (Kotsiantis et al., 2007).

4.4 Feature selection

In this section, we review some recent and relevant publications related to feature selection and extraction. However, because this is a large and active research topic, it cannot be fully covered here. Still, it is an important topic because machine learning always requires a numerical representation of the examples as an input, usually in the form of a *feature vector*. Here, a *feature* is synonymous with an input variable or attribute for a machine learning algorithm. These features describe some properties of the samples, and the values can be binary, ordinal,

categorical, or continuous. Sometimes, the samples are already in a format that can be fed directly into the machine learning algorithms, but usually an additional *feature engineering* phase is required to refine and summarize the data into a more convenient numerical format (Domingos, 2012).

Feature engineering involves constructing feature variables from the raw data. Getting this step right is vital to the successful construction of statistical or machine learning models. In fact, Domingos (2012) claims this is the differentiating factor between successful and failed ML applications. However, engineering useful features and, thus, finding a good data representation (that is, a useful set of features) for the raw data is a challenging domain specific task that requires experimentation and domain expertise (Guyon and Elisseeff, 2006). Furthermore, experimentation is needed to see how data preprocessing, features, and machine learning interact. These interactions can sometimes be surprisingly complex and delicate, which illustrates why feature engineering is sometimes considered to be more of a black art than an exact science (Domingos, 2012).

According to Guyon and Elisseeff (2006), one should be aware of information loss in the feature extraction phase and err on the side of being too inclusive, rather than risk omitting useful features. Thus, experimentation with both low- and high-level features is important (low being close to the original raw data and high being features constructed from other features). However, this may lead to the manifestation of the previously mentioned curse of dimensionality (Bellman and Dreyfus, 1962). The curse causes the learning task to become considerably more difficult and computationally intensive to construct. There is also the danger that a model is overfit or made less comprehensible (Zhao et al., 2010). To overcome these challenges caused by *irrelevant* and *redundant* features in machine learning applications, many *feature selection* (FS) approaches have been proposed. The aim in feature selection is to improve the performance and accuracy of the models by recognizing a good subset of important features from the original full feature set. Selecting a good subset is a multicriteria search problem that needs to balance between minimizing the number of features and maximizing the model accuracy and generalization ability.

Many techniques have been proposed to solve this task. They can be divided into three broad categories: *filter*, *wrapper*, and *embedded* FS techniques. The filter approaches are usually fastest as they only consider the information in the dataset itself. For example, the correlation-based feature selection (CFS) method of Hall (2000) uses feature correlations with the class label and a heuristic search strategy to recommend a set of relevant features. Similarly, in the minimum-redundancy-maximum-relevance (mRMR) method from Peng et al. (2005), the feature ranking is based on mutual information between the features and correlations between the features and the class labels. The wrapper-based methods test the candidate feature subsets on the actual target ML algorithm. This can be done using, for example, forward selection where features that improve the accuracy most are iteratively added to the set, or with backwards elimination where irrelevant features are removed one by one. As one can expect, this approach can be computationally very intensive on large feature sets. The embedded methods

are similar to wrapper methods, but they rely on the features of specific learning algorithms and use their internal information to rank or select the features. Ensemble methods, such as random forests, are a natural choice, because with them, the feature-importance measures can be calculated using the final model(s) (Saeys et al., 2008).

Besides being an important part of the preprocessing in machine learning (Saeys et al., 2008) in that they have the potential to increase the prediction accuracy and speed of the machine learning models, any of the aforementioned feature selection approaches can be used to gain a better understanding of the samples and the feature extraction process (Guyon and Elisseeff, 2006). As such, feature selection is also a powerful data analysis tool for gaining a better understanding of the underlying phenomena.

5 META-OPTIMIZATION

All heuristics are to some extent problem dependent. Their performance greatly depends on how well they can exploit the characteristics of the specific problem being solved (Arnold and Sörensen, 2019b). Hence, to achieve a good performance, algorithms need to be carefully selected and tuned for the specific problem under consideration. This makes applied work sometimes more of an art than a science. The task involves recognizing the idiosyncrasies of the problem instance class, comparing them against the strengths and weaknesses of the different algorithms, and finally, using experiments and intuition to fine-tune the most promising algorithms to solve the problem.

As one can expect, building such expertise and intuition can be difficult. Developers rarely provide any guidelines on how to select and configure their proposed algorithms (Brazdil et al., 2009). Furthermore, comparative experimental studies concentrate on showing which algorithm is better under a specific set of circumstances and often fail to generalize or consider the implications of the experimental results (Hooker, 1995; Barr et al., 1995). Most importantly, the familiarity of the researchers with some of the algorithms can have a significant effect on the results; there exists a severe danger for confirmation bias in such algorithm comparisons due to the varied manipulations and manual fine-tuning efforts the algorithms receive (Ansótegui et al., 2009). Also, the extent of these manual efforts is rarely disclosed, and it can be argued that manual selection and configuration of optimization algorithms can lead to irreproducible results (Eggensperger et al., 2019).

Fortunately, relying on expert intuition is not the only option for tackling the problem of algorithm suitability, configuration, and selection. Analogous to *meta-learning* in the machine learning literature (Brazdil et al., 2009), *meta-optimization* involves optimizing the selection and configuration of optimization algorithms. The basic idea in meta-optimization is to learn a mapping between the characteristics of optimization tasks and suitable optimization algorithms and their parameter configurations. Ansótegui et al. (2009) argued that using such techniques should be a recommended practice when planning, making, and reporting computational studies. Furthermore, it has been shown that machine learning can

improve the performance of existing optimization algorithms (Birattari, 2009). Hence, applying meta-optimization to optimization problems with significant practical importance, such as the vehicle routing problem described in Section 2, seems to offer a promising way to increase the performance and robustness of the solution methods.

Meta-optimization with its applications, automatic algorithm configuration, parameter control, and algorithm selection is closely related to the research on *hyper-heuristics* (Burke et al., 2013; Pillay and Qu, 2018). As noted earlier (Section 2.2.3), the trend in solving VRPs has shifted towards hybrid metaheuristics that combine the elements of several solution techniques. The rapidly developing field of hyper-heuristics promises to automate this hybridization; instead of proposing and experimentally validating yet another hybrid algorithm, one can build a system that is able to automatically discover a good combination of lower level heuristics and algorithmic ideas for a specific set of problem instances. In comparison with metaheuristics, hyper-heuristics work in the heuristic space instead of the solution space. That is, hyper-heuristics selects or generates the low-level heuristics from the heuristics space, which are then used to solve the original problem by searching the solutions space (Pillay and Qu, 2018).

The emphasis of hyper-heuristics is with the on-line selection and configuration of heuristics during the solving process, whereas automatic algorithm configuration and selection are off-line techniques. However, this dissertation only considers off-line meta-optimization, which leaves hyper-heuristics out of the scope of the conducted research. Still, is is useful for the reader to recognize the advances in hyper-heuristics research in parallel with that of algorithm configuration and selection (Smith-Miles, 2009; Pappa et al., 2014). While the ideas on these fields have developed to some extent independently, Pappa et al. (2014) calls for cross-fertilization among the fields.

The next sections concentrate on topics related to automatic algorithm configuration and algorithm selection. Also, there seems to be a research gap that calls for a holistic understanding and exploitation of the VRP instance and solution structure. Therefore, as the last topic of this chapter on meta-optimization, we discuss meta-features, VRP and TSP feature extraction, and VRP fitness landscape analysis.

5.1 Automatic algorithm configuration

Most state-of-the-art VRP algorithms have free parameters, and the choice of values for these parameters can have a significant impact on the performance of the optimization algorithm. This applies to exact solvers based on linear programming (Hutter et al., 2010) as well as to approximate algorithms based on heuristics or metaheuristics (Pellegrini and Birattari, 2007).

Some of the parameters are fixed by the programmer of the algorithm, but some are left to be set by the user, to allow adaption of the solver to the prob-

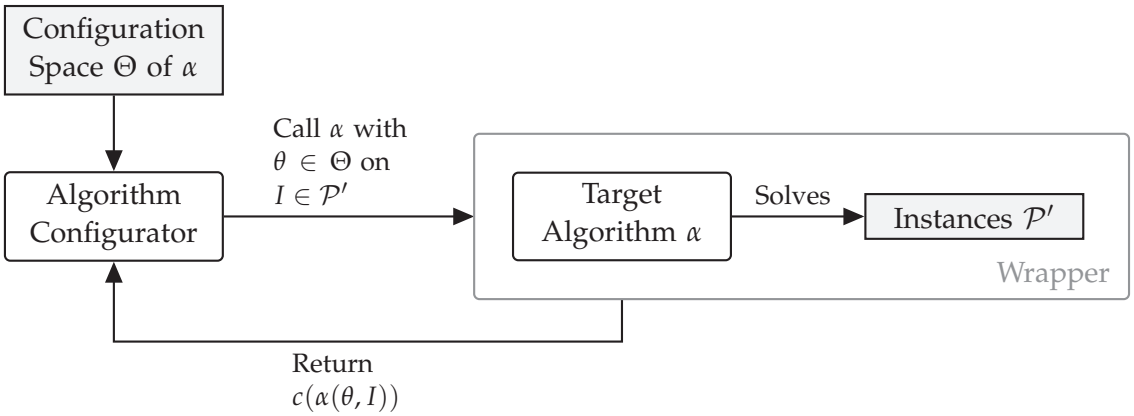


FIGURE 7 Algorithm configuration workflow (Eggensperger et al., 2019).

lem being solved (Hutter et al., 2009). The algorithm parameters adjust how the search for the optimal solution is conducted and, in the case of heuristics, how the exploration and exploitation in the search space are balanced. Traditionally, these algorithm parameters have been set manually using expertise and experimentation (Hutter et al., 2009), which takes considerable time and effort. Furthermore, experimenting with different algorithm parameters is tedious, and without the necessary experience and expertise in the optimization field, frustrating with numerous caveats in the interaction between the parameters, problem instances, and algorithms (Barr et al., 1995). For example, a metaheuristic can be composed of multiple algorithmic components (e.g., in the form of multiple built-in local search operators) and of a set of free parameters that enable and disable them. Adapting metaheuristics to a certain set of problems often requires serious configuration effort that involves selecting the correct components and experimentally finding suitable values for the parameters (Birattari, 2009, p. 3).

Kadioglu et al. (2010) argued that exposing more parameters to the user of the solver is better. Their rationale was that this allows the user to fine-tune the solver to specific needs. The downside is that the multitude of parameters can make the effective use of the algorithm difficult, especially if there are complex interactions between the parameters themselves. In fact, Cordeau et al. (2002, p. 514) argued that VRP "algorithms that contain too many parameters are difficult to understand and unlikely to be used." They suggested two strategies to combat this: fix them to some predefined meaningful value or allow the methods to self-adjust the parameters during the optimization. Fortunately, there is also a third option.

The tedious task of parameter configuration can be automated with the use of *automatic algorithm configuration* (AAC). AAC addresses the off-line task of finding a good set of parameter values, or a *parameter configuration*, prior to using the *target algorithm* in production. Because of the knowledge and effort requirements of manually finding good sets of parameters, manual configuration efforts often lead to inferior performance compared to AAC (Hutter et al., 2009). Hence, the increase in algorithm performance is perhaps the most obvious benefit. In

their experiments, Kadioglu et al. (2010) demonstrated algorithm speed increases ranging from 10- to almost 300-fold compared to the parameter defaults when configuring solvers for the well-known satisfiability problem (SAT). The additional benefit of AAC to the target algorithm is that it makes the algorithm what Battiti et al. (2010, p. 5) call self-contained; its quality and performance can be measured independently from the designer. Hence, the algorithm designer and his/her extensive expertise in tuning the algorithm manually no longer cause bias to the results. This allows objective evaluation and comparison of algorithms.

Algorithm configuration can be formalized as follows: given an algorithm α and its configuration space Θ, find a *parameter configuration* $\theta \in \Theta$ which allows the algorithm to yield the best possible solution for a problem instance $\mathbf{I} \in \mathcal{P}$ or a set of problem instances $\mathcal{P}' \subset \mathcal{P}$. Assuming the objective is to minimize the cost metric c, and we are configuring a deterministic algorithm, the task of finding the optimal parameter configuration θ^* can be given as (Eggensperger et al., 2019):

$$\theta^* \in \arg\min_{\theta \in \Theta} \sum_{I \in \mathcal{P}'} c(\theta, I). \tag{2}$$

In the case of a deterministic algorithm, the cost with a given parameter configuration is acquired simply by calling the algorithm with the candidate parameters and the problem instance as an input (see Figure 7). Depending on the cost metric c, the best parameter configuration for a *target algorithm* might be the one that minimizes the algorithm runtime or produces the best solution quality.

According to Hepdogan (2006, p. 23) an ideal AAC technique should be fast, use as few evaluations on the target algorithm as possible, be robust and suitable for many different kinds of optimization algorithms, and be able to produce algorithm configurations in a repeatable fashion. The tool should be easy to use and have very few parameters so that using it does not require extensive setup or configuration time. Also, the method should outperform simple trial-and-error parameter tuning and pure random search. A good practical guide on applying AAC methods has been recently published by Eggensperger et al. (2019). There are also many powerful AAC tools that are available that can be used to improve the performance and efficiency of existing algorithms. These include SMAC from Hutter et al. (2011), GGA from Ansótegui et al. (2009), and iterated F-Race from López-Ibáñez et al. (2016).

Coy et al. (2001) was one of the first to raise the awareness of the importance of configuring the parameters of VRP metaheuristics, and Van Breedam (2002) presented a systematic analysis of the parameters of many classical VRP heuristics. The study from Pellegrini and Birattari (2007) illustrated the benefits of AAC through a series of experiments on six VRP metaheuristics and showed that the configuration significantly improved the performance of the algorithms. Saremi et al. (2007) proposed a systemic approach based on the design of experiments for tuning a memetic metaheuristic for VRPB and Ceschia et al. (2011) used F-Race to select best local search operator composition for tabu search metaheuristic solving HFVRPTW. Also, while it is hard to estimate its significance in producing the state-of-the art results on "rich" VRPs, Vidal et al. (2013a) used

CMA-ES (Hansen, 2006) to configure their hybrid algorithm. The results of Pellegrini and Birattari and those in Paper PVI suggest that AAC might have played a significant role in achieving their impressive results. Recently, Labadie et al. (2016) noted that the trend of building hybrid metaheuristics leads to an increase in the number of free parameters. This emphasizes the importance of a solid design of experiments (Barr et al., 1995) and AAC. Lastly, for an industry perspective on configuring VRP algorithms, please refer to the Becker et al. (2005).

It should be noted that in the field of evolutionary computing, AAC usually goes by the name of *parameter tuning* (De Jong, 2007; Eiben and Smit, 2011), whereas the similar task involving the configuration of machine learning algorithms is known as *hyperparameter optimization* (Bergstra et al., 2011). The approach in all of these is similar and the proposed techniques are to a large extent interchangeable. However, the research on the topic in these fields currently seems to be quite disjoint.

5.2 Algorithm selection

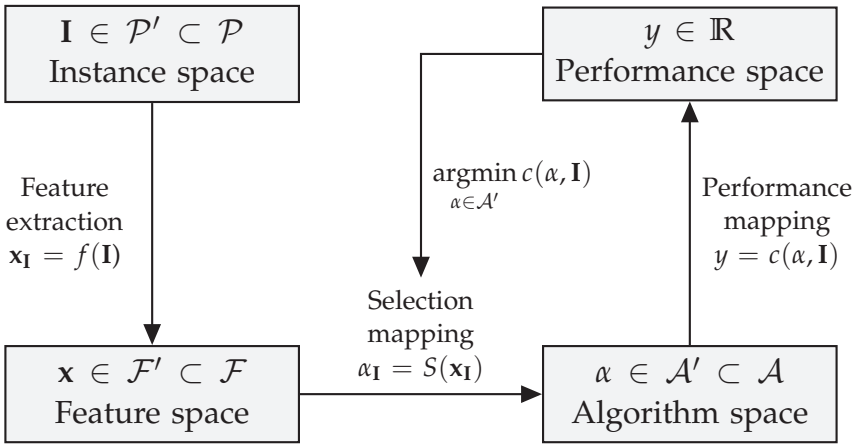


FIGURE 8 Model for algorithm selection (Rice, 1976; Smith-Miles, 2009).

The *algorithm selection* (AS) problem can be stated as follows (Rice, 1976): Given a portfolio of algorithms and their measured performance on a set of problem instances, which algorithm or algorithms should be used to solve a previously unseen problem instance? The basic idea in automating this is to use machine learning to learn a model which is then used to predict the best among a *portfolio* of different algorithms (Bischl et al., 2016). Alternatively, the model can be used to determine the ordering and time budget allocation for the different methods by predicting a ranking of the algorithms. In both approaches, the model exploits the varying algorithm performance over a set of problem instances (Bischl et al., 2016). As a result, algorithm selection allows concentrating the computational effort on those algorithms that are expected to show a good performance on a specific problem instance.

Next, a more formal definition is given. Assume that we have a set of algorithms $\mathcal{A}$ that can be used to solve problems from the problem space $\mathcal{P}$. Feature extraction for a problem instance $I \in \mathcal{P}$ can be formally expressed as a feature extractor function that transforms the problem instance to a feature vector $\mathbf{x}_I$ of n feature values, $f : \mathcal{P} \rightarrow \mathbb{R}^n$. Thus, the performance data for the algorithm selection task can be expressed as $\mathcal{D} = (\mathbf{x}, y)$. If we denote the algorithm with α, and $\alpha \in \mathcal{A}'$ where $\mathcal{A}'$ is the portfolio of algorithms, we can use the performance data to learn an algorithm recommendation mapping $S : \mathcal{P} \rightarrow \mathcal{A}'$ which can, together with the feature extraction, be used in the on-line task of predicting the best algorithm for a previously unseen problem instance $I_n \in \mathcal{P}, I_n \notin \mathcal{P}'$. Figure 8 illustrates the relations between the different spaces related to algorithm selection: problem instance space $\mathcal{P}$, problem instance feature space $\mathcal{F}$, algorithm space $\mathcal{A}$, and algorithm performance measure space.

Algorithm selection has been shown to produce state-of-the art results in many combinatorial optimization tasks (see, e.g., Xu et al., 2008). In his survey of algorithm selection targeting combinatorial optimization problems, Kotthoff (2014) recognizes that the quest for significant performance improvements has shifted the focus away from proposing new algorithms and towards investigating how to recognize the most suitable existing algorithm. Additionally, like AAC, algorithm selection can help researchers to make fairer algorithm comparisons and obtain understanding of the strengths and weaknesses of the different methods (Bischl et al., 2016).

However, one should note that there are some considerations in applying this approach (Kotthoff, 2014). It is critical to use a representative set of problem instances when building the algorithm selection model. Otherwise, there is a danger that the selected algorithms are not the best choices for new, previously unseen, instances. Also, the algorithms may seem to perform well on one set of instances, but the performance of the selected algorithms can be worse than the single best algorithm on another set. Thus, the ability to generalize should always be verified with cross-validation or with a separate testing set. It is also necessary to recognize features that can describe the problem set in the way they capture properties relevant to: a) solving the problems, b) configuring algorithm parameters, c) recognizing a set of mutually similar problems that can share a configured parameter configuration, and d) can predict algorithm performance. This is not always easy as feature engineering is always a domain-specific task (Domingos, 2012).

The extensive literature on VRP algorithms shows that the performance of different VRP algorithms varies between problem instances, and no single algorithm is superior to the rest (Tighe et al., 2005). Hence, the main motivation to solve the algorithm selection problem in the VRP context comes from the possibilities in increasing the speed and expressive power of the existing optimization technology (de la Banda et al., 2014). Algorithm selection has been previously applied to VRP by various researchers (Nygard et al., 1990; Tuzun et al., 1997; Tighe et al., 2005; Steinhaus, 2015). Nygard et al. (1990) presented empirical evidence that there was a significant dependency between the relative performance

of four different classical CVRP algorithms and the distribution of the customer points. They achieved remarkably good predictive accuracy through application of several complementary neural network classifiers, which were activated by another meta-meta-level neural network classifier depending on the inputs. Similarly, the study of Tuzun et al. (1997) concluded that using neural networks to predict the most suitable algorithm for a CVRP instance was a viable approach to solving the algorithm selection problem. Tighe et al. (2005) explored the capabilities of an adaptive system where a self-organizing fuzzy control orchestrates a dynamic PDP optimization task by changing the runtime priority of three algorithms. More recently, Steinhaus (2015) proposed using self-organizing maps in a CVRP algorithm selection. The related topic of TSP algorithm selection has also been studied by a number of individuals (e.g., Nallaperuma et al., 2013; Hutter et al., 2014; Pihera and Musliu, 2014).

One recent development is methods that simultaneously choose algorithms and optimize their parameters. An example of such work is the proposed solution for the combined ML classifier selection and hyperparameter optimization problem from Kotthoff et al. (2017). Their extensive experiments showed that a combined selection and configuration approach for classification tasks offers further benefits as opposed to considering these tasks separately.

5.3 Meta-features

In meta-learning (Brazdil et al., 2009), the dataset statistics are characterized by meta-features. They are needed to build the models that can predict the most suitable ML algorithm for a specific dataset. Analogous to meta-learning, the properties of the optimization problem need to be described to the meta-optimization algorithms; in algorithm selection and instance specific algorithm configuration, the task becomes to learn the connections between the performance of the algorithms and the problem instance features (Kanda et al., 2016). Hence, the meta-features allow understanding and detecting the particularities of a specific problem instance. However, the specific number and quality of features depends on the intended application.

The importance of good features cannot be overstated, and similar to any other ML application, they are the key to producing a useful model (Domingos, 2012). Also, a higher volume of data and a richer set of meta-features per data point is expected to result in a better learned model (Arnold and Sörensen, 2019b). Thus, a wealth of problem-specific information needs to be extracted to the heuristics to work optimally. Battiti et al. (2010, p. 14) consider three possible sources of such information that can be used in algorithm selection and tuning: *problem type related*, *instance specific*, and those based on the current position and the trace through the *search space*.

The *problem-type-related features* are usually categorical or binary. They can indicate if some certain modeling constructs are needed or not. In practice, these

features can be derived from different classifications and taxonomies of VRP variants. An example of such work is the formal classification language of Desrochers et al. (1999). However, such type-related meta-features are relevant only with multi-attribute VRP solvers that are able to solve several VRP variants such as the one from Vidal et al. (2014). Furthermore, such encoding of the problem variant in a format that can be given to a ML algorithm is currently a manual process, and we are not aware of any prior work on automating the detection of VRP variants or attributes from the raw data that specifies the problem instance.

Regarding the *problem-instance-specific features*, VRP shares the typical list with the other COPs: the size of the problem, type of numeric inputs, and the size and quantity of the constraints can can be used as meta-features (Tighe et al., 2005). There exist relatively few prior works on VRP feature extraction. Several propose features for VRP algorithm selection, but not all papers discussing the topic use meta-features. Instead, it seems typical that parameters of a problem instance generator are used as inputs for the learning algorithm (e.g., Tuzun et al., 1997; Tighe et al., 2005), which severely limits the applicability of the proposed studies. An exception to this is the work of Nygard et al. (1990) who proposed three feature extractors that were invariant under rotations of the problem and measured the distance from the depot, clustering, and angular dispersion of the customers. Each extractor produced values for 10 frequency classes, and so the total number of features was 30. The dimensionality of the input data was further reduced to a feature vector with only four values using a neural autoencoder. (Steinhaus, 2015, p. 66) proposed a set of 23 features capturing the spatial distribution of the customers, depot position, average number of customers served by a single route, and the demand structure. Additional features can be adapted to VRP from the related TSP literature (Smith-Miles and van Hemert, 2011; Mersmann et al., 2013; Nallaperuma et al., 2013; Hutter et al., 2014; Pihera and Musliu, 2014; Kanda et al., 2016).

For *describing solutions and the search space* of VRPs, we give two examples. Jin et al. (2014) described a co-operative parallel tabu search metaheuristic that uses clustering of solutions in the solution pool and bookkeeping of cluster metrics to balance between intensification and diversification. Clustering also warranted Jin et al. to define a solution similarity measure. Later, Arnold and Sörensen (2019b) proposed 10 features to characterize solutions. The features that are used to describe VRP solutions also contain information about its search space and can, thus, be used to study its structure. Generally, such information is gathered through solution attempts with a simple heuristic algorithm. The algorithm leaves a trace of candidate solutions, which may reveal rich details and patterns about the search space, and the features derived from this information may have high predictive power (Asta, 2015).

Such an approach is related to the topic of fitness landscape analysis (FLA), which can be used to analyze and increase the performance of metaheuristic algorithms (Pitzer and Affenzeller, 2012). FLA aims to characterize the problem structure through analysis of the search space and the fitness and neighborhood functions (Marmion et al., 2013). The metaphor in FLA is about imaging a two-

dimensional search space as a landscape with valleys, peaks, canyons, and other such features. The analogy helps to give an intuition on how heuristic algorithms operate, but it can be misleading in higher dimensions (Pitzer and Affenzeller, 2012). FLA can allow the problem to be understood better by the researcher, and many FLA techniques can provide useful meta-features for the ML algorithms. Typical features measured from a fitness landscape are modality (the number and distribution of local optima), existence of basins of attraction, barriers and phase transitions, landscape walks, and ruggedness. For a generic survey on FLA, refer to Pitzer and Affenzeller (2012).

There exist some studies where the fitness landscape of VRPs have been explored using the FLA technique. Kubiak (2007) presented an approach for statistical fitness-distance analysis of the CVRP instances. He proposed multiple metrics for measuring the distance between solutions and observed that most of the local optima seem to be clustered, and that this global structure of the search space might explain the success of some metaheuristics. Czech (2008) measured the ruggedness of VRPTW fitness landscapes that allowed them to rank the problem instances according to their difficulty and determine the parameters of a SA metaheuristic. Pitzer et al. (2012) and Marmion et al. (2013) used asymmetric CVRP as a case study in FLA. Marmion et al. used the relocate and exchange neighborhoods and a giant-tour-solution encoding and proposed a solution-to-solution distance measure based on a robust on-hold greedy algorithm. They studied the valley structure and the ruggedness of eight problem instances and concluded that there are definite differences in the landscapes as observed through the different local search operators. Ventresca et al. (2013) used similar analysis methods to discriminate between difficult and easy VRPTW and CVRP instances. Interestingly, instance clusters recognized in their study tended to contain only one type of local search operator. Based on this observation, they proposed an approach similar to instance-specific algorithm configuration (Kadioglu et al., 2010) involving performance profiles for the GA.

When new meta-features are proposed, it is important to consider the computational effort involved in producing the feature data. Extensive analysis of a problem instance can easily counteract any time saved in predicting the algorithm parameters or using the most suitable solution algorithm (Tighe et al., 2005). This topic has been discussed in depth, for example, by Kanda et al. (2016). They note that the computational cost of generating the meta-features should always be smaller compared to the combined cost of running several optimization algorithms. Otherwise, there is a danger that meta-optimization cannot provide any efficiency gains. Hence, the analysis of the feature importance and examination of their computational effort, as done by Kanda et al., is extremely important. A useful tool to evaluate the feature importance is feature selection (Section 4.4), which allows eliminating redundant and irrelevant meta-features. This approach was demonstrated by Pihera and Musliu (2014), where they reduced the dimensionality of feature vectors that characterized TSP instances. However, prior to our own work on the topic in Paper PV, we are not aware of earlier studies where modern feature selection methods have been applied in VRP research.

6 SUMMARY OF THE PUBLICATIONS

This chapter summarizes the original articles included in this dissertation and lists the individual contributions with special emphasis on the software aspects. The papers are not presented strictly in the order they were published. Instead, we proceed from the more general, conceptual, and exploratory works towards the more detailed topics. The last two articles on algorithm selection and automatic algorithm configuration of VRP (meta)heuristics bring the discussion back to applications of meta-optimization in the context of vehicle routing systems.

6.1 Paper PI: Automatic Customization Framework for Efficient Vehicle Routing System Deployment

This research was presented at the CM3 Conference on *Computational Multi Physics, Multi Scales and Multi Big Data in Transport Modeling, Simulation and Optimization*, held May 25-27, 2015, in Jyväskylä, Finland. The accompanying paper was peer-reviewed and published in the book *Computational Methods and Models for Transport - New Challenges for the Greening of Transport Systems*.

Aim

The aim of this work was to recognize the issues in customization and tailoring of vehicle routing systems which are holding back the widespread adaptation of vehicle route optimization technology among small- and medium-sized transportation companies. Furthermore, we aimed to provide a vision for an automated configuration workflow to address these issues. Of particular interest was the data flow and related transformations inside a typical vehicle routing system. Thus, we did not seek to recommend proven solutions for automating the design of the recognized transformations but rather explored alternative ways to increase the degree of automation in configuring such systems.

Method

Our main method in this study was to conduct well-constrained literature surveys on multiple related topics. By analyzing the trends in VRP research, we showed that the state-of-the-art has been progressing towards unified models that can simultaneously capture many attributes of "rich" vehicle routing problems and towards adaptive and learning algorithms in solving them. These trends and the earlier work (Puranen, 2012) allowed us to assume access to a generic vehicle routing system with flexible modeling capabilities and a selection of modern heuristic algorithms.

In order to present a concept of data flow and data transformations, we recognized the typical modules inside a VRS. We discussed different options for managing the transformations and argued that the one involving automatic customization would be the most natural one, especially considering the emerging VRP research trends. Similar ideas have also been presented by other authors (e.g., de la Banda et al., 2014) in related research fields around the same time, which seems to further validate our choice.

Results and contribution to the whole

We presented descriptions and related literature for automating the configuration of the data transformations in interpreting the input data, inferring the optimization model, selecting and configuring the suitable algorithms, solving the problem, interpreting the results, and producing a correctly formatted plan.

The descriptions of the transformation steps and related automation opportunities were complemented with some preliminary experimental results. The benefits of configuring three CVRP metaheuristics on a set of real-world inspired problem instances with SMAC from Hutter et al. (2011) and iterated F-race from López-Ibáñez et al. (2016) were demonstrated, and the concept and preliminary results of our data import automation prototype were presented.

This study's main contribution to this dissertation was the customization framework for automating the data transformation operations inside a vehicle routing system. Adapting a vehicle routing system to the particularities of a specific strain of VRP instances using data-driven customization seemed to be a promising research direction, and we expected it to have significant practical significance. We based our argument on the existing literature, and the preliminary results of configuring the algorithms and automating a part of the data integration process.

Summary of individual contributions

- Recognized the possibilities of a data-driven customization process based on machine learning and meta-optimization. It is expected to make the tailoring and deployment of vehicle routing systems cheaper, easier, and less labor intensive.

- Proposed a conceptual model for the seven data transformations inside a typical VRS, and described how they can be used as variation points of a software product line.

- Provided surveys, suggestions, and in two proofs-of-concept (for algorithm configuration and data integration) on how to use machine learning to automatically configure the data transformations.

6.2 Paper PII: Solution Space Visualization as a Tool for Vehicle Routing Algorithm Development

This research was first presented at the FORS40 Workshop on Optimization and Decision-making on August 20-21, 2013. The workshop was held at Lappeenranta University of Technology to celebrate 40 years of the Finnish Operations Research Society (FORS). Later, a peer-reviewed paper was published in the proceedings of the workshop.

Aim

The aim was to understand how and why VRP heuristics work has still received surprisingly little attention (Corstjens et al., 2019). Learning and reasoning about the decisions the algorithms make during a search is sometimes difficult. Fortunately, visualization has proved to be a powerful tool for understanding and gaining intuition on the operating principles behind a heuristic algorithm (Halim et al., 2006). In this paper, we proposed a technique to visualize a VRP solution and local search neighborhood landscapes. The main goal of this work was to propose a visualization technique that could be used as a viable tool by VRP algorithm designers and developers.

Method

We proposed a technique that uses multidimensional scaling to find a 2D representation for the highly multi-dimensional solution space of vehicle routing problems. Each point in this visualization space corresponds to one feasible or infeasible solution to the original problem instance. Developing the visualization method involved creating an efficient procedure to enumerate all solutions for small CVRP instances, experimentally finding a suitable measure for solution dissimilarity, experimenting with multidimensional scaling algorithms, and selecting a tool to visualize the resulting data. We also implemented the necessary tooling to calculate multiple statistical fitness landscape measures in the original search space and in the visualized space in order to verify the accuracy of the visual representation.

Results and contribution to the whole

In the spirit of constructive research approach (Lukka, 2003), we created a technique capable of visualizing vehicle routing problem solution landscapes. This allows a visual examination of VRP landscapes and analysis of how local search operators move through it. The work improved our understanding of the fitness landscapes of vehicle routing problems and the behavior of local search heuristics. Later, these tools helped us to examine and verify the correct operation of the classical routing algorithms implemented in Paper PIV. In this study, we also made a novel contribution by proposing a method for verifying the accuracy of the representation via the use of three relevant statistical fitness landscape measures. These features were later used in describing vehicle routing problem instances as part of our comprehensive set of feature extractors for vehicle routing problems (PIII).

Summary of individual contributions

- Introduced a technique for visualizing vehicle routing problems and local search trace neighborhoods.
- Proposed an analysis technique that uses fitness landscape analysis measures to verify that the visualization preserves the relevant features of the search space.
- Empirically derived a formula for the number of solutions to a symmetric CVRP.
- **Software:** Designed and programmed the visualization tool to draw interactive 3D plots of small problem instances and of local search traces.
- **Software:** Designed and programmed a fast open-source CVRP solution enumerator[1] supporting three different solution generation methods.

6.3 Paper PIII: Feature Extractors for Describing Vehicle Routing Problem Instances

This work was presented at the 5th Student Conference on Operations Research (SCOR16) held at 8-10th April 2016 in Nottingham, UK. The talk selected as the runner-up for the best talk at the conference and the peer-reviewed paper was later published in the conference proceedings.

[1] https://github.com/yorak/cvrp-solution-generator

Aim

Access to a set of high-quality meta-features is a prerequisite to efficient algorithm runtime prediction, algorithm parameter value prediction, and algorithm selection. This feature set must be diverse, comprehensive, and relevant to the machine learning task. A good set of features is capable of capturing the idiosyncrasies of the problem instances.

In this study, our aim was to find a set of VRP meta-features that could be used in VRP-related meta-optimization tasks. To that end, we needed to build a feature extractor framework intended for numerically describing the well-known problem instances from the literature as feature vectors.

Method

Based on a literature survey on the topic of describing TSPs and VRPs, a set of 76 feature extractors was adapted to characterize VRP instances. These extractors can produce a feature vector with 386 feature values for a single CVRP instance. The features were organized into groups: those related to node distribution (10 extractors), features calculated from the minimum spanning tree (3 extractors), features from probing the problem instance through 20 solution attempts with a heuristic (12 extractors) and branch-and-cut solver (4 extractors), geometric features based on an enclosing rectangle and convex hull (5 extractors), nearest-neighbor graphs (22 extractors), and VRP-domain-specific features (11 extractors). The time used to calculate the aforementioned features add a further nine features. For probing, we used a modified VRPH (Groër et al., 2010) solver and the open-source SYMPHONY (Ralphs and Güzelsoy, 2005) branch-and-cut solver with a time cutoff.

The feature extraction required some preprocessing measures. The distance matrix was adequate for describing the distribution of the customer nodes on the plane, but some features, such as those concerning the convex hull, also required the node coordinates to be known. However, there were benchmark instances that were defined using only a distance matrix. In cases where coordinates were missing, or given as geographical coordinates, their locations were approximated using multidimensional scaling (MDS) (Borg and Groenen, 2005). We also scaled the problem instance nodes to the range $x, y \in [(0, 400), (0, 400)])$ as suggested by (Smith-Miles and van Hemert, 2011) but retained the aspect ratio of the problem.

The proposed set included many features that were used to describe VRPs for the first time. For example, we proposed including the silhouette score, which measures how appropriately the data has been clustered, and an autocorrelation length of a random walk in the solution space.

The quality of this feature set was evaluated by clustering a collection of 168 CVRP benchmark instances collected from multiple sources. Also, automatic algorithm configuration of three metaheuristic CVRP algorithms was done using a state-of-the-art instance-specific algorithm configuration tool SMAC (Hutter et al., 2011).

Results and contribution to the whole

The VRP feature set presented in our study was significantly larger than those previously found in the literature. Clustering of the benchmark instances showed that the proposed feature set has a good discriminative power. Also, the results of the instance-specific algorithm configuration were promising. We were able to further improve the quality of the solutions by using the feature data, and in many cases the difference was statistically significant. We recognized that the dimensionality of the full feature set was too high for most tasks, and we had to use PCA to bring it down to more manageable levels. However, our experiments revealed the need for more advanced feature selection that could fully utilize the information captured by the features.

Our initial experiments with VRP meta-learning showed that applying machine learning approaches in this domain had the potential to improve the resulting solution quality and lead to significant algorithm performance improvements, in addition to lifting the burden of tweaking the algorithms from the shoulders of the operations researcher.

Summary of individual contributions

- Provided a literature survey on describing VRP and TSP instances.
- Proposed a comprehensive list of CVRP feature extractors.
- Demonstrated the usefulness of the features through instance-specific automatic algorithm configuration and clustering of problem instances.
- **Software:** Designed and programmed a feature extraction tool with 76 feature extractors that can produce 386 features for a given CVRP instance.

6.4 Paper PIV : Meta-Survey and Implementations of Classical Capacitated Vehicle Routing Heuristics with Reproduced Results

This report currently has a status of an unpublished manuscript.

Aim

Despite their popularity in operations research, only a few VRP algorithms have freely available implementations. Furthermore, even if an implementation is available, replication of results has not been the main focus of such reimplementation work. This also applies to the classical CVRP heuristics that were introduced between 1959 and the early 1990s. This situation makes experimental comparisons between different algorithms difficult and unreliable (Barr et al., 1995).

In addition to the criticism on the empirical testing of heuristics (Hooker, 1995; Hasle and Kloster, 2007), the themes of reproduction and replication of re-

sults and code re-use in the field of vehicle routing research have recently resurfaced (e.g., Cordeau et al., 2002; Sörensen et al., 2019). Simultaneously, the general trend in VRP algorithm research has been towards hybridization and automatic composition of solution methods through hyper-heuristics. Such approaches benefit from recognizing the relative strengths and weaknesses of different algorithmic components and solution approaches. Hooker (1995) has argued that gaining these insights requires us to shift our focus from competitive testing to understanding the effect of the design decisions made in algorithm development. Such research efforts would benefit from access to an open-source software library of classical vehicle routing heuristics. Hence, the objective of the study was to recognize the most relevant classical heuristics and then produce open-source implementations that can reproduce the original results.

This work was also motivated by the structure of this dissertation. To carry out the necessary experiments involving algorithm selection for Paper PV, we needed access to implementations of several VRP algorithms. To make these experiments less computationally intensive to conduct, we made the additional requirements for the algorithms to be deterministic to be relatively parameter free, and to be able to solve also the larger problem instances in the popular CVRP benchmark sets. Many of the classical VRP heuristics fulfill these criteria. Also, there is large variation in operating principles between the classical heuristics (Cordeau et al., 2002, p. 514), which makes them an interesting target for in-depth analysis and well suited for experimental study in algorithm selection.

Method

We conducted a literature meta-survey of 15 survey papers published between 1971 to 2014 on VRP heuristic algorithms to recognize those CVRP heuristics that can be considered to be "classical." By calculating the frequency of citations since the algorithms had been introduced, we were able to recognize 28 "classical" CVRP heuristics.

We also surveyed the currently available open-source libraries for solving vehicle routing problems. There were many alternatives offering an implementation of metaheuristics, but only a few offered several alternative algorithms, and even if they did, no results or evidence was given to demonstrate that the implemented algorithms would, indeed, replicate the results from the literature. This, together with the complexity of the existing software libraries, led us to write our own independent implementations.

Most of the classical heuristics were deterministic and some could be trivially converted. The requirement for determinism also helped us to rule out some algorithms that could be more naturally classified as metaheuristics. We also ruled out heuristics that involved interactive or manual steps and those few for which we were unable to find experimental results. This left us with a list of 15 classical deterministic CVRP heuristics introduced between 1964 and 1989. The rest of the study concerned itself with the implementation details and replications of the computational results of these 15 algorithms.

Results and contribution to the whole

The working principles of the 15 selected classical heuristics were carefully documented during the implementation process. The descriptions were complemented by notes on the many design and implementation insights and details gained through the tenacious effort of replicating the numerical results from the literature. As a result of our efforts, 10 of the implementations closely replicated the original numerical results. For the rest, the replication level was mostly satisfactory. The probable reasons for the discrepancies were also discussed; some uncertainties remained because of missing details of the original experimental setup, due to the vagueness in the algorithm description, or due to possible implementation error on our part.

The reproductions were complemented by an extensive computational study. We reported the observed time complexities of the algorithms and presented a comparison of the implemented CVRP algorithms on 454 CVRP problem instances from the literature. This extensive amount of performance data allowed us to study and analyze the different strengths and weaknesses of the heuristic algorithms through a systemic examination of their the accuracy, consistency, speed, and simplicity. To our knowledge, this is the most in-depth review and computational study on classical CVRP heuristics. The main contribution is the open-source implementations written in a modern programming language and the related replication of the original results. Later, the computational results on the 454 CVRP problem instances were used in the experimental part of Paper PV

Summary of individual contributions

- Recognized 15 deterministic CVRP heuristics that can be considered to be "classical" through a literature meta-survey.
- Presented replications of the results for those 15 heuristics and documented the related design and implementation decisions.
- Presented and analyzed the results of an extensive computational study where the classical heuristics solved 454 well-known CVRP instances. This allowed us to…
  - …experimentally provide computational complexity classes for the 15 heuristics.
  - …recognize patterns in the solution quality variation between the classical heuristics, revealing their strengths and weaknesses.
  - …derive quantitative scores of accuracy, robustness, simplicity, and speed for the 15 classical heuristics.
- **Software:** Designed and programmed an open-source software library[2] with implementations of 15 deterministic classical CVRP heuristics.

[2] https://github.com/yorak/VeRyPy

6.5 Paper PV : Feature and Algorithm Selection for Capacitated Vehicle Routing Problems

The paper was published in the proceedings of the 27th European Symposium on Artificial Neural Networks in 2019. The conference was held in Bruges, Belgium, on April 24-27, 2019.

Aim

The number of features proposed for describing combinatorial problem instances has continued to steadily grow, but there is little discussion on the trade-offs between computational effort of the feature extraction and the extent of the increase in solver performance one can expect by using meta-learning techniques. Furthermore, the differences in the importance of the different features is rarely explored in the literature.

We had recently proposed an extensive set of features for CVRP instances in Paper PIII, and recognizing the most relevant VRP features in meta-optimization tasks was set as the main objective of this study. More specifically, we sought to determine the most central features for the task of VRP algorithm selection. This involved answering the following question: which subset of the features proposed in Paper PIII should be used when predicting the most suitable classical heuristic algorithm from Paper PIV for a given CVRP instance? We were also interested in seeing what kind of increase in solution quality an ensemble of relatively simple VRP heuristics could offer. The study was inspired by a similar study from Pihera and Musliu (2014), where they used discretization and machine learning to predict a best TSP algorithm for a given TSP instance.

Method

The feature extraction framework that was originally proposed in Paper PIII, and extended by Kärkkäinen and Rasku (2019), was further modified in this study. Compared to Paper PIII, we added: features related to the local search neighborhood; a feature indicating the ratio between integer and non-integer decision variable values in the output of the exact solver; and several features measuring the tightness of an optional maximum route duration constraint. The features were preprocessed by imputing large values in the place of infinite ones, performing normalization, and creating a second discretized dataset using the multi-interval discretization of continuous-valued attributes (MDLP) algorithm from Fayyad and Irani (1993).

We used the algorithm performance data from Paper PIV, which contained the accuracy and computation time data for the implemented 15 deterministic classical CVRP heuristics, with each solving 454 CVRP instances from the literature. Using this data instead of the values reported in the literature allowed us to sidestep the issues related to stochasticity (Barr et al., 1995) and algorithm

performance data commensurability (Laporte, 2007). Out of the 15 algorithms, the sequential savings algorithm was omitted because it was not the single best algorithm on any of the 454 CVRP benchmark instances. We also prepared another algorithm selection scenario, where the task was to select the best algorithm among the three most successful on the benchmark set: GAP (Fisher and Jaikumar, 1981), Petal (Foster and Ryan, 1976), and parametrized savings (Paessens, 1988).

This led to four algorithm selection scenarios: to predict the best algorithm (with or without the feature value discretization) among the 14 classical heuristics for CVRPs, or among the three best. In addition to the algorithm performance data, the extended feature data with 433 feature values was calculated for each of the 454 problem instances. Using the optimality gap and elapsed time data, we were able to label the data. So, the best algorithm was set to be the one with the smallest optimality gap, or in the case of ties, the one with the shortest solution time.

A comparison where PCA and three feature selection methods, mRMR (Peng et al., 2005), CFS (Hall, 2000), and extremely randomized trees-based estimators (Geurts et al., 2006), were used to reduce the dimensionality of the feature data and four classifiers, 3-nearest neighbor (3-NN), multilayer perception (MLP), *C*-SVM, and random forest, were used to learn the performance patterns. All ML tasks were evaluated with a selected feature count from 1 to 100 using a leave-one-out cross-validation.

Results and contribution to the whole

The results revealed that features from probing with exact and local search algorithms were among the top 10 for all of the four feature selection/analysis methods, but also features related to the tightness of the constraints, position of the depot, and the structure of the nearest-neighbor digraph contributed significantly to the classification task. The best predictive accuracy in our experiments was reached when between 75 and 100 features were used. However, the embedded feature selection method based on random forest indicated that the feature importance decreased significantly after ten features.

The results of this study allowed us to estimate what kind of predictive accuracy one can expect in VRP algorithm selection tasks assuming a suitable feature set for describing the problem instances is available. The best predictive accuracy in the experiments ranged from 48.5% for the prediction task with 14 algorithms, through 75.3% for a more realistic case where an acceptable result was that the predicted algorithm was in the top 3, to 74% accuracy on the task of where the very best heuristic was selected among a portfolio of three proven classical CVRP heuristics. Compared to the simple strategy of using the single best algorithm, the algorithm selection yielded the following improvements: on the scenario where the algorithm was predicted among 14 alternatives, the algorithm selection allowed closing the optimality gap on average by 1.1 percentage points, which was a 28% improvement. In the easier three algorithm scenario, the

optimality gap was closed on average by 1.3 percentage points, a 34% improvement. In these comparisons, the predictions were put against the generalized savings algorithm of Paessens (1988), which was the single best all-around classical algorithm in Paper PIV.

The results verified the feasibility of the algorithm selection approach and illustrated how existing tools and techniques related to feature extraction, feature selection, and algorithm selection could be used to boost the quality of VRP solutions. Based on the experiments and their results, we expect using the algorithm selection approach in customizing a vehicle routing system to yield numerous benefits. This would require embedding the feature extraction (PIII) and algorithm selection to the vehicle routing system customization and deployment workflow (Paper PI).

Summary of individual contributions

- Identified the most relevant features in the algorithm selection of classical CVRP heuristics.
- Presented results of the automatic algorithm selection. A good heuristic was selected with an accuracy of around 75%.
- Presented empirical evidence demonstrating that the algorithm and feature selection can allow an ensemble of relatively simple heuristics to produce high quality solutions.

6.6 Paper PVI: On Automatic Algorithm Configuration of Vehicle Routing Problem Solvers

The preliminary results (Rasku et al., 2014) on this topic were published in the proceedings of the *Optimization and PDEs with Industrial Applications* conference held in June 2012 at the University of Jyväskylä, Finland. The study was further extended with new configuration targets and with an in-depth analysis of the results, and in 2019, a study containing these new contributions was published in the *Journal on Vehicle Routing Algorithms*.

Aim

Almost all algorithms for combinatorial optimization problems have a number of free parameters that influence their behavior (Birattari, 2009), and VRP algorithms are not an exception. In addition to the several free parameters, they tend to exhibit relatively long computing times which makes finding the right values for these parameters challenging (Becker et al., 2005).

Usually, a set of default parameter values is suggested by the algorithm designer who has used expertise and experiments to find them. Unfortunately,

these are typically optimized to solve a specific scientific benchmark set (Hutter et al., 2009), and using the defaults on a new set of problems may not give an accurate impression of the true performance of the algorithm. Thus, the objective in this study was to offer practical recommendations on how a systematic algorithm configuration of VRP heuristics could be carried out in order to improve their performance and allow replicable algorithm comparisons (Ansótegui et al., 2009; Eggensperger et al., 2019).

The timeliness for such an experimental study was further emphasized by the observation of Kadioglu et al. (2010) that there has been a renaissance in automatic algorithm configuration during the first decade of the 21st century. This development has led to many new and interesting automatic algorithm configuration methods being proposed. Employing them would allow automating the critical and tedious part of finding the right parameters for the VRP algorithms. Also, we would like to argue that a comprehensive critical evaluation of well-known existing configuration methods is needed to verify their suitability in configuring VRP metaheuristics.

Method

Benchmarking the surveyed state-of-the-art automatic algorithm configuration tools necessitated designing a comprehensive set of experiments that took the special needs of configuring VRP algorithms into account. To facilitate our research, we developed a design-of-tuning-experiment tool similar to the one proposed by Eggensperger et al. (2019). This scaffolding made it easier to build wrappers and experiment with different configuration scenarios.

We used solvers from two sources: the CVRP solvers from Groër et al. (2010) providing TS, SA, and RTR metaheuristics; and IRIDIA (Bianchi et al., 2005) solvers with ACO, EA, ILS, SA, and TS metaheuristics targeting VRPSDs. Both aimed to solve a specific VRP type, and both were based on local search enhanced and orchestrated by the different metaheuristics. The solvers had from 3 to 14 parameters that needed to be configured. The experiments involved using these eight metaheuristics to compute solutions to several academic problem sets on a total of 2695 configuration runs.

The extensive computational requirements of such automatic algorithm configuration experimentation required us to carefully control and fix some aspects of our experiments. For example, we required that each configuration target in our experiments had 14 problem instances and a separate training and test sets. If the original problem set was larger, we used stratified sampling to create a smaller benchmark set. Stratification made sure that the problem size composition of the instance sets resembled the original set. We used a 3-fold cross-validation for one of the problem sets which did not have enough problem instances for a separate test set. We also used fixed evaluation budgets of 100, 500, and 1000 solver invocations, with each invocation limited to a runtime of 10 seconds. To test the effect of a larger budget, the configuration methods tuning the ACO metaheuristic were also given a larger budget of 5000 evaluations.

The objective in our automatic algorithm configuration runs was to find a parameter configuration that would minimize the aggregated optimality gap over a problem instance set. Also, the metaheuristics had a stochastic component, and repetitions were required to get a reliable estimate of the performance with a given parameter configuration. Thus, the automatic algorithm configuration methods were set to expect stochastic solvers and the actual allocation of the evaluation budgets depended on how the method decided to use evaluations and handle stochasticity of the target algorithms. Furthermore, the configuration methods were also allowed to use the default parameter configuration of the target metaheuristics as a seed for the parameter search.

Regarding other parameters of the configuration methods, we used the defaults and changed them only if it was necessary to make a configuration method to respect the evaluation budget. Configuring the automatic configuration methods themselves could have improved the performance but this would have increased the already high computational effort further. It can also be argued that a good meta-optimization method should not need such meta-meta-level parameter configuration (Sörensen et al., 2008, p. 251-252).

Results and contribution to the whole

The absolute and relative improvements achieved through automatic algorithm configuration of the eight VRP metaheuristics were considered through a careful analysis of the results. This allowed gaining insights on what kind of improvement one can expect from the problem set specific configuration. Unsurprisingly, all of the tested metaheuristics benefited from automatic configuration, and they were able to surpass the solution quality of the default parameter values.

While presenting and analyzing some of the best algorithm configurations found by the configuration methods, we recognized parameter configurations that would allow producing optimal solutions for all of the Augerat (1995) problem instances in the validation set. However, there was a large variation between the different combinations of metaheuristic, configuration method, and problem instance sets. We recognized three different difficulty classes among the configuration tasks. The classification revealed how the suitability of a metaheuristic in solving a specific problem instance type can significantly effect its configurability.

Our experiments and analysis also revealed clear differences in the performance of different configuration methods. However, these were not systematic and the best method depended on configuration targets and tasks. The two most successful methods for configuring VRP metaheuristics in our study were SMAC from Hutter et al. (2011) and iterated F-race from López-Ibáñez et al. (2016). The results strongly supported the hypothesis that such methods could also be used to automatically configure the algorithms inside a vehicle routing system. Furthermore, if we assume that the best configuration method for a specific target can always be chosen, the optimality gap was closed by 70 % compared to using the default parameters. Hence, the performance increase in automatic algorithm configuration of such software systems can be significant.

If we consider a more general VRP research context, we argue that comparison studies like the one presented in this paper are needed to reveal the true robustness and performance characteristics of the algorithms targeting vehicle routing problems. We have shown that such work can provide a deeper insight into the nature and solution space structure of the routing and automatic parameter configuration problems and provide a large amount of useful empirical information on how to apply these solution techniques. As such, with a sharper focus on VRP metaheuristics, our study can complement the more generic advice from Eggensperger et al. (2019).

Summary of individual contributions

- Presented a comprehensive experimental study on automatic algorithm configuration (AAC) of VRP metaheuristics which allowed us to observe that...
  - ...in general, the algorithm with configured parameters produced significantly better solutions than one with the default parameters. This shows that existing AAC methods are suitable for configuring the tested VRP metaheuristics.
  - ...as few as 100 algorithm invitations was in most cases enough to find a better parameter configuration than the defaults.
  - ...SMAC and iterated F-race are feature-rich and robust configuration methods well suited for configuring VRP metaheuristics.
  - ...the configuration performance varied greatly, and there was no single best AAC method for VRP metaheuristics.
- Argued that AAC should be a recommended practice when VRP researchers are doing comparative experimental work.
- **Software:** Designed and programmed a tool for designing and running AAC experiments. It includes wrappers for the seven AAC methods and eight VRP metaheuristics used in this study and allows easy extendability.

6.7 Author's contributions

I was the corresponding and the main author of all six papers. Only Paper PI contains original content written by two of the co-authors: Tuukka Puranen contributed text on how the software product lines and vehicle routing systems can be combined, and Antoine Kalmbach provided a description of the automatic data integration and the preliminary results for its application. Still, it should be noted that all papers went through many writing iterations and the review and revision work from all co-authors was an instrumental part of the writing process.

Many of the ideas and approaches used in this dissertation were heavily influenced by the discussions between me and the co-authors. This dissertation can first and foremost be seen as a continuation of the work done by Tuukka Puranen, who in his doctoral dissertation proposed a metamodel and a model-driven software production line approach for building vehicle routing systems. His efforts, together with the work of the other members of the Research Group on Computational Logistics of the University of Jyväskylä, created the foundation for the presented research and heavily influenced the formulation of the research problems and objectives. Furthermore, recognizing how the data flows through a vehicle routing system, what kind of transformations are required, and what kind of variability points are exposed should be credited to Tuukka Puranen and the other members of the research group involved in creating the vehicle routing system research prototype.

Regarding the individual papers, Nysret Musliu was familiar with earlier studies on applying automatic algorithm configuration and algorithm selection techniques to related combinatorial optimization problems. His input and supervision initiated and had a major impact on the research reported in Paper PVI and Paper PV. Similarly, the earlier research carried out by Tommi Kärkkäinen shared similarities with the work in papers PII and PV. Furthermore, one of his methods for discovering important features based on hybrid autoencoders (Kärkkäinen and Rasku, 2019) was one of the feature importance analysis methods in Paper PV. However, while I co-authored that paper, my role was minor and the study is not included in this collection of original papers. Overall, the guidance, questions, and recommendations of Tommi and Nysret were invaluable in formulating the research goals and designing the required experiments for all these works. However, I worked independently on formulating the research questions, choosing the appropriate research methods, implementing the necessary tools, preparing and carrying out the necessary experiments, and formulating the main contributions of the individual papers.

I developed multiple research tools and software artifacts during the course of the outlined research. I wrote the automatic algorithm configuration framework that made it possible to carry out the computational study of Paper PVI. Similarly, the visualization tool in Paper PII, and the efficient solution generators that could produce the data to be visualized, were written solely by me. However, the local search trace and its neighborhood was generated using the research VRP solver prototype of the research group, which was slightly modified and run by Pekka Hotokka. Also, the preliminary results on automated data import and model inference in Paper PI were provided by the research prototype written by Antoine Kalmbach. For Paper PIII, I was again the sole programmer and implemented a total of 76 feature extractors. I extended this feature extraction framework twice for subsequent studies. The final version of the feature extraction tool used in Paper PV implements 100 feature extractors. However, the largest implementation effort in this work involved writing VeRyPy, which is the open-source library implementing 15 classical CVRP heuristics, many local search heuristics, and a few additional less-known constructive heuristics.

Finally, all papers included a large amount of experimental empirical work. I was responsible for setting up the computational experiments, running them, and processing their results. This involved, sometimes extensive, scripting and analysis work. In addition to this experiment management, the generation of most of the figures and tables was done using scripts designed and programmed by me.

7 DISCUSSION

This dissertation illustrates how recent advances in machine learning and combinatorial optimization allow leveraging the diversity of the existing routing algorithms which, ultimately, should allow pushing the state-of-the-art (Kotthoff, 2014). More specifically, this research has presented a vision and suggestions for realizing a next-generation vehicle routing system with increased self-adaptivity—the goal being a less-labor-intensive customization process. We have also argued that automatic customization of the system for each specific operational context yields numerous benefits. Most centrally, this is expected to improve the quality of the results and make the systems cheaper to deploy. Our results suggest that meta-optimization can be used to automate some of the tailoring work, which would otherwise require hard to find and acquire expertise.

What follows is a more detailed discussion of the results and contributions leading toward the goal of automated vehicle routing system customization. Also, the validity, limitations, and future research topics are discussed at the end of this chapter.

7.1 Discussion of the results

According to de la Banda et al. (2014), in order to deliver effective high-level, easy-to-use optimization to everybody, modern optimization technology needs to be made easier to use and deploy. To overcome this challenge, they called for major scientific contributions in three related areas: 1) *modeling*, 2) *model transformations*, and 3) *solver technology*. The focus of de la Banda et al. was constraint programming, but it is easy to see how similar ideas involving observation of the problem and solution search space to outperform more general-purpose algorithms can be applied with methods targeting vehicle routing problems.

The research presented in the six original papers that make up this dissertation aim to address this challenge by making contributions in all three areas. More specifically, the dissertation contributes by:

1) Giving the VRP researchers visual tools (PII) and descriptive metrics (PIII) to observe the problems they are investigating. This helps them to understand the structure and idiosyncrasies of problem instances. de la Banda et al. (2014) explicitly mention combinatorial substructures and stochastic information as a few of the possible ways of describing the problems, but there are many others as we have demonstrated in Paper PIII. We have also shown how such tools can be useful for debugging, explanation, and exploration of the algorithm's behavior. For example, the visualization technology presented in Paper PII contributes by allowing an intuitive understanding on the behavior of a heuristic search procedure through visual examination of its trace.

2) Proposing the idea of optimization model inference with some preliminary results (PI). Using the model transformation terminology in de la Banda et al. (2014, 3.2), we can see that this dissertation contributes to the *automatic derivation of design models* (i.e., optimization models). The presented research also provides empirical evidence on how high-level features (PIII) and VRP algorithm performance are linked (PV).

3) Presenting empirical results on applying automatic algorithm configuration (PVI) and algorithm selection (PV). Algorithm portfolios and automatic model and algorithm tuning (configuration) can be considered to be central techniques in *automatic hybridization* of combinatorial optimization solvers (de la Banda et al., 2014) and these topics are closely related to the research on hyper-heuristics (Smith-Miles, 2009; Pappa et al., 2014). A closer integration of the VRP heuristics would, indeed, lead to hybridization (if done manually) and hyper-heuristics (if automated). Due to the similarities of these approaches, we argue that the presented research provides original and useful insights on how to address this third challenge regarding solver technology. Furthermore, de la Banda et al. (2014) explicitly call for research and tools that can automatically tune optimization models and algorithms, and such work is presented in Paper PVI. They also emphasize the importance of tools that allow exploration and comparison of the performance of different optimization techniques (PIV), especially from the search space exploration point of view (PII).

Another trend in optimization technology is what de la Banda et al. (2014) call *data intensive optimization*. With machine learning, the data can be used to aid and guide the optimization technology. Data-intensive techniques have developed together with big data and increasing computational capacity. We have seen an advent in the use of massive vectorized computing with graphical processing units (GPU) and data centers with scalable parallel processing power. Optimization technology has been, and will be, benefiting greatly of these advances; these advances and techniques will allow us to build next generation optimization technology that can be utilized without needing an expert to constantly tweak the system.

Gathering the necessary data for teaching the models for algorithm selection and automatic algorithm configuration is a computationally intensive task. The advances in parallel computing power make it feasible to explore the configuration and design choice space to a unprecedented extent as shown in Papers PVI and PV. Ultimately, we have demonstrated how such data can be used to build models that capture the interactions among the triplet: problem instance, algorithm and its parameter configuration, and performance measure. In essence, such meta-optimization techniques learn the patterns in the performance data to create a model with a predictive power, be it predicting the best algorithm for a given problem instance or a good parameter configuration.

Both automatic algorithm configuration and algorithm selection are data-driven automation methods. As such, they benefit from a large number of prior solution runs. As with all machine learning, the predictive power of the models is heavily dependent of the amount and quality of the data. In this dissertation, Papers PIII and PIV aim to address this dependency by proposing an extensive feature extraction framework for characterizing the problem instances and a set of 15 open-source implementations of the classical CVRP heuristics. As such, this work also supports the call for reproducibility, replicability, and openness of the research on vehicle routing algorithms (Sörensen et al., 2019), for example, in the form of open-sourcing the relevant software artifacts.

Van Breedam (2002) analyzed the effect of parameters on VRP heuristics and in the conclusions of his article, he proposed a checklist for building a system that could automatically select and configure VRP algorithms. He pointed out that the proposed actions would include a huge amount of experiments. Thus, due to the constructive and experimental nature of our work, it is interesting to reflect the contributions of this dissertation against his list:

1. *Van Breedam: Determine the characteristics of a given VRP instance.* One major contribution of this work was the feature exaction framework proposed in Paper PIII, which, in addition to those characteristics explicitly mentioned in Van Breedam (2002), includes a comprehensive set of features derived from the literature on TSP and VRP meta-optimization and solution space analysis. Furthermore, we provide a feature selection based analysis on the relative importance of these features in Paper PV and our work on VRP visualization and fitness landscape analysis presented in Paper PII offers insights on the properties of VRP solution search spaces.

2. *Van Breedam: Determine the most efficient heuristics to solve a VRP instance.* This required access to a comprehensive selection of heuristics. For practical reasons, we wanted to limit our study to deterministic algorithms, and because such implementations were not readily available, we proceeded to recognize and implement 15 classical CVRP heuristics ourselves. The correctness of the implementations was demonstrated through a replication study. This work is covered by Paper PIV. The implementations allowed us to determine the best classical heuristic for all problem instances in the commonly used CVRP benchmark sets.

3. *Van Breedam: Determine the best parameter values for the heuristics.* This task was explored in Paper PVI, where we proposed extensive computational experiments in automatically configuring the parameters of eight VRP metaheuristics using seven automatic algorithm configuration methods. Based on that, we could give recommendations on how to do effective parameter meta-optimization with VRP metaheuristics and which kind of accuracy and improvements one can expect. In Paper PI, we showed similar results on problem instances derived from real world routing cases. Paper PIII further extended the experiments with instance-specific algorithm configuration, where the features (characteristics) of the problem instances were used to aid the prediction of good parameter values for a problem instance.

4. *Van Breedam: Generate a solution for the given VRP instance.* Feasibility of the algorithm selection approach was shown in Paper PV, where machine learning classifiers were able to predict a suitable solver with good accuracy. An ensemble of simple VRP algorithms guided by machine learning seems to be a feasible strategy to get better solutions compared to using a single well-regarded algorithm.

Thus, this dissertation addresses all of the items on the Van Breedam (2002) checklist and offers practical insights on how the diagnostic system he envisioned could be implemented.

Similar fundamental challenges were recognized by Desrochers et al. (1999) but with a stronger focus on commercial vehicle routing systems. They studied how to represent and manipulate meta-knowledge on problems and models and how to utilize it in decision support systems targeting vehicle routing problems. Their proposed approach relied on using a mathematical modeling language for mixed integer problems. However, the industry standard in solving real-life transportation optimization has moved towards metaheuristics (Laporte et al., 2014) for which the modeling is more conveniently done using alternative approaches. Related to this development, Puranen (2011, p. 52) differentiates *formulating* the problem and *modeling* it: formulating is stating the problem in the form of objective and constraints, but modeling "involves transforming the required aspects of reality into a mathematical abstraction that can be operated on by algorithmic means." A natural nonlinear encoding for a VRP is one that uses graphs. This is also the reason why the use of a routing metamodel and the associated model transformations described in (Puranen, 2011, Chapter 5) is the approach advocated by this dissertation. We have argued that assuming the availability of such modeling and solution capabilities allows automating some of the customization tasks, and is one of the main reasons our proposed approach differs from the one envisioned by Desrochers et al. (1999). Instead of manually building a knowledge base, we have applied a more modern machine learning approach to automatically discover the patterns and relations between problem types, instance features, modeling constructs, and algorithm performance from the data. This approach allows using meta-optimization to automatically fine-tune the algorithmic components inside a vehicle routing system.

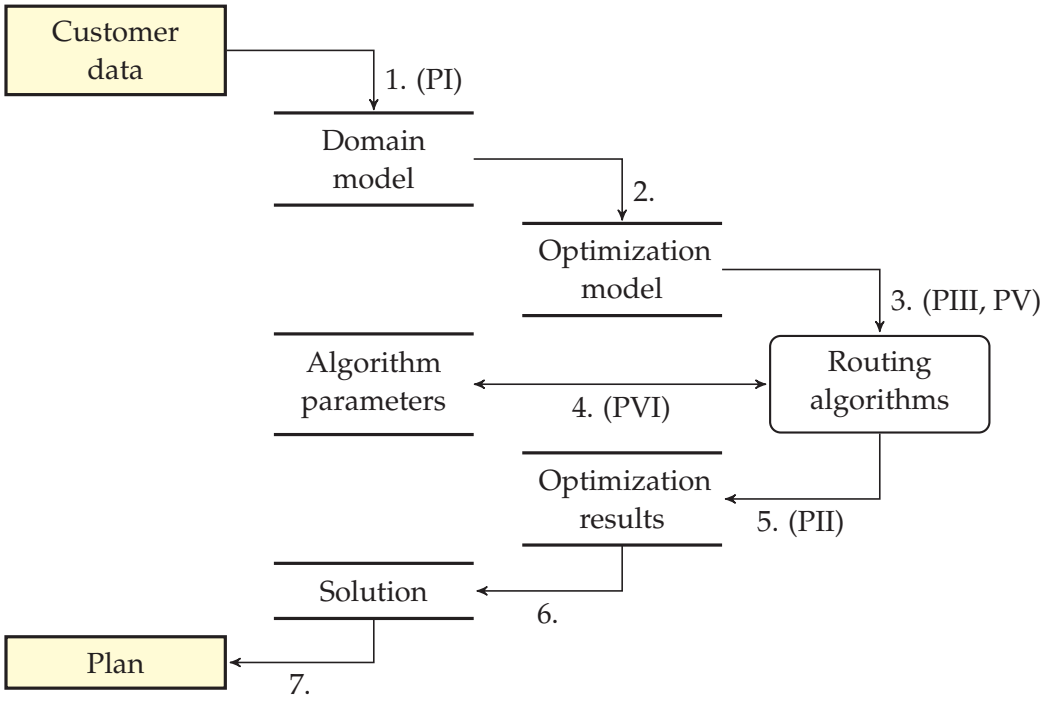


FIGURE 9 The included papers in relation to the data flow through a VRS. The model proposed in Paper PI and the other papers are assigned to their respective transformations.

Of particular interest in this dissertation was the *data flow* through a vehicle routing system. It defines how the problem instance is passed from one system module to another. This process is illustrated in Figure 9, which describes how the input data of describing a planning task ultimately leads to a transportation plan after having gone through several transformations. The transformations convert the problem between different representations and abstraction levels so that it is in the correct format used by the different modules. As one can see, all of the original papers are associated to one of these transformations.

In this disseration, we have argued that the flow of information and managing these transformations are the main aspects affecting the customization, deployment, integration, and utilization of vehicle routing systems. These transformations can be linked to the seven variation points of the software production line proposed by Puranen (2011, pp. 201–202). Below, we have reordered and grouped them to match the numbered transformations illustrated in Figure 9. The VRP is solved by:

1. populating the **domain model** by wiring **data connections** and by using the **task generation** process;

2. generating the **optimization model**;

3. selecting the **solution methodology** and

4. its parameters;

5. applying the **solution methodology**;
6. interpreting the resulting decision variables back to the concepts used in the **domain model**; and
7. producing an actionable plan using the presentation model and the operational terminology (on the **presentation layer**).

Hence, our proposed data-driven approach to automate the configuration of such customizable optimization software is similar to the one described by de la Banda et al. (2014). They envisioned how the runtime analysis of the data, conceptual/domain model, optimization model, and algorithms could be used to discover information related to the transformations between them, and how this could help in solving the problem effectively. They propose transferring information about the original problem through domain and mathematical models to the solver, and expect that this should allow faster convergence towards good solutions. This dissertation demonstrated how instance-specific algorithm configuration (PIII) and the algorithm selection (PV) can be used to realize this vision. Furthermore, Paper PI proposed some possible techniques on automating the data import phase, including some preliminary results on automatically populating the domain model from an unstructured or structured data source. Hence, our contributions allow moving towards a situation where "the benefits of ... optimization technology can be utilized without expert knowledge" (de la Banda et al., 2014).

7.2 Conclusions and future research directions

This dissertation is built around the six articles that focus on an analysis of vehicle routing problems, problem instances, heuristics, and search spaces. The motivation of this work was to explore ways vehicle routing system customization efforts could be automated.

The main contributions, in addition to the description of the customization workflow itself, are related to the applicability of the meta-optimization approach. We have shown that the use of state-of-the-art automatic algorithm configuration techniques to configure VRP metaheuristics provides several benefits: it is more thorough and less tedious than manual configuration and even modest automated configuration efforts increase the algorithm performance over defaults. We have also proposed and demonstrated the usefulness of an extensive set of features for describing vehicle routing problem instances and search spaces. These features were used in instance-specific automatic algorithm configuration, clustering of well-known academic problem instances, and in predicting the most suitable classical VRP heuristic for a given problem instance. In addition to the empirical support on the suitability of the algorithm selection approach, the last contribution also included identification of the most relevant features among the proposed set of VRP instance features.

The constructive research methodology (Lukka, 2003) used in this dissertation allowed us to contribute to the knowledge on how to design and implement a vehicle routing system that is customizable with meta-optimization, with a special focus on its solver component (transformations 3, 4, and 5 in Figure 9). The conclusions are supported by a set of extensive computational experiments and the contributions align with the recent trend in VRP research towards more unified and self-adaptive solution methods.

For operations researchers, this dissertation offers tools and techniques for visualizing, analyzing, and solving vehicle routing problems. These tools have the potential to further our understanding of the nature of vehicle routing problems and how to solve them effectively. Regarding recommendations on how to conduct experimental studies on VRP algorithms, we have demonstrated how automatic algorithm configuration is important for replicable algorithm comparisons, and how inherent properties of individual problem instances are correlated with the performance of heuristic VRP algorithms. We have also open-sourced the code of our software library implementing 15 classical CVRP heuristics. This library can be used not only to replicate results but also as a foundation for implementing novel TSP and VRP algorithms. Hence, our work contributes to the recent and important discussion on reproducibility, replicability, and openness of the research on vehicle routing algorithms (Sörensen et al., 2019).

Regarding the limitations of the conducted research, we can reflect the contributions against the transformations inside a vehicle routing system (Figure 9). This dissertation provided in-depth exploration of only the transformations involving the solver component. This focus on meta-optimization was deliberate. Our research goal was to proceed *toward* the automatic customization of vehicle routing systems and the questions related to the self-adaptivity of the solver recognized earlier by Van Breedam (2002); Sörensen et al. (2008) and Neittaanmäki and Puranen (2015) seemed most relevant to the VRP researchers. The methods proposed in this dissertation allow increasing the level of automation in vehicle routing systems, but more work is still needed to further automate the adaptation and deployment of such systems.

A natural direction for future research would be to shift the focus from off-line to on-line adaptivity, that is, to methods that adapt the algorithms during the optimization. Experimenting with hyper-heuristics (Burke et al., 2013) and related reactive search and optimization methods (Battiti et al., 2010) in the context of vehicle routing systems would verify the applicability, advantages, and disadvantages of such methods in a commercial vehicle routing context. Advances in adapting these techniques could make it easier for non-experts to use powerful vehicle route optimization tools. Furthermore, in this dissertation, we have discussed only relatively simple VRP models. Meta-learning with metaheuristics targeting rich models (Lahyani et al., 2015; Vidal et al., 2014) and stochastic or dynamic (Pillac et al., 2013) VRP variants could reveal new interesting behaviors, strengths, and weaknesses between the algorithms and models. This would also warrant extending our feature extraction framework to capture the details of these more complex VRP variants.

A more industry-motivated future research direction would be to concentrate on the first two steps of Figure 9. We have already explored some possibilities in automating the data integration, but only preliminary results exist for the statistical-inference-based mapping of the input data to the domain model (Kalmbach, 2014). It would be interesting to see this approach extended to include the transformation of the domain model to the optimization model. Here, the model transformation could perhaps be inferred using the problem instance features proposed in this dissertation. Adapting the VRP taxonomies and description languages, such as the one proposed by Desrochers et al. (1999), to encode the VRP variant information would probably be beneficial for model inference. Also, in this dissertation we concentrated on the forward-looking data transforms and did not consider the transformations back from the mathematical model to the domain level and, ultimately, into an actionable plan in the format of printed maps, delivery schedule, or load manifests. Instead, such transformations were left as a future research topic.

Finally, we acknowledge that challenges with the data quality, availability, and integration are very important to successful vehicle routing software deployments (Neittaanmäki and Puranen, 2015). However, many of these topics more naturally belong to the management side of management science and were, thus, left outside of this work. Still, automation, in the form of validation, anomaly detection, and imputation of the missing data, could be part of the solution to such data quality concerns. After the automatic customization of vehicle routing algorithms is addressed, we hope that machine-learning-powered data management approaches can be used to remove some of the remaining barriers for effortless industry-wide deployment of vehicle routing systems.

YHTEENVETO (SUMMARY IN FINNISH)

Väitöskirjassa tutkittiin kuljetustenoptimointijärjestelmien räätälöinnin automatisoimista metaoptimoinnin ja koneoppimisen avulla. Kuljetusten optimoinnissa tehtävänä on suunnitella reitit, jotka palvelevat joukon asiakkaita mahdollisimman tehokkaasti. Tehtävää on tutkittu 60-luvulta lähtien, mitä heijastelee myös saatavilla olevien ratkaisualgoritmien suuri määrä. Lisäksi erilaisten kuljetustehtävien välillä on suuria eroja, sillä reittien suunnittelussa huomioitavat erilaiset rajoitteet ja tavoitteet ovat moninaisia. Siksi algoritmeja ja järjestelmiä joudutaan lähes poikkeuksetta räätälöimään kuljetustoiminnan tarpeita vastaavaksi. Erikoisratkaisu myös yleensä huomattavasti parantaa optimoitujen suunnitelmien laatua. Tämä räätälöintityö on kuitenkin haastavaa ja kallista.

Vastatakseen näihin haasteisiin tutkijat ja järjestelmätoimittajat tasapainoilevat yleiskäyttöisyyden ja erillisratkaisuiden välillä. Yleiskäyttöiset mallinnusvälineet ovat jo laajalti käytössä, mutta algoritmeja räätälöidään ja kehitetään edelleen paljon tiettyihin yksittäisiin käyttötarkoituksiin. Väitöskirja tarjoaa ratkaisuksi metaoptimointia eli optimointimenetelmien optimointia. Mikäli tätä tehdään koneoppimisen avulla, historiatietoa voidaan käyttää optimointimenetelmien suorituskyvyn säännönmukaisuuksien löytämiseen ja hyödyntämiseen.

Ensimmäisessä väitöskirjan julkaisussa tunnistettiin ne kuljetusten optimointijärjestelmän kohteet, joita mukauttamalla räätälöintiä tehdään – ja jotka täten ovat kiinnostavia kohteita räätälöinnin automatisoinnille. Julkaisussa listattiin erilaisia vaihtoehtoja dataohjatun muunneltavuuden saavuttamiseksi ja esiteltiin konepäättelyyn perustuvan dataintegraation ja automaattisen algoritmien konfiguroinnin kautta lähestymistavan hyötyjä. Seuraavat kolme julkaisua tutkivat kuljetusten suunnittelutehtävien, niiden hakuavaruuksien ja niiden ratkaisemiseen käytettyjen heuristiikkojen ominaisuuksia. Piirteytyksen avulla kuljetusten suunnittelutehtäville voitiin laskea niiden ominaispiirteitä kuvaavia tunnuslukuja. Viidennessä julkaisussa näitä piirrearvoja käytettiin ennustamaan paras ratkaisualgoritmi viidentoista klassisen heuristiikan joukosta, joiden kuvaukset löytyvät neljännestä julkaisusta. Viimeinen, eli kuudes artikkeli sisältää laajan parametrien automaattista konfigurointia käsittelevän laskennallisen tutkimuksen.

Yhdessä julkaisut osoittavat, että metaoptimointi pystyy olennaisesti parantamaan olemassa olevien kuljetusten optimointialgoritmien suorituskykyä ja itsesäätyvyyttä. Tietomassoja hyödyntämällä osa työläästä järjestelmien räätälöintityöstä voidaan täten siirtää tietokoneelle. Tämä tekee kuljetusten optimointiohjelmistoista helpompia ja halvempia räätälöidä eri käyttötarkoituksiin. Jo nykyisellään kuljetusten optimoinnilla voidaan säästää kuljetuskustannuksissa 5-20% verrattuna käsin tehtävään suunnitteluun. Optimointitekniikan avulla suunnittelu nopeutuu, suunnitelmien laatu kasvaa ja se saadaan joustavammin tukemaan yrityksen muita prosesseja. Väitöskirjassa esitellyillä menetelmillä voidaan varmistua siitä, että käytettävissä on aina sopiva ja oikein viritetty optimointimenetelmä. Entistä paremmat suunnitelmat tuovat lopulta myös makroekonomisia hyötyjä, kuten sujuvamman liikenteen ja pienemmät kuljetuspäästöt.

REFERENCES

Ansótegui, C., Sellmann, M. & Tierney, K. 2009. A gender-based genetic algorithm for the automatic configuration of algorithms. In I. Gent (Ed.) Principles and Practice of Constraint Programming - CP'09, Vol. 5732. Berlin: Springer. Lecture Notes in Computer Science, 142-157.

Archetti, C. & Speranza, M. G. 2014. A survey on matheuristics for routing problems. EURO Journal on Computational Optimization 2 (4), 223–246.

Arnold, F. & Sörensen, K. 2019a. Knowledge-guided local search for the vehicle routing problem. Computers & Operations Research 105, 32–46.

Arnold, F. & Sörensen, K. 2019b. What makes a VRP solution good? the generation of problem-specific knowledge for heuristics. Computers & Operations Research 106, 280–288.

Asta, S. 2015. Machine learning for improving heuristic optimisation. University of Nottingham. Ph. D. Thesis.

Augerat, P. 1995. Approche polyèdrale du problème de tournées de véhicules. Institut National Polytechnique de Grenoble-INPG. Ph. D. Thesis.

Baker, E. K. 2002. Evolution of microcomputer-based vehicle routing software: Case studies in the united states. In P. Toth & D. Vigo (Eds.) The vehicle routing problem. SIAM, 353–361.

de la Banda, M. G., Stuckey, P. J., Van Hentenryck, P. & Wallace, M. 2014. The future of optimization technology. Constraints 19 (2), 126–138.

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

Battiti, R., Brunato, M. & Mascia, F. 2010. Reactive Search and Intelligent Optimization, Vol. 45. Springer. Operations Research/Computer Science Interfaces Series.

Battiti, R. 1989. Accelerated backpropagation learning: Two optimization methods. Complex Systems 3 (4), 331–342.

Beck, J. C., Prosser, P. & Selensky, E. 2003. Vehicle routing and job shop scheduling: What's the difference? In ICAPS. AAAI, 267–276.

Becker, S., Gottlieb, J. & Stützle, T. 2005. Applications of racing algorithms: An industrial perspective. In International Conference on Artificial Evolution. Springer, 271–283.

Bellman, R. E. & Dreyfus, S. E. 1962. Applied Dynamic Programming. RAND Corporation.

Bergstra, J. S., Bardenet, R., Bengio, Y. & Kégl, B. 2011. Algorithms for hyper-parameter optimization. In Advances in neural information processing systems, 2546–2554.

Bianchi, L., Birattari, M., Chiarandini, M., Manfrin, M., Mastrolilli, M., Paquete, L., Rossi-Doria, O. & Schiavinotto, T. 2005. Hybrid metaheuristics for the vehicle routing problem with stochastic demands. Journal of Mathematical Modelling and Algorithms 5 (1), 91–110.

Birattari, M. 2009. Tuning metaheuristics: A machine learning perspective. Springer.

Bischl, B., Kerschke, P., Kotthoff, L., Lindauer, M., Malitsky, Y., Fréchette, A., Hoos, H., Hutter, F., Leyton-Brown, K., Tierney, K. et al. 2016. ASlib: A benchmark library for algorithm selection. Artificial Intelligence 237, 41–58.

Bishop, C. M. 2006. Pattern recognition and machine learning. Springer.

Bodin, L., Golden, B. L., Assad, A. A. & Ball, M. O. 1983. Computers & Operations Research special issue on routing and scheduling of vehicles and crews: The state of the art. (10) 2.

Borg, I. & Groenen, P. 2005. Modern Multidimensional Scaling: Theory and Applications (2nd edition). Springer.

Braekers, K., Ramaekers, K. & Van Nieuwenhuyse, I. 2016. The vehicle routing problem: State of the art classification and review. Computers & Industrial Engineering 99, 300–313.

Bräysy, O., Dullaert, W. & Nakari, P. 2009. The potential of optimization in communal routing problems: case studies from finland. Journal of Transport Geography 17 (6), 484–490.

Bräysy, O. & Gendreau, M. 2005. Vehicle routing problem with time windows, part i: Route construction and local search algorithms. Transportation Science 39 (1), 104–118.

Bräysy, O. & Hasle, G. 2014. Software tools and emerging technologies for vehicle routing and intermodal transportation. In P. Toth & D. Vigo (Eds.) Vehicle Routing: Problems, Methods, and Applications. SIAM, 351-380.

Brazdil, P., Carrier, C. G., Soares, C. & Vilalta, R. 2009. Metalearning: Applications to data mining. Springer.

Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E. & Qu, R. 2013. Hyper-heuristics: A survey of the state of the art. Journal of the Operational Research Society 64 (12), 1695–1724.

Calleja, G., Olivella, J. & Vilà, M. 2019. Operations research and emergent technologies. In Management Science: Foundations and Innovations. Springer, 183–197.

Calvet, L., de Armas, J., Masip, D. & Juan, A. A. 2017. Learnheuristics: hybridizing metaheuristics with machine learning for optimization with dynamic inputs. Open Mathematics 15 (1), 261–280.

Carić, T., Galić, A., Fosin, J., Gold, H. & Reinholz, A. 2008. A modelling and optimization framework for real-world vehicle routing problems. In Vehicle Routing Problem. IntechOpen.

Ceschia, S., Di Gaspero, L. & Schaerf, A. 2011. Tabu search techniques for the heterogeneous vehicle routing problem with time windows and carrier-dependent costs. Journal of Scheduling 14 (6), 601–615.

Clarke, G. & Wright, J. W. 1964. Scheduling of vehicles from a central depot to a number of delivery points. Operations research 12 (4), 568–581.

Cordeau, J.-F. & Laporte, G. 2003. The dial-a-ride problem (DARP): Variants, modeling issues and algorithms. Quarterly Journal of the Belgian, French and Italian Operations Research Societies 1 (2), 89–101.

Cordeau, J.-F., Gendreau, M., Laporte, G., Potvin, J.-Y. & Semet, F. 2002. A guide to vehicle routing heuristics. Journal of the Operational Research Society 53 (5), 512–522.

Cordeau, J.-F., Laporte, G., Savelsbergh, M. W. & Vigo, D. 2007. Vehicle routing. In Handbook in OR & MS, Vol. 14. Elsevier, 366–427.

Corstjens, J., Depaire, B., Caris, A. & Sörensen, K. 2019. A multilevel evaluation method for heuristics with an application to the VRPTW. International Transactions in Operational Research.

Cox, T. F. & Cox, M. A. 2000. Multidimensional scaling. Chapman and hall/CRC.

Coy, S. P., Golden, B. L., Runger, G. C. & Wasil, E. A. 2001. Using experimental design to find effective parameter settings for heuristics. Journal of Heuristics 7, 77-97.

Crainic, T. G., Gendreau, M. & Potvin, J.-Y. 2009. Intelligent freight-transportation systems: Assessment and the contribution of operations research. Transportation Research Part C: Emerging Technologies 17 (6), 541–557.

Czech, Z. J. 2008. Statistical measures of a fitness landscape for the vehicle routing problem. In IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2008). IEEE, 1–8.

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

De Jong, K. 2007. Parameter setting in EAs: a 30 year perspective. In Parameter setting in evolutionary algorithms. Springer, 1–18.

Derigs, U. & Vogel, U. 2014a. Experience with a framework for developing heuristics for solving rich vehicle routing problems. Journal of Heuristics 20 (1), 75–106.

Derigs, U. & Vogel, U. 2014b. Experience with a framework for developing heuristics for solving rich vehicle routing problems. Journal of Heuristics 20 (1), 75–106.

Desaulniers, G., Desrosiers, J., Solomon, M. M., Soumis, F., Villeneuve, D. et al. 1998. A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. In Fleet management and logistics. Springer, 57–93.

Desaulniers, G., Madsen, O. B. & Ropke, S. 2014. The vehicle routing problem with time windows. In P. Toth & D. Vigo (Eds.) Vehicle Routing: Problems, Methods, and Applications. SIAM, 119–159.

Desrochers, M., Jones, C. V., Lenstra, J. K., Savelsbergh, M. W. & Stougie, L. 1999. Towards a model and algorithm management system for vehicle routing and scheduling problems. Decision Support Systems 25 (2), 109–133.

Domingos, P. M. 2012. A few useful things to know about machine learning. Communications of the ACM 55 (10), 78–87.

Dondo, R. & Cerdá, J. 2007. A cluster-based optimization approach for the multi-depot heterogeneous fleet vehicle routing problem with time windows. European Journal of Operational Research 176 (3), 1478–1507.

Dorigo, M. & Di Caro, G. 1999. Ant colony optimization: a new meta-heuristic. In Proceedings of the 1999 congress on evolutionary computation (CEC99), Vol. 2. IEEE, 1470–1477.

Drexl, M. 2012. Rich vehicle routing in theory and practice. Logistics Research 5 (1-2), 47–63.

Du, T. C. & Wu, J.-L. 2001. Using object-oriented paradigm to develop an evolutional vehicle routing system. Computers in Industry 44 (3), 229–249.

Dueck, G. 1993. New optimization heuristics: The great deluge algorithm and the record-to-record travel. Journal of Computational physics 104 (1), 86–92.

Ecorys, Fraunhofer, TCI, Prognos & AUEB-RC/TRANSLOG 2015. Fact-finding studies in support of the development of an EU strategy for freight transport logistics Lot 1: Analysis of the EU logistics sector.

Eggensperger, K., Lindauer, M. & Hutter, F. 2019. Pitfalls and best practices in algorithm configuration. Journal of Artificial Intelligence Research 64, 861–893.

Eiben, A. E. & Smit, S. K. 2011. Parameter tuning for configuring and analyzing evolutionary algorithms. Swarm and Evolutionary Computation 1 (1), 19–31.

Eksioglu, B., Vural, A. V. & Reisman, A. 2009. The vehicle routing problem: A taxonomic review. Computers & Industrial Engineering 57 (4), 1472–1483.

Ester, M., Kriegel, H.-P., Sander, J., Xu, X. et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96), 226–231.

European Commission 2018. EU Transport in figures. Statistical Pocketbook, Publications Office of the European Union.

Eurostat 2019. EU Open Data Portal. http://data.europa.eu/euodp/en/home.

Fayyad, U. M. & Irani, K. B. 1993. Multi-interval discretization of continuous valued attributes for classification learning. In Proceedings of the 13th International Joint Conference on Artificial Intelligence. Morgan-Kaufmann, 1022–1029.

Feng, L., Ong, Y.-S., Lim, M.-H. & Tsang, I. W. 2014. Memetic search with interdomain learning: A realization between CVRP and CARP. IEEE Transactions on Evolutionary Computation 19 (5), 644–658.

Fisher, M. L. & Jaikumar, R. 1981. A generalized assignment heuristic for vehicle routing. Networks 11 (2), 109–124.

Fisher, M. 1995. Vehicle routing. In M. O. Ball, T. L. Magnanti, C. L. Monma & G. L. Nemhauser (Eds.) Network Routing, Vol. 8. Elsevier. Handbooks in Operations Research and Management Science, 1–33.

Fortun, M. & Schweber, S. S. 1993. Scientists and the legacy of world war ii: The case of operations research (or). Social Studies of Science 23 (4), 595–642.

Foster, B. A. & Ryan, D. M. 1976. An integer programming approach to the vehicle scheduling problem. Journal of the Operational Research Society 27 (2), 367–384.

Gacias, B., Cegarra, J. & Lopez, P. 2012. Scheduler-oriented algorithms to improve human–machine cooperation in transportation scheduling support systems. Engineering Applications of Artificial Intelligence 25 (4), 801–813.

Garrido, P. & Riff, M. C. 2010. DVRP: a hard dynamic combinatorial optimisation problem tackled by an evolutionary hyper-heuristic. Journal of Heuristics 16 (6), 795–834.

Geiger, M. J. 2018. Algorithms, pseudo-codes, implementations–some reflections on interdependencies and potential implications. Electronic Notes in Discrete Mathematics 69, 37–44.

Geisberger, R., Sanders, P., Schultes, D. & Delling, D. 2008. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In International Workshop on Experimental and Efficient Algorithms. Springer, 319–333.

Gendreau, M., Hertz, A. & Laporte, G. 1994. A tabu search heuristic for the vehicle routing problem. Management Science 40 (10), 1276–1290.

Geoffrion, A. M. 2013. Structured modeling. In S. I. Gass & M. C. Fu (Eds.) Encyclopedia of Operations Research and Management Science. Springer, 1499–1503.

Geurts, P., Ernst, D. & Wehenkel, L. 2006. Extremely randomized trees. Machine learning 63 (1), 3–42.

Glover, F. & Laguna, M. 1998. Tabu search. In Handbook of combinatorial optimization. Springer, 2093–2229.

Golden, B., Wasil, E., Kelly, J. & Chao, I. 1998. The impact of metaheuristics on solving the vrp: algorithms, problem sets, and computational results. In T. Crainic & G. Laporte (Eds.) Fleet Management and Logistics. Springer, 33–56.

Golden, B. L., Raghavan, S. & Wasil, E. A. 2008. The vehicle routing problem: latest advances and new challenges, Vol. 43. Springer.

Groër, C., Golden, B. & Wasil, E. 2010. A library of local search heuristics for the vehicle routing problem. Mathematical Programming Computation 2 (2), 79–101.

Guyon, I. & Elisseeff, A. 2006. An introduction to feature extraction. In Feature extraction. Springer, 1–25.

Halim, S., Yap, R. H. & Lau, H. C. 2006. Viz: a visual analysis suite for explaining local search behavior. In Proceedings of the 19th annual ACM symposium on User interface software and technology. ACM, 57–66.

Hall, M. A. 2000. Correlation-based feature selection for discrete and numeric class machine learning. In Proceedings of the Seventeenth International Conference on Machine Learning (ICML '00). Morgan Kaufmann, 359–366.

Hansen, N. 2006. The CMA evolution strategy: a comparing review. In Towards a new evolutionary computation. Springer, 75–102.

Hasle, G. & Kloster, O. 2007. Industrial vehicle routing. In G. Hasle, K.-A. Lie & E. Quak (Eds.) Geometric modelling, numerical simulation, and optimization. Springer, 397–435.

Hepdogan, S. 2006. Meta-RaPS: Parameter Setting And New Applications. University of Central Florida. Ph. D. Thesis.

Hoff, A., Andersson, H., Christiansen, M., Hasle, G. & Løkketangen, A. 2010. Industrial aspects and literature survey: Fleet composition and routing. Computers & Operations Research 37 (12), 2041–2061.

Holland, J. H. 1975. Adaptation in natural and artificial systems. The University of Michigan Press.

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

Horner, P. 2018. Innovation powers dynamic VR sector. OR/MS Today 45 (1).

Huggins, D. et al. 2009. Panorama of transport (Sixth edition). Eurostat.

Hutter, F., Hoos, H. H. & Leyton-Brown, K. 2011. Sequential model-based optimization for general algorithm configuration. In International Conference on Learning and Intelligent Optimization. Springer, 507–523.

Hutter, F., Hoos, H. H., Leyton-Brown, K. & Stützle, T. 2009. ParamILS: An automatic algorithm configuration framework. Journal of Artificial Intelligence Research 36, 267-306.

Hutter, F., Hoos, H. & Leyton-Brown, K. 2010. Automated configuration of mixed integer programming solvers. In A. Lodi, M. Milano & P. Toth (Eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Vol. 6140. Springer. LNCS, 186-202.

Hutter, F., Xu, L., Hoos, H. H. & Leyton-Brown, K. 2014. Algorithm runtime prediction: Methods & evaluation. Artificial Intelligence 206, 79–111.

Irnich, S. 2008. A unified modeling and solution framework for vehicle routing and local search-based metaheuristics. Journal on Computing 20 (2), 270–287.

Irnich, S., Toth, P. & Vigo, D. 2014. The family of vehicle routing problems. In P. Toth & D. Vigo (Eds.) Vehicle Routing: Problems, Methods, and Applications. SIAM.

Jin, J., Crainic, T. G. & Løkketangen, A. 2014. A cooperative parallel metaheuristic for the capacitated vehicle routing problem. Computers & Operations Research 44, 33–41.

Kadaba, N., Nygard, K. E. & Juell, P. L. 1991. Integration of adaptive machine learning and knowledge-based systems for routing and scheduling applications. Expert Systems with Applications 2 (1), 15–27.

Kadioglu, S., Malitsky, Y., Sellmann, M. & Tierney, K. 2010. ISAC - instance-specific algorithm configuration. In H. Coelho, R. Studer & M. Wooldridge (Eds.) 19th European Conference on Artificial Intelligence - ECAI 2010, Vol. 215. Amsterdam, Netherlands: IOS Press. Frontiers in Artificial Intelligence and Applications, 751-756.

Kalmbach, A. 2014. Fleet inference: importing vehicle routing problems using machine learning. University of Jyväskylä. Master's Thesis.

Kanda, J., de Carvalho, A., Hruschka, E., Soares, C. & Brazdil, P. 2016. Meta-learning to select the best meta-heuristic for the traveling salesman problem: A comparison of meta-features. Neurocomputing 205, 393–406.

Kärkkäinen, T. & Rasku, J. 2019. Application of a knowledge discovery process to study instances of capacitated vehicle routing problems. In P. Diez, P. Neittaanmäki, J. Periaux, T. Tuovinen & P.-P. Jordi (Eds.) Computation and Big Data for Transport - Digital innovations in surface and air transport systems. Springer. (To appear in).

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

Kotsiantis, S., Kanellopoulos, D. & Pintelas, P. 2006. Data preprocessing for supervised leaning. International Journal of Computer Science 1 (2), 111–117.

Kotsiantis, S. B., Zaharakis, I. & Pintelas, P. 2007. Supervised machine learning: A review of classification techniques. Emerging artificial intelligence applications in computer engineering 160, 3–24.

Kotthoff, L. 2014. Algorithm selection for combinatorial search problems: A survey. AI Magazine 35 (3), 48–60.

Kotthoff, L., Thornton, C., Hoos, H. H., Hutter, F. & Leyton-Brown, K. 2017. Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. The Journal of Machine Learning Research 18 (1), 826–830.

Kritzinger, S., Tricoire, F., Dörner, K. F., Hartl, R. F. & Stützle, T. 2017. A unified framework for routing problems with a fixed fleet size. International Journal of Metaheuristics 6 (3), 160–209.

Kubiak, M. 2007. Distance measures and fitness-distance analysis for the capacitated vehicle routing problem. In K. F. Dörner, M. Gendreau, P. Greistorfer, W. Gutjahr, R. F. Hartl & M. Reimann (Eds.) Metaheuristics. Springer, 345–364.

Labadie, N., Prins, C., Prodhon, C., Monmarché, N. & Siarry, P. 2016. Metaheuristics for Vehicle Routing Problems. Wiley.

Lahyani, R., Khemakhem, M. & Semet, F. 2015. Rich vehicle routing problems: From a taxonomy to a definition. European Journal of Operational Research 241 (1), 1–14.

Laporte, G. 2007. What you should know about the vehicle routing problem. Naval Research Logistics 54 (8), 811–819.

Laporte, G. 2009. Fifty years of vehicle routing. Transportation Science 43 (4), 408–416.

Laporte, G., Ropke, S. & Vidal, T. 2014. Heuristics for the vehicle routing problem. In P. Toth & D. Vigo (Eds.) Vehicle Routing: Problems, Methods, and Applications. SIAM, 87–116.

Laporte, G. & Semet, F. 2002. Classical heuristics for the capacitated VRP. In P. Toth & D. Vigo (Eds.) The vehicle routing problem. SIAM, 109–128.

Lenstra, J. K. & Kan, A. H. G. R. 1981. Complexity of vehicle routing and scheduling problems. Networks 11 (2), 221–227.

Leviäkangas, P. 2016. Digitalisation of finland's transport sector. Technology in Society 47, 1–15.

Li, F., Golden, B. & Wasil, E. 2007. A record-to-record travel algorithm for solving the heterogeneous fleet vehicle routing problem. Computers & Operations Research 34 (9), 2734–2742.

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

Lourenço, H. R., Martin, O. C. & Stützle, T. 2003. Iterated local search. In Handbook of metaheuristics. Springer, 320–353.

Lukka, K. 2003. The constructive research approach. In Case study research in logistics. Publications of the Turku School of Economics and Business Administration, Series B, 83–101.

Lysgaard, J., Letchford, A. N. & Eglese, R. W. 2004. A new branch-and-cut algorithm for the capacitated vehicle routing problem. Mathematical Programming 100 (2), 423–445.

Marmion, M.-É., Jourdan, L. & Dhaenens, C. 2013. Fitness landscape analysis and metaheuristics efficiency. Journal of Mathematical Modelling and Algorithms in Operations Research 12 (1), 3–26.

Marsland, S. 2009. Machine learning: an algorithmic perspective. Chapman and Hall/CRC.

Maturana, S., Ferrer, J.-C. & Barañao, F. 2004. Design and implementation of an optimization-based decision support system generator. European Journal of Operational Research 154 (1), 170–183.

Mersmann, O., Bischl, B., Trautmann, H., Wagner, M., Bossek, J. & Neumann, F. 2013. A novel feature-based approach to characterize algorithm performance for the traveling salesperson problem. Annals of Mathematics and Artificial Intelligence 69 (2), 151–182.

Mitchell, T. M. 1997. Machine learning. McGraw-Hill.

Nallaperuma, S., Wagner, M., Neumann, F., Bischl, B., Mersmann, O. & Trautmann, H. 2013. A feature-based comparison of local search and the christofides algorithm for the travelling salesperson problem. In Proceedings of the twelfth workshop on Foundations of genetic algorithms XII. ACM, 147–160.

Neittaanmäki, P. & Puranen, T. 2015. Scalable deployment of efficient transportation optimization for SMEs and public sector. In Advances in Evolutionary and Deterministic Methods for Design, Optimization and Control in Engineering and Sciences. Springer, 473–484.

Neri, F. & Cotta, C. 2012. Memetic algorithms and memetic computing optimization: A literature review. Swarm and Evolutionary Computation 2, 1–14.

Newman, S. 2015. Building microservices: designing fine-grained systems. O'Reilly Media.

Nygard, K. E., Juell, P. & Kadaba, N. 1990. Neural networks for selective vehicle routing heuristics. ORSA Journal on Computing 2 (4), 353–364.

Olafsson, S., Li, X. & Wu, S. 2008. Operations research and data mining. European Journal of Operational Research 187 (3), 1429–1448.

Osman, I. H. 1993. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. Annals of operations research 41 (4), 421–451.

Paessens, H. 1988. The savings algorithm for the vehicle routing problem. European Journal of Operational Research 34 (3), 336–344.

Pappa, G. L., Ochoa, G., Hyde, M. R., Freitas, A. A., Woodward, J. & Swan, J. 2014. Contrasting meta-learning and hyper-heuristic research: the role of evolutionary algorithms. Genetic Programming and Evolvable Machines 15 (1), 3–35.

Parragh, S. N., Dörner, K. F. & Hartl, R. F. 2008. A survey on pickup and delivery problems. Journal für Betriebswirtschaft 58 (1), 21–51.

Partyka, J. & Hall, R. 2012. Vehicle routing software survey: On the road to innovation. OR/MS Today 39 (1), 38–45.

Partyka, J. & Hall, R. 2014. Vehicle routing software survey: VR delivers the goods. OR/MS Today 41 (1), 40–46.

Pecin, D., Pessoa, A., Poggi, M. & Uchoa, E. 2017. Improved branch-cut-and-price for capacitated vehicle routing. Mathematical Programming Computation 9 (1), 61–100.

Pellegrini, P. & Birattari, M. 2007. Implementation effort and performance. In T. Stützle, M. Birattari & H. H. Hoos (Eds.) Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics, Vol. 4638. Berlin: Springer. Lecture Notes in Computer Science, 31-45.

Peng, H., Long, F. & Ding, C. 2005. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. IEEE Transactions on pattern analysis and machine intelligence 27 (8), 1226–1238.

Pihera, J. & Musliu, N. 2014. Application of machine learning to algorithm selection for tsp. In 2014 IEEE 26th International Conference on Tools with Artificial Intelligence. IEEE, 47–54.

Pillac, V., Gendreau, M., Guéret, C. & Medaglia, A. L. 2013. A review of dynamic vehicle routing problems. European Journal of Operational Research 225 (1), 1–11.

Pillay, N. & Qu, R. 2018. Hyper-Heuristics: Theory and Applications. Springer. Natural Computing Series.

Pitzer, E. & Affenzeller, M. 2012. A comprehensive survey on fitness landscape analysis. In Recent advances in intelligent engineering systems. Springer, 161–191.

Pitzer, E., Vonolfen, S., Beham, A., Affenzeller, M., Bolshakov, V. & Merkuryeva, G. 2012. Structural analysis of vehicle routing problems using general fitness landscape analysis and problem specific measures. In 14th International Asia Pacific Conference on Computer Aided System Theory, 36–38.

Pohl, K., Böckle, G. & van Der Linden, F. J. 2005. Software product line engineering: foundations, principles and techniques. Springer.

Potvin, J.-Y. 2009. A review of bio-inspired algorithms for vehicle routing. In Bio-inspired algorithms for the vehicle routing problem. Springer, 1–34.

Potvin, J.-Y., Lapalme, G. & Rousseau, J.-M. 1990. Integration of ai and or techniques for computer-aided algorithmic design in the vehicle routing domain. Journal of the Operational Research Society 41 (6), 517–525.

Potvin, J.-Y., Lapalme, G. & Rousseau, J.-M. 1994. A microcomputer assistant for the development of vehicle routing and scheduling heuristics. Decision Support Systems 12 (1), 41–56.

Puranen, T. 2011. Metaheuristics meet metamodels: a modeling language and a product line architecture for route optimization systems. University of Jyväskylä. Ph. D. Thesis. (Jyväskylä studies in computing 134).

Puranen, T. 2012. Producing routing systems flexibly using a VRP metamodel and a software product line. In D. Klatte, H.-J. Lüthi & K. Schmedders (Eds.) Operations Research Proceedings 2011. Berlin, Heidelberg: Springer, 407–412.

Ralphs, T. K. & Güzelsoy, M. 2005. The SYMPHONY callable library for mixed integer programming. In The next wave in computing, optimization, and decision technologies. Springer, 61–76.

Rasku, J., Musliu, N. & Kärkkäinen, T. 2014. Automating the parameter selection in VRP: An off-line parameter tuning tool comparison. In W. Fitzgibbon, A. Y. Kuznetsov, P. Neittaanmäki & O. Pironneau (Eds.) Modeling, Simulation and Optimization for Science and Technology, proceedings of Optimization and PDEs with Applications Workshop, June 18-19, 2012, University of Jyväskylä, Finland, Vol. 34. Springer. Computational Methods in Applied Sciences, 191–209.

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

Rincon-Garcia, N., Waterson, B. J. & Cherrett, T. J. 2018. Requirements from vehicle routing software: Perspectives from literature, developers and the freight industry. Transport Reviews 38 (1), 117–138.

Rizzoli, A. E., Montemanni, R., Lucibello, E. & Gambardella, L. M. 2007. Ant colony optimization for real-world vehicle routing problems. Swarm Intelligence 1 (2), 135–151.

Ropke, S. & Pisinger, D. 2006a. A unified heuristic for a large class of vehicle routing problems with backhauls. European Journal of Operational Research 171 (3), 750–775.

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

Saeys, Y., Abeel, T. & Van de Peer, Y. 2008. Robust feature selection using ensemble feature selection techniques. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer, 313–325.

Sáez, D., Cortés, C. E. & Núñez, A. 2008. Hybrid adaptive predictive control for the multi-vehicle dynamic pick-up and delivery problem based on genetic algorithms and fuzzy clustering. Computers & Operations Research 35 (11), 3412–3438.

Saremi, A., ElMekkawy, T. & Wang, G. 2007. Tuning the parameters of a memetic algorithm to solve vehicle routing problem with backhauls using design of experiments. International Journal of Operations Research 4 (4), 206–219.

Savelsbergh, M. W. 1985. Local search in routing problems with time windows. Annals of Operations research 4 (1), 285–305.

Schwemmer, M. 2015. Top 100 in European transport and logistics services 2017/2018. Fraunhofer Center for Applied Research on Supply Chain Services.

Smith-Miles, K. A. 2009. Cross-disciplinary perspectives on meta-learning for algorithm selection. ACM Computing Surveys (CSUR) 41 (1), 6.

Smith-Miles, K. & van Hemert, J. 2011. Discovering the suitability of optimisation algorithms by learning from evolved instances. Annals of Mathematics and Artificial Intelligence 61 (2), 87–104.

Solakivi, T., Ojala, L., Laari, S., Lorentz, H., Töyli, J., Malmsten, J., Lehtinen, N. & Ojala, E. 2017. Finland state of logistics 2016. University of Turku, Finland.

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

Sörensen, K., Arnold, F. & Palhazi Cuervo, D. 2019. A critical analysis of the "improved Clarke and Wright savings algorithm". International Transactions in Operational Research 26 (1), 54–63.

Sörensen, K., Sevaux, M. & Schittekat, P. 2008. Multiple neighbourhood search in commercial VRP packages: Evolving towards self-adaptive methods. In Adaptive and multilevel metaheuristics. Springer, 239–253.

Speranza, M. G. 2018. Trends in transportation and logistics. European Journal of Operational Research 264 (3), 830–836.

Steinhaus, M. 2015. The Application of the Self Organizing Map to the Vehicle Routing Problem. University of Rhode Island. Ph. D. Thesis.

Taillard, É. D. 1999. A heuristic column generation method for the heterogeneous fleet vrp. RAIRO-Operations Research 33 (1), 1–14.

Tarantilis, C. D. & Kiranoudis, C. T. 2002. Using a spatial decision support system for solving the vehicle routing problem. Information & Management 39 (5), 359–375.

Tighe, A., Smith, F. S. & Lyons, G. 2005. Optimisation enhancement using self-organising fuzzy control. Kybernetes 34 (9/10).

Toth, P. & Vigo, D. 2014. Vehicle Routing: Problems, Methods, and Applications. Philadelphia, PA, USA: SIAM.

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

UK Department for Transport 2005. Computerised Vehicle Routing and Scheduling (CVRS) for Efficient Logistics. Freight Best Practice www.freightbestpractice.org.uk.

Uchoa, E., Pecin, D., Pessoa, A., Poggi, M., Vidal, T. & Subramanian, A. 2017. New benchmark instances for the capacitated vehicle routing problem. European Journal of Operational Research 257 (3), 845–858.

Van Breedam, A. 2002. A parametric analysis of heuristics for the vehicle routing problem with side-constraints. European Journal of Operational Research 137 (2), 348–370.

Ventresca, M., Ombuki-Berman, B. & Runka, A. 2013. Predicting genetic algorithm performance on the vehicle routing problem using information theoretic landscape measures. In Proceedings of the Evolutionary Computation in Combinatorial Optimization (EvoCOP 2013), Vol. 7832. Springer. LNCS, 214–225.

Vidal, T., Crainic, T. G., Gendreau, M. & Prins, C. 2013a. A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. Computers & operations research 40 (1), 475–489.

Vidal, T., Crainic, T. G., Gendreau, M. & Prins, C. 2013b. Heuristics for multi-attribute vehicle routing problems: A survey and synthesis. European Journal of Operational Research 231 (1), 1–21.

Vidal, T., Crainic, T. G., Gendreau, M. & Prins, C. 2014. A unified solution framework for multi-attribute vehicle routing problems. European Journal of Operational Research 234 (3), 658–673.

Waters, C. D. J. 1992. Expert systems for vehicle scheduling. In Artificial Intelligence in Operational Research. Springer, 215–225.

Watson-Gandy, C. & Foulds, L. R. 1981. Vehicle scheduling problem: a survey. New Zealand Operational Research 9 (2), 73–92.

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

Xu, D. & Tian, Y. 2015. A comprehensive survey of clustering algorithms. Annals of Data Science 2 (2), 165–193.

Xu, L., Hutter, F., Hoos, H. H. & Leyton-Brown, K. 2008. SATzilla: portfolio-based algorithm selection for sat. Journal of artificial intelligence research 32, 565–606.

Xu, R. & Wunsch, D. I. 2005. Survey of clustering algorithms. IEEE Transactions on Neural Networks 16 (3), 645.

Zhang, D. & Tsai, J. J. 2005. Machine learning applications in software engineering, Vol. 16. World Scientific.

Zhao, Z., Morstatter, F., Sharma, S., Alelyani, S., Anand, A. & Liu, H. 2010. Advancing feature selection research. ASU feature selection repository.

ORIGINAL PAPERS

PI

AUTOMATIC CUSTOMIZATION FRAMEWORK FOR EFFICIENT VEHICLE ROUTING SYSTEM DEPLOYMENT

by

Jussi Rasku, Tuukka Puranen, Antoine Kalmbach, Tommi Kärkkäinen 2016

Proceedings of ECCOMAS Thematic Conference : CM3 - Computational Multi Physics, Multi Scales and Multi Big Data in Transport Modeling, Simulation and Optimization, Computational Methods and Models for Transport - New Challenges for the Greening of Transport Systems, Springer

https://doi.org/10.1007/978-3-319-54490-8_8

Automatic Customization Framework for Efficient Vehicle Routing System Deployment

Jussi Rasku, Tuukka Puranen, Antoine Kalmbach, and Tommi Kärkkäinen

Abstract Vehicle routing systems provide several advantages over manual transportation planning and they are attracting growing attention. However, deployment of these systems can be prohibitively costly, especially for small and medium-sized enterprises: the customization, integration, and migration is laborious and requires operations research expetise. We propose an automated configuration workflow for vehicle routing system and data flow customization, which provides the necessary basis for more experimental work on the subject. Our preliminary results with learning and adaptive algorithms support the assumption of applicability of the proposed configuration framework. The strategies presented here equip implementers with the methods needed, and give an outline for automating the deployment of these systems. This also opens up new directions for research in vehicle routing systems, data exchange, model inference, automatic algorithm configuration, algorithm selection, software customization, and domain-specific languages.

1 Introduction

Globalization, increased goods consumption, and economic changes pose challenges on transportation logistics. With increasing scale, tightened competition, and environmental concerns, dispatchers stretch their planning capabilities to the limit. Handling all the factors may even be impossible for the human planner [21], which has spawned an interest in commercial automated route planning systems. Combined with the rapid development of IT, this has created a transportation planning tools industry serving operators working with the increasingly complex logistics systems [14].

The advantages of these systems are well known: savings up to 30% in operational costs, reduced planning time, and minimization of human error [42]. Drexl [9] also note the macroeconomic benefits such as improved traffic flow and lowered emissions. If ap-

Jussi Rasku, e-mail: jussi.rasku@jyu.fi · Tommi Kärkkäinen, e-mail: tommi.karkkainen@jyu.fi
Department of Mathematical Information Technology, University of Jyväskylä, FI-40014, Finland

Tuukka Puranen e-mail: tuukka.puranen@nfleet.fi · Antoine Kalmbach
NFleet Oy, Pajatie 8 F, Jyväskylä, FI-40630, Finland

plied at a large scale, deployment of Vehicle Routing Systems (VRSs) can lead to significant economic and environmental benefits.

A VRS is described in Drexl [9] as follows: it is an operational planning software that can read in data with vehicles, drivers, depots, customers, and their respective requests connected to geographical locations. The data defines the specific problem scenario. A map view is often used for visual data verification. A VRS then allows manual, interactive or fully automated (optimization-based) construction of routes. The algorithms, that can build a routing plan conforming to the operational rules such as work time regulations, are the key feature of the system. Finally, the system interacts with an existing resource management system, or allows saving and printing the plans for operational use.

The operation environment for a VRS is complex and dynamic [42, 5, 31] and poses hard to match requirements for commercial software. In a survey from an industry viewpoint, Hoff et al [14] raised a concern that while academic Vehicle Routing Problem (VRP) research has provided efficient algorithms for these problems, they typically use idealized models which omit important facets such as driver breaks, work time regulations, turning restrictions, variation of service times, and congestion. According to Partyka and Hall [32] the providers are having difficulties in providing holistic solutions due to this complexity.

In addition to shortcomings of the idealized models, different logistic operators have differing requirements [5]. As it is not commercially viable to build a unique VRS for each of them [42] the product is made customizable. Here, a VRS designed for easy deployment needs to capture the features of the common VRP variants. Additionally, solving the problems effectively calls for robustness, adaptivity, and reactivity [42].

According to Partyka and Hall [32] routing installations require heavy customization which is mainly done manually. A survey of the Dutch VRS market by Kleijn [21] agrees: most of the software was at least partially tailor-made. The issue has been identified also in academic research. Puranen [36] proposes the use of Software Product Lines (SPL) as a mass-customization strategy. It is a promising approach, as these techniques exploit commonalities in a system to effectively manage variation. Applying SPL in other application domains report order of magnitude reductions in time-to-market, engineering overhead, error rates, and cost [24]. Preliminary experiments in [36] suggest that these benefits are achievable also with VRSs. The extended rationale for the work, as presented in [31], is that the underutilization of route optimization is not due to the shortcomings of models and algorithms, but due to problems in deployment.

The challenges in implementation and deployment call for an approach that could forward the adoption of optimization. In this paper, we propose such an approach as a set of actions and techniques to automate the flow of data through a VRS. Acting upon presented ideas allows utilization of the recent advances in Software Engineering, Machine Learning, and Operations Research.

In related works Desrochers et al [8] describe a VRS that could be used by a consultant with a basic understanding of mathematical programming. Similarly, Maturana et al [30] describe a decision support system generator that substantially lowers the cost of developing such systems, although in their solution the model and data structures has to be still defined manually. Also, Hoff et al [14] envision a tool exhibiting some of the properties presented here. Despite these ideas, a planning and decision support system that allows as extensive automation as ours has not been previously described. No customization framework for the needed automation methods has existed, and their interaction within VRSs has not been previously explored. In this paper, we address this by providing an automated configuration workflow for VRSs and a review on the automation methods for different

phases of the process. The customization framework should be of interest not only to operations researchers, but also to providers of VRSs.

For an overview, Section 2 provides a review of the trends in vehicle routing research. In Section 3 we recognize the opportunities for automation in customizing from data flow perspective. Section 4 we present our proposition for solving some pressing problems in VRS deployment and Section 5 reviews our preliminary experimental results. In Section 6 we conclude our study.

2 Trends in Vehicle Routing Problem Research

VRP has been under intensive research ever since was first introduced by Dantzig and Ramser [7]. VRP concerns the task of finding optimal *routes* for a *fleet* of *vehicles* leaving typically from a *depot* to serve a specified number of *requests*. Objectives can be anything from minimizing the number of vehicles or total travel distance to complex multiobjective business goals. Over 50 years of academic interest has experienced many shifts of research focus. The trends in VRP research as recognized, e.g., by Puranen [36] are illustrated in Figure 1.

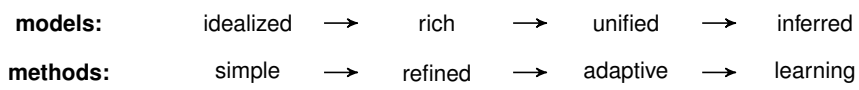


Fig. 1: Trends in vehicle routing research. Adapted from [36]

The early models were **idealized**, partly due to the limitations of computational hardware and solution methods of the time. Since the early days, the trend has been towards more complex and more realistic **"rich"** problems [42, 14]. Rich models extend the classical formulation with complex decisions and objectives as well as can introduce many operational constraints. In addition, a number of new aspects have been proposed; for example Hoff et al [14] call for more explicit handling of stochasticity and risk in the models, and stress the need for research on real time and dynamic routing. For reviews on rich VRP research in the context of commercial solutions, we refer to Hasle and Kloster [13], Drexl [9], and Bräysy and Hasle [5]. Recently, there has been efforts to develop **unified** modeling approaches with generic and flexible modeling structures that can capture the aspects of different VRP variants [40, 13, 18, 48, 37]. Vidal et al [45] make a synthesis on previous research and propose a naming scheme for these variants. Unified modeling frameworks often provide a Domain Specific Language (DSL) for describing the problems. One of the contributions of this paper concerns the rightmost transition in modeling: we argue that the advances in unified modeling enable model **inference** where *composite* optimization model can be automatically or semi-automatically formed by inferring the composition of the model from the problem instance data.

The solution methodologies have followed a similar trend. The first methods relied on **simple** heuristics or on mathematical programming as in the original paper by Dantzig and Ramser [7]. The growing problem size and model complexity led to interest in more **refined** and sophisticated methods. However, due to scale of the real-life problems, exact

solution methods cannot always be used. Thus, a number of heuristics and metaheuristics have been proposed. For surveys in solving VRPs, see, e.g., Toth and Vigo [44], Cordeau et al [6], and Laporte [26]. Recently, there has been interest in **adaptive** and self-adjusting methods where algorithms observe the optimization progress and react accordingly. This trend was recognized, for example, in the survey by Vidal et al [46]. A newer trend is the application of **learning** hyperheuristic systems, which involve using data-driven techniques that enable and disable different algorithms depending on the observed search space. This involves identifying situations similar to those found in the history data or knowledge model. For a survey on using hyperheuristics in combinatorial optimization we refer to Kotthoff [23].

The disadvantage of unified and "rich" models and refined versatile solving methods is that they may make the deployment more complicated [31]. Also, note that in most of the case studies in the aforementioned surveys, the derivation of the model, selection of the algorithms, and fine tuning of the methods is done manually by the researchers based on their expertise. Unfortunately, this does not scale in a commercial setting and poses a barrier for the deployment of VRSs.

3 Data Flow in a Vehicle Routing System

In this section, we outline the *data flow* through a VRS, or more specifically, how the problem instance is passed from system module to another. The flow of information is one of the main aspects affecting the deployment, integration, and utilization of the system. Describing the modular structure of a typical VRS in detail is omitted, and the reader is referred to Drexl [9], Puranen [36], and Bräysy and Hasle [5].

The data flow can be divided into phases as illustrated in Figure 2. First, the data is read from a data storage, such as files or relational database, and then constructed into a domain model (1). Domain model offers primitives for concepts such as truck, driver, and request. The domain model is then translated into optimization model (2). This involves describing the decision variables, the objective function, and the necessary constraints. Note that a DSL or similar technique has to be used to capture the aspects of the specific routing problem. Result of this transformation is a mathematical optimization model that can be then completed with the problem instance specific variable values. The modeled problem can now be fed to the routing algorithms residing in the solver module (3). Effectiveness of the algorithms depends heavily on the algorithm parameters [15]. Thus, when adapting a VRS care is needed to derive a set of suitable parameter values (4). After the optimization (5) the results can be transformed back to the primitives of the domain model (6) which in turn are translated into an actionable plan (7).

Provided that the VRS is generic enough to model a wide set of different and "rich" VRP variants, and that it includes a set of modern metaheuristics and local search based routing algorithms, the biggest effort in adopting a VRS for a new customer is to make sure that the data is read and processed correctly [31]. VRS providers have several options to manage the data flow:

(a) Force an identical data flow for all customers. This will remove much of the flexibility and only a narrow set of problem types can be efficiently addressed by the VRS.

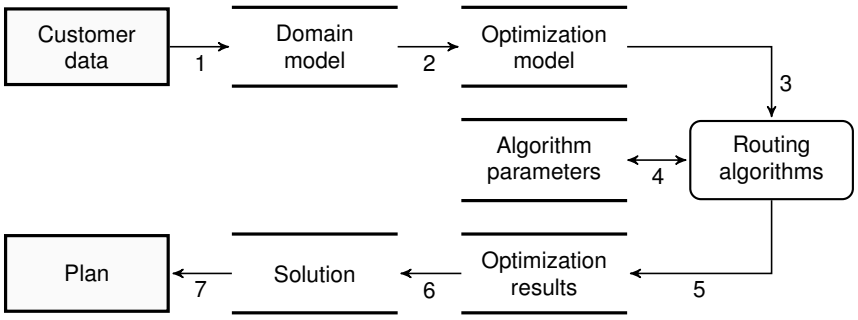


Fig. 2: Data flow of a problem instance trough a VRS.

(b) Customize the data flow manually on a case-by-case basis. Here multiple model variants and use cases can be supported, but the customization heavily depends on manual work and expertise.
(c) Outsource the customization to a third party or to the customer by providing a way to externally configure the system via, e.g. a DSL. The challenge is to provide enough training and sufficient tools for the third party.
(d) Automate the customization so that fixing any given set of functionality inside the VRS is done automatically based on the customer input and data. In addition, if the provider has access to the history data during the customization, the automation might even be learning, that is, with every new modeled problem and deployment the software gains experience.

Designing the software in a way that the flexibility is maximized makes the system applicable in larger number of different contexts, thus making the approaches (c) and (d) feasible. The challenge for (c) is that many logistic operators are small, and lack the necessary expertise to understand the inherent complexity in selecting, configuring and deploying VRSs [31]. Therefore, out of these, the automation based approach (d) is the one that is more scalable and cost-effective. This validates the need for the proposed customization framework.

Each of the data conversion phases Figure 2 expose a potential point of customization. In practice, this *variability* is exposed by configurable behavior of the software system, and it needs to be managed somehow. From a theoretical perspective, this has been addressed by the techniques in the area of software product line engineering (SPLE) [35]. In SPLE, the developed system is divided into two layers: domain layer and application layer. The domain layer of the system captures the generic properties of the current domain, and the application layer is used to define customized application instances with *variability points*. It is a predefined point in the system, in which variation between the applications occur [19]. The specialized expertise required in the customization of VRSs prohibits manually managed mass customization. Instead, we suggest the use of machine-learning based adaptive mass customization techniques, and argue that these represent one of the key technologies in achieving cost-effective routing system deployment.

4 The Automation and Customization Framework

Our main contribution is an outline, or a vision, of how highly automated and easy-to-deploy VRS could be constructed. This customization framework could also enable experimentation with different automation approaches, but here we concentrate on the techniques we have either successfully applied ourselves, or see as pragmatic solutions to the presented opportunities for workflow automation. We limit ourselves to well-known methods used in related fields, and assume a generic solver module capable of expressing a wide set of "rich" VRP variants. The section follows the structure of Figure 2 with each phase having a corresponding subsection.

4.1 Interpreting Customer Data

Input data → Domain model

Interpreting the customer data and transforming it into routing problem starts with the creation of a domain model, which represents the real world entities that form the routing problem. The transformation task consists of taking the problem data as an input and then extracting the data into the domain model. In the simplest case, one can specify a data format that is required and the VRS simply parses this data into a model. It becomes problematic if the parser needs to support different formats. Maintaining numerous many-to-one mappings can quickly become an onerous task.

A likely scenario for data integration is a relational dataset, such as relational database, but in general, any kind of flat dataset with interconnected files can be used. To illustrate, one part of the dataset could consist of ordinary files that pertain to drivers and vehicles, and the other deliveries and locations. Finding semantic links between the relations in these datasets is what we call *join inference*, which in turn is based on foreign key discovery [1, 41]. We propose join inference as a model that can learn the semantic links between a set of relations. It is used to produce a cohesive union of data, the joined relation.

After join inference has been done, we propose the use of schema mapping [4] to extract the required information from the data. Schema mapping consists of finding pairings between two schemas. A *schema* is a formal description of the information contained in a relation; crudely, this would be a set of data attributes, or column headers. Having to find these attribute pairings makes the problem a kind of *data exchange problem* [22], where the goal is to take data from different sources and assimilate it, in this context, to the domain model of a VRS.

4.2 Inferring the Optimization Model

Domain model → Optimization model

After mapping the input data to the domain model, it must be translated into a format understood by the VRP solver. This includes choosing an optimization model. We were unable to find related work on automating this step. Therefore, we proceed to propose four approaches for implementing such an automated transformation:

1. **Separate models:** methods from Section 4.1 can be used on domain model to map it against a selection of optimization models. Out of these, the one with the best fit is selected and completed with instance data. This is suitable approach only if a VRS has support for a limited number of VRP variants.
2. **Coupled models:** a number of domain and optimization models are coupled together with predefined pairwise transformations. Data interpretation from Section 4.1 is done with all domain models in the coupling set and then the one with the best schema mapping (along some criteria) is selected. This has the same constraints as the previous approach.
3. **Model composition:** the optimization model is composed of different objects that may correspond to partial objectives, decisions, or constraints. Filled domain model is matched against each optimization model component and if a threshold is crossed, the component is included to the composite model.
4. **Model reduction:** alternatively, the initial optimization model may be "complete" or unified in a sense that is capable of expressing all the supported VRP features. After doing schema mapping between the domain and optimization models, the unused elements, for which the variable values were not set, are removed from the optimization model.

Besides domain model, other sources for deducing the optimization model include e.g. the vocabulary used in the data. To illustrate, the field revealing that the transportation involves people, refers to use of a dial-a-ride optimization model. The unified naming convention for VRP variants in Vidal et al [46] might prove to be useful in recognizing the different modeling constructs for the model inference. We note that the feasibility of applying automation in this phase is uncertain, mostly because of the lack of prior published research on the topic.

4.3 Selecting the Suitable Optimization Algorithms

Optimization model → Algorithm performance predictions

As mentioned earlier, industrial solutions tend to favor algorithms based on heuristics [42, 5], and many implement a collection of different algorithms to gain extra flexibility. It is also known, that the performance of an algorithm varies greatly between different routing variants and even problem instances [15]. Therefore, it is important to use an algorithm that is efficient in solving the given problem. Portfolio-based algorithm selection techniques such as SATzilla [49] use statistical models to select the algorithm for solving a given problem instance. In VRS this approach could be applied to select the higher level algorithmic components: a metaheuristic could be selected based on the instance characteristics and predicted performance.

Another way to improve solver performance is the utilization of so called *hyperheuristics*. Instead of using a single algorithm or a manually constructed combination, a hyperheuristic acts as a high level learning "supervisor" algorithm that *selects and combines* lower level algorithms from a portfolio on the fly.

Similar ideas have been tried in VRP, for example by using several simple heuristics in varied order to escape local optima. Pisinger and Ropke [34] proposed a mehtod, where adaptive heuristic selection is done among intensification and diversification heuristic op-

erators. Garrido has proposed the use of hyperheuristics to select local search operators in solving different VRP variants [11]. VRP was also one of the problem domains in Cross-domain Heuristic Search Challenge (CHeSC2011) where a number of domain independent hyperheuristics were evaluated [47].

We note that these schemes should be useful when adapting an industrial VRS to a new set of end user provided sample problem instances. Our experimental work to explore these possibilities is in preparation.

4.4 Configuring the Optimization Algorithm

Optimization model & Observed performance → Algorithm parameter values

The algorithms used to solve hard computational problems often reveal parameters that can be used to change the behaviour of the algorithm and adapt it into solving a specific problem instance [15]. The settings of the algorithm parameters have a substantial effect on the performance of the algorithm. However, setting them manually is a non-trivial task requiring expertise and effort trough experimentation [15]. Therefore, automatic search approaches have been proposed to what is in literature known as the problem of *automated algorithm configuration (AAC)*.

In practice, AAC can be used to automatically adapt the a routing solver for each VRS deployment. This allows the VRS provider to get the best performance out of the implemented algorithms. Also, after enough experimentation, archetypes of routing problems might emerge. With this history data the previous configuration effort could be reused to provide more varied algorithm defaults for the solver. In fact, several AAC methods have proven successful also with VRP [33]. Of particular interest in this context is the work in Becker et al [3], where they tuned the parameters of a commercial VRP solver with real-world routing problem instances. Our recent experimental work [39] verifies this and gives suggestions on which AAC methods to use to configure VRP metaheuristics.

Current state-of-the-art methods like SMAC from Hutter et al [17] or I/F-Race from Balaprakash et al [2] support all parameter types, are able to use extra information like the parameter structure, interactions or hierarchies, and use several intensification techniques that aim to save computationally expensive parameter configuration evaluations. The benefits of can be striking: Hutter et al [16] were able to achieve up to 50-fold speedup over the default parameters of the CPLEX solver.

The main challenge of applying AAC in routing, however, is that the runtime on real world routing cases may be hours, especially in presence of complex constraints [3]. Fortunately, the focus of the AAC research has been recently shifting onto overcoming these challenges, see e.g. Mascia et al [29].

4.5 Solving the VRP Problems

Optimization model & Algorithms and their parameter values → Optimization results

The solver module is responsible of performing the optimization, which contains the tasks of mapping of tasks to vehicles as well as routing the vehicles as efficiently as possible

according to the objective function. The search is performed until a predefined stopping criterion has been met, or the user ends the process.

From the process perspective, the ability to predict and adjust the runtime is a major concern. The same system may be operated under tight time-constraints for planning, whereas some users prefer the added robustness of a more thorough search. It is probable that this variability is exposed e.g. as stopping criteria.

Another viewpoint to solver module customization is the availability of computational resources. In many cases, the routing system is still run in a desktop environment, but increasingly, optimization services are available through cloud services [5]. This opens a new dimension in the customization, namely the allocated computing time, resources and priority based on the customer requirements, service level agreements, and instance characteristics, which all adds in to the complexity of deploying the system.

4.6 Interpreting the Optimization Results

Optimization model & Optimization results → Domain model (solution)

The optimization solver module usually returns the resulting plan in the mathematical format it uses internally. The interpretation of the optimization results has a direct connection to the construction of the optimization model. Whereas in model construction the decision variables are selected based the data in the domain model, in this phase the values of the decision variables need to be interpreted back to the relations and values of the entities in the domain model. We can use an inverse transformation of the one in Subsection 4.2 to decode the solution.

One issue in the interpretation of the results is the type of the decoding. It may be that the decoding is not one-to-one. That is, there may be multiple possible plans the optimized solution can be decoded to. For example, in a classical VRP all the trucks are identical and it does not matter how the vehicles and routes are mapped Puranen [36]. This potential unambiguity may have undesired consequences if it is not taken into account.

4.7 Producing a Formatted Plan

Domain model (solution) → Output plan

Ultimately the user of a VRS needs to apply the plan into practice. Different users have different formats, output data requirements, and reporting needs, so in the final data transformation step an automated VRS could adapt its output to the format most convenient to the end user.

If the interfaced system includes plan generation, it could be enough to do the schema mapping procedure from Section 4.1 in reverse. The existing system would then compose the output document to that is to be handed to the drivers. Another option is to infer the structure of an example document using methods such as table extraction, visual object and information extraction, and entity identification [27, 25]. This produces a template which then can be filled with the relevant data from the solver. Similar technique has been used, for example, in web page content and structure extraction to reformat the web page content for mobile clients [25].

5 Preliminary Experimental Results

To demonstrate the potential of automatic configuration of route optimization algorithms, we configured the three metaheuristics (Record-to-Record travel *RTR*, simulated annealing *SA*, and ejection *EJ*) provided by the VRPH library [12] on four real world based benchmark instances from the literature. For details of the experimental setup see Rasku et al [39].

The target problem instances were `F-n45-k4`, `F-n72-k4`, and `F-n135-k7` from [10] with 45, 72 and 135 requests and the 385 request instance `tai385` from Taillard [43]. The `tai385` instance is generated based on the locations and census of population data from canton of Vaud in Switzerland, whereas the instances `F-n45-k4` and `F-n135-k7` are from a day of grocery deliveries from the Ontario terminal of National Groceries Limited. The `F-n72-k4` instance data is obtained from Exxon associated case involving the delivery of tires, batteries and other accessories to gas stations. We used SMAC [17] (version 2.3.5) and Iterated F-Race [2] implementation described in [28] (version 1.0.7) and restricted to evaluation budget of 500 invocations. Each metaheuristics was configured separately for each of the target problem instance. A 30 second cutoff was used for the solvers.

Table 1: Average AAC results for all solver-instance pairs.

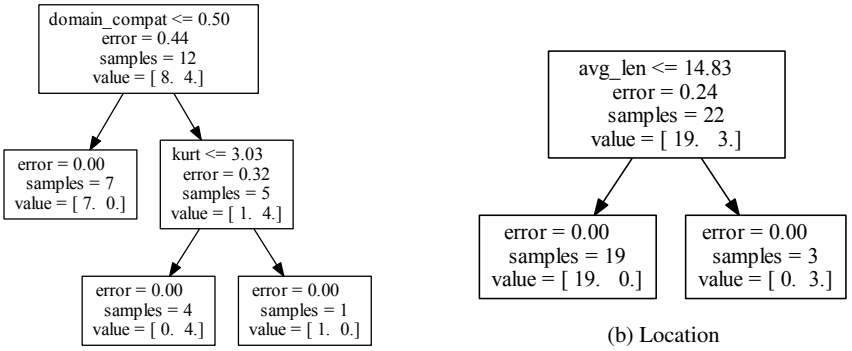
Results are given as percentage from the best known solution (relative optimality gap). Statistically better ($p < 0.05$) results are bolded, evaluation budget violations of more than 5% are in italics, and the standard deviation over 100 VRP solutions is given in parentheses.

	Quality on the target instance			Quality on the other instances		
Target	Defaults	I/F-Race	SMAC	Defaults	I/F-Race	SMAC
F-n45 EJ	0.49 (0.35)	**0.12 (0.23)**	**0.15 (0.25)**	2.57 (2.19)	**2.21 (1.41)**	2.70 (2.07)
F-n45 RTR	0.48 (0.40)	0.01 (0.04)	**0.00 (0.00)**	11.25 (0.40)	**5.32 (3.01)**	**6.02 (3.56)**
F-n45 SA	0.30 (0.34)	**0.03 (0.14)**	**0.01 (0.09)**	8.91 (1.54)	**6.55 (4.97)**	**7.68 (6.70)**
F-n72 EJ	0.99 (2.15)	**0.19 (1.03)**	**0.16 (1.11)**	**1.98 (0.54)**	2.15 (0.88)	2.11 (0.82)
F-n72 RTR	6.63 (0.28)	**0.00 (0.00)**	**0.00 (0.00)**	4.94 (0.51)	**3.86 (1.01)**	**3.66 (1.02)**
F-n72 SA	3.80 (1.75)	**0.05 (0.15)**	**0.02 (0.09)**	5.06 (0.52)	**2.66 (1.11)**	**3.06 (1.41)**
F-n135 EJ	0.24 (0.29)	**0.19 (0.28)**	**0.17 (0.15)**	2.96 (2.58)	2.01 (1.61)	**1.88 (0.92)**
F-n135 RTR	1.62 (0.07)	0.06 (0.08)	**0.02 (0.03)**	9.94 (0.57)	**4.71 (3.00)**	5.65 (2.17)
F-n135 SA	0.11 (0.07)	*0.14 (0.14)*	**0.08 (0.06)**	8.99 (1.57)	***6.42 (3.65)***	**6.25 (2.89)**
tai385 EJ	1.23 (0.28)	1.10 (0.23)	**1.02 (0.18)**	1.92 (2.46)	**0.72 (0.53)**	**0.76 (0.43)**
tai385 RTR	2.91 (0.27)	1.00 (0.22)	**0.88 (0.18)**	8.61 (0.48)	3.99 (2.09)	**3.47 (1.85)**
tai385 SA	4.67 (0.40)	***1.04 (0.24)***	1.18 (0.27)	3.74 (2.06)	***2.15 (2.51)***	5.79 (4.63)

Results of the configuration runs are presented in Table 1. On average, the quality of the results was improved by 1.65 percentage points with the use of AAC, which means that the relative optimality gap was closed by 73%. Furthermore, the performance of the metaheuristics was more consistent when configured, as can be observed from the standard deviations.

Because the metaheuristics were configured for each instance separately, we acknowledge the danger of overtuning [16]. To observe the effect, the rightmost three columns of Table 1 present the performance of the resulting parameter configurations on the other

three remaining instances. These columns can be interpreted as the result of a 4-fold cross-validation. The configurators overfit only for the `F-n72-k4`, and for all other targets the solution quality is statistically significantly improved on average by 2.4 percentage points (a 37% improvement). The solver behavior becomes slightly more erratic as can be perceived from the standard deviations. However, this is likely to be a byproduct of the improved solution quality and the more rugged fitness landscape of a multi-instance problem set. As suggested by the results in [39], if tuned on the entire instance set, the robustness of the solvers on similar instances is expected to improve.



(a) Capacity

(b) Location

Fig. 3: Decision trees for the schema mapping of two domain model attributes

Our proposed solution to increase the level of automation in the data import phase is presented in [20]. To summarize, Kalmbach [20] provided a formulation for the data import and model inference problem, presented a decision trees [38] based approach for join inference and schema mapping, and explored its applicability in importing of schemaless routing instance data. Two decision trees for *capacity* and request *location* mapping are provided in Figure 4.4 as an illustration of the generated inference rules. The proof-of-concept tool is able to recognize the nature of each column in a column-oriented input for the generated test data, and is thus capable of generating simple mapping rules between input and the domain model.

6 Conclusions

Vehicle routing systems provide several advantages over manual transportation planning, but the deployment of these systems is in many cases laborious and costly. In addition, migration from the current system with associated customization and integration challenges create practical obstacles that prevent the latest advances in operations research from being disseminated to wide use. The focus in academic research is in modeling and solving efficiency whereas in commercial routing systems usability, flexibility, and scalability are more important. Tighter interaction between the two is needed in order to effectively solve real-world routing problems [5].

The advances in technologies such as GPS and RFID, and drop in data warehousing prices, have made transportation big data collection possible and economical. Concurrently, logistics operators have begun to see the information as a vital asset that can be used in decision making. This opens new possibilities for machine learning, for which the accuracy is dependent of the amount and availability of data that can be used to train the models. Therefore, these trends have paved the road for a new generation of vehicle routing systems that can utilize machine learning to automate the customization and deployment. This in turn has the potential to increase the effectiveness and robustness as the system can be adapted automatically to the particularities of a problem instance. The goal is to diminish the importance of an operations researcher in the deployment process and consequently to permit higher scalability and more widespread deployment of route optimization.

In this paper, we have outlined a customization framework for the automation of data transformation operations inside a routing system. Our framework recognizes seven transformation steps, each open for system customization. We also provide suggestions on automating these steps. Our preliminary empirical results are promising, but further experimental work is required to establish whether all the proposed techniques are fully applicable in practice.

To evaluate the proposed customization framework, we reflect it against the framework for analyzing VRS deployment published in [31]. Neittaanmäki and Puranen [31] recognize several practical adoption and deployment barriers for the VRSs. They see the involvement of an optimization expert as a prohibiting investment and call for an increased automation of the deployment process. Their deployment process is split into three phases: data, process, and system integration. To see in which extent our proposed customization framework can resolve the 18 barriers they recognized, we proceed to give some possible solutions to the recognized issues: In data integration step, the missing, low quality and incomplete data could be automatically imputed, or at least recognized with machine learning. The data structure inference from Subsection 4.1 can help when acquiring and combining the data from existing systems. In addition, because of the techniques proposed in Subsection 4.2, it takes less expertise to generate the optimization model. As demonstrated by our experiments, the plan quality can be improved, sometimes significantly, using automatic algorithm configuration (Subsection 4.4). Use of automation results into lower perceived complexity and improved usability that can instill trust in the users to the system and to the plans it generates. On the system integration level, the automation makes integration easier and faster, which in turn can make the system deployment cheaper, less dependent on expertise and other resources, and flexible to the current and future changes in operations.

Taken together, we argue that in order to bring the latest academic routing knowledge to the hands of logistics operators in a massive scale, the automatic configuration approach, as presented in this paper, is needed. The recent trends in VRP research seem to converge towards generic reusable modeling and highly adaptive and configurable modeling frameworks, but we have shown that several other areas in practical system integration and deployment need to be considered in order to effectively apply these into practice. This requires extensive further studies in several disciplines, but should provide a promising area of research with a potential for a wide array of practical benefits.

References

[1] Acar AC, Motro A (2009) Efficient discovery of join plans in schemaless data. In: Proceedings of the 2009 International Database Engineering & Applications Symposium, ACM, New York, NY, USA, IDEAS '09, p 111

[2] Balaprakash P, Birattari M, Stützle T (2007) Improvement strategies for the F-Race algorithm: Sampling design and iterative refinement. Tech. rep., IRIDIA, Université Libre de Bruxelles

[3] Becker S, Gottlieb J, Stützle T (2006) Applications of racing algorithms: an industrial perspective. In: Proceedings of the 7th international conference on Artificial Evolution, Springer-Verlag, Berlin, Heidelberg, EA'05, pp 271–283

[4] Bellahsene Z (2011) Schema Matching and Mapping. Springer

[5] Bräysy O, Hasle G (2014) Software Tools and Emerging Technologies for Vehicle Routing and Intermodal Transportation, SIAM, chap 12, pp 351–380. MOS-SIAM Series on Optimization

[6] Cordeau JF, Gendreau M, Hertz A, Laporte G, Sormany JS (2005) New heuristics for the vehicle routing problem. In: Logistics Systems: Design and Optimization, Springer-Verlag, New York, chap 9, pp 279–297

[7] Dantzig GB, Ramser JH (1959) The truck dispatching problem. Management Science 6(1):80–91

[8] Desrochers M, Jones CV, Lenstra JK, Savelsbergh MWP, Stougie L (1999) Towards a model and algorithm management system for vehicle routing and scheduling problems. Decision Support Systems 25(2):109–133

[9] Drexl M (2011) Rich vehicle routing in theory and practice. Tech. Rep. 1104, Gutenberg School of Management and Economics, Johannes Gutenberg University Mainz

[10] Fisher ML (1994) Optimal solution of vehicle routing problems using minimum k-trees. Operations Research 42(4):626–642

[11] Garrido P, Riff MC (2010) DVRP: a hard dynamic combinatorial optimisation problem tackled by an evolutionary hyper-heuristic. Journal of Heuristics 16(6):795–834

[12] Groër C, Golden B, Wasil E (2010) A library of local search heuristics for the vehicle routing problem. Mathematical Programming Computation 2(2):79–101

[13] Hasle G, Kloster O (2007) Industrial vehicle routing. In: Geometric modelling, numerical simulation, and optimization, Springer, pp 397–435

[14] Hoff A, Andersson H, Christiansen M, Hasle G, Løkketangen A (2010) Industrial aspects and literature survey: Fleet composition and routing. Computers & Operations Research 37(12):2041–2061

[15] Hoos HH (2012) Automated algorithm configuration and parameter tuning. In: Autonomous Search, Springer, pp 37–71

[16] Hutter F, Hoos HH, Leyton-Brown K (2010) Automated configuration of mixed integer programming solvers. In: CPAIOR, Springer, Lecture Notes in Computer Science, vol 6140, pp 186–202

[17] Hutter F, Hoos H, Leyton-Brown K (2011) Sequential model-based optimization for general algorithm configuration. In: Learning and Intelligent Optimization, Springer, pp 507–523

[18] Irnich S (2008) A unified modeling and solution framework for vehicle routing and local search-based metaheuristics. INFORMS Journal on Computing 20(2):270–287

[19] Jacobson I, Griss M, Jonsson P (1997) Software reuse: architecture, process and organization for business success. ACM Press/Addison-Wesley Publishing Co. New York, USA
[20] Kalmbach A (2014) Fleet inference : importing vehicle routing problems using machine learning. Master's thesis, University of Jyväskylä, Department of mathematical information technology
[21] Kleijn MJ (2000) Tourenplanungssoftware: ein vergleich für den niederländischen markt. Internationales Verkehrswesen 52(10):454–455
[22] Kolaitis PG (2005) Schema mappings, data exchange, and metadata management. In: Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, ACM, New York, NY, USA, PODS '05, p 6175
[23] Kotthoff L (2014) Algorithm selection for combinatorial search problems: A survey. AI Magazine 35(3):48–60
[24] Krueger CW (2002) Easing the transition to software mass customization. In: Linden F (ed) Software Product-Family Engineering, Lecture Notes in Computer Science, vol 2290, Springer Berlin Heidelberg, pp 282–293
[25] Krüpl-Sypien B, Fayzrakhmanov RR, Holzinger W, Panzenböck M, Baumgartner R (2011) A versatile model for web page representation, information extraction and content re-packaging. In: Proceedings of the 11th ACM symposium on Document engineering, ACM, New York, USA, pp 129–138
[26] Laporte G (2007) What you should know about the vehicle routing problem. Naval Research Logistics 54(8):811–819
[27] Lin X, Hui C, Nelson G, Durante E (2006) Active document versioning: from layout understanding to adjustment. In: Taghva K, Lin X (eds) Document Recognition and Retrieval XIII, SPIE, SPIE Proceedings, vol 6067
[28] López-Ibánez M, Dubois-Lacoste J, Stützle T, Birattari M (2011) The irace package, iterated race for automatic algorithm configuration. Tech. rep., IRIDIA, Université Libre de Bruxelles
[29] Mascia F, Birattari M, Stützle T (2013) An experimental protocol for tuning algorithms on large instances. In: Learning and Intelligent Optimization, Springer
[30] Maturana S, Ferrer JC, Barañao F (2004) Design and implementation of an optimization-based decision support system generator. European Journal of Operational Research 154(1):170–183
[31] Neittaanmäki P, Puranen T (2015) Scalable deployment of efficient transportation optimization for smes and public sector. In: Advances in Evolutionary and Deterministic Methods for Design, Optimization and Control in Engineering and Sciences, Springer, pp 473–484
[32] Partyka J, Hall R (2012) Software survey: Vehicle routing. OR/MS Today 39(1)
[33] Pellegrini P, Birattari M (2006) The relevance of tuning the parameters of metaheuristics. Tech. rep., IRIDIA, Université Libre de Bruxelles
[34] Pisinger D, Ropke S (2007) A general heuristic for vehicle routing problems. Computers & Operations Research 34(8):2403–2435
[35] Pohl K, Böckle G, van der Linden FJ (2005) Software Product Line Engineering: Foundations, Principles and Techniques. Springer
[36] Puranen T (2011) Metaheuristics meet metamodels – a modeling language and a product line architecture for route optimization systems. PhD thesis, University of Jyväskylä, Jyväskylä studies in computing;1456-5390;134

[37] Puranen T (2012) Producing routing systems flexibly using a VRP metamodel and a software product line. In: Operations Research Proceedings 2011, Springer, pp 407–412
[38] Quinlan JR (1986) Induction of decision trees. Machine learning 1(1):81–106
[39] Rasku J, Musliu N, Kärkkäinen T (2014) Automating the parameter selection in VRP: An off-line parameter tuning tool comparison. In: Modeling, Simulation and Optimization for Science and Technology, Computational Methods in Applied Sciences, vol 34, Springer, pp 191–209
[40] Ropke S, Pisinger D (2006) A unified heuristic for a large class of Vehicle Routing Problems with Backhauls. European Journal of Operational Research 171(3):750–775
[41] Rostin A, Albrecht O, Bauckmann J, Naumann F, Leser U (2009) A machine learning approach to foreign key discovery. In: 12th International Workshop on the Web and Databases (WebDB)
[42] Sörensen K, Sevaux M, Schittekat P (2008) Multiple neighbourhood search in commercial VRP packages: Evolving towards self-adaptive methods. In: Adaptive and multilevel metaheuristics, Springer, pp 239–253
[43] Taillard E (1993) Parallel iterative search methods for vehicle routing problems. Networks 23(8):661–673
[44] Toth P, Vigo D (eds) (2002) The vehicle routing problem. SIAM
[45] Vidal T, Crainic TG, Gendreau M, Prins C (2012) A unified solution framework for multi-attribute vehicle routing problems. Tech. rep., CIRRELT
[46] Vidal T, Crainic TG, Gendreau M, Prins C (2013) Heuristics for multi-attribute vehicle routing problems: A survey and synthesis. European Journal of Operational Research 231(1):1 – 21
[47] Walker JD, Ochoa G, Gendreau M, Burke EK (2012) Vehicle routing and adaptive iterated local search within the hyflex hyper-heuristic framework. In: Learning and Intelligent Optimization - 6th International Conference, Springer, Lecture Notes in Computer Science, vol 7219, pp 265–276
[48] Welch PG, Ekárt A, Buckingham C (2011) A proposed meta-model for combinatorial optimisation problems within transport logistics. In: MIC 2011: The IX Metaheuristics International Conference, vol IX
[49] Xu L, Leyton-brown K (2008) SATzilla : Portfolio-based Algorithm Selection for SAT. Artificial Intelligence 32:565–606

PII

SOLUTION SPACE VISUALIZATION AS A TOOL FOR VEHICLE ROUTING ALGORITHM DEVELOPMENT

by

Jussi Rasku, Tommi Kärkkäinen, Pekka Hotokka 2013

Proceedings of the Finnish Operations Research Society 40th Anniversary Workshop (FORS40)

Solution Space Visualization as a Tool for Vehicle Routing Algorithm Development

Jussi Rasku*, Tommi Kärkkäinen* and Pekka Hotokka*
*Department of Mathematical Information Technology,
University of Jyväskylä, P.O. Box 35, FI-40014, Finland
Email: {jussi.rasku, pekka.hotokka, tommi.karkkainen}@jyu.fi

***Abstract*—Understanding the exploration of search space of vehicle routing problems is important in designing efficient algorithms. These spaces are multidimensional and hard to visualize, which makes it difficult to examine the trajectories of the search. We use a technique based on multidimensional scaling to visualize objective function surfaces for such problems. The technique is used to examine a full objective function surface of small VRP and TSP problem instances, and the neighbourhood as observed by heuristic search algorithms.**

I. Introduction

Combinatorial optimization problems are usually inherently multidimensional. For example, the well known travelling salesman problem (TSP) has n^2 binary decision variables, where n is the number of cities to visit. Understanding, let alone visualising such solution spaces is challenging. A visualization technique that could plot the trajectory of an algorithm in search space would be a useful tool for algorithm developers (Halim and Lau, 2007).

In this study we present a visualisation technique based on multidimensional scaling (MDS). We address the limitations and challenges of the presented technique, such as the definition of the distance metric between solutions. We will also show how to use the presented visualization technique to depict two well known routing problems, the TSP and the capacitated vehicle routing problem (CVRP). To our knowledge, this is the first time this kind of technique is used to make observations of routing problem solution spaces.

In Section II we give a brief overview to the vehicle routing problem. In Section III we review previous relevant work on understanding VRP solution spaces. Section IV describes our visualization technique and Section V gives examples of the different landscapes and addresses the accuracy of the method. We conclude our study in section VI.

II. The Vehicle Routing Problem

VRP is one of the most studied $\mathcal{NP}$-hard combinatorial optimization problems (Toth and Vigo, 2002). The task in VRP is to find optimal *routes* for *vehicles* leaving from a *depot* to serve a specified number of *requests*. In its archetypical form each customer must be visited once by exactly one vehicle. Each vehicle must leave from the depot and return there after serving customers on its route. Typical objective is to minimize the total length of the routes.

Following Toth and Vigo (2002) a graph formulation for VRP can be given. Let $V = 0, \ldots, n$ be the set of vertices where the depot has the index 0 and where the indices $1, ..., n$ correspond to the customers. The graph $G = (V, E)$ is complete with each edge $e = (i, j) \in E$ having an associated non-negative c_{ij} that is the cost of traversal from vertex i to j. Each of the edges $(i, j) \in E$ has an adjoining binary decision variable x_{ij} to decide whether traverse the edge. Using the notation of the graph formulation, a linear programming model for VRP can be given (Toth and Vigo, 2002).

$$\min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \tag{1}$$

$$\text{s.t.} \sum_{i \in V} x_{ij} = 1, \ \sum_{i \in V} x_{ji} = 1 \qquad \forall j \in V \setminus \{0\} \tag{2}$$

$$\sum_{i \in V} x_{i0} = K, \ \sum_{j \in V} x_{0j} = K \tag{3}$$

$$\sum_{i \notin S} \sum_{j \in S} x_{ij} \geq \gamma(S) \qquad \forall S \subseteq V \setminus \{0\}, S \neq \emptyset \tag{4}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in V \tag{5}$$

Different routing problem variants can derived by defining specialized objectives and by adding additional constraints. Out of these, the most usual one, CVRP is used as an illustrative example in this work. In CVRP a set of identical vehicles, with a *capacity* of C, serve requests with a non-negative demand of d_i. The problem is to find the lowest cost solution with minimum number of tours K^* so that the capacity C is not exceeded by any of the tours.

III. On Solution Space Analysis

Statistical analysis of fitness landscapes has proven to be useful approach in understanding solution space structure (Weinberger, 1990). Previous approaches for examining routing problem search spaces, and observations of thereof, are explored in e.g. (Fonlupt et al., 1999; Kubiak, 2007; Czech, 2008; Pitzer et al., 2012). These methods involve probing the solution space in order to calculate statistical measures that describe the behaviour of the objective function.

These studies have revealed that routing problem search space is rugged, multimodal and tends to have a "big valley" structure where the local optima are clustered close to each other. Landscape ruggedness describes the expected amount of local variance of the objective function values around any given neighbourhood.

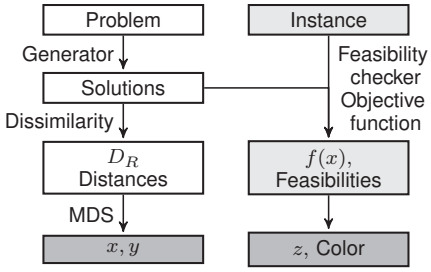
Halim and Lau (2007) and Mascia and Brunato (2010) have previously introduced tools that can be used in stochastic local search (LS) landscape visualization and analysis. The aim of these studies has been to further the understanding of the problems. Because of the differences in underlying nature of combinatorial optimization problems, we argue that problem structure exploration tools need to be adapted to different domains.

IV. Description of the used method

Biggest challenge in visualizing the search space of VRP comes from the multidimensional nature of the problem. A combinatorial optimization problem, such as VRP, may have hundreds or thousands of binary decision variables. A technique called multidimensional scaling (MDS) can be used to bring the dimensionality down to two or three dimensions. (Brunato and Battiti, 2010) Following tasks are expected:

(A) Find a solution distance (or dissimilarity) measure.
(B) Implement an effective method of enumerating the solutions for a full search space visualization.
(C) Select the MDS method.
(D) Draw the visualization.

Fig. 1: Visualization process



Our visualization technique uses a process that is described in Figure 1. The problem with the size n; and instance with capacity constraint, Q distance matrix D, and demands d_i; can be treated separately. For the problem, all solutions are generated and their distances to each other calculated using a dissimilarity measure. MDS produces the x and y coordinates for the visualization. Then, feasibility and objective function values are calculated for each solution of the instance. From objective values we get the z coordinate and from fitness the colour that is used to differentiate feasible and infeasible solutions.

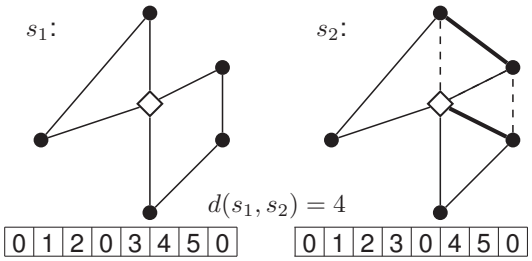
A. Solution similarity

Let $s_1, s_2 \in R$, be two solutions of a symmetric VRP. The distance between solutions $d(s_1, s_2)$ can be defined as the number of edges *not shared* by the solutions. Thus, d is a dissimilarity measure of VRP solutions. For operators like 2-opt and relocate (Fig. 2), this Manhattan distance has been shown to be a good metric to be used with routing problems (Fonlupt et al., 1999). However, other binary dissimilarity measures such as Dice, Jaccard, Yule, Russel-Rao, Rogers-Tanimoto and Sokal-Sneath (Choi et al., 2010) can also be used, and we will present the first comparison of these measures in the domain of routing problems.

B. Enumerating VRP solutions

We needed an effective way of enumerating all solutions without capacity, time window, or similar constraints. Note that the node degree (2), (3), subtour elimination (4), and variable type (5) constraints from Section (II) are still respected. If we let $\forall K : K \in \{1 \dots n\}$, all possible solutions are created. We consider only the symmetric case where the graph is undirected. Therefore, routing problems where the direction of the tour is important (PDP, VRPTW etc.), are not considered in this study.

Fig. 2: Relocate* operator and giant tour encoding



We implemented three algorithms to enumerate the solutions. 1) A permutation based algorithm lists all possible choices for the first points for K routes. All permutations of remaining points are then divided to the routes in all possible ways. 2) A matrix based algorithm that uses an integer interpreted as binary upper strict triangle matrix of decision variables. Increase by 1 gives a new solution. With constraints to the sum of a row/column we can prune entire branches of constraint breaking solutions. 3) An algorithm based on generation of all Hamiltonian paths after which visits to the depot are recursively added. A pseudocode of a subprocedure for the last method is given below.

Require: giant tour encoded solution s
Require: a, b are indexes of s: $a, b \in \{0, \dots, |s| - 1\}$
  $R_t \leftarrow \emptyset$
  if $b - a \geq 2$ **then**
    for $c = a$ **to** b **do**
      $t_{c^-}, t_{c^+} \leftarrow$ the tours of s before and after c
      $t_{b^+} \leftarrow$ the tour of s after b
      $C_1 \leftarrow s[c] \neq 0$ **and** $s[c+1] \neq 0$
      $C_2 \leftarrow t_{c^-}.first() < t_{c^-}.last()$
      $C_3 \leftarrow t_{c^+}.first() < t_{c^+}.last()$
      $C_4 \leftarrow t_{c^-}.first() < t_{c^+}.first()$
      $C_5 \leftarrow |t_{b^+}| > 0$ **and** $t_{c^+}.first() < t_{b^+}.first()$
      if C_1 **and** C_2 **and** C_3 **and** C_4 **and** C_5 **then**
        $s^* \leftarrow$ copy of s with visit to depot at c
        $R_t \leftarrow R_t \cup \{s^*\}$
        $R_t \leftarrow R_t \cup$ call algorithm recursively with
          $s' \leftarrow s^*, a' \leftarrow a, b' \leftarrow c + 1$
        $R_t \leftarrow R_t \cup$ call algorithm recursively with
          $s' \leftarrow s^*, a' \leftarrow c + 1, b' \leftarrow b$
      end if
    end for
  end if
  return R_t

We use giant tour encoding (Toth and Vigo, 2002) with 0 indicating the depot as illustrated in Figure 2. In addition we

use rules that require the tours to be directed so that the first node has always smaller index than the last node (conditions C_2, C_3 of the procedure) and that tours are ordered according to their first nodes (conditions C_4, C_5). These enforce unique encoding of the solutions.

Require: $N \in \mathcal{N} \leftarrow$ Size of the CVRP instance
  $R \leftarrow \emptyset$
  for all $t \in$ Hamiltonian paths **do**
    if $t.first() < t.last()$ **then**
      $R \leftarrow R \cup \{t\}$
      $R \leftarrow R \cup$ call algorithm with
        $s' \leftarrow t, a' \leftarrow 1, b' \leftarrow N$
    end if
  end for
  return R

The algorithm described above is then used to generate all solutions of an asymmetric VRP using the subprocedure described earlier. Generation of Hamiltonian cycles can simply be done generating permutations of the request indices.

C. Multidimensional Scaling

MDS is a name for a family of techniques of dimensionality reduction for data analytics and visualization. Input is proximity data in the form of a dissimilarity matrix $D^{N\times N}$ of high-dimensional coordinate points and the output embeddings of these points into a lower dimensional space $\mathcal{R}^d$. Several methods exist, but in this work we use the popular SMACOF stress majorization algorithm (Borg and Groenen, 2005). As the minimization target we used the Kurskal stress function:

$$\sigma(X, D) = \sqrt{\sum_{i<j} (d_{ij} - \hat{d}_{ij})^2} \tag{6}$$

We used the implementation from Orange Data Mining library[1] with Torgerson initialization (Borg and Groenen, 2005) that gives an analytic solution to the MDS. The SMACOF was terminated after 100 iterations or when termination condition $\sigma < 0.001$ was met.

D. Drawing the Visualization

To draw the visualization we used Mayavi2[2] library for scientific 3D data visualization. We used Delaunay 2D triangulation to create a mesh which was then plotted with Mayavi surface pipeline. LS algorithm trace is drawn using 3D parabolas as "jumps" from solution to another.

The accuracy of the visual representation can be verified using fitness landscape analysis measures. This idea is one the major contributions of this paper. We want to examine if important solution space features, such as the autocorrelation length λ of a random walk (RW) (Weinberger, 1990), number of local optima $\#LO$ discovered with random initialization and LS, and distribution of the aforementioned local optima $(\bar{d}(LO_i, LO_i^*),\ \sigma_d)$, are preserved. To do this we need to define a LS operator in the visualization space. In this work we chose to use k-nearest neighbour search, where k is the length giant tour encoding of the current solution.

[1] http://orange.biolab.si/

[2] http://code.enthought.com/projects/mayavi/

V. Results

The described generation method is capable of producing around 220 000 solutions per second on a 2.13GHz Intel Core 2 Duo workstation. For any practical purposes this is enough.

TABLE I: Number of solutions with different n

n	3	4	5	6	7	8	9	10
$\lvert R_{\text{CVRP}}\rvert$	7	34	206	1486	12412	117692	1248004	14625856
$\lvert R_{\text{TSP}}\rvert$	1	3	12	60	360	2520	20160	181440

Because all three implemented algorithms produced the same number of solutions at least up to $n = 10$ we feel confident that all VRP solutions are enumerated. Enumeration with different number of customers yields a sequence shown in Table I. To allow comparison, we have also listed the number of possible solutions for a corresponding TSP expressed with $|R_{\text{TSP}}| = (n-1)!/2$.

Our empirical study of the number of solutions to symmetric CVRP leads to Formula (7) (Wilf, 2012). We have verified it produces the correct number of solutions least up to $N = 13$. By presenting the result, we would like to open discussion of the implications of knowing the exact number of solutions.

$$|R_{\text{CVRP}}| = \sum_{k=0}^{n} |s_1(n,k)|\, b(k), \quad \text{where} \tag{7}$$

$$s_1(n,m) = \sum_{k=0}^{n-m} \binom{n-1+k}{n-m+k} \binom{2n-m}{n-m-k} s_2(n-m+k, k) \tag{8}$$

$$s_2(n,m) = \sum_{k=0}^{m} \frac{\binom{m}{k} k^n}{m!(-1)^{(k-m)}} \tag{9}$$

$$b(n) = \sum_{k=0}^{n} \sum_{i=0}^{k} \frac{(-1)^i (k-2i)^n \binom{k}{i}}{2^k k!}, \quad b(0) = 1 \tag{10}$$

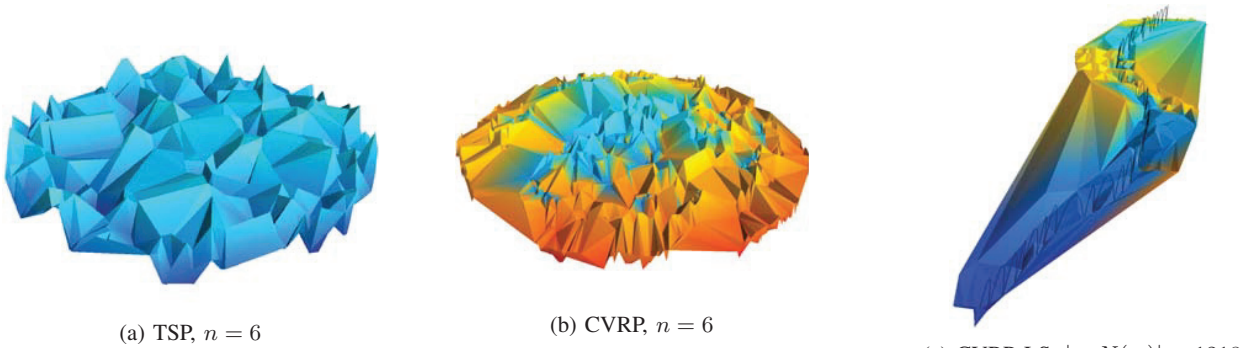
In order to examine the suitability of the distance measures we compare MDS stress. As a reference we use fitting of a uniformly random point cloud inside a unit 3D ball down to 2D coordinates. Reference provides us an intuition of how "tight" it gets when fitting CVRP solutions into 2D. The results are presented in Table II. Dice and Manhattan seem to be the ones showing stable behaviour and least stress.

TABLE II: MDS stress comparison

	CVRP				3D ball
$\lvert R\rvert$	Dice	Manhattan	Jaccard	Yule	euclidean
7	0.008	**0.004**	0.010	0.027	0.003
34	**0.019**	0.021	0.032	0.037	0.008
206	**0.035**	0.039	0.042	0.056	0.010
1486	0.049	0.048	0.073	**0.043**	0.011

In Figure 3 we present visualizations of randomly generated TSP and CVRP instances produced with our method. In addition, the rightmost plot visualizes a search trajectory of Clarke-Wright construction heuristic followed by few iterations of 2-opt. Our visualization tool allows interactive examination with freeform rotation around the visualizations[3].

[3] http://youtu.be/LW4DnTHpXoE
http://youtu.be/i6JsS6VjsEU
http://youtu.be/y53Qh2kC0Xo



(a) TSP, $n = 6$

(b) CVRP, $n = 6$

(c) CVRP LS, $|\cup N(s_i)| = 1218$

Fig. 3: Example visualizations produced using the described method

TABLE III: Perseverance of fitness landscape features

Measure	$\text{CVRP}_{n=4}$	$\text{CVRP}_{n=5}$	$\text{CVRP}_{n=6}$
λ	0.22	0.70	0.67
$\#LO$	0.73	0.24	0.01
$\bar{d}$ (σ_d)	0.78 (0.53)	0.26 (0.32)	0.58 (0.59)

We calculated solution space metrics for 20 random CVRPs and compared them with the ones for corresponding visualizations. Each RW and LS with maximum length of 500 steps was repeated 100 times. Finally the results were averaged for reliability. Table III shows correlations between these results. It becomes clear that the accuracy of the described method drops on transition from $n = 5$ to $n = 6$. Especially the placement of local optima in a topology persevering way becomes hard. Luckily, the problem is smaller with neighbourhood plots as there is more room for the MDS to work. In addition, the visualization of neighbourhoods is more convenient in algorithm development as it allows e.g. plotting of different LS strategies into one visualization for comparison or troubleshooting.

VI. Conclusions and Future Topics

We have presented a technique capable of visualizing vehicle routing problem solution landscapes. Because of this technique, we were able to visually observe the nature of the vehicle routing problem instance landscape for the first time. We also examined the accuracy of our method. We have shown that the properties and possibilities make the presented technique a viable tool for routing algorithm designers, especially when dealing with local search neighbourhoods. We also presented the first closed form function for the number of solutions for symmetric CVRPs.

Further research might be extended the comparison to recent routing specific distance measures (Løkketangen et al., 2012). Also, accelerated methods such as Split-and-Combine MDS or Landmark MDS might be needed to visualize larger solution spaces and search algorithm neighbourhoods. Another bottleneck was the handling of big distance matrices. Experimentation with approximation and compression techniques could lead to faster calculations and reduced memory footprint. Our aim is to create an online gallery of problem variants (CVRP, TSP, VRPTW, PDP etc.) and search trajectories of different popular heuristic operators and metaheuristics. Also the source code of the tool will be published with the gallery.

References

S. Halim and H. C. Lau, "Tuning tabu search strategies via visual diagnosis," in *Meta-Heuristics: Progress as Complex Systems Optimization*. Kluwer, 2007, pp. 365–388.

P. Toth and D. Vigo, *The vehicle routing problem*. SIAM, 2002.

E. Weinberger, "Correlated and uncorrelated fitness landscapes and how to tell the difference," *Biological Cybernetics*, vol. 63, no. 5, pp. 325–336, Sep. 1990.

C. Fonlupt, D. Robilliard, P. Preux, and E.-G. Talbi, "Fitness landscapes and performance of meta-heuristics," in *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Kluwer, 1999, pp. 257–268.

M. Kubiak, "Distance measures and fitness-distance analysis for the capacitated vehicle routing problem," in *Metaheuristics: Pregress in Complex Systems Optimization*. Springer, 2007, pp. 345–364.

Z. J. Czech, "Statistical measures of a fitness landscape for VRP," in *Proceedings of IPDPS*. IEEE, 2008, pp. 1–8.

E. Pitzer, M. Affenzeller, A. Beham, and S. Wagner, "Comprehensive and automatic fitness landscape analysis using heuristiclab," in *EUROCAST'11*. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 424–431.

F. Mascia and M. Brunato, "Techniques and tools for search landscape visualization and analysis," in *Proceedings of Stochastic Local Search 2009, Brussels, Belgium*, ser. LNCS, T. Stützle, M. Birattari, and H. Hoos, Eds., vol. 5752. Springer Berlin / Heidelberg, 2010, pp. 92–104.

M. Brunato and R. Battiti, "Grapheur: a software architecture for reactive and interactive optimization," in *LION'10*. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 232–246.

S. S. Choi, S. H. Cha, and C. Tappert, "A Survey of Binary Similarity and Distance Measures," *Journal on Systemics, Cybernetics and Informatics*, vol. 8, no. 1, pp. 43–48, 2010.

I. Borg and P. Groenen, *Modern Multidimensional Scaling: Theory and Applications*. Springer, 2005.

H. S. Wilf. (2012) A011800: number of labeled forests of n nodes each component of which is a path. [Online]. Available: http://oeis.org/A011800

A. Løkketangen, J. Oppen, J. Oyola, and D. L. Woodruff, "An Attribute Based Similarity Function for VRP Decision Support," *Decision Making in Manufacturing and Services*, vol. 6, no. 2, pp. 65–83, 2012.

PIII

FEATURE EXTRACTORS FOR DESCRIBING VEHICLE ROUTING PROBLEM INSTANCES

by

Jussi Rasku, Tommi Kärkkäinen, Nysret Musliu 2016

Proceedings of 5th Student Conference on Operational Research (SCOR 2016),
OpenAccess Series in Informatics (OASIcs)

Feature Extractors for Describing Vehicle Routing Problem Instances

Jussi Rasku[1], Tommi Kärkkäinen[1], and Nysret Musliu[3]

1 **Department of Mathematical Information Technology, University of Jyväskylä, P.O. Box 35, FI-40014, Finland**
`jussi.rasku@jyu.fi`
2 **Department of Mathematical Information Technology, University of Jyväskylä, P.O. Box 35, FI-40014, Finland**
3 **Institute of Information Systems, Vienna University of Technology, A-1040 Vienna, Austria**

Abstract

The vehicle routing problem comes in varied forms. In addition to usual variants with diverse constraints and specialized objectives, the problem instances themselves – even from a single shared source – can be distinctly different. Heuristic, metaheuristic, and hybrid algorithms that are typically used to solve these problems are sensitive to this variation and can exhibit erratic performance when applied on new, previously unseen instances. To mitigate this, and to improve their applicability, algorithm developers often choose to expose parameters that allow customization of the algorithm behavior. Unfortunately, finding a good set of values for these parameters can be a tedious task that requires extensive experimentation and experience. By deriving descriptors for the problem classes and instances, one would be able to apply learning and adaptive methods that, when taught, can effectively exploit the idiosyncrasies of a problem instance. Furthermore, these methods can generalize from previously learnt knowledge by inferring suitable values for these parameters. As a necessary intermediate step towards this goal, we propose a set of feature extractors for vehicle routing problems. The descriptors include dimensionality of the problem; statistical descriptors of distances, demands, etc.; clusterability of the vertex locations; and measures derived using fitness landscape analysis. We show the relevancy of these features by performing clustering on classical problem instances and instance-specific algorithm configuration of vehicle routing metaheuristics.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.1.6 Optimization, G.1.10 Applications, I.2.6 Learning, I.2.8 Problem Solving, Control Methods, and Search

Keywords and phrases Metaheuristics, Vehicle Routing Problem, Feature extraction, Unsupervised learning, Automatic Algorithm Configuration

Digital Object Identifier 10.4230/OASIcs.SCOR.2016.7

1 Introduction

The quality and the required computational effort of algorithmically optimized vehicle routing solutions are heavily dependent on the problem instance, the solution method, and using the right parameters for the algorithms [5]. Fortunately, it has been shown that automatic algorithm configuration and algorithm selection can be used to improve the solver performance. Thus, in order to make routing algorithms more robust and adaptive, we propose applying machine learning to help the algorithms more effectively adapt to the problem being solved. However, as a prerequisite, we need a way to describe the problem instances to the learning algorithms.

The vehicle routing problem (VRP) can be considered to be a generalization of the traveling salesman problem (TSP). Therefore, studies where TSP instances are described, e.g. [20, 8, 12, 6, 13], are highly relevant. Smith-Miles and van Hemert [20] proposed 12 features for predicting the most suitable optimization algorithm for a TSP instance. The feature set is well-rounded containing features derived from the distance matrix, clustering, nearest neighbors, and geometry of an instance. Kanda et al. [8] had a similar goal, but they relied only on features based on the problem size and statistical description of the distance matrix, whereas Mersmann et al. [12] proposed a set of 47 features in order to build a model that could be used to discriminate between hard and easy TSP instances. Hutter et al. [6] proposed a set of new approaches such as describing minimum spanning trees, ruggedness, and probing with TSP solvers. Probing involved analyzing and describing the solution attempts with a heuristic and branch-and-cut solvers. Pihera and Musliu [13] further extended this feature set, which allowed algorithm selection for a TSP instance.

Literature of VRP descriptors is scarce. The only studies on VRP feature extraction from the machine learning perspective we are aware of are the dissertation of Steinhaus [22] and algorithm performance prediction in [25]. Steinhaus [22] explores the use of a self organizing maps in solving VRPs and in algorithm selection. She proposes 23 features specifically for VRP problems and explores the discrimination power of this feature set across 102 VRP benchmark problems. Most features she proposed are based on earlier literature on describing TSPs, but they are complemented with features describing the demand distribution of the nodes, vehicle capacity, and their relations. Studies from a VRP *fitness landscape analysis* perspective, e.g. [21, 14, 25], do exist, but as these metrics are mainly used to gain deeper understanding of the problem, they need to be adapted before they can be used for performance prediction or algorithm selection. This was the approach chosen by Ventresca et al. [25].

In this article, a set of feature extractors gathered from the aforementioned sources is adapted to describe capacitated vehicle routing problem (CVRP) instances. Our goal is to recognize problem types and better understand instance properties that may affect solving them. A set of features that is this comprehensive has not been previously used to describe vehicle routing problems. Moreover, the feature set is validated experimentally with clustering of benchmark instances, automatic algorithm configuration [5], and instance specific algorithm tuning [7].

Our contributions are threefold: First, we give a review on feature extraction of vehicle routing problems. Second, proposed features are used in automatic configuration of three metaheuristic CVRP solvers to prove their usefulness for self-adaptive and learning solution techniques. Finally, we do clustering on 168 well known CVRP benchmark instances and make observations on their similarities. To the best of our knowledge, this is the first study that proposes the use of features acquired by probing CVRP instances with exact and heuristic solution methods. This is also the first study to explore the possibility of using features to improve the performance of automatic configuration of vehicle routing algorithms.

This paper is organized as follows: In Section 2 the automatic algorithm configuration problem is defined. Section 3 introduces the vehicle routing problem in detail, with handling of common solution approaches. It is followed by listings of feature extractors and descriptors for these problems, also including those presented in this study. Section 4 describes the experimental setup and the results for verifying the proposed feature set. Finally, we conclude our study in Section 5.

2 Instance Specific Algorithm Configuration Problem

The task of automatic algorithm configuration (AAC) involves the off-line task of finding a "good" set of parameter values, or *a parameter configuration*, for a *target algorithm* in a way that the algorithm achieves the best possible performance. It is critical to use a representative set of problem instances when configuring the algorithm parameters. This ensures that the performance advantage manifests also on new, previously unseen, instances.

If a good generalized performance is needed and the problem set is not homogeneous, i.e. the instances are very different from each other, the use of AAC may even be disadvantageous: a parameter configuration may enable an algorithm to perform well on some instances, but be inferior to algorithm defaults on another. One possible solution in a situation like this is to use instance specific algorithm configuration as described e.g. in Kadioglu et al. [7]. The idea is to configure the parameters for each group of mutually similar problem instances separately, and when a new problem instance needs to be solved, the automatically configured parameters of the most similar instance group is used. For a study on instance specific algorithm configuration of a TSP metaheuristic we refer to [18].

The task of algorithm selection is closely related to AAC. In algorithm selection, the problem instance properties are used to choose the algorithm with best predicted performance. Usually the algorithm is selected out of a portfolio, and the model for algorithm performance is built earlier during an off-line learning phase. The approach has proven successful: during the last decade many state-of-the-art results in combinatorial optimization competitions have been achieved using algorithm selection from an algorithm portfolio [27, 10].

Please note that both algorithm selection and instance specific algorithm configuration need a way to describe the problem instances. Therefore, a good set of feature extractors is a critical prerequisite for employing these learning meta-optimization techniques. It is necessary to experimentally discover which features can characterize a problem set in such a way they capture properties relevant to a) solving the problems b) configuring algorithm parameters c) recognizing a set of mutually similar problems that can share a configured parameter configuration, and d) ability to predict algorithm performance.

3 The Vehicle Routing Problem

The Vehicle Routing Problem (VRP) involves finding optimal routes for *vehicles* leaving from a *depot* to serve number of *clients*. Each client must be visited exactly once by exactly one vehicle. Each vehicle must leave from the depot and return there after serving the clients on its *tour*. There are numerous variants of VRPs, each with their own additional constraints [23]. In this study, only the classic Capacitated Vehicle Routing Problem (CVRP) is considered. In the CVRP, each of the identical vehicles has a maximum carrying capacity of Q that cannot be exceeded at any point of the tour. Each of the clients, indexed with i, have a demand q_i that has to be within $0 < q_i \leq Q$. The number of vehicles, denoted by k, is the primal minimization target, followed by the total travel distance of the k vehicles. Extending this notation, the CVRP can be written in a graph formulation adapted from Toth and Vigo [23] as follows: Let $V = \{0, \ldots, n\}$ be the set of vertices where the depot has the index 0 and where the rest correspond to the clients. The size of the problem is denoted by $N = |V|$. Let $E = \{(0,1), \ldots, (i,j), \ldots, (n-1,n)\}$ be the set of edges, where $i, j \in V, i \neq j$. Therefore, the graph $G = (V, E)$ is complete with each edge $e = (i,j) \in E$ having an associated non-negative weight c_{ij} that is the cost of traversal from vertex i to j. The weights can be also given as a distance matrix D. For each of the edges $(i,j) \in E$ there is a binary decision variable x_{ij} to decide whether the edge is traversed.

3.1 On Solving Vehicle Routing Problems

The solution approaches of VRPs can be divided into two main families: exact and heuristic methods. Heuristic methods are often augmented with metaheuristics to avoid entrapment in the first local optima the search encounters. Laporte [9] further divides heuristic methods into constructive and improvement heuristics. Constructive heuristics insert unassigned clients on the routes, and improvement heuristics improve the solution quality through small moves until no improving steps can be taken. Improvement heuristics can be seen as a building block of the Local Search (LS), a key element in modern metaheuristics.

The (meta)heuristic approach is the most feasible approach when solving larger CVRP problems. However, exact methods are still relevant as the (meta)heuristic methods make no guarantees in reaching the globally optimal value. According to Lysgaard et al. [11], the most promising exact solution technique for CVRP appears to be branch-and-cut (BnC). In BnC cutting planes are iteratively added to a relaxed linear programming model to ultimately narrow down on the global optimum.

3.2 Descriptors for the Problem

The features for routing problem instances are usually calculated either using the distance matrix or the 2D coordinates. Therefore, to calculate all features, both the node coordinates and the distance matrix need to be known. If D was not given in an instance file, a distance matrix was produced using the depot and client coordinates. Likewise, if a benchmark instance provided only a distance matrix, we used multidimensional scaling (MDS) [2] to generate x and y coordinates for the instance. We also followed the example of Smith-Miles and van Hemert [20] and scaled the coordinates into the $[(0,0),(400,400)]$ rectangle to make the geometrical features comparable between problem instances. However, we retain the shape (scale) of the problem when normalizing the problem to avoid distortion of the distance matrix, i.e. we maintained the x/y ratio. To maintain the connection between the coordinates and D, we scaled the distance matrix D using the same multiplier as with coordinates. This preprocessing produces a commensurable distance matrix D^n and coordinate set P^n that can be used to calculate geometrical and graph features.

Table 1 (p. 5) presents the CVRP feature extractors used in this study. The features proposed in this study are marked with bold typeface. The table also shows how many feature values each extractor produces. Usually the features are statistical descriptors explaining the distribution of measured values. If the number of statistical descriptors is five, it includes statistical moments (mean, standard deviation, skewness and kurtosis) and coefficient of variation; whereas if 11 descriptors are given, the former are complemented by minimum, maximum, median, number of modes, frequency of the mode value, and the mode itself (or average of modes). An even more complete set of 14 descriptors adds quartiles.

Table 1a. The first feature set is for describing the node distribution on a 2D plane. The most often used feature involves statistically describing the distance matrix (cost matrix, without the diagonal). Smith-Miles and van Hemert [20] used the standard deviation, which Kanda et al. [8] and Hutter et al. [6] complemented with a more comprehensive set of statistical descriptors. We normalized the distance matrix to the rectangle $[(0,0),(400,400)]$, similarly to [20], and calculate 11 statistical descriptors for the distance distribution. Smith-Miles and van Hemert [20] also proposed counting the distinct distances found in the distance matrix using different precision. We used four levels of precision, like in [6]. Also, the centroid of the coordinates and the euclidean distance from each point to the centroid were calculated. The average of these distances is the "radius" feature from [20].

Table 1 The feature extractors for CVRPs, grouped by type.

(a) Node distribution features

ID	Feature	#
ND1	Distribution of distance matrix calues [20, 8, 6]	11
ND2	Fraction of distinct distances (with 1,2,3,4 decimals) [20, 6]	4
ND3	Centroid of the nodes (x,y) [20]	2
ND4	Distance to the centroid [20]	5
ND5	# of clusters (abs.,rel.) [20]	2
ND6	# of core, edge and outlier cluster points (rel.) [20]	3
ND7	Reach of the clusters [20]	5
ND8	Normalized cluster sizes [6]	5
ND9	**Silhouette coefficient**	1
ND10	Minimum bottleneck cost [6]	5

(b) Minimum spanning tree (MST) features

ID	Feature	#
MST1	MST edge cost [12, 6]	5
MST2	MST node degree [12, 6]	5
MST3	MST depth **from the depot**	5

(c) Local search (LS) probing features

ID	Feature	#
LSP1	Solution quality after construction phase [6]	5
LSP2	Solution quality after LS [6]	5
LSP3	Improvement per LS step [6]	5
LSP4	LS steps to local minimum [6]	5
LSP5	Distance of local minima [6]	5
LSP6	% for edges in local optima [6]	5
LSP7	Solution edge lengths per quartile (**5** $\times$ **4** quartiles) [13]	**20**
LSP8	Segment length [13]	5
LSP9	Segment edge count [13]	5
LSP10	Segment edge length [13]	5
LSP11	Intra-tour intersections [13]	5
LSP12	**Autocorrelation length**	5

(d) Branch-and-cut probing features

ID	Feature	#
BCP1	Improvement per added cut [6]	5
BCP2	Ratio between upper and lower bound [6]	1
BCP3	Solution value after probing [6]	1
BCP4	Lower bound [6]	1

(e) Geometric features

ID	Feature	#
G1	Area of the enclosing rectangle ("squareness") [20, 6]	1
G2	Convex hull (CH) area [12]	1
G3	Ratio of points on the hull [12]	1
G4	Distance of enclosed points to the CH contour [13]	11
G5	Edge lengths of the CH [13]	11

(f) Nearest neighborhood (NN) features

ID	Feature	#
NN1	Distance to 1st NN [20, 6]	5
NN2,9,15	Node input degree in directed kNN graph (DkNNG) for $k \in \{3, 5, 7\}$ [13]	14
NN3,10,16	# of strongly connected components (SCCs) in DkNNG	11
NN5,11,17	Size of SCCs in DkNNG [13]	11
NN6,12,18	# of Weakly Connected Components (WCCs) in DkNNG	11
NN7,13,19	Size of WCCs in DkNNG [13]	11
NN8,14,20	Ratio of SCCs/WCCs [13]	1
NN21	Angle between edges to two NNs [12, 13]	11
NN22	Cosine similarity between edges to two NNs [13]	11

(g) VRP specific features

ID	Feature	#
DC1	Number of clients [8]	1
DC2	The depot location (x, y) [22]	2
DC3	**Distance between the centroid and the depot**	1
DC4	**Client dist. to the depot**	5
DC5	Client Demands [22]	5
DC6	Ratio of total demand to total capacity (the "tightness") [22]	1
DC7	Ratio of max. cluster demand to vehicle capacity [22]	1
DC8	Ratio of cluster outlier to overall demand [22]	1
DC9	Ratio between the largest demand and the capacity [22]	1
DC10	Average number of clients per vehicle [22]	1
DC11	Minimum number of trucks [22]	1

Some heuristics rely heavily on the existence of clusters. Therefore, features capturing this aspect are expected to be useful in algorithm selection. DBSCAN clustering has been used, at least, in [20, 8, 12, 6, 13, 22] to extract features for routing problem instances. We calculated features for cluster count (absolute and relative to N); cluster size; and a relative number of core, edge and outlier points. Mersmann et al. [12] used three different values for the ϵ (maximum allowed distance for two points belonging to a same cluster), while Steinhaus [22] experimented with four alternative methods to find a good ϵ value. In our study we decided to use the minimum cluster size of 4, and with that $\epsilon = A_{\text{bb}}/(\sqrt{N} - 1)$, which is an approximation of the 4th nearest neighbor distance if the nodes are assumed to be uniformly distributed on a lattice within a square with an area of A_{bb} [22]. To include a feature measuring the quality of the DBSCAN clustering, we propose the silhouette score [19] as a novel addition to the feature set.

The node distribution features are completed with the Minimum Bottleneck Cost (MBC) as proposed by [6]. It is used to describe the clusterability of TSP instances. The bottleneck cost is defined to be the weight of the longest edge on a path from node i to j, $i \neq j$. We get the minimum bottleneck cost by taking a minimum of bottleneck costs over all possible paths from node i to j. By calculating the bottleneck cost for all possible node pairs $i, j \in V$, $i \neq j$, we get a distribution that can then be described with statistical moments.

Table 1b. A minimum spanning tree (MST) was calculated for the fully connected normalized graph G^n. As suggested in [12, 6], the distribution of edge costs and node degrees of the MST were described using statistical moments. Mersmann et al. [12] included the spanning tree node depth as well, which we adapted for the VRP by calculating it with the depot as the root. We omitted the sum of the MST tree cost proposed in [12], as it can be inferred from the average MST cost.

Table 1c Probing features are computed with a solution attempt on a problem instance. An algorithm is run for predefined time or steps and the trajectory of the search is recorded. The approach is general and applicable to a variety of problems. Probing has been shown to be useful, e.g. for predicting the performance of an algorithm [6].

To adapt the TSP LS probing features from [13], we used the VRPH heuristic search algorithm library [3], or more specifically, its `vrp_init` application that is based on the Clarke-Wright construction heuristic. It was modified to accept a shape parameter γ that affects the savings calculation [28]. The parameter can be selected randomly to produce varied initial solutions. After construction, the solution is improved with intra-route multi-neighborhood search using best accept strategy with one-point-move, two-point-move and two-opt local search heuristics [3] until no improving move is found.

By repeating the probing 20 times, we could calculate the statistical descriptors in Table 1c. Some of the features closely resemble those we have used previously to validate visualization technique for VRPs with solution space analysis (SSA) [16]: the first is the distribution of Manhattan distances between the local optimum solutions (`LSP5`), calculated from the differences in edge traversal decision variable values between the solutions. This feature is a measure of the multimodality and indicator for the existence of a "big-valley" structure [14]. The second SSA feature `LSP6` describes the distribution of probabilities of all edges in locally optimal solutions, which aims to reveal the existence of a backbone [26], that is, a common structure between good solutions.

The features `LSP8-10` involve the concept of a segment. A segment is a continuous path of consecutive edges on a tour, from which the longest edges are removed as specified in [13]. Pihera and Musliu [13] also proposed another extension to the set of local search

features, which is the number of intra-tour intersections, i.e. the times the edges of a tour cross each other.

The final local search probing feature is the autocorrelation length λ_{ACL} of a random walk through a series of best-accept one-point-move neighborhoods (heuristic described e.g. in [3]). This closely relates to the autocorrelation coefficient used in [6]. For calculating the autocorrelation length we used a method adapted from [21] and [4], with a random walk length of $2N$. The walk is repeated and the length calculated 10 times.

Table 1d. Besides heuristics, also exact solvers can be used to probe the problem. We used the open source mixed-integer programming package SYMPHONY 5.6.15 and its VRP application that can solve CVRPs [15]. Unfortunately, we were not able to compile the VRP application with heuristics support for this version of SYMPHONY. Therefore, the upper bound is set only after the first feasible solution is found. Because of this, is possible that the feature `BCP2` is left undefined for the larger instances if no feasible solution is found. SYMPHONY also requires the number of vehicles k as an input when solving an instance. If k was not known we divided the total demand with some margin (+5%) with the vehicle capacity Q to get the value for k, i.e. $k = \lceil 1.05 \sum q_i/Q \rceil$. For branch-and-cut probing we used a wall time cutoff of 3.0 seconds.

Table 1e. Geometric features try to capture information of the overall shape of the problem. The area of an enclosing rectangle, when normalized with the area of the scaled problem, describes the "squareness" of the problem. Mersmann et al. [12] suggested two features concerning the convex hull: the hull area and the fraction of nodes that are on the hull contour. According to their experiments, convex hull features allow accurate separation of easy and hard TSP instances. Pihera and Musliu [13] added statistical descriptors for distances of inner nodes to the hull contour. It is assumed that the more evenly distributed the nodes are inside the convex hull, the more difficult it is to solve. Therefore, all these were included in our feature set.

Table 1f. Because heuristic solution methods operate by navigating through the search space using a local search neighborhood, the Nearest Neighbors (NN) of the nodes can offer important insight to the structure of the problem. In our study, the distribution of the 1st nearest neighbor distances over all nodes is statistically described as done in [20, 6, 13]. We also included the extended nearest neighbor features presented in [13], which involve building a directed graph by taking only $k \in 3, 5, 7$ shortest edges for each node from the complete normalized graph G^n. The node degree, number and size of strongly and weakly connected components, and their ratios are calculated and statistically described.

Table 1g. In describing the demands and capacity, we followed [22]. As an extension to the VRP specific features, we propose measuring the distance between the depot and the centroid of the client points. Also, describing the shape of the distribution of distances from clients to the depot is included in our feature set. Furthermore, the size of the problem (number of clients) is included here. Refer to [22] for details on these features.

In addition to the features presented in Table 1, we recorded the per instance feature computation time as proposed in [6]. These are reported as timing features `T1-T9` that match the feature groups (Tables 1a–1g), with the exception of the autocorrelation and bottleneck cost features, which are timed separately.

To summarize this section, we have adapted and proposed 76 feature extractors for CVRPs which generate 386 features in total. The feature extractors were implemented in Python version 2.7.10, with the aid from numerical libary Numpy (version 1.9.2), machine learning library Scikit-learn (version 0.16.1), and statistical library Scipy (0.15.1). VRPH and SYMPHONY were built with GNU g++ 5.3.0 compiler from the Mingw-w64 project.

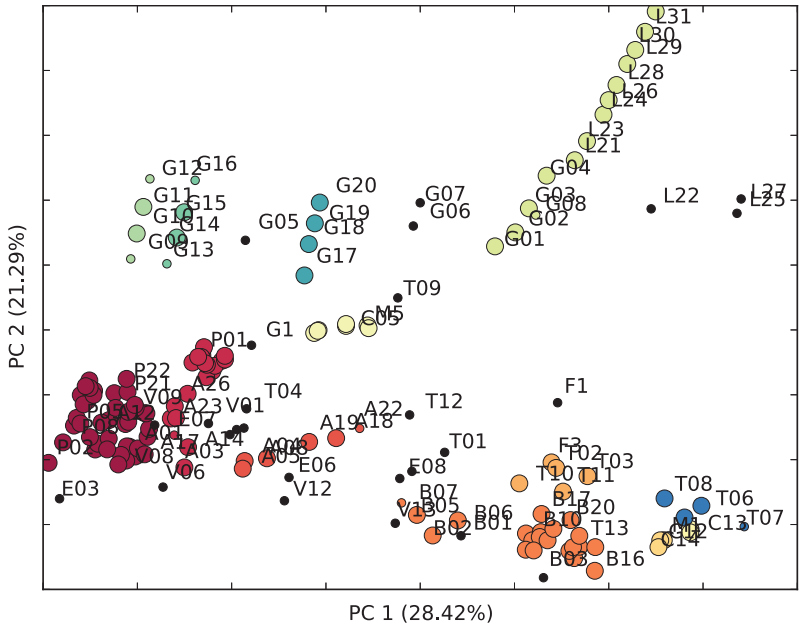


Figure 1 The clustering of the benchmark instances. Black dots are non-clustered outliers. The plot axes are the first two principal components, with the ratio of explained variance given in parenthesis.

All feature extraction in this study was done on a laptop with dual-core 2.53 GHz Intel Core i5 520M processor, 8 GB of memory and 64-bit Windows 7 Enterprise operating system.

4 Experimental Evaluation of the Features

4.1 Clustering

To evaluate the quality of the proposed feature set, we computed the 386 features for each of the 168 problem instances in CVRPLIB, which is a collection of CVRP benchmark instances [24]. However, the high dimensionality of the resulting data had to be addressed before clustering. Hutter et al. [5] suggests using principal component analysis (PCA) to reduce the computational complexity when building a surrogate model for the automatic algorithm configuration tool SMAC. We share some of the concerns regarding the computational cost. However, in our case a larger issue is the curse of dimensionality, where the space volume grows very rapidly as the dimensionality increases. This makes the dataset too sparse to provide a representative sample of the high dimensional space. A related problem is the irrelevancy of the distance metric in high dimensional data, where all data points seem to be similarly close to each other [1]. To overcome these issues, we reduced the dimensionality of the feature space with PCA.

To do the actual clustering, the feature data was first normalized by scaling all features independently to a range $[0.0, 1.0]$. Then, PCA was applied to bring the dimensionality of the data down from 386 to 7 following the example of [5]. These seven principal components together explain 71 % of the overall variance in the data. Finally, to do the unsupervised learning, we used the DBSCAN with a minimum cluster size of 3. The ϵ parameter was set to 0.20 through experimentation. Resulting clusters for the 168 benchmark instances are presented in Figure 1.

The clusters in the lower left corner seem to be the small-to-medium easy-to-solve instances. Unsurprisingly, the A and P sets overlap, as P is based on A. The difference between A and B is that clients in A are uniformly generated whereas in B they are clustered. Also the Taillard (T) and Fisher (F) sets contain clustered clients, which can be observed as an overlap with the set B. Interestingly, the benchmark set Golden (G) is separated into four clusters and some outliers. The benchmark set contains points in geometric shapes like stars, squares, circles and rays, and it seems that our features are able to discriminate between these. The Li (L) and Golden benchmark sets are similar and clustering them together is expected. For a more accurate analysis of clusters we would need to do a more extensive experimentation with the solvers, since probing does not necessarily allow reliable estimation of the hardness and computational difficulty of an instance. The complete list of problem instance abbreviations and the clustering in a table format, together with an interactive zooming visualization of the clustering, can be found from the supplementary online appendix at `http://users.jyu.fi/~juherask/features/`

4.2 Instance Specific Algorithm Configuration

As the automatic algorithm configuration targets, we used the three metaheuristic solvers provided by the VRPH package from Groër et al. [3]. Each solver employs different metaheuristic: Record-to-Record travel (VRPH-RTR, 6+8 free parameters), simulated annealing (VRPH-SA, 6+5), and ejection (VRPH-EJ, 6+3). We omit the descriptions of the algorithms and solver parameters and refer the reader to [3], whereas a detailed description of the automatic algorithm configuration setup can be found in [17].

As a configurator, we used SMAC [5]. SMAC is a state-of-the-art AAC method that alternates between fitting a random forest model to the observed behavior of the target algorithm, and using that model to predict the performance of generated parameter configuration candidates – evaluating only those that are most promising on the solver. SMAC offers an option to complement the problem instances with feature values, which are used when building and updating the random forest model. In our experiments this approach is called fSMAC. fSMAC already does PCA to the feature vectors, but as an additional preprocessing step we took 50 features that showed the highest correlation with the solution quality in heuristic and branch-and-bound probing. SMAC is not an instance specific algorithm configuration tool like ISAC from Kadioglu et al. [7], but we can follow a similar scheme to create IS-fSMAC. This variant uses k-Means clustering on the preprocessed feature data to split the problem instance set to subsets. These subsets supposedly share similar solving characteristics and can be configured separately.

In our configuration experiments we used a set of 14 instances taken with stratified sampling from the CVRPLIB set A. The problem set is the same one that we used in [17], which makes it possible for the interested reader to compare the proposed approach against other configurators. Also, each configuration task was run with three different evaluation budgets (EBs): 100, 500, and 1000 time capped (10 s) runs of the target algorithm. In the case of configuring the clustered instances, the budget was distributed according to the cluster size. Because the algorithms are stochastic, the experiments were repeated 10 times.

The results of the configuration tasks are presented in Table 2. The use of features seems beneficial, especially with a budget of 100. This is unsurprising, as the use of features is expected to provide more initial information when building the surrogate model of the parameter-solution quality response surface. Especially VRPH-SA target seems to benefit from using the features. The advantage gained by using features is smaller for VRPH-EJ and VRPH-RTR targets, but the effect still exists. However, the results of instance specific

Table 2 Median tuning results for VRPH metaheuristics. Results are given as percentages from the aggregated best known solution (relative optimality gap). The best known solution values are from CVRPLIB. Statistically better results are in bold ($p < 0.05$ with Bonferroni adjustment). If no single best was found, a test for a best pair was made.

Target	**VRPH-SA**			**VRPH-EJ**			**VRPH-RTR**		
Defaults	0.83 (0.12)			0.50 (0.13)			1.42 (0.07)		
Method \ EB	**100**	**500**	**1000**	**100**	**500**	**1000**	**100**	**500**	**1000**
SMAC	**0.40**	0.29	0.26	0.39	**0.35**	0.34	**0.16**	0.09	0.10
	(0.11)	(0.08)	(0.06)	(0.07)	**(0.04)**	(0.04)	**(0.05)**	(0.01)	(0.02)
fSMAC	**0.39**	0.26	**0.23**	**0.36**	0.36	0.35	**0.15**	0.09	**0.07**
	(0.15)	(0.06)	**(0.05)**	**(0.06)**	(0.05)	(0.06)	**(0.06)**	(0.04)	**(0.03)**
IS-fSMAC	0.56	0.27	0.25	**0.35**	**0.33**	0.34	0.19	0.09	0.09
	(0.11)	(0.07)	(0.05)	**(0.08)**	**(0.07)**	(0.06)	(0.06)	(0.03)	(0.01)

parameter tuning are not as good as expected. The clustering to problem classes seems to be beneficial only for VRPH-EJ targets. It may be that the features are unable to capture the differences (unlikely), the clustering is handicapped by the curse of dimensionality (likely), or the evaluation budget split among clusters is too small for SMAC to converge to good parameter configurations (likely). Still, the most probable cause is the homogeneity of the problem set. All of the instances in the set A come from the same generator, thus showing similar solving characteristics. Additional experiments are needed to identify the largest factor preventing the instance specific tuning from giving comparable advantage to what has been reported in e.g. in [7]. Nonetheless, every resulting parameter configuration is superior compared to the defaults.

All automatic configuration was done on a computing server with 64 Intel(R) Xeon(R) CPU E7 2.67 GHz cores, and 1 TB of RAM running 64-bit OpenSUSE version 13.2 (codename Harlequin). We enforced a 10 second cutoff for all evaluations of the CVRP solver.

5 Conclusions

In this article, we set out to find feature extractors for capacitated vehicle routing problem (CVRP) instances, mostly by adapting Traveling Salesman Problem (TSP) descriptors from the literature. We implemented 76 feature extractors for almost every descriptor that had been reportedly used in algorithm selection and automatic algorithm configuration of routing algorithms and proposed some novel ones. The presented set of 386 features for CVRP is unparalleled in its extent. Additionally, we are not aware that probing with heuristic and branch-and-cut solvers has been previously used to produce features for CVRP meta-optimization. The suitability of these features was verified with feature assisted automatic algorithm configuration with the state-of-the-art tool SMAC. We also presented clustering of 168 well-known benchmark instances from the CVRPLIB collection. Clustering shows good discrimination ability between the known properties of these problems. However, a more complete analysis of the clustering is warranted to get novel insights.

We can conclude that automatic algorithm configuration can benefit from using the proposed features. Out of the tested CVRP metaheuristics, the simulated annealing (VRPH-SA) benefited the most. We also experimented with instance specific configuration, where it was possible to further improve the configured solver performance of the ejection metaheuristic (VRPH-EJ). However, the overall increase in performance when using an instance specific

algorithm configuration scheme was modest. This is probably due to our problem set being relatively small and homogeneous. Therefore, a more extensive experimentation with different targets, instance specific configurators, and problem sets is required to make a judgment on applicability of instance specific parameter configuration of vehicle routing solvers. Also, please note that in our automatic algorithm configuration experiments we did not test for over-tuning (cf. overfitting), which may manifest as poor generalizability of the configured parameter configuration.

Other future research topics include: Feature selection that should help us recognize the most useful features, as currently the high dimensionality of the feature vector seems to confuse unsupervised learning and algorithm configuration efforts. We would also like to extend our feature extractors to describe other well-known VRP variants such as vehicle routing problem with time windows (VRPTW) and pickup and delivery problems (PDP). This could potentially reveal new interesting similarities between the problem types and sets. We would also like to extend our study towards algorithm selection.

It has been shown that applying feature based machine learning approaches, such as the one presented here, in solving combinatorial optimization problems, can lead to significant improvements in on-line algorithm performance and resulting solution quality. Adapting this approach in solving VRPs has shown promise and warrants further research.

References

1 Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. On the surprising behavior of distance metrics in high dimensional space. In Jan Bussche and Victor Vianu, editors, *Proceedings of Database Theory (ICDT 2001): 8th International Conference*, pages 420–434, Berlin, Heidelberg, 2001. Springer.

2 I Borg and P Groenen. Modern multidimensional scaling: theory and applications. *Journal of Educational Measurement*, 40(3):277–280, 2003.

3 Chris Groër, Bruce Golden, and Edward Wasil. A library of local search heuristics for the vehicle routing problem. *Mathematical Programming Computation*, 2(2):79–101, April 2010.

4 Wim Hordijk. A measure of landscapes. *Evolutionary computation*, 4(4):335–360, 1996.

5 Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization*, pages 507–523. Springer, 2011.

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

7 Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, and Kevin Tierney. ISAC – instance-specific algorithm configuration. In Helder Coelho, Rudi Studer, and Michael Wooldridge, editors, *Proceedings of 19th European Conference on Artificial Intelligence (ECAI 2010*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 751–756. IOS Press, 2010.

8 Jorge Kanda, Andre Carvalho, Eduardo Hruschka, and Carlos Soares. Selection of algorithms to solve traveling salesman problems using meta-learning. *International Journal of Hybrid Intelligent Systems*, 8(3):117–128, 2011.

9 Gilbert Laporte. What you should know about the vehicle routing problem. *Naval Research Logistics*, 54(8):811–819, 2007.

10 Marius Lindauer, Holger H. Hoos, Frank Hutter, and Torsten Schaub. AutoFolio: An automatically configured algorithm selector. *Journal of Artificial Intelligence Research*, 53:745–778, 2015.

11 Jens Lysgaard, Adam N Letchford, and Richard W Eglese. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, 100(2):423–445, 2004.

12 Olaf Mersmann, Bernd Bischl, Heike Trautmann, Markus Wagner, Jakob Bossek, and Frank Neumann. A novel feature-based approach to characterize algorithm performance for the traveling salesperson problem. *Annals of Mathematics and Artificial Intelligence*, 69(2):151–182, 2013.

13 Josef Pihera and Nysret Musliu. Application of machine learning to algorithm selection for TSP. In *Tools with Artificial Intelligence (ICTAI), IEEE 26th International Conference on*, pages 47–54. IEEE, 2014.

14 E. Pitzer, S. Vonolfen, A. Beham, M. Affenzeller, V. Bolshakov, and G. Merkuryeva. Structural analysis of vehicle routing problems using general fitness landscape analysis and problem specific measures. In *14th International Asia Pacific Conference on Computer Aided System Theory*, pages 36—38, 2012.

15 Ted K Ralphs. Parallel branch and cut for capacitated vehicle routing. *Parallel Computing*, 29(5):607–629, 2003.

16 Jussi Rasku, Tommi Kärkkäinen, and Pekka Hotokka. Solution space visualization as a tool for vehicle routing algorithm development. In Mikael Collan, Jari Hämälainen, and Pasi Luukka, editors, *Proceedings of the Finnish Operations Research Society 40th Anniversary Workshop (FORS40)*, volume 13, pages 9–12. LUT Scientific and Expertise Publications, 2013.

17 Jussi Rasku, Nysret Musliu, and Tommi Kärkkäinen. Automating the parameter selection in VRP: an off-line parameter tuning tool comparison. In William Fitzgibbon, A. Yuri Kuznetsov, Pekka Neittaanmäki, and Olivier Pironneau, editors, *Modeling, Simulation and Optimization for Science and Technology*, pages 191–209. Springer, 2014.

18 Jana Ries, Patrick Beullens, and David Salt. Instance-specific multi-objective parameter tuning based on fuzzy logic. *European Journal of Operational Research*, 218(2):305–315, 2012.

19 Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

20 Kate Smith-Miles and Jano van Hemert. Discovering the suitability of optimisation algorithms by learning from evolved instances. *Annals of Mathematics and Artificial Intelligence*, 61(2):87–104, 2011.

21 Peter F Stadler. Landscapes and their correlation functions. *Journal of Mathematical Chemistry*, 20(1):1–45, 1996.

22 Meghan Steinhaus. *The Application of the Self Organizing Map to the Vehicle Routing Problem.* PhD thesis, University of Rhode Island, 2015.

23 Paolo Toth and Daniele Vigo, editors. *The vehicle routing problem.* SIAM, 2002.

24 E. Uchoa, D. Pecin, A. Pessoa, M. Poggi, A. Subramanian, and T. Vidal. New benchmark instances for the capacitated vehicle routing problem. Technical report, UFF, Rio de Janeiro, Brazil, 2014.

25 Mario Ventresca, Beatrice Ombuki-Berman, and Andrew Runka. Predicting genetic algorithm performance on the vehicle routing problem using information theoretic landscape measures. In Martin Middendorf and Christian Blum, editors, *Proceedings of the Evolutionary Computation in Combinatorial Optimization: 13th European Conference (EvoCOP 2013)*, pages 214–225, Berlin, Heidelberg, 2013. Springer.

26 Toby Walsh and John Slaney. Backbones in optimization and approximation. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, 2001.

27 Lin Xu and Kevin Leyton-brown. SATzilla : Portfolio-based algorithm selection for SAT. *Artificial Intelligence*, 32:565–606, 2008.

28 P.C. Yellow. A computational modification to the savings method of vehicle scheduling. *Operational Research Quarterly*, 21:281–293, 1970.

PIV

META-SURVEY AND IMPLEMENTATIONS OF CLASSICAL CAPACITATED VEHICLE ROUTING HEURISTICS WITH REPRODUCED RESULTS

by

Jussi Rasku, Tommi Kärkkäinen, Nysret Musliu 2019

Manuscript

Meta-Survey and Implementations of Classical Capacitated Vehicle Routing Heuristics with Reproduced Results

MANUSCRIPT

Jussi Rasku, Tommi Kärkkäinen, and Nysret Musliu

September 24, 2019

Abstract

In this study, we present open source implementations for a comprehensive collection of 15 classic construction heuristics and seven local search operators targeting the capacitated vehicle routing problem. While the literature on the classical algorithms has been widely cited, existing vehicle routing software libraries only offer implementations for a few selected construction heuristics. Additionally, while some of the approaches are dated, and currently the state-of-the-art results are achieved using adaptive metaheuristics, classical construction heuristics still remain a viable approach when a solution should be produced deterministically or with limited computational resources. Construction heuristics are also a vital component in the more advanced solvers, where they are used to produce initial solutions that metaheuristics then seek to improve. The heuristics in this study were programmed in Python according to the functional programming paradigm. The source code was written to be accessible, easy-to-extend, well tested, and well documented. This is further supported by the detailed commentary on the implementation and design decisions. The provided algorithms can be used to solve traveling salesman and capacitated vehicle routing problems, with or without a maximum route duration constraint. We present an extensive experimental study that aims to replicate the results published in the original papers and provide comparative results on 454 capacitated vehicle routing problem instances from the literature. We were able to replicate most of the classical results and could confirm many of the earlier observations regarding the early heuristics. We were also able to recognize their individual strengths and weaknesses, and rank them by their simplicity, accuracy, consistency, and speed. Consequently, our work addresses the recent concerns on repeatability, reproducibility, and sharing the source code of vehicle routing heuristics.

Keywords: heuristics, vehicle routing problem, software, replication

1 Introduction

Between its inception in 1959 (Dantzig and Ramser, 1959) and early 1990s, many algorithms had been proposed to solve the vehicle routing problem (VRP). The algorithms of this era are usually referred to as the *classical heuristics*. Most of these methods were composed of a straightforward greedy heuristic with an improvement or mixed integer programming (MIP) step, and their purpose was well-defined: when given a problem instance definition including the location of the depot and customers, their demands, and vehicle capacity, the algorithm promptly produces a feasible 'good enough' solution. This solution is usually not optimal, but the algorithms were often simple to implement and extend (Laporte and Semet, 2002). Due to their practical value, the classical heuristics have remained relevant also for the later research on the topic (Braekers et al., 2016). Today, these algorithms are typically used to give an initial solution for the more advanced methods such as the many metaheuristics proposed since late 1980s (Section 4.4, p. 90- Toth and Vigo, 2014). Hence, the humble construction heuristic still matters, notably since it has been shown that the more sophisticated methods benefit from using a good initial solution (Thompson and Psaraftis, 1993).

The classical algorithms were usually proposed for solving the capacitated vehicle routing problem (CVRP), which can be seen as a generalization of the widely-known NP-hard traveling salesman problem (TSP). Instead of just one traveling entity, as in TSP, in CVRP there are multiple identical *vehicles* serving a number of *customers* from one central *depot*. Fixed or unlimited number of vehicles are deployed from the depot to carry out *routes* before returning to the depot. Moreover, the *demand* of each customer must be satisfied by a visit of exactly one vehicle, and the combined demand of the customers on a route must not exceed the vehicle *capacity*. The task is to minimize the transportation costs (usually length of the routes). Some classical problem instances also include a maximum route length/duration/cost constraint, and this variant is sometimes called the duration-constrained VRP (DVRP).

According to Sörensen (2015), it is important to be knowledgeable of the recent and previously proposed methods in order to avoid doing "research" where renaming existing concepts is regarded to be a contribution. Furthermore, without acknowledging the roots of a given field, the newly proposed methods will not be properly positioned in the literature and existing taxonomies of heuristics. Additionally, as Sörensen points out, "the ultimate goal of science is to understand". While it is clear that the classical algorithms, of which some are several decades old, do not always offer state-of-the-art performance, studying them can be useful. Recognizing the principles behind the most prominent classical heuristics allows one to gain intuition on which kind of algorithmic structures seem beneficial for certain situations or problems, and innovate and build upon this foundation (Hooker, 1995). Moreover, proper replication of computational experiments is a fundamental prerequisite for reliable algorithm comparisons (Črepinšek et al., 2014). The recent replication difficulties of a VRP algorithm reported by Sörensen et al. (2019) can be seen as an example of the importance of this topic. Unfortunately, it seems that classical algorithms are rarely reimplemented in the academic literature in their original form, and a concentrated effort on independent replication of their results seems to be missing from the literature. We are aware that some of the classical algorithms

have been reimplemented individually (e.g., in Golden et al., 1977; Cordeau et al., 2002; Laporte and Semet, 2002; Renaud et al., 1996), but the replication of the results has not been the main focus of these studies. It seems that the incentives to re-implement existing algorithms and replicate their results are sorely missing in operations research (Sörensen et al., 2019), which to some extent endangers the quality of the research building on top of the earlier studies.

The situation leads to our research questions: Which of the early CVRP algorithms actually should be considered to be "classical"? Are their originally published results replicable? And, how much, and which kind of variation there is in the quality of the solutions and in the runtime between these heuristics when they are used to solve well-known CVRP problem instances from the literature.

Regarding the last research question, it is difficult to find studies that reliably compare the performance differences between the classical algorithms. While publications usually include result tables with multiple algorithms on a handful of problem instances, the results may have been combined from multiple sources (e.g., Fisher, 1995; Cordeau et al., 2002; Laporte and Semet, 2002), which makes comparison unreliable due to differing test environments. There may be variation in the implementation and tuning effort used, in the numeric representation accuracy of the computational hardware, and in the software versions that were used to produce the results (Barr et al., 1995, p. 19). Furthermore, in the early VRP research the rounding and truncating conventions differed from author to author (Mole, 1979; Gendreau et al., 1994), which further complicates the comparison of algorithms (Fisher, 1995; Cordeau et al., 2002). Sometimes, two publications do not even share the primary optimization objective, and the exact parameter values and the number of runs needed to produce the reported solution are not disclosed. All this makes comparing, reproducing, or replicating the results difficult (Barr et al., 1995). The comparisons become even less meaningful for the computation times. The classical results have been run on a variety of computers over a large time span and implemented in different programming languages (Laporte, 2009). In addition, besides the implementation details and the computation environment, which themselves have a definite effect on the algorithm performance, usually only a limited number of problem instances are tested (Laporte and Semet, 2002). Finally, our work revealed several misprints and inconsistencies, which may have had effect on many earlier comparisons where the experiments are drawn from several sources.

Due to these issues, we argue that answering the research questions is impossible using only the results published in the literature. The conclusions would be drawn based on limited experimental data, or worse, selective experimental testing, which has the danger of leading to unremarkable and misleading results (Sörensen, 2015). Conducting the comparison correctly, therefore, requires building an experimental setup where the classical heuristic implementations from a single source are used to solve a comprehensive set of problem instances on a fixed computing environment. Only this would allow a solid comparison of the classical heuristics and answering the questions regarding the replicability of the results reported in the classical papers.

Unfortunately, while there are many software packages for solving the traveling salesman problem (TSP) (for a survey, see Lodi and Punnen, 2007), and some for VRP metaheuristics (e.g., Schrimpf et al., 2000; Groër et al., 2010; De Smet et al., 2016), we are not aware of a CVRP library that offers implementations for a com-

prehensive selection of classical algorithms. It seems that some of the classical VRP heuristics are rarely implemented because they are often considered to be too complicated to understand or implement (Cordeau et al., 2002). This is disconcerting, as freely available implementations would surely benefit the research field.

The most complete reimplementation effort of classical VRP heuristics is found from the PhD thesis of Van Breedam (1994). Unfortunately, his work was carried out before the open source movement gained momentum (Newman, 1999) and the C++ implementations of the algorithms are not publicly available. Also, his work (Van Breedam, 1994, 2002) concentrated on analyzing the effects that heuristic parameters and side constraints can have on the resulting quality of the solutions, and did not present replications of the earlier results, nor analyze the individual variation between the algorithms. Thus, it seems that the field is sorely missing accurate and freely available implementations of the most central classical heuristics. In this study, we did set out to fill this void by creating a software library of classical vehicle routing heuristics called VeRyPy.

But first, to recognize the most prominent algorithms, we did a meta-survey on the topic of classical CVRP heuristics. In order to narrow down the scope of this study, we limited ourselves to deterministic heuristics with well-defined termination rules that have no, or only few, free parameters. This is because algorithms with many parameters are hard to understand (Laporte and Semet, 2002) and replicating their results is notoriously difficult. Furthermore, stochastic methods with arbitrary stop conditions require extensive computation and experimentation to gain statistically valid estimates on their expected performance. Note that these criteria also naturally exclude methods which are better classified as metaheuristics. Additionally, for an algorithm to be included in our library, we wanted it to be relevant for solving large scale problems, i.e., with at least 1000 customer points. We also required that all the necessary details to re-implement the algorithm were given in the original paper (see guidelies given in Barr et al., 1995). This requirement included computational experiments on a standard set of CVRP benchmark instances to make it possible to verify the correct operation of our independent implementations.

The most laborious task in our study consisted of writing the (re)implementations of the established classical CVRP heuristics. However, as we have argued, this implementation work was necessary to allow answering the research questions of the replication accuracy and algorithm performance. To differentiate from the existing open source VRP software libraries, and to make our CVRP library as accessible as possible, we decided to implement it in Python. This decision is further supported by Python's recent popularity in many scientific fields (Millman and Aivazis, 2011). Its expressive and compact syntax, built-in interactive environment, comprehensive standard library with a rich collection of data structures, and a selection of useful scientific extension modules creates a productive *computational ecosystem* for high-level scientific computing (Perez et al., 2011). At the same time, we are aware that Python is not the best option for scientific applications involving intensive computations (Cai et al., 2005). However, this was not a major concern in this study, because the speed or the state-of-the-art performance were not our top priority. Instead, we followed the recommendation of Hooker (1995) and concentrated on the scientific testing of the algorithms instead of competitive bout of coding skill. Using a high level language for this task was, therefore, well-justified. Also, the algorithms

should still be reasonably fast because special care was taken to use suitable data structures and low level algorithms.

Besides programming language, there are many implementation decisions and trade-offs to be made when implementing an algorithm library, and we needed to set guidelines how to balance between them. Cordeau et al. (2002) argued that a good VRP heuristic should be estimated using four criteria: *accuracy*, *speed*, *simplicity*, and *flexibility*. Our main concerns were to make the implementations as simple as possible, followed by accuracy in the sense that they should replicate the results from the literature. As mentioned above, absolute speed was not of paramount importance in our study. Regarding the last criteria, we argue that simplicity also helps to achieve flexibility, because it makes the code easy to reuse and extend (e.g., to solve other VRP variants).

Related to accuracy and speed of the methods, the extensive computational experiment presented in this paper compared the 15 implemented classical heuristics on 454 capacitated vehicle routing problem instances from the literature. This allowed us to recognize the different strengths and weaknesses of the algorithms and to independently verify many earlier observations from the literature. Based on the results, three of the heuristics stood out: the Petal heuristic from Foster and Ryan (1976) is very good at solving problems with a maximum route duration constraint; the generalized assignment heuristic from Fisher and Jaikumar (1981) can usually find good solutions but is computationally expensive, especially on the larger problems with a maximum route duration constraint; and the extension from Paessens (1988) to the well-known savings heuristic of Clarke and Wright (1964) turned out to be accurate, robust, and fast, while still being reasonably simple to implement and extend. Despite these three heuristics standing out, there is not a single heuristic that always dominates the others: All but one of the tested heuristics can be stated to be the recommended algorithm for at least one of the 454 problem instances. These results, together with the detailed descriptions of the heuristics in this study, should help algorithm developers to a) more easily recognize and borrow promising ideas from the extensive literature on the topic and b) offer them insights for novel combinations of heuristic components when proposing hybrid solvers.

Our work also plays a part in addressing the concerns recently discussed by Sörensen et al. (2019) regarding repeatability, replicability, and improving peer review standards of the VRP research. Freely available implementations that have emphasis on simplicity and flexibility could help peer-reviewers to independently verify new results proposed for publication. Our contribution, a publicly available, clearly documented, and well written code library of classical vehicle routing problems, allows this. Also, with this ideal in mind, we strove to replicate the classical computational results. Prior to our work, a comprehensive study concentrating on reproduction and replication of classical VRP algorithm results was completely missing from the VRP literature.

The remainder of this report is organized as follows: Local search, which is a central building block in many heuristics, is discussed first in Section 2, together with listing the operators implemented by our library. This forms a necessary background for the literature meta-survey on classical vehicle routing heuristics (Section 3.1) and for the survey on open source software VRP libraries (Section 3.2). The surveys are, in turn, used to determine which classical heuristics to implement in Section 4. This

section also contains the detailed descriptions of the heuristics, their implementations, and replication of their results. The next section, Section 5, is intended to the users of our classical CVRP heuristics software library, and it gives examples on how to use the heuristics to solve problem instances and some performance considerations for operations research practitioners. Section 6 presents summary of the replication results, a description of the experimental study, its results, and analysis thereof. The report is concluded in Section 7, where the implications and future directions for our research are discussed.

2 Local Search

Local search is a key element not only in many classical but in almost all modern heuristics and metaheuristics (Funke et al., 2005). Familiarity to the concept is crucial to understanding the working principle of the algorithms described later in this report. Local search involves making *moves* (small changes) to a route or a solution. The search progresses by repeatedly exploring candidate moves and then applying those that lead to other, usually better, solutions.

Following the notation of Laporte (2007), we can define a concept of *neighborhood*, which is central to the operating principle of local search. Assume we are trying to improve a solution $s \in S$, where S is the set of all solutions. Local search operators see a neighborhood $N(s) \subset S$, which is a set of solutions reachable from the current solution by applying a operator specific transformation. When such a move is made, a new solution $s' \in N(s)$ is selected from the neighborhood. One such neighborhood is depicted in Figure 1.

Among the popular search strategies are the *first accept strategy* which involves taking the first improving move that is encountered, or the *best accept strategy* where the entire neighborhood for solution s^* is searched for the move that would give the best improvement to the objective function value:

$$s' = \underset{s \in N(s^*)}{\arg\min}\, c(s).$$

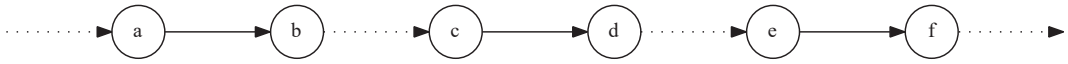
Here, $c : S \to \mathbb{R}$ is the objective value of a solution and we assume that we are minimizing the function. Thus, local search has reached a local optima if there is no better solution in the neighborhood, that is, $c(s') \leq c(s)\, \forall s \in N(s')$.

To facilitate the implementation of classical heuristics, we selected a set of well known local search operators (see Figures 2 and 3). We did not aim for feature parity nor competitive performance with the existing libraries such as VRPH from Groër et al. (2010). Instead, we wanted to implement a representative selection of operators that would avoid code duplication in implementing the classical heuristics. Note that the naming of local search operators is not consistent in the literature. This report roughly follows the naming used by Funke et al. (2005), Bräysy and Gendreau (2005), and Groër et al. (2010). The implemented operators are:

`do_2opt_move`, which takes one route and reconnects the ends of two edges that cross. In practice, this means reversing a sequence of nodes within one route. In a symmetric euclidean case this always improves the solution. The 2-opt neighborhood is a subset of 3-opt neighborhood (see Figure 1).

Figure 1: The 3-opt neighborhood on edges a-b, c-d, e-f. Dotted edges represent sequences of customers and may contain arbitrary number of visits.

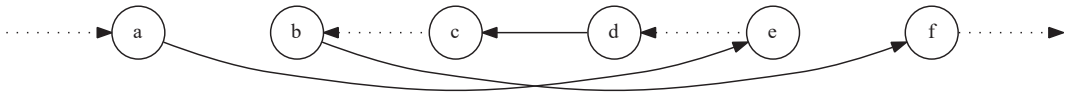
(a) Initial state



(b) 2-opt move: c-e, d-f

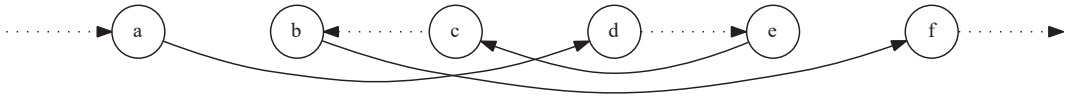
(c) 2-opt move: a-c, b-d

(d) 2-opt move: a-e, b-f

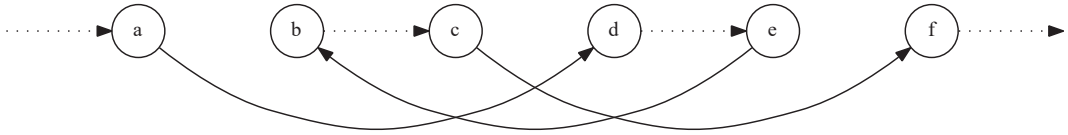


(e) 3-opt move: a-e, d-b, c-f

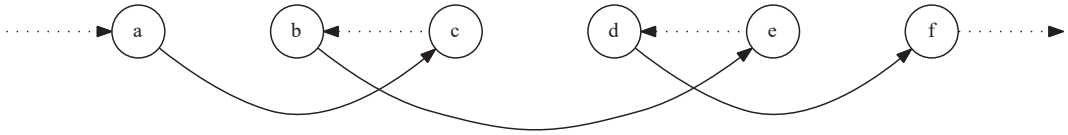
(f) 3-opt move: a-d, e-c, b-f



(g) 3-opt move: a-d, e-b, c-f



(h) 3-opt move: a-c, b-e, d-f



`do_3opt_move`, which removes three edges from one route and reconnects the node sequences in a way that improves the solution if such configuration exists.

`do_relocate_move` checks if a customer can be moved to a different position within a route in a way that the route is improved. The move is a special case of `do_3opt_move` operator, and, thus, the one-point move neighborhood is a subset of 3-opt neighborhood.

`do_exchange_move` checks if a route can be improved by swapping a customer on the route with another on the same route.

`do_1point_move` checks if an improving move can be found when a customer on a route can be moved to any position on another route. Thus, this can be thought as an inter-route version of `do_relocate_move`.

`do_2point_move` checks if a customer on a route can be swapped to a position on another route in a way that the solution would be improved. Thus, this is an inter-route version of `do_exchange_move`.

`do_insert_move` inserts previously non-routed customers to a route and to the position with least impairment to the objective function value.

`do_2optstar_move` is similar to `do_2opt_move`, but the edges can be from two separate routes. Hence, more combinations are available for reconnecting the tour segments.

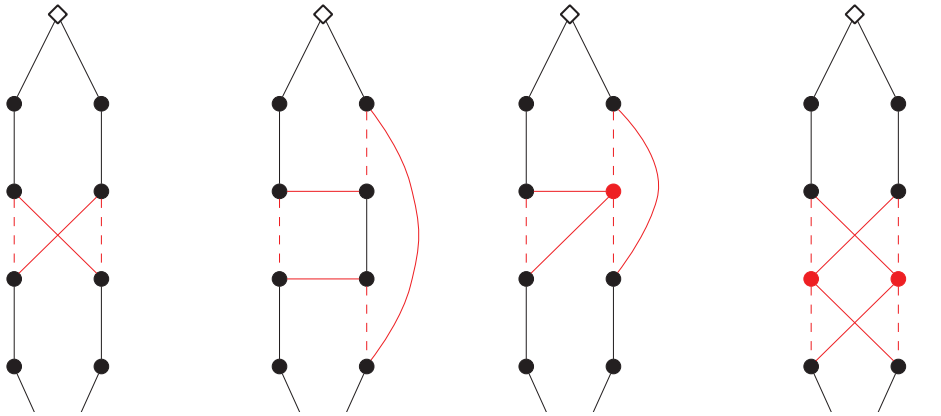
`do_3optstar_move` is similar to `do_3opt_move`, but the edges can be from any route and, thus, more combinations are available and checks are needed to avoid generation of subtours.

`do_chain_move` is a computationally expensive move which involves three routes. The complete canonical name for this type of operator seems to be node-ejection chains (Rego, 1998). It tries to find an improving move where a customer on the second route is replaced with a customer from the first route. The replaced customer must then be inserted on a third route. The move was introduced by Wren and Holliday (1972) and later explored in depth by Rego (1998).

The names *one-point-move* and *two-point-move* are used to differentiate the inter-route moves, from the intra-route (TSP) versions *relocate* and *exchange*. Here, the inter-route operators move customers or swap edges between multiple routes, whereas intra-route operate on a single route at the time. As one can see, we also implemented inter-route variants of *2-opt* and *3-opt*, often coined as *2-opt** and *3-opt**. However, operators that consider several nodes at the time such as *Or-opt*, or *cyclic transfers*, are not included in the local search part of VeRyPy library because they were not used by any of the implemented classical heuristics.

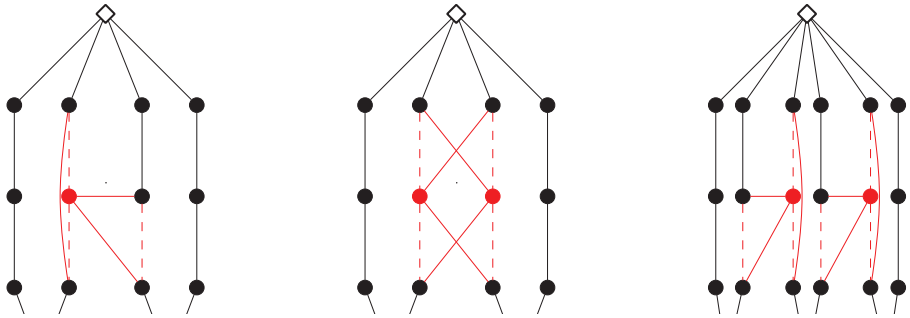
Regarding implementation of local search operators, we would like to point out that there are several design choices that affect how the local search is carried out, and what is the resulting solution:

Figure 2: The four intra-route local search operators in VeRyPy



(a) 2-opt (b) 3-opt (c) Relocate (d) Exchange

Figure 3: Three of the inter-route local search operators in VeRyPy



(a) 1-pt.-move (b) 2-pt.-move (c) Chain

1. Which local search operators are used?
2. In which order the operators are applied?
    a) Run one operator until there are no improvements before moving on to the next operator.
    b) Run all operators, one after another, and repeat as long as at least one of the operators finds an improving move.
3. The strategy choice, e.g., between first-accept or best-accept.
4. In what order the routes and the customers on the route are considered? Depending on the order, a different best move can be chosen if there are several equally good ones.

The local optima that is eventually reached through the local search depends on how the search was conducted, of the operator order, or any other detail outlined above. These decisions may greatly influence the resulting quality of the solution. Unfortunately, the aforementioned details are not usually described in full detail, even when the local search is a critical part of the operation of a heuristic. This makes replicating the results of the methods with local search problematic.

In general, the weakness of local search heuristics is their tendency to getting stuck on local optimum, as they usually quickly converge and stop there. Therefore, numerous different metaheuristics that complement the local search with exploration capabilities have been proposed. Metaheuristics are outside of the scope of this study, but the interested reader is referred to Chapter 4 of Toth and Vigo (2014) and the survey part of the work of Vidal et al. (2013).

3 Literature Survey

The literature survey is divided into two parts. In the first part, we present a meta-survey of the vehicle routing problem literature discussing algorithms and heuristics to recognize the most prominent classical algorithms. The survey also includes a high level descriptions for these algorithms. In the second part, we give the reader an overview on the currently available open source vehicle routing software libraries.

3.1 Classical Vehicle Routing Heuristics

The concept of *classical vehicle routing heuristics* seems to be formed around the time tabu search algorithms for the vehicle routing problem were first proposed (Gendreau et al., 1994; Laporte, 1992). While the concept is used widely even in the modern literature (see e.g. Braekers et al., 2016), it is not always clear which algorithms should be included in the list of classical VRP algorithms. To remedy this, we present a meta-survey on the topic and briefly describe the most central early vehicle routing heuristics.

One of the distinguishing features of these methods is that they are capable of starting from an empty solution and building a good (although, rarely optimal) feasible solution. Therefore, a name *construction or constructive heuristics* is sometimes used when referring to these methods. Cordeau et al. (2002) refined their definition with an observation that the classical heuristics are able to obtain a feasible solution quickly, and that the solution can then be improved with a *postoptimization* procedure. However, Cordeau et al. (2002) also noted that only few heuristics are sufficiently well-known to be truly 'classical'.

Laporte and Semet (2002) further narrowed down the concept of classical heuristics to concern the VRP methods that had been developed mostly between 1960 and 1990. This is a natural distinction, because after the late 1980s the focus in VRP research moved on to more complex search algorithms (Eksioglu et al., 2009). Many of the proposed methods of this later era relied on local search operators which were orchestrated by a supervisor algorithm or *a metaheuristic*. These orchestration techniques were varied, but metaheuristics can use stochasticity, perturbations, restarts, self-adaptivity, or maintain diversity, e.g., via a population of solutions (Gendreau and Potvin, 2010). The purpose of these techniques is to make the effective exploration of the solution space possible.

It was the advances in computing that made these methods possible and, consequentially, the metaheuristics often require significantly larger computational resources than the early heuristics (Eksioglu et al., 2009). Furthermore, due to their built-in stochasticity methods, the metaheuristic algorithms are not usually deter-

ministic and the ultimate accuracy of the method is dependent on how the termination rule is selected. Thus, the experimental results have been since reported, e.g., as the best of 10 runs. Most metaheuristics also expose a number of parameters that, while making them more versatile, require expertize and experimentation to set them correctly. The main issue with these features is that the quality of the solutions can vary greatly depending on the random seed and the values of these parameters. Hence, a tuned set of parameter values are needed to get the algorithm to consistently perform well on a specific set of problem instances. This makes testing and comparing metaheuristics challenging (Laporte et al., 2000). Still, VRP metaheuristics are currently the recommended method to solve multi-attribute and large VRP instances (Vidal et al., 2013). Compared to them, the results provided by the first generation of VRP heuristics were poor, with a typical gap of 10-15 % to the best known solution (Renaud et al., 1996).

However, the early classical heuristics remain relevant to this day. In their recent and extensive review of the VRP research from 2009 to 2015, Braekers et al. (2016) showed that around 10 % of the published research still relied entirely on the classical heuristics. This is probably because the performance of the simple classical heuristics is often "good enough" for practical, real world vehicle routing. Only if the operational fleet is capable of executing tightly planned routes, it is worth to pay the price in extra complexity brought in by a more sophisticated method. And, even then, the classical heuristics are usually used to provide a feasible initial solution or upper bound for the more sophisticated methods (Laporte et al., 2014).

To get a balanced view on the topic, we looked into surveys from different eras and from multiple different authors. We also considered the other meta-surveys on the topic: In 2009 Eksioglu et al. did a taxonomic review of the VRP literature that had, by then, become quite disjointed. In their review, they listed three works in the category of 'Survey, review or meta-research'. Out of these only the survey of Laporte et al. (2000) is relevant to CVRP, because the other two surveys concentrated on different extensions and variants. Also, the earlier version of the aforementioned study from 1992 was mentioned when Eksioglu et al. (2009) discussed solution methods of the classical algorithms era. Furthermore, based on the number of citations they have received, it seems that the earlier surveys from Eilon et al. (1971), Turner et al. (1974), Golden (1978), Mole (1979), Christofides et al. (1979), and Watson-Gandy and Foulds (1981) have a prominent presence in the literature. There is an earlier survey from Pierce (1969) with a similar topic, but that mainly surveyed the formulations and exact solution methods. Our set of included surveys is completed by a number of more recent ones from Cordeau et al. (2002), Cordeau et al. (2007), Laporte (2009), Vidal et al. (2013), and Laporte et al. (2014).

We intended to use our meta-survey to recognize the most central heuristic algorithms and assumed that they are the ones most often mentioned when classical constructive heuristics are discussed and surveyed. Also, while surveying the literature, we kept in mind our additional criteria: to be implemented in this study the algorithm needed to be deterministic, (relatively) parameter free, capable of solving problem instances at least up to 1000 customer points, and the original publication was required to contain enough details and an experimental study to allow replication of the original results.

Eilon et al. (1971)

Moving in a chronological order, we start from one of the earliest surveys on the topic of vehicle routing algorithms. A chapter on vehicle scheduling in the book Distribution management (Eilon et al., 1971) had a review on the prevailing solution methods.

The approach from Clarke and Wright (1964) had a dedicated section in the survey of Eilon et al. (1971). This well-known savings algorithm starts from a solution where each customer is served independently. Then, from the largest savings value (best improvement) first, the algorithm merges the routes until there are no feasible merges left. The alternative savings calculation methods from Gaskell (1967), and variants that explore a larger set of possible merges from Knowles (1967) and Tillman and Cochran (1968), were also briefly described by the survey.

After the savings algorithm and its variants, the method from Hayes (1967) was discussed. This algorithm mimics the way a human dispatcher would built the routes. However, according to the description of Eilon et al. (1971), the algorithm contains a random element and many free parameters, e.g., in the form of weights for different characteristics that are used to determine which nodes should be connected next. This makes the Hayes (1967) algorithm an unsuitable implementation candidate for this study.

The survey also included a description of the "r-optimal tour method" from Christofides and Eilon (1969), which works by improving a feasible random tour serving all customers; a route is first improved with intra-route and inter-route 2-opt procedure and then made 3-optimal. However, the efficiency of the algorithm relies on the stochasticity in the initial solution generation, and, hence, it does not satisfy our criteria either.

Turner et al. (1974)

Turner et al. (1974) surveyed the literature, formulations, exact methods, and heuristics for the 'transportation routing problem'. Out of the listed heuristics for VRP, the algorithm by Dantzig and Ramser (1959) gives an impression that it relies on steps that are intended to be carried out manually. Furthermore, their algorithm is generally considered to been superseded by the algorithm from Clarke and Wright (1964) and its extensions (Gaskell, 1967; Knowles, 1967; Tillman and Cochran, 1968). All these different savings variants were mentioned in the survey.

The survey mentioned also the man-machine algorithms from Hayes (1967) and Krolak et al. (1972). The approach from Krolak et al. (1972) has a manual phase, which is applied after an initial solution has been built by clustering the customers and using local search to force the routes feasible and to improve them. Manual interaction seems to be required to set up the many parameters of the algorithm. This together with the built-in stochasticity excludes the surveyed algorithms from Krolak et al. (1972), Hayes (1967), and Christofides and Eilon (1969) from our study.

Another algorithm covered by this survey is the one from Gillett and Miller (1974). Their sweep algorithm assumes that the customers are located on a plane. The algorithm assigns them to routes by their polar coordinates and constraints with a sweep centered at the depot. The assignment procedure is complemented by an improvement phase and a TSP optimization of the resulting routes.

Golden (1978)

Golden (1978) presented a survey on 'recent computational experiments with various algorithms' of the time. Some of the algorithms they listed are outside of the scope of this study. For example, while the tree based search enumeration scheme from Pierce (1969) is guaranteed eventually find the optimal solution, the search space and runtime is expected to grow prohibitively large even for medium sized instances. Another such algorithm mentioned in the survey was written by Russell (1977). He proposed a heuristic called MTOUR that works by improving randomly perturbed (and sometimes manually crafted) initial solutions via edge exchange operations. It is not entirely clear from the MTOUR description if he considered only intra-route or also inter-route improvements. However, the fact that the algorithm involves random perturbations and interactive steps excludes it from our study.

Of the methods that fill our criteria the ones from Clarke and Wright (1964), Gaskell (1967), Knowles (1967), and Tillman and Cochran (1968) were discussed. Also, other extensions to the savings approach were listed: method proposed by Yellow (1970) is an acceleration technique for parametrized savings function that uses polar coordinates to calculate only those savings that are probable to lead to a valid merge. However, due to the speed of modern CPUs and abundance of memory, the technique is no longer relevant. Holmes and Parker (1976) proposed yet another 'look ahead' savings modification to the Clarke and Wright (1964) algorithm. Earlier similar schemes had been proposed by Knowles (1967) and Tillman and Cochran (1968). Another popular extension to the Clarke and Wright (1964) algorithm has been to apply r-opt route improvement step. For example, Robbins and Turner proposed an intra-route 2-opt phase for the savings algorithm. Note that Golden (1978) referred to the preliminary work from the authors, but the final paper seems to have been published later (Robbins and Turner, 1979).

Moving on from the savings algorithms, Golden (1978) continued to describe Gillett and Miller (1974), Tyagi (1968), and Newton and Thomas (1969) algorithms. While the research of Newton and Thomas (1969) concerned routing school buses, the proposed algorithm can also be used in solving CVRPs. The algorithm solves a VRP instance first as a TSP by omitting the depot and only routing the customer points. Then, the TSP tour is split into separate routes according to the constraints. Also the algorithm by Tyagi (1968) uses TSP algorithms; It works by assigning the customers to routes using a sequential nearest neighbor route construction scheme, which is then followed by two-point move and TSP optimization phases.

Please note that in a related study (Golden et al., 1977), Golden noted that the algorithms from Clarke and Wright (1964), Tyagi (1968), and Gillett and Miller (1974) had already been used to solve problems up to 1000 customers. This happens to be one of the criteria for the algorithms to be included in this study, and the observation supports inclusion of these algorithms as relevant to our study.

Mole (1979)

Many algorithms surveyed by Golden (1978) were also mentioned by Mole (1979) in his survey. Algorithm from Clarke and Wright (1964), with its extensions (Gaskell, 1967; Knowles, 1967; Yellow, 1970; Holmes and Parker, 1976), were listed. However, the idea of improving the savings solution with a 2-opt and 3-opt route improvement

steps was attributed to McDonald (1972) and Beltrami and Bodin (1974) instead of Robbins and Turner (1979) like it was done in Golden (1978). Thus, it seems that this idea had been independently proposed by several different authors.

The surveyed algorithms that targeted TSP, the multi-depot case (MDVRP), or problems with delivery time windows (VRPTW) were ignored. Furthermore, the CVRP algorithms that are not applicable to our study were: Christofides and Eilon (1969) due to random initial solutions, Russell (1977) due to random perturbations, Krolak et al. (1972) due to its manual phase, and Dantzig and Ramser (1959) that was superseded by the savings algorithm. Furthermore, generating a full set of savings values is very fast on a modern computer, which makes the discussion on saving the computational effort somewhat obsolete (including, e.g., Yellow, 1970).

The survey discussed the insertion heuristic from Mole and Jameson (1976) in depth. It is useful to compare it to the Clarke and Wright (1964) savings where the savings values are calculated in the beginning and the merges can only be applied between the route ends. However, insertion algorithms use a more general savings criterion which can merge, or *insert*, single customers or partial tours to arbitrary position on the emerging routes. This involves updating the savings list whenever these insertions are made. In addition, the insertion heuristic of Mole and Jameson (1976) keeps the routes 2-optimal thorough the optimization and includes a "refine" procedure. This procedure tries to move customers from one route to another if the operation improves and maintains the feasibility of the solution.

The other two studies that were mentioned in the survey were from Wren and Holliday (1972); Gillett and Miller (1974), and Foster and Ryan (1976). The method from Wren and Holliday (1972) is very similar to the one of Gillett and Miller (1974) in that the customers are assigned to vehicles by 'sweeping' the plane by their polar coordinates. The sweeps are done in clockwise direction from four initial directions. The main difference, however, is in how the improvement procedure is activated. The Gillett and Miller (1974) heuristic alternates between the improvement procedure and the clustering, but in (Wren and Holliday, 1972) the improvement procedure is based on local search operators and is carried out after the initial assignment of the customers on the routes is done. Similarly to the algorithms from Gillett and Miller (1974) and Wren and Holliday (1972), the algorithm of Foster and Ryan (1976) uses a sweep procedure to generate a large number feasible candidate routes or 'petals'. This is followed by a phase that alternates between solving a set covering problem, where these petals are used to find a *cover*, and a heuristic inter-route improvement phase that generates new petals for the next iteration.

Christofides et al. (1979)

A chapter in the book Combinatorial Optimization (Christofides et al., 1979) was dedicated to the vehicle routing problem. The heuristic algorithms from Clarke and Wright (1964), Mole and Jameson (1976), and Gillett and Miller (1974) were described in detail. The sequential savings approach and its steps were also discussed, but it was not attributed to Webb (1964), where it seems to originate. The studies of Dantzig and Ramser (1959), Gaskell (1967), Hayes (1967), and Tillman and Cochran (1968) were briefly mentioned when the terminology is discussed, but their algorithms were not described.

Besides surveying existing methods, Christofides et al. (1979) also proposed two new heuristic algorithms targeting CVRPs. The first is a tree-based search heuristic, which, even if it is quite hard to see from the algorithm description, requires user to set its parameters and involves randomized decisions in selecting the customer to branch on. Furthermore, no default parameter values nor the number of repetitions to achieve the reported quality for the solutions were given, which excludes the tree-search based heuristic of Christofides et al. (1979) from our study. The other proposed heuristic is a two-phase algorithm. In its first phase it uses a score to greedily build routes around seed customers and in the second phase the heuristic seeks to rebuild the routes using a more sophisticated procedure. Also this second algorithm seems to rely on random choices of the seed points, but it seems feasible to make these decisions deterministic. This would involve using a selection heuristic for the seed points of the first phase. With these modifications the two-phase algorithm of Christofides et al. (1979) is eligible for implementation in this study.

Watson-Gandy and Foulds (1981)

Few years later, the survey from Watson-Gandy and Foulds (1981) presented the latest ideas from the literature of the time. The heuristic approaches were now categorized into three classes: *route-first (RF)*, *cluster-first (CF)*, and *relaxed optimization (RO)* procedures.

The savings heuristic from Clarke and Wright (1964) with its various modifications and extensions (Gaskell, 1967; Knowles, 1967; Tillman and Cochran, 1968; Yellow, 1970; Holmes and Parker, 1976) were noted to belong to the RF class. Note that Watson-Gandy and Foulds (1981) attributed the sequential savings algorithm to Webb (1964), which seems to be the earliest publication where this variant is described. However, we were not able to gain access to this article or the conference paper cited in (Gaskell, 1967), and, hence, we had to rely on the descriptions in secondary sources such as Christofides et al. (1979, pp. 327–328). Also, the insertion approach from Mole and Jameson (1976), as well as the two-phase algorithm from Christofides et al. (1979), were described under discussion of RF algorithms. However, in our opinion, it could be argued that the first phase of the Christofides et al. (1979) algorithm defines the clusters by generating the seed points and that the second phase uses these assignments to do the actual routing, thus making the algorithm a CF heuristic.

Out of the CF methods, the algorithms from Wren and Holliday (1972) and Fisher and Jaikumar (1981) were mentioned. The Fisher and Jaikumar (1981) algorithm uses a sweep-like procedure to generate a number of seed points. Then it relaxes the VRP and solves it as a generalized assignment problem (GAP). The objective function is kept linear by approximating the route length using the distances to the seed points. We would like to note that because mathematical programming is used to solve the GAP, the Fisher and Jaikumar (1981) heuristic could also be categorized as a RO procedure. In addition to it, another two RO algorithms were mentioned; the first being the Petal algorithm from Foster and Ryan (1976) and the second the truncated branch-and-bound, tree-search based algorithm method of Christofides et al. (1979).

Some algorithms surveyed by Watson-Gandy and Foulds (1981) were omitted

from this meta-survey. As explained earlier, the algorithm from Christofides and Eilon (1969) involves random initial tours, and thus, does not meet our criteria. The same applies to the heuristic of Buxey (1979), where randomization of the savings merges is used to improve the resulting quality of the solutions. Also, some of the surveyed works concentrated their solving effort on other VRP variants, or did not describe a complete procedure that could be written as a computer program.

Bodin et al. (1983)

In 1983, Bodin et al. wrote a special issue to the Computers and Operations Research journal. The issue was entirely devoted to vehicle routing and crew scheduling, including a classification of solution strategies used in vehicle routing (Section 2.4.4 Bodin et al., 1983, p. 98). The survey did not limit itself to CVRP, and, due to the scope of this study, we did not consider the surveyed algorithms specializing in stochastic, arc-routing, multi-depot, or fleet size and mix problems. Furthermore, the special issue contained an extensive bibliography on the topic, but not all classical heuristics listed there are discussed in the text. This meta-survey aims to concentrate only on the most central and influential classical heuristics, and, hence, only the methods included in the main discussion on the VRP solution strategies are noted.

From each class of algorithms only few meets our criteria: the *cluster-first, route-second* (CFRS) sweep algorithm from Gillett and Miller (1974), the *route-first, cluster-second* (RFCS) algorithm from Newton and Thomas (1969), and the *savings* algorithm from Clarke and Wright (1964). Also, out of the relaxed *mathematical programming approaches* the algorithms from Fisher and Jaikumar (1981) and Stewart and Golden (1984), fulfill our criteria. The heuristic from Stewart and Golden, later published in (Stewart and Golden, 1984), borrows the idea of Lagrangian relaxation used in mathematical programming and uses it to steer an infeasible initial solution towards better and feasible solutions through a series of 3-opt* moves.

Of the other heuristics mentioned in the survey, the two-phase *improvement* algorithms from Christofides and Eilon (1969) and Russell (1977) rely on stochasticity and are, therefore, omitted. Similarly, the *interactive optimization* approaches, such as the one of Krolak et al. (1972), and *exact procedures* are outside of the scope of this study.

Laporte (1992)

Gilbert Laporte has written several algorithm surveys to serve the vehicle routing community. His review article Laporte (1992) surveyed the main algorithms published before the year 1992. Of particular interest in his survey to us was the section that describes four heuristic algorithms for solving CVRPs. These included the familiar savings algorithm from Clarke and Wright (1964) with its well-known extensions (Gaskell, 1967; Yellow, 1970). As a more recent extension, the experimental study from Paessens (1988) was mentioned. What makes his savings variant interesting is the a computationally effective strategy to select the most suitable values for parametrized savings function. Therefore, while the Paessens savings algorithm contains parametrized components, there is no need to manually configure them.

In the next section of the survey, the both variants of the sweep algorithm (Wren and Holliday, 1972; Gillett and Miller, 1974) were described, followed by a description of the two-phase algorithm of Christofides et al. (1979). The heuristics based on mathematical programming were discussed next under the section on exact algorithms. Pure exact methods are outside of the scope of this study, but out of the methods discussed earlier, the heuristics from Fisher and Jaikumar (1981) and Foster and Ryan (1976) were mentioned.

Laporte (1992) also described the tabu search algorithm. While the algorithm can be implemented to be deterministic, it is heavily parametrized. Also, the tabu list size and iteration count can always be further increased to achieve even better results. In fact, tabu search describes a more general framework for solving VRPs, does not rely on any specific local search operator, and its termination rule must be chosen carefully. Hence, it is usually classified as a metaheuristic. Its appearance in the surveys can be considered to mark the transition from the era of classical heuristics to the one of metaheuristics. As with other metaheuristics, we refrained from including tabu search into our study.

Fisher (1995)

Fisher (1995) discussed the vehicle routing algorithms in three parts. The first part was reserved for *the first generation of simple heuristics*. Here, the savings algorithm from Clarke and Wright (1964) with its extensions from Gaskell (1967) and Yellow (1970) were discussed once more. The sequential variant was mentioned but, again, it was not attributed to Webb (1964). As a new addition to the family of savings algorithms, the matching heuristic from Altinkemer and Gavish (1991) was mentioned. It was noted to offer significant improvements over the existing experimental results by iteratively solving a maximum matching problem.

The Christofides and Eilon (1969) algorithm with its 3-optimized random initial solutions, as well as the algorithms from Russell (1977), and the work of Thompson, that seems to be published later in (Thompson and Psaraftis, 1993), were mentioned in the survey. However, all these three heuristics rely on stochasticity, so them, and the other studies that concentrate primary on other VRP variants or on local search, were outside the scope of our study. After the discussion on stochastic local improvement methods, the two-phase heuristic of Gillett and Miller (1974) was described and the heuristics of Christofides et al. (1979) and Tyagi (1968) were briefly mentioned.

The second part of the Fisher (1995) survey discussed *mathematical programming based heuristics*. Some of the mentioned papers are theoretical in nature and do not present any experimental results that could be used to estimate their applicability. However, also more practical algorithms were discussed; the GAP approach used in Fisher and Jaikumar (1981) was described in depth including their method for seed generation. Similarly, the heuristic of Bramel and Simchi-Levi, later published in (Bramel and Simchi-Levi, 1995) was described: The algorithm determines the seed locations by solving a capacitated concentrator location problem and, hence, shares similarities with the GAP approach. This was followed by a detailed discussion of the set partitioning approach. The one algorithm that fulfilled to our criteria is that from Foster and Ryan (1976). Furthermore, out of the many papers discussed by

Fisher (1995) it is seems to be the only heuristic that concentrates on CVRPs and does not involve interaction with a human operator. The other cited papers mainly concentrated on modeling side constraints like time windows or split deliveries and were, therefore, omitted.

In the last part of the survey Fisher (1995) discussed the third generation of VRP algorithms. This included the first metaheuristics that were surveyed together with the latest advances in local search, followed by a thorough discussion of exact methods. However, these topics are not relevant to the topic of this study and are not listed here.

Laporte et al. (2000); Laporte and Semet (2002)

Laporte et al. (2000) and Laporte and Semet (2002) significantly extended the previous survey from 1992. In fact, an entire chapter in (Toth and Vigo, 2002b) was dedicated to the classical heuristics for the CVRP, which indicates the concept of had been more or less stabilized by the year 2002.

These surveys are almost identical, and both begin with a discussion on the savings algorithms. There, the sequential version of the savings algorithm was explicitly mentioned and its operating principle explained. While the method seems to originate from Webb (1964), it is not attributed to him in either of the surveys. The usual enhancements to the Clarke and Wright (1964) savings heuristic were mentioned (Gaskell, 1967; Yellow, 1970; Paessens, 1988). Mentioned are also the works of Golden et al. (1977) and Nelson et al. (1985) which mainly contribute to the implementation aspects of the savings algorithms. As a new category of the savings approach, the matching algorithms (Desrochers and Verhoog, 1989; Altinkemer and Gavish, 1991; Wark and Holt, 1994) were discussed. Of these, the repeated matching algorithm from Wark and Holt (1994) relies on stochasticity in certain circumstances, while the matching based approach presented by Desrochers and Verhoog (1989) and extended in Altinkemer and Gavish (1991) appear to be computationally quite intensive. The Mole and Jameson (1976) insertion variant of the savings approach and the Christofides et al. (1979) two-phase algorithm were discussed under the title of *sequential insertion heuristics.* However, including the Christofides et al. (1979) algorithm to this class of heuristics seems a bit misleading because the method does not, in fact, use insertions. Instead, it works by associating customers to clusters based on their distance to a set of seed customers.

The discussion of *two-phase methods* included a general description of the sweep algorithm with relevant references to both Wren and Holliday (1972) and Gillett and Miller (1974). Of the cluster-first, route-second algorithms, the Fisher and Jaikumar (1981) algorithm as well as a similar but later and more computationally demanding capacitated concentrator location based algorithm from Bramel and Simchi-Levi (1995) were briefly described. The truncated branch-and-bound tree-search based algorithm from Christofides et al. (1979) was mentioned, but its stochastic and parametric nature was not noted, even if it would have been relevant when discussing the computational effort of the algorithms.

When discussing the algorithms solving the VRP as a weighted set covering problem, it was mentioned that the Petal algorithm from Foster and Ryan (1976) is expected to be able to solve large problems in a reasonable time. It was also

noted that it can be further accelerated using the more efficient solution techniques from Ryan et al. (1993). The Petal algorithm of Renaud et al. (1996) uses a different method for generating the petals, with allegedly improved the quality of the solutions. However, this algorithm also comes with many parameters.

In these surveys, the idea of route-first, cluster-second approach was attributed to Beasley (1983), although from the earlier surveys has become evident that the idea had been proposed earlier, for example, by Newton and Thomas (1969). However, it seems Beasley (1983) was the first who evaluated the performance of the route-first, cluster-second approach on standard vehicle routing problems. Out of the surveyed works, Beasley (1983) also seems be the only one of that presented experimental results, whereas other similar works concentrated on bounds and theoretical analysis of the approach. The Beasley (1983) algorithm uses random initial TSP tours through all customers in the routing phase, one can see how a deterministic variant could be built using an optimal TSP tour.

From here, the surveys (Laporte et al., 2000; Laporte and Semet, 2002) continued with the description of *improvement heuristics*. Most of the literature surveyed for this topic concerned descriptions of different local search move operators. Complete VRP algorithms are rare, but the Lagrangian relaxation heuristic from Stewart and Golden (1984) can independently solve CVRPs. Also, the survey mentioned the work from Fahrion and Wrede (1990), where they proposed yet another alternative for improving the result of the Clarke and Wright algorithm with a chain-exchange local search post-optimization step. Also, the algorithm of Thompson and Psaraftis (1993) comes close to fulfilling our criteria, as they provide a complete algorithm built around a cyclic transfer local search neighborhood. However, their local search operator is parametrized and the proposed heuristic starts from a random initial solution.

After discussion on other VRP variants and metaheuristics, Laporte and Semet (2002) concluded their survey to a remark that there 'is little room left for significant improvement in the area of classical improvements' and that the time has come to shift the research focus towards local search and metaheuristics. This observation further solidifies the concept of classical VRP heuristics.

Cordeau et al. (2002)

Cordeau et al. (2002) surveyed and summarized several of the most important heuristics for the VRP. They made a clear distinction between the classical heuristics, metaheuristics with local search, and population search. Of the classical heuristics, the algorithm from Clarke and Wright (1964) with an optional 3-opt post-optimization step was described. Savings improvements from Gaskell (1967) and Yellow (1970) were mentioned as well as the usual computational enhancements (Nelson et al., 1985; Paessens, 1988). The sequential savings was discussed, but, again, it was not attributed to Webb (1964). Also, the savings matching algorithms (Desrochers and Verhoog, 1989; Altinkemer and Gavish, 1991; Wark and Holt, 1994) were recognized as an another stream of research on improving the Clarke and Wright (1964) algorithm.

The sweep algorithm was described with appropriate references (Gillett and Miller, 1974; Wren and Holliday, 1972). The survey also recognized the connec-

tion between sweep algorithms and the Petal algorithm (Foster and Ryan, 1976) and its extensions (Ryan et al., 1993; Renaud et al., 1996). The survey included a description of the Fisher and Jaikumar (1981) GAP algorithm, but also mentioned issues Cordeau et al. (2002) had had in replicating their results. Finally, location based heuristic from Bramel and Simchi-Levi (1995) was briefly mentioned before the focus of the survey turns towards metaheuristics.

Cordeau et al. (2007)

While the book Transportation edited by Barnhart and Laporte (1993) took a more general view to the research developments in transportation, it also contained a chapter dedicated to VRP (Cordeau et al., 2007). Classical heuristics were surveyed using the algorithm classification from Laporte and Semet (2002). Out of the *route construction heuristics*, parallel savings (Clarke and Wright, 1964) and sequential savings (mentioned, not cited, Webb, 1964), insertion (Mole and Jameson, 1976), the two-phase heuristic (Christofides et al., 1979), and the matching algorithms (Desrochers and Verhoog, 1989; Altinkemer and Gavish, 1991; Wark and Holt, 1994) were listed. The *cluster-first, route-second* part of the survey on *two-phase heuristics* contained a familiar set of algorithms (Wren and Holliday, 1972; Gillett and Miller, 1974; Fisher and Jaikumar, 1981; Bramel and Simchi-Levi, 1995; Foster and Ryan, 1976; Ryan et al., 1993; Renaud et al., 1996) and the truncated branch-and-bound tree based method of Christofides et al. (1979). Of the *route-first, cluster-second* methods the following were listed: Beasley (1983); Haimovich and Rinnooy Kan (1985); and Bertsimas and Simchi-Levi (1996). However, only Beasley (1983) describes a complete algorithm together with experimental results on CVRP instances.

Before surveying metaheuristics, Cordeau et al. (2007) also discussed classical route improvement heuristics. However, out of the cited works, only the one from Thompson and Psaraftis (1993) can be considered to describe a complete CVRP heuristic.

Laporte (2007, 2009)

To avoid bias in including too many surveys from the same authors, the two surveys by Laporte (2007, 2009) are examined together. Besides giving a survey on the several families of exact algorithms for the VRP, they discuss classical heuristics in depth. According to Laporte (2007), the qualificative trait in classical heuristics is that they perform descents, i.e., they always move towards better solutions. This is in contrast to metaheuristics that may allow non-improving moves, or even moves that make the solution temporarily infeasible.

The algorithm from Dantzig and Ramser (1959) was explicitly mentioned in Laporte (2009), followed by the savings algorithm (Clarke and Wright, 1964) and enhancements that can speed up the computations (Golden et al., 1977; Nelson et al., 1985; Paessens, 1988). In this survey the parametrized savings variant was attributed to Golden et al. (1977), even though the idea originates from earlier works of Gaskell (1967) and Yellow (1970). Of the matching approach, the variants from Altinkemer and Gavish (1991) and Wark and Holt (1994) were explicitly mentioned.

The methods surveyed under set partitioning problem heuristics included Gillett and Miller (1974), Foster and Ryan (1976), and the extensions to the latter from Ryan et al. (1993) and Renaud et al. (1996). Under cluster-first, route-second heuristics, the algorithm from Fisher and Jaikumar (1981) is mentioned together with its extension from Baker and Sheasby (1999). Finally, in the discussion preceding the one on metaheuristics the improvement heuristics due to Thompson and Psaraftis (1993) is briefly described.

Vidal et al. (2013)

While the motivation for the work of Vidal et al. (2013) came from making a synthesis of the recent research of multi-attribute VRPs, they also included a comprehensive survey on the classical heuristics. Also, the paper has been influential in the recent VRP research , which further supports its inclusion to this meta-survey.

The surveyed algorithms included the savings heuristic of Clarke and Wright (1964), with its usual extensions from Gaskell (1967) and Yellow (1970), as well as the insertion algorithm of Mole and Jameson (1976). The sweep algorithm of Gillett and Miller (1974) was briefly described, as well as the *route-first, cluster-second* approach that was attributed to several authors (Newton and Thomas, 1974; Beasley, 1983, are those relevant to this study). The last classical constructive heuristic surveyed was the *cluster-first, route-second* algorithm from Fisher and Jaikumar (1981). When the local-improvement heuristics were discussed, also the cyclic transfers of Thompson and Psaraftis (1993) are mentioned.

Laporte et al. (2014)

The last survey in our meta-survey is the most recent update to the review series on classical VRP heuristics from Laporte et al. (2014). They reviewed methods that "have withstood the test of time or present some interesting distinctive features" and the classical heuristics are discussed under *constructive* and *improvement heuristics*.

The parallel savings heuristic of Clarke and Wright (1964) was described together with some of its enhancements (Nelson et al., 1985; Paessens, 1988). The other approach that they recognized of still having a practical value is the Petal algorithm of Foster and Ryan (1976) with its extensions (Ryan et al., 1993; Renaud et al., 1996). Also, the very first VRP heuristic from Dantzig and Ramser (1959) is briefly mentioned. Interestingly, Laporte et al. (2014) mentioned that it may have inspired the later matching based heuristics from Desrochers and Verhoog (1989); Altinkemer and Gavish (1991); Wark and Holt (1994). Again, of the listed *improvement heuristics* only the work of Thompson and Psaraftis (1993) describes a complete deterministic CVRP algorithm. Other surveyed works propose new local search moves, or can already be considered to be basic metaheuristics.

The survey from Laporte et al. (2014) completes our meta-survey on classical vehicle routing heuristics. Please find the analysis of the results of this meta-survey in the beginning of Section 4 that is presented after the survey on open source vehicle routing software below.

3.2 Vehicle Route Optimization Libraries

VRPH[1] is a high performance library that has been shown to produce solutions to CVRPs that are within one percent of the best-known ones on many of the well-known academic benchmark problems (Groër et al., 2010). Also, this performance can be further improved through automatic algorithm configuration (Rasku et al., 2019b). VRPH implements seven local search move operators, three metaheuristics, and two classical construction heuristics: the parametrized parallel savings algorithm from Gaskell (1967) and the first phase of the sweep algorithms (Wren and Holliday, 1972; Gillett and Miller, 1974). However, the library is geared towards implementing metaheuristics and because VRPH is written in C++, integrating external exact integer programming solvers that are needed to implement certain classical heuristics would have been laborious. Also, VRPH does not include any tests for the correctness of the algorithms, and its development seems to have ceased. In our earlier experiments and comparisons (Rasku et al., 2016, 2019b) we have found there to be room for improvement in the code of VRPH, which makes it suboptimal foundation to build on. However, the implementations of the heuristics are fast and the library is published under the very permissive Common Public License (CPL), which makes it a good tool for computational experiments on metaheuristics.

The open source **OptaPlanner**[2] is 'a lightweight, embeddable constraint satisfaction engine which optimizes planning problems' (De Smet et al., 2016). It is written in Java and is licensed under very liberal Apache Software License 2.0 (ASLv2). OptaPlanner is capable of solving VRPs and other combinatorial problems. It comes with generic construction heuristics such as cheapest insertion and local search heuristics such as hill climbing variable neighborhood descent (VND). While its heuristics could be used as a building blocks to implement classical vehicle routing heuristics, the architecture as a generic combinatorial optimization suite is quite sophisticated. The limitations of the architecture could potentially become restrictive when implementing the classical heuristics, especially since the software is quite extensive and geared towards general use. This generality would necessitate modifying and including large amount of code not directly related to classical VRP heuristics. Furthermore, properly building on top of OptaPlanner could require including support for other variants of VRP (e.g., VRPTW and PDP).

Jspirit[3] (Schröder and GraphHopper team, 2018) is an open source toolkit for solving rich TSPs and VRPs. It is in many ways similar to OptaPlanner, as both are written in Java and have a company committed to developing the software further. They even share the same ASLv2 open source license. On the solver side, Jspirit uses a single generic metaheuristic to handle many vehicle routing variants. The metaheuristic was described in (Schrimpf et al., 2000) and was later extended with ideas from Pisinger and Ropke (2007). The toolkit seems to be actively developed, documented, well-tested, flexible, and easy-to-use. However, due to its many dependencies, generality, and being written in Java, the architecture of the library seems quite complex and it proved to be difficult to estimate the feasibility and effort of extending it to support classical CVRP construction heuristics.

[1] `https://projects.coin-or.org/VRPH`
[2] `https://www.optaplanner.org/`
[3] `https://github.com/graphhopper/jsprit`

OscaR[4] (OscaR team, 2012) is a Scala toolkit for solving Operations Research problem. It is published with GNU Lesser Genral Public License 3.0 (LGPLv3). Among other solution approaches, it comes with constraint-based local search library implementing standard neighborhoods: insertion, one-point-move, two-point-move, relocate, exchange, cross-exchange, chain, 2-opt, and 3-opt. However, the main focus of the toolkit is not in heuristics nor in VRP, and the same comments as with OptaPlanner or Jspirit related to architecture, generality, and chosen programming language apply.

Google Optimization Tools or **OR-Tools**[5] is a ASLv2 licensed software suite for solving combinatorial optimization problems (Google, 2018) including vehicle routing library for TSP and VRP. Of the construction heuristics OR-Tools has nearest neighbor, cheapest insertion, sweep, and parametrized savings. It also comes with local search with a large neighborhood search and other metaheuristics (guided local search, simulated annealing, and tabu search). Of the local search operators the library implements relocate, exchange, cross, cross-exchange, 2-opt, Or-opt, and Lin-Kernighan usage heuristics. The code is well documented C++ with F#, Python, Java and C# bindings with multiple examples. Also some tests exists, but seem to be mostly concentrate on regression testing. OR-Tools would be a good platform to extend, but as our aim was academic solver concentrating on simplicity and readability, we were hesitant to use such an extensive library as a foundation for our implementations.

Open-VRP[6], while no longer being actively developed, is a well documented academic open source software framework capable of solving different routing problems, including CVRPs. The library is written in Common Lisp and licensed under Lisp lesser GNU public license (LLGPL). The architecture of the framework is clean, and it seems easy to extend. However, currently only insertion heuristic and tabu search metaheuristic are implemented and, thus, for implementing classical heuristics, many local search heuristics should be added. Also, MIP software packages rarely offer Lisp bindings, which would make implementing relaxed mathematical programming VRP methods laborious.

While mainly being a generic open-source (under Eclipse Public License 1.0) MIP solver, **SYMPHONY** (Ralphs and Güzelsoy, 2005) also has extensions and an application for solving VRPs. The application has built-in heuristics for finding the upper bound for the exact solver and its C source code [7] implements relocate and exchange local search heuristics. The construction heuristics include a custom clustering insertion; nearest neighbor; sequential savings; and a route-first, cluster-second scheme with several TSP initialization methods. We tried to compile the current version of 5.6.16 with the heuristics enabled, but this failed. After some research into the issue, it became apparent that it is hard to estimate the extent of the modifications to reuse the code. In addition there were significant amount of code duplication, and using the SYMPHONY codebase would have required additional refactoring effort.

[4] https://bitbucket.org/oscarlib/oscar/wiki/Home

[5] https://developers.google.com/optimization/

[6] https://github.com/mck-/Open-VRP

[7] https://github.com/coin-or/SYMPHONY/tree/master/SYMPHONY/Applications/VRP/src/Heuristics

VROOM (Coupey, 2018) is another open-source (under BSD 2-clause license) optimization engine for VRPs. It is written in C++14 and currently offers algorithms for shortest path computation and solving TSPs using local search heuristics. It's being actively developed with a clustering CVRP heuristic to be added in the near future. However, it is still in early stages of its development, and does not currently offer a stable foundation for the classical heuristics.

While not being a traditional software library, another popular option for solving VRP seems to be the **Open Source Spreadsheet based Solver** from Erdoğan (2017). Microsoft Excel is undoubtedly a good solution when trying to remove barriers preventing the use of vehicle route optimization, but this technological choice makes it hard for us to extend the solver to offer other algorithms than the large neighborhood search (LNS) metaheuristic that the package currently offers.

To conclude the literature review on open source vehicle routing libraries, we note that there is a wide selection of readily available models and algorithms for solving typical academic VRPs. However, while some of the libraries implement one or two classical algorithms, there is no VRP library with a comprehensive collection of classical heuristics. Most of the surveyed libraries are implemented with a specific metaheuristic in mind and one that includes, or would make easy to include, relaxed mathematical programming approaches is missing. Also, while not surveyed here, some of the more general purpose frameworks for designing local search algorithms such as EasyLocal++ from Di Gaspero and Schaerf (2003) could be used as a foundation for building VRP heuristics.

However, the existing libraries typically follow the object oriented paradigm (OOP) (Booch, 1998) and are written in object oriented programming languages (mainly Java and C++). While this paradigm allows building powerful abstractions, it can also lead to unnecessary structural complexity and inflexibility (Subramanyam and Krishnan, 2003). Hence, using a framework and OOP is to some extent in conflict with our aims related to understanding, implementing, and replicating the results of algorithms and producing a simple and easy to understand software library for classical VRP heuristics. Thus, based on the earlier meta-survey on classical vehicle routing algorithms, and the survey on the available open source implementations, a decision was made to implement an independent open source library of classical vehicle routing algorithms in Python following the functional programming paradigm. This involved choosing the algorithms to be implemented, which is discussed next.

4 The Implemented Classical Heuristics

Based on our meta-survey of classical heuristics in Section 3.1, we proceed to give abbreviations to those methods that had been cited at least twice and in at least in 20% of the surveys published after the method had been proposed. These requirements are somewhat arbitrary, but also necessary to rule out methods that have failed to stay relevant. Additionally, we use a key after the algorithm abbreviation that indicates if the method requires human interaction ($\times$), if it is parametrized ($*$), if it is stochastic ($\dagger$), or if there are no published experimental results or we were unable to gain access to the paper ($\circ$).

Table 1: Algorithms from the meta-survey of classical VRP heuristics.

	One phase									Matching			Improvement			RFCS		CFRS						Relaxed optimization					
	DR59-EM	CW64-PS	We64-SS	Ga67-PS|$\pi\lambda$	Kn67-TBS$^{\circ}$	TC68-2PS$^{\circ}$	Ye70-PS|G1P*	HP76-PS|MS*	MJ76-INS	DV89-MBSA	AG91-MM	WH94-MM†	Ru77-MTOUR$^{\dagger\times}$	Pa88-PS|G2P*	TP93-PS|CT$^{\dagger *}$	CE69-rOPT$^{\dagger *}$	Be83-RFCS†	Ty68-NN	Kr72-CLS$^{\times *}$	WH72-SwLS	GM74-SwRI	CMT79-2P†	FR76-1PTL	CMT79-DTS†	FJ81-GAP	LR3OPT†	SG84-	BSL95-CLP	RBL96-2PTL
Eilon et al. (1971)		✔		✔	✔	✔		—	—	—	—	—	—	—	—	✔			—	—	—	—	—	—	—	—	—	—	—
Turner et al. (1974)	✔	✔		✔	✔	✔		—	—	—	—	—	—	—	—	✔			✔		✔	—	—	—	—	—	—	—	—
Golden (1978)	✔	✔		✔	✔	✔	✔	✔		—	—	—	✔	—	—	✔	✔	✔	✔		✔	—		—	—	—	—	—	—
CMT (1979)		✔	✔						✔	—	—	—		—	—						✔	✔		✔	—	—	—	—	—
Mole (1979)	✔	✔	✔	✔	✔		✔	✔	✔	—	—	—	✔	—	—	✔			✔	✔	✔		✔		—	—	—	—	—
W-GF (1981)	✔	✔	✔	✔	✔	✔	✔	✔	✔	—	—	—	✔	—	—	✔		✔		✔	✔	✔	✔	✔	✔	—	—	—	—
Bodin et al. (1983)		✔								—	—	—	✔	—	—	✔	✔				✔			✔	✔	✔	—	—	—
Laporte (1992)		✔		✔			✔					—		✔	—					✔	✔	✔	✔		✔		—	—	—
Fisher (1995)		✔	✔	✔			✔				✔		✔		✔	✔		✔			✔	✔	✔	✔	✔		✔	—	—
Laporte et al. (2000)		✔	✔	✔			✔		✔	✔	✔	✔		✔	✔		✔			✔	✔	✔	✔		✔	✔	✔	✔	✔
Cordeau et al. (2002)		✔	✔				✔			✔	✔	✔		✔						✔	✔		✔		✔		✔	✔	✔
Cordeau et al. (2007)		✔	✔						✔	✔	✔	✔			✔		✔			✔	✔	✔	✔	✔	✔		✔	✔	✔
Laporte (2007)		✔					✔				✔	✔	✔	✔	✔						✔		✔		✔				✔
Vidal et al. (2013)		✔		✔			✔		✔						✔		✔				✔				✔				
Laporte et al. (2014)	✔	✔								✔	✔	✔		✔	✔								✔						✔
Cited in (%)	33	100	47	60	33	27	60	23	46	50	75	71	46	63	86	47	33	20	21	43	93	50	69	42	90	22	57	83	

$\times$ is interactive, $*$ is parametrized, † has a stochastic component, $\circ$ no experimental results or the paper is not publicly available

The results of the meta-survey are summarized in Table 1, which lists how many times the methods were mentioned in the literature. On the summary row, only those surveys published after the introduction of the method are considered in the calculation of the citing percentage. Please note that there is some room for consideration in what is counted as a reference depending on the nature of the survey and on the context in which the original work was mentioned. For example, the fundamental VRP paper from Dantzig and Ramser (1959) is cited in every survey, but their proposed algorithm for solving the truck dispatching problem is explicitly mentioned only in few of them. Similarly, the sequential savings approach is not always attributed to Webb (1964), but to keep the citation counts commensurable, all mentions of the sequential savings heuristic are marked in the column of We64-SS. Also, please note that the route-first, cluster-second heuristic had already been proposed in (Newton and Thomas, 1969, 1974) for solving a related school bus routing problem, but in Table 1 we count all citations to those studies towards a later study from Beasley (1983), who applied the approach to CVRP.

The list of algorithms to be implement was further shortened by using the specific criteria used in this study. Consequently, in the abbreviation list below, the methods chosen to be implemented in VeRyPy are marked with ✔ and those omitted are marked with ✘. In case an algorithm has been omitted, the main reason for its elimination is given. In addition to our criteria, those methods that lacked experimental results on CVRP benchmark instances (e.g., Beltrami and Bodin, 1974) or for which we were unable to verify that such data exists (Knowles, 1967; Tillman and Cochran, 1968) were not chosen.

As we saw in the meta-survey, there are many ways to categorize and classify VRP heuristics. The categories are usually somewhat overlapping and we are aware that the taxonomy used in Table 1 and in the list below is somewhat arbitrary. However, the used taxonomy allows clarifying the differences between the general heuristic approaches.

One Phase Heuristics

✘ **DR59-EM**, the edge matching based heuristic of Dantzig and Ramser (1959). Omitted from this study as it has been superseded by CW64-PS (according to, e.g., Fisher, 1995).

✔ **CW64-PS** is the original parallel savings algorithm of Clarke and Wright (1964).

✔ **We64-SS** or sequential savings, attributed to Webb (1964) where it seems to originate. The fact that another sequential algorithm from Yellow (1970) was often cited, contributed to implementing the original sequential savings heuristic from Webb (1964) in our study.

✔ **Ga67-PS**$|\pi/\lambda$, the (parallel) π and λ-savings algorithms of Gaskell (1967).

✘ **Kn67-TBS**$^\circ$ the tree-search savings method of Knowles (1967). It is omitted because we were able to gain access only a secondary source description of the algorithm (Eilon et al., 1971), and also because its similarity to HP76-PS$|$IMS*.

✘ **TC68-2PS**$^{\circ}$ is the look-ahead savings variant of Tillman and Cochran (1968) for which we were also unable to find the published paper. This made replicating the results impossible and it was omitted from this study.

✘ **Ye70-PS|G1P*** the sequential parametrized savings criteria heuristic of Yellow (1970). It has been partly superseded by Pa88-PS|G2P* that further generalizes the savings criteria. The sequential route building behavior is sometimes attributed to Yellow (1970), but the idea becomes more apparent in We64-SS.

✔ **HP76-PS|IMS*** is the iterative parallel savings merge suppression heuristic of Holmes and Parker (1976).

✔ **MJ76-SI** is the sequential insertion heuristic of Mole and Jameson (1976).

Matching Heuristics

✔ **DV89-MBSA**, the Desrochers and Verhoog (1989) maximum matching based savings heuristic.

✘ **AG91-MM**, the Altinkemer and Gavish (1991) two-phase maximum matching heuristic with dummy nodes. Omitted due to being introduced quite late to be considered classical heuristic. It is also only a marginal improvement over DV89-MBSA.

✘ **WH94-MM**† Wark and Holt (1994) cluster splitting maximum matching heuristic. It has a stochastic component and there is no straightforward modification leading to a deterministic variant. Thus, it was omitted.

Improvement Heuristics

✘ **Ru77-MTOUR**$^{\dagger\times}$ is the MTOUR local search improvement heuristic of Russell (1977). It is omitted due to its random perturbation and manual crafting of initial solutions, both of which would make it hard to replicate the results.

✔ **Pa88-PS|G2P** is the Paessens (1988) two parameter parallel savings heuristic with an intra-route 3-opt improvement phase and a parametrized savings function auto-tuning strategy.

✘ **TP93-PS|CT**$^{\dagger *}$ uses random initial solution with cyclic transfer improvement phase (Thompson and Psaraftis, 1993). It is omitted because of its stochasticity and parameters.

Route-First, Cluster-Second Heuristics

✘ **CE69-rOPT**†, is the stochastic r-optimal improvement approach (Christofides and Eilon, 1969; Eilon et al., 1971). Omitted due to stochasticity.

✔ **Be83-RFCS**†, the route-first, cluster-second algorithm for CVRP of Beasley (1983), which was proposed for bus routing in (Newton and Thomas, 1969, 1974).

Cluster-First, Route-Second Heuristics

- ✔ **Ty68-SNN** is the sequential nearest neighbor insertion heuristic of Tyagi (1968).
- ✔ **WH72-SwLS**, sweep with local search improvement phase of Wren and Holliday (1972).
- ✔ **GM74-SwRI**, sweep with TSP solver and inter-route improvement of Gillett and Miller (1974).
- ✔ **CMT79-2P**†, the Christofides et al. (1979) two-phase heuristic.

Relaxed Mathematical Programming Heuristics

- ✔ **FR76-1PTL**, the set covering Petal heuristic of Foster and Ryan (1976)
- ✘ **CMT79-TTS**†, the truncated tree search heuristic of Christofides et al. (1979). Omitted due to stochasticity.
- ✔ **FJ81-GAP**, the generalized assignment problem (GAP) heuristic of Fisher and Jaikumar (1981).
- ✔ **SG84-LR3OPT**†, the Lagrangian relaxation inspired 3-opt method of Stewart and Golden (1984).
- ✘ **BSL95-CLP**, the capacitated concentrator location problem seed generation for the GAP heuristic from Bramel and Simchi-Levi (1995). Omitted due to being computationally too expensive (see below). It has also been published quite late to be considered a classical heuristic.
- ✘ **RBL96-2PTL**, the 2-Petal extension of Renaud et al. (1996). Omitted due to being a computationally expensive extension to FR76-1PTL and due to late publication date for a classical heuristic.

There are few omission decisions in the above list for which we would like to give more details on: To decide if the Bramel and Simchi-Levi (1995) BSL95-CLP algorithm could be included in our study, we estimated the computational effort required to solve a 1000 customer problem. If we assume[8] that BM POWER1 CPU, on which the results of Bramel and Simchi-Levi (1995) were computed, produces around 70 VAX MIPS (around 40 MWIPS), then a modern day Xeon processor is around 70 to 100 times more powerful (Longbottom, 2014). Extrapolating the computational effort reported by Bramel and Simchi-Levi (1995) suggests that it would take IBM POWER1 from around 55 hours to 860 hours to produce a solution for the hypothetical 1000 point problem. Thus, even the most optimistic estimate would give a solution time in the ballpark of four hours of single threaded computation on a modern 3.7 Ghz Intel i7 CPU, which makes it an infeasible approach to use on large problems. This is also supported by Table 2 in Fisher (1995), which suggests

[8] S/6000 350; POWER1 @ 41.6 Mhz, 40.7 MWIPS, 70 VAX MIPS, 44.7 MWIPS/DP **VS.** Intel; Core i7 4820K @ 3700 MHz, 3063 MWIPS, 7270 VAX MIPS, 3257 MWIPS/DP

that the (Bramel and Simchi-Levi, 1995) algorithm is significantly more computationally intensive than the other mathematical programming based heuristics such as the ones from Fisher and Jaikumar (1981) and Foster and Ryan (1976).

Similar estimation can be done for the heuristics which rely on repeatedly solving the maximum matching problem. Fitting a power curve to the computational times given in Table 4 of Desrochers and Verhoog (1989) gives an estimate of 220 hours for the solution time on a 1000 customer problem on their unspecified DAC VAX/VMS computer. Assuming a top of the line VAX 8700 model of the era, and referring the benchmark data of (Longbottom, 2014), a naive best case estimate of computing a 1000 customer problem on a modern 3.7 Ghz Intel i7 CPU processor would be around 80-120 minutes assuming single threaded computation[9]. Thus, based on this estimation, it seems somewhat feasible to use a simple matching problem based approach to solve large CVRP instances on a modern CPU. However, the extensions from Altinkemer and Gavish (1991) (AG91-MM) and Wark and Holt (1994) (WH94-MM) increase the computational complexity, and would probably require excessive computing times on the larger problem instances. Based on this estimate AG91-MM and WH94-MM were omitted from our study.

Also the omission of CMT79-TTS (Christofides et al., 1979) should be discussed. It is a tree based search method inspired by a set partitioning formulation of the VRP. Each tree node represents a feasible route and at each branching the algorithm selects a route to serve nodes not yet served by any of the parent node routes. However, the method seems to rely strongly on stochastic components, is parametrized, and involves use of another independent construction heuristic to get a upper bound for the tree search. Thus, we decided to omit it from this study. However, the results in Table 11.1 Christofides et al. (1979, p. 335) indicate that the tree based algorithm is capable of producing high quality solutions with reasonable CPU effort. Also, the approach is different from the others, which makes it potentially interesting future addition to the relaxed mathematic programming heuristics offering of VeRyPy.

Looking ahead to the assessment of the algorithm features and their implementation details given later in this section, Cordeau et al. (2002) presents a useful list of desirable criteria for a vehicle routing heuristic: *accuracy*, which usually is measured with a gap between the algorithm provided solution and the optimal value; *consistency* of producing good accuracy over a large number of different problem instances or parameter values; *speed*, or how quickly the algorithm is able to produce results with an acceptable accuracy; *simplicity*, which is a measure on how hard the algorithm is to understand and implement (including the number of free parameters); and *flexibility* as a measure on how easy it is modify the algorithm to solve different VRP variants.

Please note that flexibility is linked to simplicity because simpler algorithms are usually easier to extend to accommodate new side constraints. Furthermore, each of our criteria for an algorithm to be implemented in this study is closely linked to these five attributes: the main reason behind the requirements that the algorithm should be parameter free and deterministic is to allow the results to be replicable. This is best served through consistency and simplicity. On the other hand, ability to solve

[9]DAC VAX; VAX 8700 P8 @ 22.2 MhZ, 4.98 WMIPS, 1.16 MFOPS, 5.81 VAX MIPS, 3.41 MWIPS/DP **VS.** Intel; Core i7 4820K @ 3700 MHz, 3063 MWIPS, 7270 VAX MIPS, 755 MFLOPS, 3257 MWIPS/DP

large instances requires speed and flexibility. Later in this section, the descriptions of the heuristics and their results, as presented in the original publications, are considered carefully against these requirements and attributes.

4.1 General Remarks on Replicating the Results

Reproducibility of the results should be central when the reliability of the research field is considered. This is especially true in computational sciences, where effectiveness of the proposed methodologies is usually demonstrated via empirical testing (Barr et al., 1995). Barr et al. advocated that in addition to evaluating the contributions with a necessary scientific rigor and reporting them objectively, one should document the experimental design details and publish the necessary software artifacts required to independently verify the results. This is an important prerequisite not only to estimation of the validity of the research but also to the accumulation of knowledge in science.

Regarding computational testing of heuristic algorithms, Barr et al. (1995) called for defining and documenting the termination rules, algorithm parameters (or rules to set them), and all implementation details. Also, the circumstances and factors affecting the experiment and the implementation of the algorithm should be discussed. Unfortunately, not nearly all details of the algorithms and their computational benchmarking have been disclosed in the literature on classical VRP heuristics. Hence, whenever we had to make design decisions or experimentation with the experimental setup for the replication tests, we discuss these topics along with the replicated results.

One such major factor that significantly affects the values coming from the computational experiments in the literature is that of distance and objective rounding or truncating conventions. These vary from author to author (Cordeau et al., 2002; Laporte, 2007), and, based on our survey, it seems that at least following conventions are used:

- rounding to a specified number of digits after the decimal point,
- truncating after a specified number of digits after the decimal point,
- using exact distances (R) with the full floating point accuracy of 16/24/32/64 bits,
- rounding to the nearest integer (I), or
- truncating down to the closest integer (floor, F).

Unfortunately, the exact rounding convention is rarely explicitly specified, which makes the replication of the results difficult and tedious. Sometimes the convention can be inferred from the results or appendix of the paper, but not always. Furthermore, the solution value may have been independently rounded or truncated. That is, the problem instance is solved using exact distances, but the result is given as an integer. In the discussion of the replication results, we mention whenever we believe that rounding convention had an effect on the replication results. Nonetheless, the

considerations of rounding should be kept in mind when interpreting the replication results. We give the type of rounding used with each benchmark problem instance.

In the tables for replication results following shorthands are used for constraints: C is used for capacitated vehicle routing problems (CVRP) and D for distance or duration constrained routes (DVRP, distance/duration constrained vehicle routing problem). These can be combined with the labels for rounding conventions. Thus, for example, I^{CD} stands for a **C**apacitated and **D**istance/duration constrained VRP instance with distance matrix values rounded to the nearest **I**nteger.

Cordeau et al. (2002) have made an observation that 'Accurate reporting of computing time is another delicate issue in the scientific literature .. [and] strict standards are not consistently enforced by VRP researchers when it comes to assessing the speed of algorithms.' In our study, we were able to sidestep this to some extent because we were more interested in the correctness of the implementation. Even though we made sure suitable data structures and low level algorithms were used, we mostly ignored the replication of runtimes of the implemented algorithms.

4.2 One Phase Heuristics

In this category we discuss the group of straightforward constructive heuristics that build the solution using a single simple procedure and rules. However, we have also included methods that extend these heuristics through iterations or an improvement step. That is, the simple procedure can be repeated several times with different parameters or with some of the previous decisions marked forbidden. Note that all methods with a TSP algorithm based improvement step could be considered to be cluster-first, route-second heuristics, but those whose origins can be traced back to the one phase heuristics are discussed here instead.

The most well known of the one phase heuristics are the *savings algorithms*. These heuristics build a solution by merging routes according to a sorted list of *savings* that describe how much the solution is improved by merging the two routes. The merges are applied from the largest saving first while ignoring those that would break the constraints.

The original savings algorithm from Clarke and Wright (1964) is one the best-known heuristics in operations research (Sörensen et al., 2019). Thus, it is not a surprise that a large number of authors have proposed extensions and variations to it through the years. Webb (1972) presented a computational comparison of several savings variants, and Watson-Gandy and Foulds (1981) suggested that the popularity of the extensions might be due to the relative simplicity of the procedure. Still, Webb (1972) and related experimental work have demonstrated that despite its apparent simplicity, minor changes in the merge order can have a significant and hard to predict effect on the emerging route structures and, ultimately, resulting quality of the solutions(Mole, 1979).

Implementation details and acceleration techniques for one phase algorithms can be found, e.g., from the works of Yellow (1970); Golden et al. (1977); Nelson et al. (1985) and Paessens (1988). However, due to the rapid growth of computational resources, these contributions have become somewhat irrelevant (Cordeau et al., 2002) and are not described in depth. Also, based on our meta-survey, only the most well-known savings variants are discussed in this section.

Overall, the main advantages of the one phase algorithms are related to their simplicity and speed. Meanwhile, their accuracy and flexibility are not that good, and adding new side constraints often leads to the algorithm's already modest accuracy to quickly deteriorate (Cordeau et al., 2002).

4.2.1 Savings

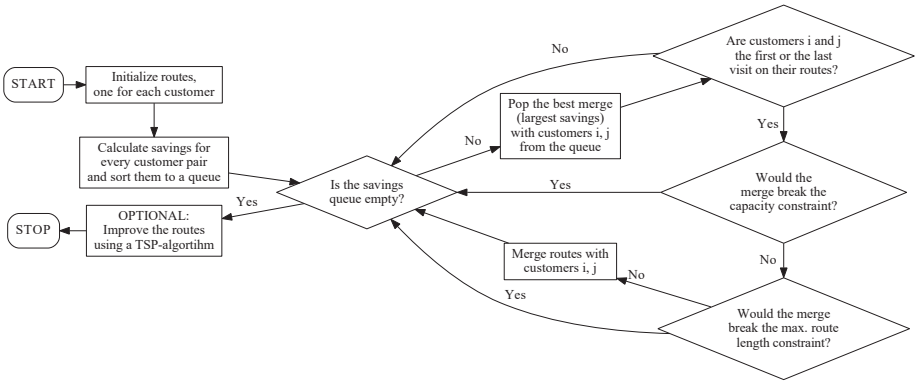


Figure 4: Operating principle of the parallel savings algorithm

Clarke and Wright (1964) were the ones who first proposed the idea for the savings algorithm (**CW64-PS**), and showed how it can be used to solve vehicle routing problems. The parallel approach starts from an initial state where each customer is served individually by a route. Then, *best feasible merge* is made at each step of the algorithm, until no further feasible merges are in the savings list. The routes are build in parallel and, thus, each merge reduces the number of routes by one. Clarke and Wright (1964) proposed to calculate the savings from merging two routes by connecting customers i and j with:

$$s(i,j) = c_{0i} + c_{0j} - c_{ij}. \tag{1}$$

Here, c_{ij} is the cost of traveling from i to j and the savings value $s(i,j)$ describes the distance saved by merging the two routes by connecting i and j with an edge, given that that i and j are connected to the depot (indicated by 0) with an edge prior to the merge operation.

Gaskell (1967) explored the trade-offs of different savings functions and the order of merges. He proposed two new savings function alternatives together with some experimental results. These are referred to as Gaskell's π and Gaskell's λ criteria or **Ga67-PS**$|\pi/\lambda$. Given $\bar{c_0}$ as the average cost of all edges leaving from the depot, the two criteria can be written out as follows:

$$s_\lambda(i,j) = s_{ij}(\bar{c_0} + |c_{0i} - c_{0j}| - c_{ij}), \tag{2}$$
$$s_\pi(i,j) = c_{0i} + c_{0j} - 2c_{ij}. \tag{3}$$

In his final comments Gaskell (1967) proposes one more savings criteria with parameter θ, but this variant is not included in his experimental study. Later, Yellow (1970) called this a *parameterized savings* function:

$$s_\theta(i,j) = c_{0i} + c_{0j} - \theta c_{ij}. \tag{4}$$

An computational study on the effect route shape parameter θ can be found in (Webb, 1972) and (Golden et al., 1977). Webb (1972) compared savings criteria (1), (2), (3), and (4) with different parameter values, but his results were inconclusive. Later, Paessens (1988) proposed a further generalization of the parameterized savings function by introducing two multipliers g and f. His function was of the form:

$$s_{gf}(i,j) = c_{0i} + c_{0j} - gc_{ij} + f|c_{0i} - c_{0j}|. \tag{5}$$

What makes the savings criterion of Paessens (1988) interesting to our study, is that his algorithm **Pa88-PS|G2P** uses two strategies (M1 and M4) to find suitable values for the parameters g and f. Because parallel savings is computationally cheap algorithm generating a solution eight times, as it is done with the strategy M4, is perfectly feasible.

Yet another alternative for the savings function is the *proximity criterion* (Gaskell, 1967) with the savings function $s_p(i,j) = c_{ij}$. That is, the distance between the two merge candidates is the savings value. Using this criterion is equivalent to a variant of parallel nearest neighbor heuristic, where shortest edges are connected as long as there are no more feasible route merges to be made.

Besides the savings function itself, the ordering of the savings values matters. Usually, the savings algorithm does the merge with largest saving value first. Additionally, a merge is subject to the following restrictions:

1. customers i and j are already not on the same route,
2. routes to be merged have an edge from i and j to the depot, and
3. the merge will not violate capacity or maximum route cost constraints.

If any of these restrictions is violated, the savings is rejected and the next candidate is checked. However, if one wishes to minimize the number of vehicles used, also those merges with a negative savings value can be applied.

As Gaskell (1967) pointed out, it is very possible for two or more merge candidates to have identical savings values. In fact, our experiments revealed that this is quite common. Originally, Clarke and Wright (1964) proposed that the choice between the equally good alternatives is made randomly. Later, Gaskell (1967) proposed a rule based scheme, where the priority is determined "by distance apart". That is, the customers closest to one another are connected whenever a choice between multiple merge alternatives must be made. Later, Hallberg and Kriebel (1979) further validated this idea by showing that this approach ensures better quality solutions for problems where customers are situated in a grid, while not producing worse solutions for routing problems that do not have this structure. As we only wanted to include deterministic algorithms to VeRyPy, we used the rule based approach of Gaskell (1967).

The routes in a savings algorithm are typically build in parallel, that is, the number of routes decreases as the algorithm merges the routes with largest savings.

However, also a sequential version of the algorithm exists. We decided to follow Watson-Gandy and Foulds (1981) and attribute the sequential savings heuristic **We64-SS** to Webb (1964). However, as we were not able to gain access to his original paper, we had to rely on secondary sources for its implementation details (Christofides et al., 1979; Van Breedam, 1994, 2002). Similarly, we had to use the experimental study of (Gaskell, 1967) to gain access to the reference results for the sequential savings heuristic.

Sequential savings algorithm works via 'route extension', where a single emerging route is built at the time (see Figure 5). The advantage of the approach is that it is not necessary to initially generate all n^2 savings values which makes it fast, albeit less accurate (Laporte and Semet, 2002). Another disadvantage of the sequential savings approach is that the emerging routes need to be initialized with a seed customer, which is another delicate choice that has a great impact on the resulting quality of the solutions.

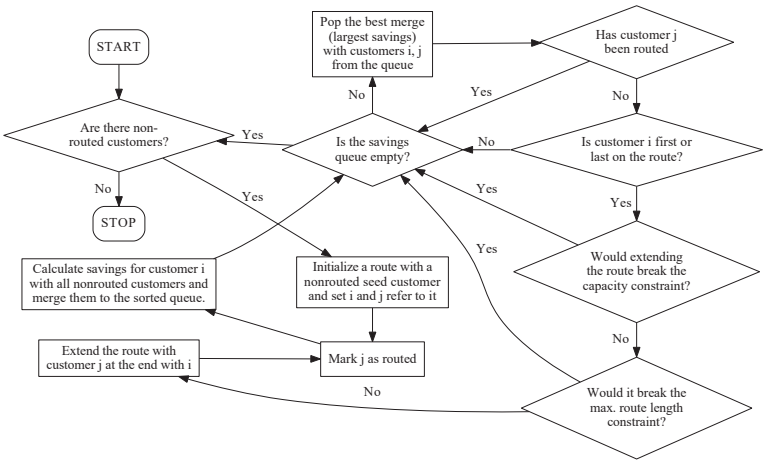


Figure 5: Operating principle of the sequential savings algorithm

Regarding the VeRyPy implementation of the savings algorithm, computing the savings values for the parallel version is an $\mathcal{O}(n^2)$ operation. VeRyPy is intended for solving at most 1000 customer problems, so storing the savings value in the main memory is not a problem. To sort it, we can use the Python's builtin list sorting algorithm. When applying the potential merges, the list is just iterated through, which makes it computationally cheap as the list is not modified in any way. If at some point larger problem instances with tens of thousands of customers needs to be solved with VeRyPy, the iterative computation method of Paessens (1988) or some other acceleration technique (see, e.g., Yellow, 1970; Golden et al., 1977; Nelson et al., 1985) can be used to enhance the computational efficiency of the savings implementation.

We now move on to reporting the replication results comparing the quality of the solutions between the implementations of VeRyPy and those published in their original papers. Laporte and Semet (2002) has observed that "there is a great variability in the numerical results for the saving heuristics". For example, Webb (1972) reported that he was unable to replicate Gaskell's results. His work concerns the sensitivity analysis on the parametrized savings function parameters, but he

discounts the discrepancies to erratic rounding errors and differences in manual calculations.

One of the earliest and highly cited savings publications is the one from Gaskell (1967), where he proposes three new savings criteria and experimentally compares two of these against the original one from Clarke and Wright (1964). Also, the sequential route building variant of the savings heuristic of Webb (1964) is included in the comparison, even if it is not described in detail. Our replications of the (Gaskell, 1967) results are shown in Table 2 and Table 3. As one can see, the accuracy of the original Clarke and Wright (1964) is replicated almost perfectly and the average quality of the solutions of our Gaskell's π and λ implementations is within our replication target of 0.1 %. The variant with savings function s_λ has a slightly higher standard deviation, but the overall level in the quality of the solutions is close enough on both to their targets.

The emerging route initialization method for the sequential savings heuristic was not described in (Gaskell, 1967), but initializing it with the farthest non-routed customer from the depot seems to give similar results to those reported in the paper. For this target, the average quality of the solutions is within 0.2 % of the reported results, but there is more variation between the instances (see Table 2). Our experimentation with different route initialization methods shows that the sequential variant is very sensitive to the choice of route seeds. The replication level of the sequential savings algorithm is adequate given that no description or implementation details are given for the algorithm in (Gaskell, 1967) and that we had to rely on secondary sources for its implementation.

Table 2: Replicated results of parallel and sequential savings algorithms with Clarke and Wright (1964) (CW64) criteria, as reported by Gaskell (1967) (Ga67). $*$ = our solution uses one additional vehicle.

Problem				CW64-PS			We64-SS		
no.	source	size	type	f_{ref}	f	gap(%)	f_{ref}	f	gap(%)
1	Ga67	36	I^{CD}	923	923	0.00	947	947	0.00
2	CW64	31	I^{C}	1427	1427	0.00	1427	1434	0.49
3	Ga67	32	I^{CD}	839	839	0.00	850	883	3.88
4	Ga67	21	I^{CD}	598	598	0.00	648	640	-1.23
5	Ga67	29	I^{CD}	963	963*	0.00*	1017	982	-3.44
6	Ga67	22	I^{CD}	955	958	0.31	949	963	1.48
			average			0.05			0.20
			st.dev.			(0.12)			(2.26)

There would be other possible targets for savings related replication experiments like the parallel and sequential savings algorithms with 3-opt post-optimization step with best and first accept in Laporte and Semet (2002), but we decided to include a comparison to the results published in (Paessens, 1988). As one can see, the results produced by our implementation (Table 4) are very similar. When comparing the results to those of Paessens (1988) please note that we have not included the allowance (modeled as a service time) to the result values on the (Gaskell, 1967) instances to use the same convention in all results tables.

We have also included the unmodified parallel savings algorithm results, because

Table 3: Replication results with Gaskell (1967) savings criteria.

Problem				Ga67-PS|λ			Ga67-PS|π		
no.	source	size	type	f_{ref}	f	gap (%)	f_{ref}	f	gap (%)
1	Ga67	36	I^{CD}	913	907	-0.66	857	857	0.00
2	CW64	31	I^{C}	1434	1456	1.53	1500	1500	0.00
3	Ga67	32	I^{CD}	821	821	0.00	850	850	0.00
4	Ga67	21	I^{CD}	602	602	0.00	598	598	0.00
5	Ga67	29	I^{CD}	979	960	-1.94	943	944	0.11
6	Ga67	22	I^{CD}	988	1004*	1.62*	1015	1019	0.39
			average			0.09			0.08
			st.dev.			(1.23)			(0.14)

they were given in (Paessens, 1988). However, our parallel savings implementation uses the secondary savings value sort criteria of Gaskell (1967) instead of the random choice used in (Clarke and Wright, 1964). This is the most probable explanation for the slightly better results of our implementation (C&W and C&W+3-opt of Table 4). The quality of the solutions with the M1 parameter search strategy is replicated almost perfectly. The small discrepancies are probably due to the usual differences in floating point accuracy and local search operator order. The results of applying the M4 strategy are also very closely replicated. The two smaller noteworthy differences are the slightly better solutions our implementation finds on the instance C2, and the slightly better solutions of the Paessens (1988) implementation on C10 (both instances from (Christofides et al., 1979)). However, there is one single larger difference on instance G4 (M4 strategy). Paessens (1988) does not give the exact quality values for the solutions prior to applying 3-opt, but, instead, a percentage improvement achieved with improvement stage. We have calculated the target objective values in Table 4 for the solutions prior to the improvement phase based on these improvement percentages. The value 0.3% given for G4/M4 in Table 5 of Paessens (1988) is repeated twice on that very line, which might indicate a misprint. The fact that our 3-optimal solution is very close to the reported on this target, even if the initial solution is allegedly very different, seems to support this hypothesis.

Taken together, our implementation of the Paessens (1988) M1 and M4 strategies replicates the results very close to those published in (Paessens, 1988). If we ignore the single outlier value in the M4 strategy tests (with 3.44 % gap) we are within the 0.1 % target we have set for perfectly replicating the results.

Regarding other extensions to the savings algorithm, in his survey Mole (1979) concludes that some elaborations of the savings method only produce marginal improvements while substantially increasing the computation times. On the other hand, other improvements that make the procedure more computationally effective have become irrelevant (e.g. Yellow, 1970). However, two extensions to the savings approach, namely exploring several of the alternative savings merges (see Knowles, 1967; Tillman and Cochran, 1968; Holmes and Parker, 1976), and augmenting the savings algorithm with a post-optimization step (see, e.g., Robbins and Turner, 1979; Beltrami and Bodin, 1974; McDonald, 1972; Paessens, 1988) stood out in our literature survey. It would be trivial to recreate, for example, the Robbins and Turner (1979) algorithm by adding a 2-opt* post-optimization step to the parallel savings

Table 4: Replicated results of the Paessens (1988) parametrized savings criteria with 3-opt improvement phase (Pa88-PS |G2P) compared to the original Clarke and Wright (1964) savings algorithm.

Problem				CW64-PS			CW64-PS+3-opt			M1			M1+3-opt			M4			M4+3-opt		
no.	name	size	type	f_{ref}	f	gap(%)	f_{ref}	f	gap(%)	f_{ref}	f	gap(%)	f_{ref}	f	gap(%)	f_{ref}	f	gap(%)	f_{ref}	f	gap(%)
1	C1	50	R^C	585	585	0.00	579	579	0.00	564	564	0.00	557	557	0.00	571	571	0.00	570	570	0.00
2	C2	75	R^C	907	900	-0.77	902	888	-1.55	866	870	0.46	861	861	0.00	889	870	-2.14	876	861	-1.71
3	C3	100	R^C	889	887	-0.22	880	879	-0.11	866	866	0.00	858	858	0.00	877	878	0.11	869	869	0.00
4	C4	100	R^C	834	834	0.00	824	824	0.00	826	826	0.00	822	821	-0.12	826	826	0.00	823	821	-0.24
5	C5	100	R^{DC}	876	876	0.00	869	869	0.00	870	870	0.00	870	869	-0.11	872	872	0.00	871	869	-0.23
6	C6	120	R^C	1068	1068	0.00	1047	1046	-0.10	1065	1065	0.00	1046	1046	0.00	1068	1068	0.00	1047	1046	-0.10
7	C7	120	R^{DC}	1593	1592	-0.06	1583	1583	0.00	1584	1591	0.44	1568	1572	0.26	1591	1591	0.00	1580	1581	0.06
8	C8	150	R^C	1140	1133	-0.61	1134	1128	-0.53	1102	1102	0.00	1083	1083	0.00	1112	1111	-0.09	1106	1104	-0.18
9	C9	150	R^{DC}	1288	1288	0.00	1285	1285	0.00	1222	1222	0.00	1221	1221	0.00	1222	1222	0.00	1221	1221	0.00
10	C10	199	R^C	1395	1396	0.07	1387	1387	0.00	1370	1370	0.00	1359	1357	-0.15	1370	1384	1.02	1359	1373	1.03
11	C11	199	R^{DC}	1539	1539	0.00	1522	1522	0.00	1486	1486	0.00	1476	1476	0.00	1515	1515	0.00	1504	1504	0.00
12	G1	21	R^{DC}	389	389	0.00	389	389	0.00	375	375	0.00	375	375	0.00	388	388	0.00	388	388	0.00
13	G2	22	R^{DC}	736	736	0.00	736	736	0.00	736	736	0.00	736	736	0.00	736	736	0.00	736	736	0.00
14	G3	29	R^{DC}	674	674	0.00	672	672	0.00	648	648	0.00	647	647	0.00	648	648	0.00	647	647	0.00
15	G4	32	R^{DC}	521	521	0.00	520	520	0.00	494	494	0.00	492	492	0.00	494	511	3.44	492	494	0.41
16	GJ1	249	R^C	5568	5510	-1.04	5546	5491	-0.99	5380	5417	0.69	5348	5358	0.19	5476	5464	-0.22	5449	5435	-0.26
			average			-0.17			-0.21			0.10			0.00			0.13			-0.08
			st.dev.			(0.32)			(0.43)			(0.21)			(0.10)			(1.04)			(0.52)

C instances from Christofides et al. (1979), G instances from Gaskell (1967), and GJ instance adapted from Gillett and Johnson (1976)

heuristic. However, as outlined above, the advantage of the Paessens (1988) over its competition is the strategy for finding the parameter values for the generalized savings criteria. Of the savings merge search / look-ahead approaches, we were only able to gain access to the paper from Holmes and Parker (1976). Their algorithm **HP76-PS|IMS*** iterates on the parallel savings heuristic with Clarke and Wright (1964) criteria and on each iteration suppresses some of the best merges from the previous iterations.

Table 5: Replicated results of the Holmes and Parker (1976) savings suppression heuristic.

Problem				HP76-PS|IMS		
no.	source	size	type	f_{ref}	f	Gap (%)
1	Hayes (1967)	6	I^C	114	114	0.00
2	Christofides and Eilon (1969)	50	I^C	573	580	1.22
3	Christofides and Eilon (1969)	75	I^C	886	868	-2.03
4	Christofides and Eilon (1969)	100	I^C	876	876	0.00
					average	-0.20
					st.dev.	(1.17)

As can be seen from Table 5, our implementation replicates the average quality of the Holmes and Parker (1976) results to a reasonable degree. We were unable to gain access to two of the three smaller problems, but differences between implementations would be in any case expected to manifest only on the larger instances. Regarding variation in the results, note that Holmes and Parker (1976) did not specify a strategy for the situation where two merges share a savings value, and this is the probable cause for variation in our replication results.

Overall, implementing HP76-PS|IMS* proved that our parallel savings implementation is quite flexible, and a similar approach could be used to implement the other savings extensions (e.g., Knowles, 1967) in the future.

4.2.2 Insertion Heuristics

In 1976, Mole and Jameson proposed a generalization to the savings algorithm. Instead of allowing merging two routes only between the first or the last customers of the routes, the routes are build sequentially one at the time and the non-routed customers can be inserted between any two consecutive nodes on the emerging route. Their motivation was to seek for new solution methods which would require less computation time than the previously proposed extensions to the savings algorithm. This goal leads one to assume that the insertion heuristic **MJ76-SI** is capable of solving large problem instances.

For each insertion, the algorithm needs to decide a) which non-routed customer to insert and b) where to insert it. These decisions are made according to two criteria. The first calculates the increase of *strain* in inserting the customer u between consecutive nodes i and j on the emerging route:

$$s_1(i,u,j) = c_{iu} + c_{uj} - \mu c_{ij}. \tag{6}$$

The insertion must be feasible, i.e., it should not violate the capacity or maximum route constraints. The second criteria considers the distance to the depot and is defined using the first criteria:

$$\begin{aligned} s_2(i,u,j) &= \lambda c_{0u} - s_1(i,u,j), \\ s_2(i,u,j) &= \lambda c_{0u} - c_{iu} - c_{uj} + \mu c_{ij}. \end{aligned} \tag{7}$$

Also, for what seems to be for performance reasons, Mole and Jameson (1976) proposed determining the insertion specifics in two steps. First, s_2 is used to select the next customer u^* to insert. Then, s_1 determines the nodes i^* and j^*, and the customer is inserted between these nodes:

$$\begin{aligned} u^* &= \operatorname*{argmax}_u s_2(i,u,j), \\ i^*, j^* &= \operatorname*{argmin}_{i,j} s_2(i,u^*,j). \end{aligned}$$

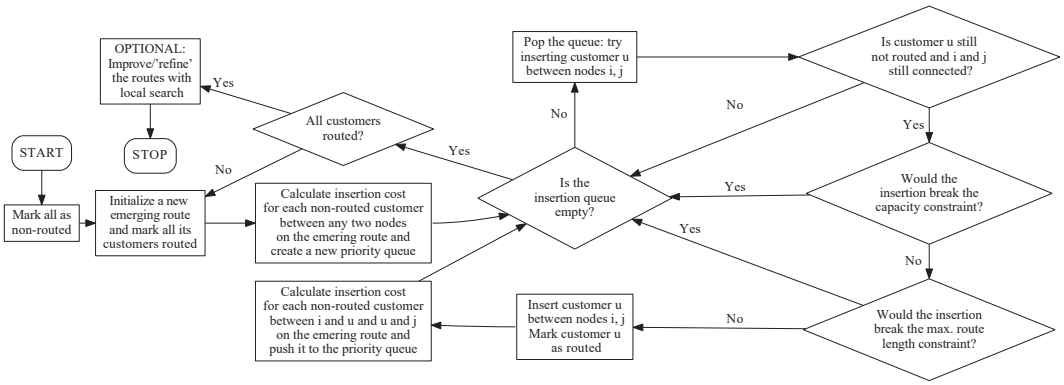


Figure 6: Operating principle of the sequential insertion heuristic

We decided to use a more straightforward approach: because the term λc_{0u^*} is the same for all considered insertion positions of u^* in the emerging route, we can find the customer to be inserted and its position in one step by getting the largest savings value s_2 over all feasible combinations of (i,u,j).

To avoid recalculating the savings values, we store the s_2 values with the insertion details in a priority queue, as implemented in the Python module `heapq`. The binary tree structure makes pushing and popping from the heap computationally inexpensive. Both operations are used extensively; whenever the emerging route grows the insertion queue is updated accordingly. The route may have changed since the insertion was pushed to the heap, which (as can be seen from Figure 6) means that following conditions are checked after an insertion candidate is popped from the heap:

1. u^* has not already been inserted,
2. there is still an edge between i^* and j^*, and
3. the insertion will not violate constraints.

Route initialization is an important detail that can have large effect on the resulting quality of the solutions. In the algorithm of Mole and Jameson (1976), the emerging route initialization depends on the values of parameters λ and μ: if $(\lambda-1) \neq \mu$, the emerging route is initialized by a farthest non-routed customer from the depot. However, if $(\lambda - 1) = \mu$, then the emerging routes are initialized with the customers p and q that have the largest savings criteria value $s_2(0, p, q)$. This corresponds to the savings function:

$$s'(p, q) = \mu c_{0p} - c_{pq} + \mu c_{0q}.$$

Similarly to the original savings algorithm, it is possible (and required to avoid a corner case of leaving one customer unrouted) that $p = q$. Also, because two pairs can easily have the same savings value, a secondary criteria needs to be used to make the choice deterministic.

The results of Mole and Jameson (1976, Table 3) suggest that the heuristic is very sensitive to the order in which the insertions are made. Furthermore, our experiments showed that strain value collisions are very common (i.e., the value given by Equation (7) is the same for two or more possible insertions). The method of resolving these situations has a great effect on the quality of the resulting solution. Unfortunately, Mole and Jameson (1976) do not specify in which order the customers are inserted if the stress values are equal. Through experimentation, we decided to use the Equation (6) values as the secondary insertion criteria, and because the collisions still were occasional, we calculated a savings value for each insertion (that is, how much worse the solution would be if the customer would be served individually) and use that as tertiary sorting criteria. This allowed us to replicate the results reported by Mole and Jameson (1976) to some extent. While there is a lot of individual variation in the results on a instance-to-instance basis, Table 6 shows that, in aggregate, our results are 0.1 % better than those reported in (Mole and Jameson, 1976). The differences are most likely due to the different insertion order, but verifying this is not possible because the tie breaking method used in the original work of Mole and Jameson (1976) was not documented. The fact that sometimes the difference between our solution and the one reported in Mole and Jameson (1976) can be as high as 8% (assuming this is not due to a misprint) suggests that the choices of individual insertions a great impact on the quality of the final solution with this heuristic. Hence, using a single stress function will cause inconsistent accuracy, but that the method stabilizes if all proposed stress functions are considered.

Another source for replication discrepancies may be in the use of local search. The algorithm keeps the route 2-optimal, which means that the 2-optimality is checked after each insertion. If the route needs to be changed to make it 2-optimal, then the insertion values are updated accordingly. After all customers have been routed, there is a post-optimization refinement phase, where the routes created by the insertion heuristic are improved using three local search operators: one-point-move, 2-opt, and redistribute. The redistribution is attempted only on the route with smallest total demand. As explained in Section 2, the result of local search is dependent on the order of the operators and on the number of combinations that are tried. The description of the refinement procedure in Mole and Jameson (1976) is vague. For example, the redistribution heuristic can try different permutations of

Table 6: Replicated results of the Mole and Jameson (1976) insertion heuristic (MJ76-INS).

Problem				Proximity			Min. strain			C&W			Parameterized*			Parameterized**			Between best
no.	source	size	type	f_{ref}	f	gap(%)	f_{ref}	f	gap(%)	f_{ref}	f	gap(%)	f_{ref}	f	gap(%)	f_{ref}	f	gap(%)	gap_{bb}(%)
1	Ha67	6	CI	119	119	0.00	121	119	-1.65	119	119	0.00	119	119	0.00	119	119	0.00	0.00
2	DR59	12	CI	290	290	0.00	294	294	0.00	290	290	0.00	290	290	0.00	290	290	0.00	0.00
3	Ga67#4	21	I^{CD}	694	668	-3.75	633	633	0.00	642	597	-7.01	598	597	-0.17	594	593	-0.17	-0.17
4	Ga67#6	22	I^{CD}	994	1012	1.81	994	994	0.00	979	958	-2.15	949	958	0.95	949	949	0.00	0.00
5	Ga67#5	29	I^{CD}	937	936	-0.11	951	953	0.21	923	981	6.28	890	922	3.60	891	938	5.27	3.60
6	CW64	31	CI	1422	1424	0.14	1465	1512	3.21	1422	1425	0.21	1422	1415	-0.49	1422	1425	0.21	-0.49
7	Ga67#3	32	I^{CD}	836	901	7.78	856	854	-0.23	834	833	-0.12	812	833	2.59	834	833	-0.12	2.59
8	CE69#8	50	CI	672	651	-3.12	603	597	-1.00	597	579	-3.02	590	555	-5.93	575	573	-0.35	-3.48
9	CE69#9	75	CI	1065	976	-8.36	1017	994	-2.26	934	912	-2.36	910	910	0.00	919	898	-2.29	-1.32
10	CE69#10	100	I^{C}	948	869	-8.33	882	914	3.63	955	955	0.00	926	930	0.43	882	881	-0.11	-1.47
			average			-1.39			0.19			-0.82			0.10			0.25	-0.08
			st.dev.			(4.55)			(1.79)			(3.18)			(2.37)			(1.81)	(1.89)

Proximity = proximity ranking insertion criteria ($\lambda = 0, \mu = 0$)
Min. strain = minimum strain insertion criteria ($\lambda = 0, \mu = 1$)
C&W = Insertion with Clarke and Wright savings criteria ($\lambda = 2, \mu = 1$)
Parameterized * = augmented/parametrized Gaskell savings criteria (best of $\lambda = 1.25, 1.5, 1.75, 2.0$; $\mu = \lambda - 1.0$)
Parameterized ** = augmented/parametrized minimum strain criteria (best of $\lambda = 0.0, 0.5, 1.0, 1.5, 2.0$; $\mu = 1.0$)

customers and routes to find an insertion combination that allows inserting all of the redistributed customers, but it is not specified if a greedy or more exhaustive strategy is used.

As some of the results are quite far from identical to those reported in Mole and Jameson (1976), we cannot verify that our implementation of the algorithms is in all parts equivalent to theirs. The operating principle and overall accuracy level are similar, but we did not meet our replication target for all of the stress functions because even after extensive experimentation we still could not determine for certain which secondary insertion criteria (if any) Mole and Jameson used. However, if the accuracy is aggregated over all of the stress functions, our implementation produces a slightly better average quality for the solutions than the original implementation of Mole and Jameson (1976).

4.2.3 Nearest Neighbor Heuristics

The nearest neighbor heuristics are greedy heuristics that build the routes by inserting one of the nearest neighbors of the previously added customers on the emerging route. The algorithm was one of the first algorithms proposed for solving TSP and can be adapted to VRP by starting a new route from the depot whenever a capacity or route length constraints are violated. Reflecting upon the attributes for heuristics in Cordeau et al. (2002), the nearest neighbor approach scores high in simplicity, as the implementation is rather trivial, and also in speed, as they are typically very fast. Furthermore, the flexibility of the algorithm is good as it is easy to extend it given that the problem is not too constrained. Incorporating alternative objectives is more difficult and, for example, we are not aware of a deterministic variant that is capable of minimizing the number of vehicles used. Also, in the case of VRP, the heuristic has a tendency to produce poor quality routes where the return trip to the depot from the last customer on the route is unnecessarily long. We were unable to gain access to the report from Klincewicz (1975) where he presented a computational study on nearest neighbor heuristics in VRP, but according to (Golden et al., 1977) Klincewicz's implementation of the Tyagi nearest neighbor algorithm produced solutions with a poor quality. Despite this, and for the sake of completeness, a nearest neighbor algorithm **Ty68-SNN** from Tyagi (1968), along with two other nearest neighbor variants similar to those presented in (Van Breedam, 1994, 2002)), have been included in VeRyPy.

The nearest neighbor heuristic presented in (Tyagi, 1968) can be considered to be a cluster-first, route-second algorithm, but here we categorize it as a constructive heuristic with a post-optimization phase. The algorithm works as follows: The customers are first divided into groups using a greedy nearest neighbor heuristic, where the algorithm initializes each group G_i with a ("first customer"), and then proceeds to add the nearest neighbor to the previously added customer until the capacity constraint C is violated. This far the Tyagi method has followed the generic approach of the nearest neighbor heuristic illustrated in Figure 7.

However, here the algorithm diverges from the this simple idea. This is due to the algorithm prioritizing minimizing the number of vehicles by interchanging (similar to the local search two-point-move operator) the customer that was first or last added to G_i with the customer whose inclusion to G_i would have violated the constraint,

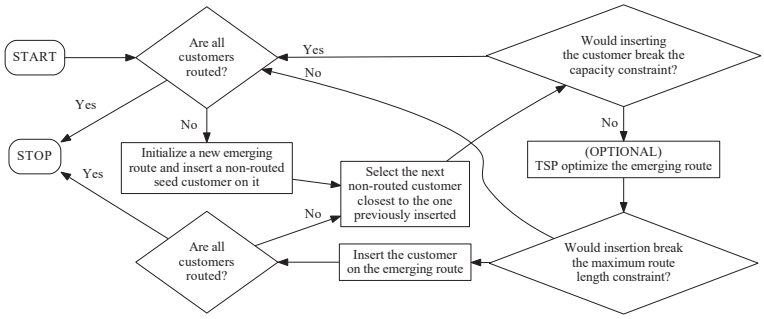


Figure 7: Operating principle of the sequential nearest neighbor heuristic

but only if the interchange brings the route demand closer to the capacity limit C. After the interchange is accepted or rejected, a new group is initialized and the process repeated until all customers have been assigned into one of the groups. In a case there is a group with only a single customer, the grouping of the first iteration is discarded and the best single customer is selected from a group of special *single route candidates* in such a way that the utilization of vehicles is maximized. This means running as many iterations of the algorithm as there are these single route candidates. After the clustering phase, the groups $G_i \forall i \in \{1 \dots K\}$ are routed as a symmetric TSPs. Tyagi (1968) describes a heuristic for this task, but in our implementation we opted to use the LKH implementation (Helsgaun, 2000, 2009) of the more established Lin-Kernighan algorithm for TSP routing (Lin and Kernighan, 1973).

The inaccuracies in the description of the Tyagi (1968) algorithm suggests that it has not been applied manually and not coded to a computer. Studying the provided illustrative example carefully suggests that Tyagi (1968) uses different undocumented variant of his heuristic to solve this 12 customer problem from (Dantzig and Ramser, 1959). The discrepancies are listed below:

1. The first inconsistency is that even though the customer 5 seems to be chosen to be served individually with a separate route, it does not fulfill the criteria for the single customer as specified in (Tyagi, 1968, p. 80).

2. Considering the group G_2 and looking at the distance matrix presented in Table 1 (Tyagi, 1968, p. 80), the customer 7 would be the nearest to the customer 6 which is the first customer. However, it is omitted and customers 8 and 9 are included instead. The interchange operator does not help to explain the discrepancy, as one gets the impression from (Tyagi, 1968, p. 80) that only the first (beginning) or the last (end) point of a group G_i can be interchanged.

3. Finally, the presented grouping is not the one that "makes the sum of the deliveries more near to C". The customer 5 is not the one with the biggest or smallest demand and we see no reason for the algorithm, as presented in (Tyagi, 1968, p. 80), to select the alternative where customer 5 is served by an individual route.

In order to implement the algorithm, some details, like how to maximize the utilization of the vehicles in the last point of the list above, are missing and need

to be addressed. The choice of using the first customer to initialize a group seems arbitrary (and it is not explicitly specified in the description of the algorithm). Also, it has not been specified what to do if a single customer group is part of the grouping but there are no customers that fulfill the criteria for customers that can be served individually with a single route. Therefore, to replicate the grouping of the customers in the illustrative example of Tyagi (1968, p. 86), we needed to slightly modify the algorithm, which leads to the following changes and options in our implementation:

1. Allow *any* customer to be served individually if the algorithm produces such a solution after the first iteration instead of allowing only the customers fulfilling certain criteria. This is required for the customer 5 to be served individually in the illustrative example.

2. Allow *any* customer in a group to be interchanged with the nearest neighbor of the last added customer. This allows grouping customer 9 instead of customer 7 in the illustrative example group G_2.

3. Do not select the grouping with largest total demand in non-single customer routes, but the one that has smallest penalty from traveling with unused capacity. The penalty is the sum of distance traveled times unused capacity. We assumed from the description that this penalty can be expressed as an equation

$$\sum_{r \in 1..K} (C - d_r)c_r, \tag{8}$$

where c_r is the total cost (length) of the route r, and d_r is total demand. The description of the penalty calculation is not very clear (Tyagi, 1968, p. 79), but our interpretation allows our implementation to make the same choice that is done in the illustrative example: that is, the penalty selects the customer number 5 to be served individually.

Table 7: Replicated results of the Tyagi (1968) sequential nearest neighbor algorithm (Ty68-SNN).

problem		original			modified		
source	size	f_{ref}	f	gap (%)	f_{ref}	f	gap (%)
Dantzig and Ramser (1959)	12	290	342	17.9	290	290	0.0

The grouping of the customers in the illustrative example is replicated with the aforementioned modifications. Furthermore, the modifications allowed us to replicate the results and the corresponding quality of the solution (see Table 7). Unfortunately, it is impossible to reliably judge if our proposed modifications are similar to those used by Tyagi (1968). Therefore, as shown also in Table 7, our implementation allows running the algorithm as described in the paper (that is, without the modifications).

Note that VeRyPy implements sequential (vB94-SNN) and parallel (vB94-PNN) versions of the nearest neighbor heuristic. These lack the route improvement procedures of the Tyagi algorithm, and instead roughly follow the descriptions in

(Van Breedam, 1994, 2002). Unfortunately, we were unable to find experimental results on the performance of these algorithms. Therefore, we were unable to do replication examination on these nearest neighbor variants.

4.3 Cluster-First, Route-Second Heuristics

The cluster-first, route-second (CFRS) algorithms work in two phases. First they group the customers to clusters and then apply a TSP algorithm to route the clustered customers. To ensure that the solution is feasible, it might be necessary to repeat the process. Alternatively, the clustering may be used to ensure that it is possible to do a feasible routing.

The approach used in the two-phase algorithm of Christofides et al. (1979) is hard to classify. It shares similarities to the savings approach, but was discussed with insertion heuristics in (Cordeau et al., 2007), even though it does not actually calculate insertion costs. It works by associating customers to seed customers and then optimizes the emerging routes. Therefore, we chose to describe it here under the CFRS algorithms.

Both the nearest neighbor Tyagi (1968) and Fisher and Jaikumar (1981) can be considered to be CFRS algorithms (although, in this study they are discussed under different categories), but perhaps in it's purest form the approach exists in the family of sweep algorithms. The idea behind the algorithm was proposed in (Wren and Carr, 1971; Wren and Holliday, 1972), but also in Gillett and Miller (1974) who gave the sweep its name.

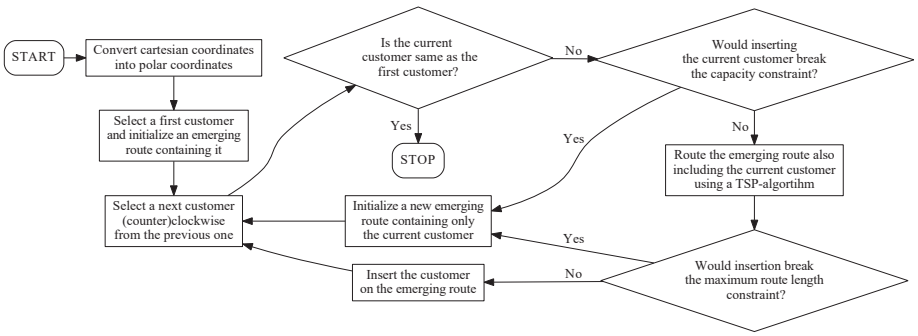


Figure 8: Operating principle of the basic sweep approach

The basic operating principle (Figure 8) of the sweep algorithm is simple: The algorithm assumes that the coordinates of the customers and the depot are known, the distances of the VRP are symmetric, and, furthermore, that the points are located on a plane. The Catresian coordinates of these customers in relation to the depot are converted to polar coordinates (ρ, ϕ), and subsequently sorted by their ϕ values. Starting from an arbitrary customer a new emering route is initialized and then, going to the clockwise or counterclockwise direction, the next adjacent non-routed customers are inserted until a constraint is violated. When this happens, the procedure repeated from the next customer until all customers are routed. Usually, the routes are then optimized using a TSP algorithm.

We would like to point out some practical optimizations to the simple procedure presented in Figure 8. Computing the TSP repeatedly, even with warm starting, is

computationally inefficient. Instead, if there is no maximum route length constraint, computing the optimal TSP tour for the route can be done after all customers have been assigned to routes. For the same reason, computing an upper bound for the TSP solution and updating the exact route length only when the upper bound violates the constraint is beneficial. Depending on how tight the constraints are compared to one another, the TSP tour can also be computed after the capacity constraint is violated. Then, the sweep is backtracked (Gillett and Miller, 1974) and the customers removed from the emerging route until the maximum route length constraint is no longer violated.

The basic sweep algorithm is simple to implement, but loses to the savings approach both in terms of accuracy and speed. Also, it depends heavily on the assumption that constraints can be satisfied by using the planar structure (Cordeau et al., 2002). This had to be into account when implementing the two different sweep variants. The algorithms are described in detail below. First is the algorithm from Wren and Holliday (1972) and the second from Gillett and Miller (1974). The main differences between the algorithms are in the number of sweeps and in the route improvement procedures.

4.3.1 Wren and Holliday Sweep Heuristic

In the Wren and Holliday (1972) algorithm **WH72-SwLS**, the sweeps are made from four evenly spaced seed customers. Out of the generated four initial solutions, the best is selected to be improved in the next phase.

The improvement procedure involves applying several refining processes consecutively and iteratively until no further improvements can be made. In the modern heuristics literature, the refining processes are known as the ocal search operators. The iterative approach proposed in Wren and Holliday (1972) is very similar to the variable neighbourhood descent (VND) metaheuristic (Hansen and Mladenović, 1999). More specifically, the improvement phase consists of applying what Wren and Holliday call *inspect, single, pair, complain, delete, combine, and disentangle* heuristics. Most of these heuristics have a modern local search counterpart but with a different name: *inspect* is the intra-route 2-opt operator, *single* corresponds to combination of the relocate and one-point move operators, and *pair* is similar as the chain move. For details on these refinement heuristics we refer to (Wren and Holliday, 1972) and Section 2.

The improvement procedure is illustrated in Figure 9 adapted from (Wren and Holliday, 1972). We implemented the necessary local search procedures and the improvement scheme and followed the order of applying the operators as specified in (Wren and Holliday, 1972). However, the exact order in which the search space is explored by the local search operators was not specified in the original paper. Thus, all of the caveats listed in Section 2 apply, and the local optima reached via VeRyPy local search can differ from the one reported in (Wren and Holliday, 1972). For example, our implementation of the *single* heuristic uses `do_relocate_move` and `do_1point_move` operators sequentially, which may cause the improvements to be applied in different order than in the implementation of (Wren and Holliday, 1972).

Replicating the results published (Wren and Holliday, 1972) proved in some parts challenging as can be observed from the replication results in Table 8. Based on our

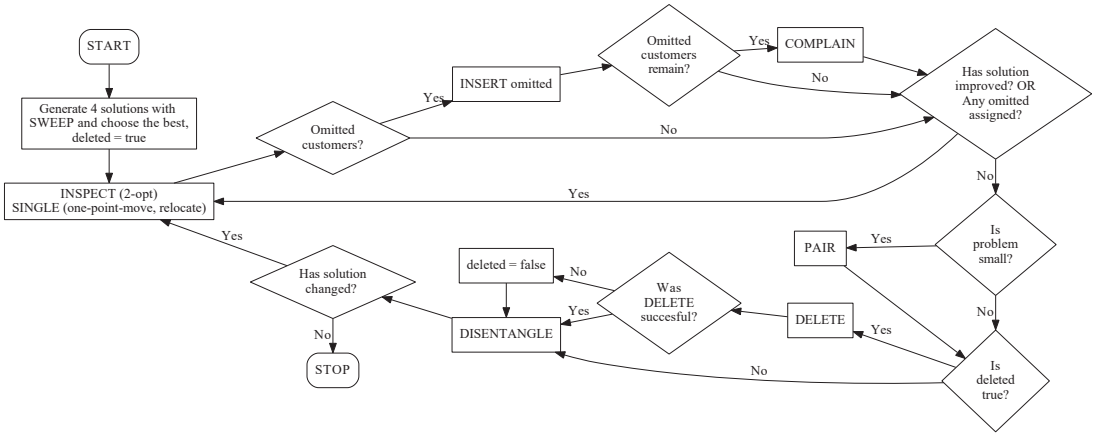


Figure 9: Operating principle of the Wren and Holliday route improving sweep algorithm (Wren and Holliday, 1972)

Table 8: Replicated results of the Wren and Holliday (1972) sweep algorithm (WH72-SwLS).

Problem				Rule based term.			Full convergence			Best sweep position		
no.	source	size	type	f_{ref}	f	Gap (%)	f_{ref}	f	Gap (%)	f_{ref}	f	Gap (%)
1	Ga67	36	I^{CD}	851	846	-0.59	851	846	-0.59	842	838	-0.48
2	CW64	31	I^{C}	1406	1417	0.78	1406	1417	0.78	-	1401	-0.36
3	Ga67	32	I^{CD}	812	822	1.23	812	809	-0.37	-	809	-0.37
4	Ga67	21	I^{CD}	593	609	2.70	593	605	2.02	-	599	1.01
5	Ga67	29	I^{CD}	888	**948**	**6.76**	888	**948**	**6.76**	-	873	-1.69
6	Ga67	22	I^{CD}	964	952	-1.24	954	949	-0.52	-	949	-0.52
7	CE69	50	I^{C}	551	**522**	**-5.26**	551	**522**	**-5.26**	-	**521**	**-5.44**
8	CE69	75	I^{C}	900	863	-4.11	863	863	0.00	-	857	-0.70
9	CE69	100	I^{C}	851	828	-2.70	851	828	-2.70	-	825	-3.06
			average			-0.27			0.01			-1.29
			st.dev.			(3.47)			(3.09)			(1.80)

Ga67 = Gaskell (1967), CW64 = Clarke and Wright (1964),
CE69 = Christofides and Eilon (1969)
Results with a z-score over 1.0 are in bold typeface.

experiments, the quality of the solutions with the basic sweep approach is highly dependent on the customer where the sweep is initiated. Furthermore, depending on how the Cartesian coordinates are interpreted and how the conversion to Polar coordinate system is made, the direction of increasing angular coordinates may be clockwise or counterclockwise. Looking from the depot, Wren and Holliday (1972) suggest starting from the direction where the customers are most sparse. However, they do not give formal definition on how this, or the other four seed customers for the initial sweep solutions, are chosen. The description (Wren and Holliday, 1972, p. 337) leaves some ambiguity how the pole of the polar coordinate system is chosen, to which direction the sweep is carried out, and to which interval the angular coordinates are limited. Additionally, it is unclear if the list of polar angles is sorted in the descending or ascending order, and if the four starting points are

taken from the list by index or by evenly spaced angles. In our implementation, we determine the most sparse direction by calculating the average x and y coordinates over all customers with the depot as the origin, weighted by their demand. Then, the polar coordinate of this average point in relation to the depot is calculated. The most sparse direction is thus 180 degrees from this direction. The seed customers are determined by segmenting the full circle by 90 degree intervals and selecting the next or previous customer to this dividing ray depending on the sweep direction.

These ambiguities, in addition to those of the local search, create a serious source of result replication errors. The initial solution for the local search after the sweep can be different than the one used in (Wren and Holliday, 1972), and the local search may converge to different local optima. We tried to alleviate this issue to some extent by including sweeps in both directions in our experiments. However, the average accuracy of our implementation is reasonably close to the one of Wren and Holliday (1972), which is a good result considering the many potential sources for the replication issues: the significantly different implementation technique (Fortran 66 vs. Python 2.7), the computational hardware (36-bit IBM 7090 vs. 64-bit Intel Core i5 520M), ambiguities in sweep seed selection, sweep direction, and even in termination criteria. In addition, there are the usual difficulties concerning how the local search was originally applied (see Section 2).

While there is large variation in the replication results from instance to instance, there is very slight difference in the average quality of the solutions for the algorithm when using the rule based convergence. Also, the average quality produced by our implementation at full convergence is extremely close to the that of Wren and Holliday (1972) (at an average gap of 0.01 %). However, for the problem instance number 7 the difference is so large that we cannot rule out the possibility of a different problem instance. If the outliers (cases number 5 and 7) are omitted, the general algorithm accuracy level of our implementation is slightly better than the original (average gap of -0.20 % for the full convergence case) while also significantly lowering the variability of the replicated results.

Taken together, our implementation was unable to fully replicate all of the results, but, omitting outliers, replicates the general solution accuracy level almost perfectly. The rightmost columns of Table 8 shows the results if all initial solutions are explored. Only one such result, namely the cost of 842 for problem number 1 is given in (Wren and Holliday, 1972), and, for the others we use the values given for the full convergence results in (Wren and Holliday, 1972, Tables 1 & 2). As one can see, we are able to reach the same, or better, solutions for all but one instance. Furthermore, note that our local search is able to improve the 36 customer case by 0.48 %. Also, this allows our implantation to find a good solution for the case number 5, which it fails to do with rule based termination and full convergence. This further illustrates how the algorithm's ability to find good solutions is dependent on selecting a correct seed solution, which is addressed only in a high level discussion of Wren and Holliday (1972, p. 337) paper.

4.3.2 Gillett and Miller Sweep Heuristic

In contrast to Wren and Holliday (1972), who had proposed an post-optimization local search step to the sweep approach, Gillett and Miller (1974) had independently

discovered a similar method that works in tandem with the sweep. The operating principle of their algorithm **GM74-SwRI** is illustrated in Figure 10.

There are few critical differences to the basic sweep scheme: whenever adding a next customer on the route would break the capacity constraint, if applicable, an additional feasibility check is made. If the maximum route length constraint is set and violated, the route will be made feasible by removing previously added customers and stepping the sweep back in the order of the polar coordinate angles. The second difference is the improvement scheme (the black octagon of Figure 10). Using a selection criteria, again based on polar coordinates, the algorithm checks if swapping a customer on the emerging route with one or two customers close to the last added customer, and each other, would improve the expected quality of the solution. Here, in addition to the length of the emerging route, also the expected length of the route to be built next is taken into account. During this check, both the emerging and expected routes are optimized with a TSP algorithm. The emerging route must naturally also be feasible after any modifications.

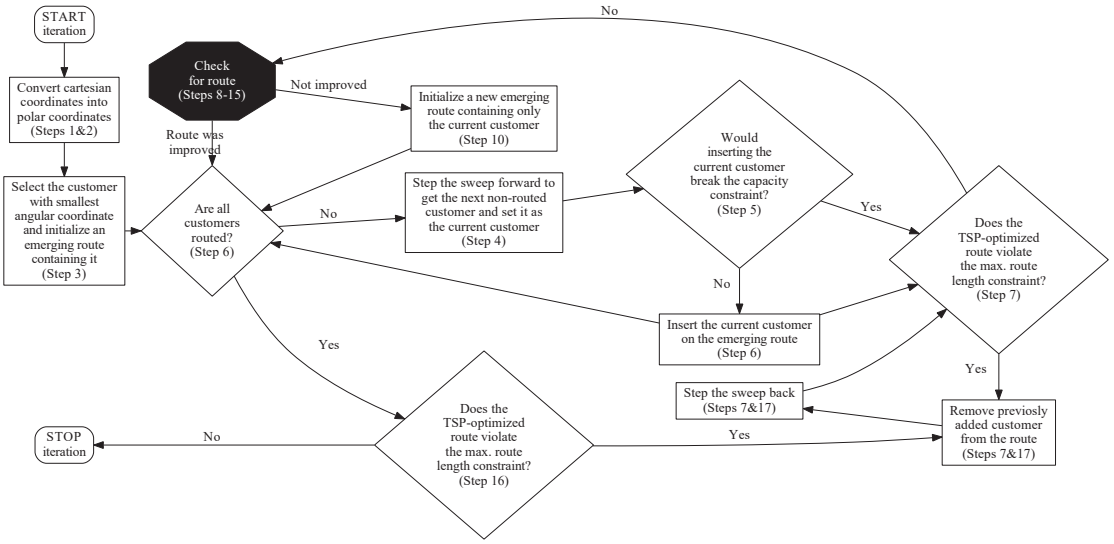


Figure 10: Operating principle of the Gillet and Miller sweep algorithm. The step numbers refer to Appendix A of (Gillett and Miller, 1974).

The algorithm, if implemented as described step-by-step in Appendix A of (Gillett and Miller, 1974), does not seem to have a mechanism to handle the customers that are swapped out during the route improvement phase. The main text is similarly vague: "the replaced location is left unassigned, to be picked up by a later formed route.". However, the index J iterating the sweep in the Appendix A only steps backwards with $J = J - 1$ when the maximum route length constraint violation is lifted, and terminates the sweep on $J = N$. This will potentially leave some customers unassigned. Also, the Steps 11 and 13 of the Gillett and Miller (1974) algorithm require that the insertion candidate customers are situated after the current customer on the sweep. This makes it impossible for the later formed route to "pick up" the swapped and skipped customers. Every execution path that goes through Step 4 advances the sweep leaving the current customer without routing if an improvement to the emerging route was accepted. Thus, the algorithm described

in Appendix A of (Gillett and Miller, 1974) does not necessarily route all of the customers and can produce an infeasible solution.

Unfortunately, also the textual algorithm description (p. 342 Gillett and Miller, 1974) has many ambiguities. From the description, one gets an impression that the improvement heuristic takes over and tries to move customers from one route to another first only after the sweep is completed. However, in the algorithm provided in the appendix of (Gillett and Miller, 1974), the improvement heuristic is run only if adding a customer to the route will not break the capacity constraint and it has been made sure that the route satisfies the maximum tour length constraint. Additionally, not all steps of the improvement procedure given in the appendix are explicitly stated and explained in the main text and vice versa. For example, after the emerging route cannot be further improved, checking for the inclusion of the customer two steps further on the sweep is missing from Appendix A, but explicitly mentioned in the main text.

There is another clear issue that may lead to infeasible solutions. If the Steps 16 and 17 of Appendix A (Gillett and Miller, 1974) are followed as written, last route can remain infeasible due to how the maximum route length constraint is checked and feasibility is proposed to be regained: the Step 17 allows only removing one customer before the route is recorded. It can be expected that regaining feasibility requires removing several previously added customers. We have rectified this issue in our implementation as illustrated in Figure 10 by returning the control flow to the procedure of Step 7 if an infeasibility is detected. This means that Step 17 is not needed and it is not, hence, present in Figure 10.

Regarding smaller issues, the improvement heuristic (Steps 8-15, Appendix A, (Gillett and Miller, 1974)) can include any of the four future sweep customers to the current route. However, in Steps 4-6, where the active customer of the sweep is updated, there is no check if the customer is already routed. To remedy these issues, our implementation follows the structure of Figure 10 by keeping track of the customers that have been routed and a backlog of customers that were not routed during the sweep. In case some customers remain in the backlog after the full sweep, they are handled as if the sweep would still continue with the backlog customers. Also, minimizing the distance is referenced in at least three different ways in the Appendix of (Gillett and Miller, 1974)): "calculate the minimum distance", "evaluate the minimum distance", "determine the distance". It is left unclear if all of these refer to optimizing the route with a TSP algorithm, or something subtly different. For example, is unclear if the route is re-optimized each time a customer is removed from the optimized solution to regain feasibility. On top of this, Gillett and Miller (1974) did not specify the TSP algorithm, but the earlier PhD thesis of Miller (1970) indicates that 3-opt was used.

One of the most blatant problems was in selecting the customer to be removed during the improvement steps. Gillett and Miller (1974) suggest using a function:

$$i_{KII} = \text{argmin}(\rho_i + \phi_i \bar{\phi}), \quad \forall i \in R_e,$$

where R_e is the set of customers on the emerging route, ρ_i the distance of the customer i to the depot, ϕ_i is the connected polar angle, and $\bar{\phi}$ is the average depot distances of all customers. Our major critique is that the ϕ_i angle is not scaled in any way for the route. Thus, in different parts of the sweep the angle value will

be bigger and the significance of the distance term is diminished. Analytically this makes no sense. The other issue with the formula is in that because the function is minimized, and ϕ_i grows as the sweep progresses, the formula prioritizes customers **farthest** from the next route. This is in conflict with the text description (Gillett and Miller, 1974, p. 342), which clearly states that "this provides a location that is close to the depot and also close to the next route". If implemented as given, a customer/location close to the depot, but farthest from the next route gets selected. Our implementation inverts the sign of the ϕ values in the formula for this calculation to remedy this.

Regarding replication of the results, in the Appendix B of Gillett and Miller (1974) only one of the solutions produced by their algorithm is printed. The solution is for Problem 3 with 29 customers, allowance of 10 units, vehicle capacity of 4500 and route mileage limit of 240 originating from Gaskell (1967). We were able to replicate this specific result perfectly. Additional solutions produced by an earlier version of the Gillett and Miller (1974) heuristic can be found from the PhD dissertation of Miller (1970). However, if contains further errors. For example, some of the problem instance descriptions have misprints and some of the given solutions are infeasible. As the solutions and problems were not reprinted in (Gillett and Miller, 1974), it is impossible to tell if the same implementation was used or if the feasibility of the resulting solutions was checked at all. Some of the best solutions in Miller (1970) that are feasible, such as the 22 and 23 customer problems originating from Gaskell, seem to be impossible to replicate using the procedure described in (Gillett and Miller, 1974).

Table 9: Replicated results of the Gillett and Miller (1974) sweep algorithm.

Problem				GM74-SwRI		
no.	source	size	type	f_{ref}	f	Gap (%)
1	Ga67	21	R^{DC}	591	**608**	**2.88**
2	Ga67	22	R^{DC}	956	968	1.26
3	Ga67	29	R^{DC}	875	875	0.00
4	Ga67	32	R^{DC}	810	810	0.00
5	CE69	50	R^{C}	546	**532**	**-2.56**
6	GM74 (CE69)	75	R^{C}	1127	1135	0.71
7	Ga67	75	R^{C}	865	884	2.20
8	GM74 (CE69)	75	R^{C}	754	757	0.40
9	GM74 (CE69)	75	R^{C}	715	706	-1.26
10	GM74 (CE69)	100	R^{C}	1170	1183	1.11
11	CE69	100	R^{C}	862	849	-1.51
12	Mi70	249	R^{DC}	5794	5830	0.62
					average	0.32
					st.dev.	(1.48)

Ga67 = Gaskell (1967), CE69 = Christofides and Eilon (1969), GM74 (CE69) = Gillett and Miller (1974) instances modified from CE69 instances, Mi70 = Miller (1970)
Outliers (z-score > 1.5) are in bold typeface.

Despite the basic sweep approach being very simple, the extension of Gillett and Miller (1974) was one of the trickiest algorithms in VeRyPy to implement because

many implementation details had to be fixed trough trial-and-error to get performance reasonably close to those reported in (Gillett and Miller, 1974), while still keeping the implementation true to its original description.

Taking the issues outlined above into account, it is not surprising that we were unable to fully replicate the reported results. However, we were eventually able to reach generally the same level of accuracy. Our implementation was, on average, 0.32 % worse on the 12 problem instances included in the computational results of Gillett and Miller (1974) (see Table 9). Additionally, there is quite a lot of variation from instance to instance as indicated by the standard deviation of 1.48 percentage points. As noted above, the algorithm description and its computational details have many ambiguities, and it is very hard to reliably estimate where the replication issues originate.

As a final note on the Gillett and Miller (1974) algorithm, we would like to point out that there is an alternative way of implementing the algorithm, which would apply the basic sweep approach first, followed by a local search post-optimization step with one-point move and two-point move. This implementation would be a relatively close approximation of the basic idea of the algorithm. However, as our intention was to implement the algorithms as close to their original descriptions as possible, we opted to follow the improvement scheme described in the Appendix A of Gillett and Miller (1974) with the difficulties and results presented above.

4.3.3 Christofides, Mingozzi, and Toth 2-Phase Heuristic

Besides introducing the popular CMT benchmark set with 14 problem instances for CVRP, the book chapter (Christofides et al., 1979) also proposes two heuristic algorithms for solving those problems. First one is a heuristic tree based search method, which, due to our requirements for speed and determinism, was not implemented. However, the second algorithm is a novel two-phase algorithm **CMT79-2P**, which uses two different phases with different criteria to associate customers to routes with seed customers. The operating principle of the algorithm is illustrated in Figure 11.

In the first phase of the algorithm, a customer i_h is selected in random to initialize a route, and then the association score values $\delta_l = c_{0l} + \lambda c_{li_h}$ are calculated for all non-routed customers. Here, λ is a parameter for specifying how closely the customer is associated to the route seed i_h. Then, from the smallest δ first, all feasible insertions to an emerging route are made. The emerging route is kept 3-optimal (Lin and Kernighan, 1973) through these insertions. When no feasible insertions remain, a new route is initialized with a new random customer, and the association scheme is repeated until no customers remain.

The K seed customers of the first phase are carried over to the second phase, where K routes, each serving different seed node i_r, are constructed. Then, another set of customer to route association criteria values $\epsilon_{rl} = c_{0l} + \mu c_{li_r} - c_{0i_r}$ are computed for each seed and non-routed customer pair. Using these values, each non-routed customer l is associated with a route with minimal ϵ value $\epsilon_{\bar{r}l}$. In the next step, one route is chosen (represented by the index $\bar{r}$) from the set of routes S. The route is then removed from S and scores $\bar{\delta}_l = \epsilon_{rl} - \epsilon_{\bar{r}l}$ are computed for all customers associated with the route. Here, ϵ_{rl} is the ϵ the customer l was associated with $\bar{r}$, and $\epsilon_{\bar{r}l}$ is the smallest epsilon for l among the remaining routes of S. All feasible

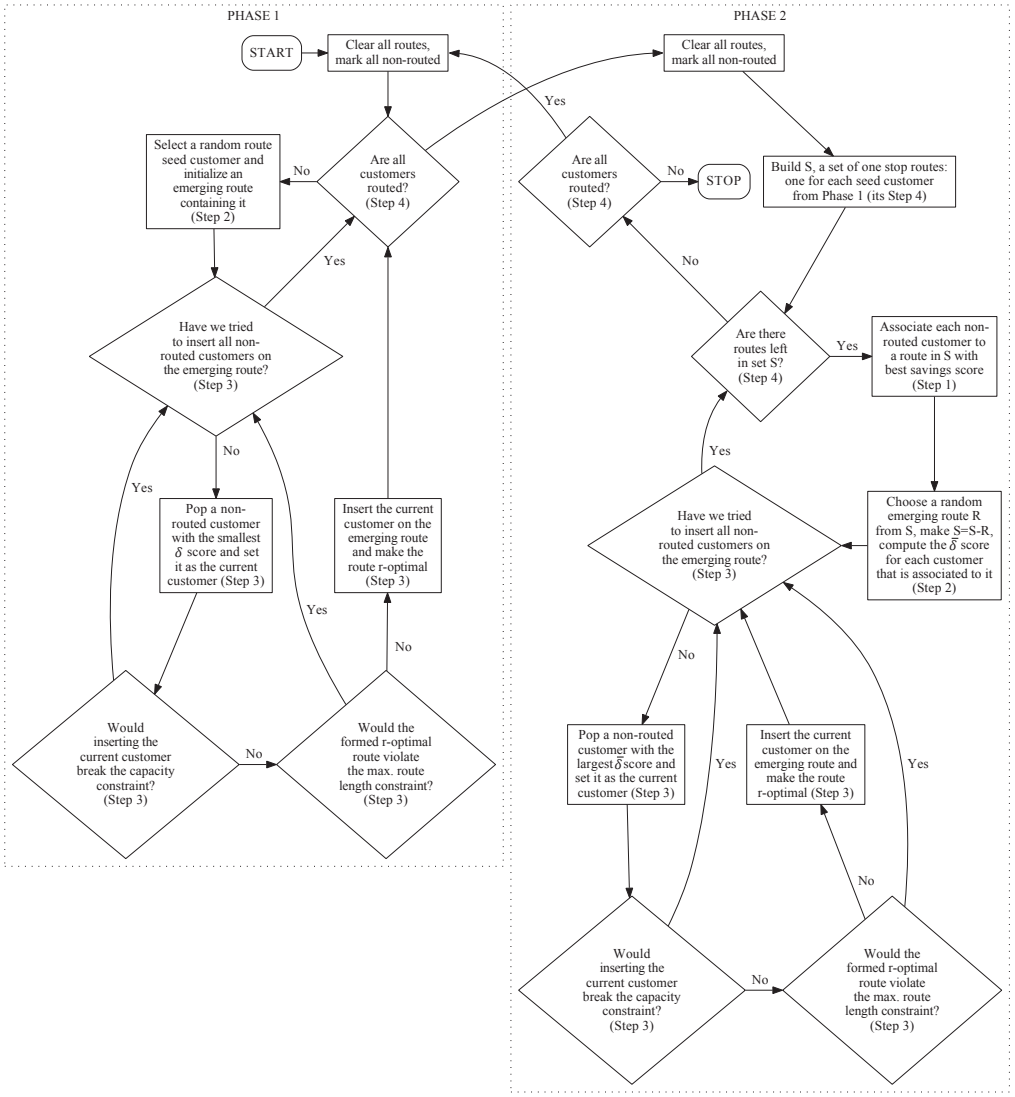


Figure 11: Operating principle of the Christofides et al. (1979) 2-phase heuristic.

insertions to the chosen route among the customer associated to it are made in descending $\bar{\delta}_l$ order. Like in Phase 1, the route is kept 3-optimal throughout. New routes are chosen and filled until the set of routes S is empty. If non-routed customers remain after completing Phase 2, steps of Phase 1 are repeated followed by Phase 2 with different set of (random) seed customers from Phase 1. When the Phase 2 is successfully completed, the algorithm is terminated.

The description of the heuristic is very concise in Christofides et al. (1979), which creates an illusion of simplicity. However, writing an effective and simple implementation of the algorithm proved tricky, mostly due to the two layer customer association scheme of Phase 2. As a result, implementing the algorithm led to a quite complex and verbose Python code.

The description of the heuristic in Christofides et al. (1979) does not cover all special cases. Especially regarding the implementation of the Phase 2 we would like to point out an ambiguity in the algorithm description (Christofides et al., 1979, chapter 11.4.5, Phase 2, Step 2, p. 333): If choosing the route $\bar{R}_{\bar{r}}$ and removing it from the set of routes S leaves no routes in S, which happens every time when the customers associated to the last route are processed, it is undefined how the $\bar{\delta}$ should be computed. Disregarding the calculation of $\bar{\delta}$ is not an option, as this would leave the customers that could be assigned to the last remaining route unrouted. We decided to omit the calculation of ϵ_{rl}, meaning that $\bar{\delta} = -\epsilon_{\bar{r}l}$ when $S = \varnothing$, which fixes the issue.

According to the algorithm description, Step 4 of Phase 2 jumps the heuristic back to the start of Phase 1 if some customers are left unrouted. In our preliminary experiments we repeated the algorithm 100 times with parameters values taken from an uniform distributions $\lambda \in [1.0, 3.0]$ and $\mu \in [0.0, 2.0]$, but it turned out that the Phase 2 was able produce a solution only for 7.1 % of the trials. Furthermore, while Phase 2 solution was better than that of Phase 1 in 98% of these cases, the successful Phase 2 solutions were concentrated only on three of the 14 problem instances. As suggested in (Phase 2, Step 4 Christofides et al., 1979), the probability of successfully completing Phase 2 can be increased by redoing the seed customer generation of Phase 1. However, it is possible, and according to our experiments, even probable, that Phase 2 is not able to produce a solution with *any* seed configuration. Thus, a procedure with a rule requiring a valid Phase 2 solution might get stuck in an infinite loop. To address the issue in our implementation, we set a cap of 10 random retries on the algorithm in case Phase 2 solution is not found. This increases the chance of finding Phase 2 solution with a single run of the algorithm from 7.1% to 28.4%. This way, with 100 repetitions and with different random seeds, at least one matching Phase 2 solution was found for 10 of the 14 problem instances reported in (Christofides et al., 1979).

The ambiguities in the description of the algorithm, its stochastic nature, and the complexities in the implementation made replicating the results of (Christofides et al., 1979, p. 335) Table 1.11 very difficult. Christofides et al. (1979) did not specify how the distance matrix was calculated, but the fact that quality of the solutions is given as integers together with our replicated results seem to indicate that the distances were truncated or rounded to the nearest integer. However, the exact distances are typically used with this problem set and we decided to follow this modern convention.

However, the most problematic omission from the replication point of view is that the values or even ranges for the λ and μ parameters are not given, nor is the number of repetitions that were required to produce the solutions of Table 1.11 (Christofides et al., 1979, p. 335). We assumed that the final result given in the table was best of some, undefined, number of runs.

To summarize, the results depend on:

1. if the distance matrix costs are exact or if they are rounded or truncated;
2. whether the number of vehicles or the total mileage is the primary optimization target;
3. how many times the stochastic algorithm is run;
4. if some statistical descriptor such as best, median or mean is given as the measure of the quality of the solutions; and
5. which parameter λ and μ values are used, if they are tuned per problem instance, and from which random distribution they are drawn from if they are chosen randomly.

Unfortunately, these details are not disclosed in (Christofides et al., 1979) and we had to specify our own experimental design to replicate the results of the algorithm (E in Table 1.11 (Christofides et al., 1979, p. 335)). The algorithm was run 100 times with parameters values drawn from uniform distributions $\lambda \in [1.0, 3.0]$ and $\mu \in [0.0, 2.0]$. If Phase 2 was unable to produce a feasible solution, each run was allowed to try to repeat Phase 1 seed generation followed by Phase 2 attempt at most 10 times.

Our implementation optimizes the emerging route with an intra-route 3-opt operator. To save computational effort, this is done after an insertion only if maximum route length constraint L is specified and the upper bound updated by naively inserting the new customer to the end of the route indicates that the constraint may be violated. If the maximum route length constraint is not set, the routes are made 3-optimal after all customer to route associations have been made.

The replicated results of the stochastic CMT two-phase algorithm as proposed in (Christofides et al., 1979) are presented in Table 10. Average and standard deviation for the quality of the solutions are given for each problem instance. It seems that in 1979 it was customary to report only the best result over several runs (Christofides and Eilon, 1969; Mole and Jameson, 1976), and, therefore, the best of 100 trials is also given in columns f^* and gap*.

As can be seen from Table 10, best results from our implementation seem to be able to replicate the results, save for the instances CMT1, CMT5, and CMT10. The average gap to the reported results without these three instances is 0.15%. And, even with the all 14 instance of the problem set, the stochastic results are within 0.39% of the ones published in (Christofides et al., 1979).

Our replication experiments revealed that the CMT79-2P heuristic is extremely sensitive to the insertion order and λ and μ parameter values, which is reflected by the large variation in the quality of the solutions between the trials (Table 10, the standard deviations given inside parenthesis). Therefore, due to the stochastic

Table 10: Replicated results of the original stochastic Christofides et al. (1979) two-phase algorithm, with 100 repetitions.

Problem		CMT79-2P, stochastic				
no.	size	f_{ref}	$\bar{f}$	$\overline{\text{gap}}$ (%)	f^*	gap* (%)
1	50	547	604.9(45.2)	10.6(8.3)	534	**-2.4**
2	75	883	1079.0(108.1)	22.2(12.2)	883	0.0
3	100	851	988.0(89.5)	16.1(10.5)	849	-0.2
4	150	1093	1368.8(107.3)	25.2(9.8)	1104	1.0
5	199	1418	1741.9(139.2)	22.8(9.8)	1452	2.4
6	50	565	639.1(36.1)	13.1(6.4)	576	1.9
7	75	969	1079.7(49.0)	11.4(5.1)	975	0.6
8	100	915	1019.6(66.6)	11.4(7.3)	915	0.0
9	150	1245	1411.6(131.1)	13.4(10.5)	1255	0.8
10	199	1508	1731.9(149.5)	14.8(9.9)	1566	**3.8**
11	120	1066	1181.2(113.4)	10.8(10.6)	1050	-1.5
12	100	827	929.3(132.4)	12.4(16.0)	829	0.2
13	120	1612	1717.0(147.1)	6.5(9.1)	1583	-1.8
14	100	876	990.8(124.4)	13.1(14.2)	881	0.6
average				14.57		0.39
st.dev.						(1.59)

CMT79 = Christofides et al. (1979)
Outliers (z-score > 1.5) are in bold typeface.

nature of the algorithm, due to its sensitivity to the parameters, and due to missing description of the experimental setup in (Christofides et al., 1979), some uncertainty remains on how close our implementation actually is to the from Christofides et al. (1979). According to Paessens (1988), there has been a previous reimplementation and replication effort from Heins (1981), but unfortunately we were not able to gain access to this paper.

One of the criteria for an algorithms to be included in VeRyPy was that it had to be deterministic. In case of the two-phase algorithm there is stochasticity in two steps: selecting seed customers in Phase 1 and selecting the order in which the routes are built in Phase 2. To build a deterministic version of the Christofides et al. (1979) two-phase algorithm, we propose always selecting the non-routed customer that is farthest from the depot as the route seed point in Phase 1, and selecting the route with *most associated candidate customers* in Phase 2. We remind that from the preliminary experiments we learned that the algorithm's performance is depended on successfully completing the Phase 2. To further increase this chance, we implemented an option in the algorithm to reset the set of routes S and run Steps 2-4 of Phase 2 repeatedly until no insertions are made in case the possibility of changing route associations allow further insertions of the customers on the emerging routes.

With the proposed deterministic approach, the impact of suitable values for the algorithm parameters λ and μ becomes even more pronounced. To configure the parameters of the deterministic version for the 14 problem instances in (Christofides et al., 1979), we used the automatic algorithm configuration tool SMAC (Hutter et al., 2011). Using our previous work (Rasku et al., 2019b) as a guideline, we set the

Table 11: Results of the deterministic Christofides et al. (1979) two-phase algorithm. Outliers (z-score > 1.5) are in bold typeface.

Problem		CMT79-2P, deterministic					
no.	size	f_{ref}	f	gap (%)	λ	μ	$t(s)$
1	50	547	550.5	0.65	1.9	1.3	0.0
2	75	883	893.3	1.17	2.7	0.8	0.0
3	100	851	860.7	1.14	2.0	1.0	0.1
4	150	1093	1089.2	-0.35	2.0	1.0	0.2
5	199	1418	1379.8	**-2.69**	2.0	1.0	0.2
6	50	565	586.4	**3.79**	1.9	1.3	0.2
7	75	969	957.6	-1.18	1.9	1.3	0.2
8	100	915	913.0	-0.22	2.7	0.8	1.2
9	150	1245	1249.5	0.36	2.0	1.0	2.8
10	199	1508	1492.1	-1.05	2.0	1.0	4.2
11	120	1066	1064.3	-0.16	2.0	1.0	0.2
12	100	827	832.3	0.64	2.0	1.0	0.1
13	120	1612	1579.4	-2.02	1.9	1.3	2.3
14	100	876	875.0	-0.11	2.7	0.8	0.5
			average	0.00			
			st.dev.	(1.51)			

evaluation budget to 2000 and ran four parallel configuration tasks. Unfortunately, SMAC was not able to find a single parameter configuration that would provide satisfactorily quality for the solutions for all 14 instances, so we configured the algorithm separately for instances number 12 and 6 which were the two outliers giving poor solutions when using the default parameters $\lambda = 2.0$ and $\mu = 1.0$. With the default values augmented by two sets of configured parameters, we got a similar level of performance as with the original stochastic algorithm of (Christofides et al., 1979) (see Table 11). However, because our proposed variant is deterministic, it uses only a fraction of the computational effort required by the stochastic algorithm to reach similar quality for the solutions.

4.4 Route-First, Cluster-Second Heuristics

The algorithm **Be83-RFCS** from Beasley (1983) relies on a route-first, cluster-second (RFCS) approach. The general idea had been proposed earlier, for example, by Newton and Thomas (1969), but Beasley was the first to test the approach experimentally against the standard CVRP benchmark instances.

During the first routing phase, the Beasley (1983) algorithm generates a 2-optimal TSP tour (Hamiltonian path) that visits all customers. The algorithm omits the depot from this TSP tour to give more flexibility in the second phase of the algorithm, where a required number of visits to the depot are added on the TSP tour using an auxiliary graph. Here, all costs of feasible 2-optimal routes of consecutive customers on the TSP tour are calculated and collected to a cost matrix indexed by the start and the end of the route. Infeasible sequences have their costs set to infinity. Thus, the matrix represents a directed graph of shortcuts from the beginning of routes to their ends. A shortest path around the TSP tour through

these shortcuts can be found using the Floyd-Warshall algorithm and the visits to the depot are added according to this path. Finally, Beasley (1983) make sure the resulting routes are 3-optimal. The general principle of the algorithm is illustrated in Figure 12.

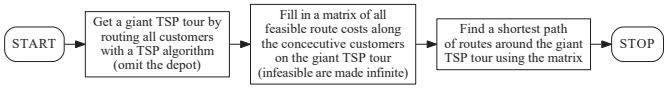


Figure 12: Operating principle of a route-first, cluster-second heuristic.

The algorithm presented in (Beasley, 1983) is stochastic, but due to the lack of RFCS heuristics, and because it was easily converted to be deterministic, an variant of the algorithm was implemented and included into VeRyPy. Instead of multiple randomly generated 2-optimal initial routes, our variant generates a single optimal (or near optimal) TSP solution, which is then partitioned into paths of feasible consecutive routes that satisfy problem constraints such as the capacity constraint or maximum route length constraint. As with (Beasley, 1983), the routes are then individually made 3-optimal and the resulting solution is returned as the solution to the original problem.

Related to the proposed use of separate TSP local search operators for shortcuts and for final routing, our extensive experimentation (see Section 6) revealed a rarely occurring issue in the algorithm, which is also connected to general issues with local search outlined in Section 2. A final 3-optimal route can sometimes, albeit very rarely, be worse than the tour that was kept 2-optimal when building it one customer at the time. This is because the TSP local optima of the final routes depends on the order of moves and the strategy (e.g., first accept or best accept). This may cause a significant problem if a problem has a maximum route cost constraint. Now, the 2-optimal route can be feasible, but there is a possibility that the final 3-optimal route is not. Our proposed solution to fix this was to build new 2-optimal routes also after applying the Floyd-Warshall algorithm. The 3-opt improvement heuristic is then applied on these 2-optimal routes.

For generating the initial TSP tour, our implementation relies on an external TSP solver. This is by default the state-of-the-art TSP solver LKH (Helsgaun, 2000, 2009) with a fixed seed. Because our deterministic implementation cannot improve its performance through the use of randomized restarts, the closest comparison target is the *1 trial* results in (Beasley, 1983) (see Table 13). However, the deterministic variant loses clearly to the stochastic with, for example, 10 repetitions. The results by the stochastic variant of our implementation are given in Table 12.

In stochastic solvers the randomness plays a major part. Unfortunately, Beasley (1983) reported only the best results after 1, 5, 10, and 25 repeated runs without averages nor standard deviations. Thus, it is hard to estimate the distribution on the quality of the solutions from the repeated trials. To replicate the results we did each of the 1, 5, 10, and 25 trial targets 10 times and reported the average and standard deviation. As expected, it is statistically improbable to perfectly match the solutions published in (Beasley, 1983), especially when there are only one or five repetitions, but the accuracy profiles become really close to one another when the number of trials increases. The final average gap is at -0.10% in 25 trial case, which

Table 12: Replicated results of the Beasley (1983) route-first, cluster-second TSP tour partitioning algorithm (Be83-RFCS).

Problem				1 trial			5 trials		
no.	name	size	type	f_{ref1}	$\bar{f}$	$\overline{\text{gap}}$ (%)	f_{ref5}	$\bar{f}$	$\overline{\text{gap}}$ (%)
1	Ha67	7	I^C	114	114.0(0.0)	0.0(0.0)	114	114.0(0.0)	0.0(0.0)
2	DR59	13	I^C	296	293.2(2.9)	-0.9(1.0)	290	291.8(2.3)	0.6(0.8)
3	Ga67	22	R^{DC}	608	595.0(5.3)	-2.1(0.9)	585	588.4(2.5)	0.6(0.4)
4	Ga67	23	R^{DC}	1017	1007.2(12.1)	-1.0(1.2)	994	987.1(16.2)	-0.7(1.6)
5	Ga67	30	R^{DC}	879	876.6(1.9)	-0.3(0.2)	876	875.1(0.3)	-0.1(0.0)
6	CW64	31	I^C	1360	1376.1(68.1)	1.2(5.0)	1298	1359.2(19.0)	4.7(1.5)
7	Ga67	33	R^{DC}	848	827.5(17.0)	-2.4(2.0)	815	816.1(7.7)	0.1(0.9)
8	CE69	51	R^C	564	570.2(8.3)	1.1(1.5)	564	561.2(8.0)	-0.5(1.4)
9	CE69	76	R^C	906	898.5(14.7)	-0.8(1.6)	895	882.3(7.1)	-1.4(0.8)
10	CE69	101	R^C	902	895.5(15.7)	-0.7(1.7)	880	876.2(9.7)	-0.4(1.1)
			average			-0.6			0.3
			st.dev.			(1.1)			(1.67)

Problem				10 trials			25 trials		
no.	name	size	type	f_{ref10}	$\bar{f}$	$\overline{\text{gap}}$ (%)	f_{ref25}	$\bar{f}$	$\overline{\text{gap}}$ (%)
1	Ha67	7	I^C	114	114.0(0.0)	0.0(0.0)	114	114.0(0.0)	0.0(0.0)
2	DR59	13	I^C	290	290.4(1.2)	0.1(0.4)	290	290.0(0.0)	0.0(0.0)
3	Ga67	22	R^{DC}	585	587.6(2.8)	0.4(0.5)	585	585.4(1.2)	0.1(0.2)
4	Ga67	23	R^{DC}	968	975.6(14.9)	0.8(1.5)	956	959.7(11.1)	0.4(1.2)
5	Ga67	30	R^{DC}	875	875.2(0.4)	0.0(0.0)	875	875.0(0.0)	0.0(0.0)
6	CW64	31	I^C	1280	1331.4(33.1)	4.0(2.6)	1280	1299.7(39.9)	1.5(3.1)
7	Ga67	33	R^{DC}	814	813.6(0.8)	-0.0(0.1)	822	813.4(0.9)	-1.0(0.1)
8	CE69	51	R^C	564	553.9(5.4)	-1.8(0.9)	552	551.2(5.7)	-0.1(1.0)
9	CE69	76	R^C	895	878.9(4.6)	-1.8(0.5)	884	878.1(4.8)	-0.7(0.5)
10	CE69	101	R^C	878	872.9(5.5)	-0.6(0.6)	873	863.1(7.5)	-1.1(0.9)
			average			0.1			-0.1
			st.dev.			(1.5)			(0.7)

Ha67 = Hayes (1967), DR59 = Dantzig and Ramser (1959), Ga67 = Gaskell (1967), CW64 = Clarke and Wright (1964), CE69 = Christofides and Eilon (1969)
Exact distances are used, with the quality of the solutions rounded to the nearest integer.

Table 13: Results of deterministic variant of Beasley (1983) route-first, cluster-second TSP tour partitioning algorithm. Same problem instances and rounding as with the stochastic variant in Table 12

Problem				Be83-RFCS (deterministic)				
no.	name	size	type	f	f_{ref1}	$\text{gap}_1(\%)$	f_{ref10}	$\text{gap}_{10}(\%)$
1	Ha67	6	CI	114	114	0.00	114	0.00
2	DR59	12	CI	298	296	0.68	290	2.76
3	Ga67	21	R^{DC}	608	608	0.00	585	3.93
4	Ga67	22	R^{DC}	993	1017	-2.36	968	2.58
5	Ga67	39	R^{DC}	875	879	-0.46	875	0.00
6	CW64	30	CI	1360	1360	0.00	1280	6.25
7	Ga67	32	R^{DC}	814	848	-4.01	814	0.00
8	CE69	50	CR	564	564	0.00	564	0.00
9	CE69	75	CR	907	906	0.11	895	1.34
10	CE69	100	R^{C}	870	902	-3.55	878	-0.91
			average			-0.96		1.60
			st.dev			(1.60)		(2.14)

means the results of the stochastic algorithm are replicated almost perfectly.

Please note the large variation in the resulting quality for the problem number 6, which is the 30 customer problem instance from Clarke and Wright (1964). Closer examination of the results reveals that with this instance the algorithm is extremely sensitive to the routing of the generated TSP tour, and it is not guaranteed that the good quality solution reported in (Beasley, 1983) is found, even after 25 trials. The other problem instances showing similar, but not as distinctive, behavior are problems number 4 and 7 (22 and 32 customer problems from Gaskell (1967)). Despite these discrepancies, the results suggest that the implementations produce remarkably similar solutions and the differences can be attributed to statistical variation.

To summarize, we have implemented the stochastic algorithm from Beasley (1983) and derived a deterministic variant. Solid statistical testing on the replicated stochastic method results is not possible due to insufficient data in the original publication, but getting the average quality within 0.1% of those reported in (Beasley, 1983) gives high confidence on the correctness of the results and our implementation. Beasley (1983) proposes ways of including handling of additional constraints with the algorithm and in their recent survey of RFCS algorithms for VRPs Prins et al. (2014) give further examples in incorporating additional VRP constraints. Thus, in addition to scoring high on simplicity, the RFCS approach also seems to be rather flexible.

4.5 Relaxed Mathematical Programming Heuristics

Exact methods for solving vehicle routing problems have been under extensive study over the years (Laporte and Nobert, 1987; Toth and Vigo, 2002a). Because even the modern methods are able to solve only small to mid-sized problem instances, some authors have drawn inspiration from the research on exact methods to derive hybrid approaches for larger problems. These heuristic algorithms usually include

a step where a mixed integer programming techniques are used to solve a relaxation or a subproblem of VRP. In the literature these hybrids are referred to as *relaxed optimization* (Watson-Gandy and Foulds, 1981), *mathematical programming based approaches* (Bodin et al., 1983; Fisher, 1995), or, more recently, *matheuristics* (Maniezzo et al., 2010; Archetti and Speranza, 2014).

The main limitation of these hybrid methods is that the mathematical programming models and computation times can still grow prohibitively long when the problem size increases. Also, unlike the exact methods, they do not guarantee to find the optimal solution, and on some problem instances it is even possible that the final solution is inferior of that from the simpler heuristics.

Based on our literature meta-survey, we have chosen to implement four heuristics inspired by mathematical programming models. Three of these use a MIP solver as a part of the solution process, and the fourth relies on local search and draws inspiration from Lagrangian relaxation where some of the constraints are moved to the objective function. Each of these four approaches are detailed together with the replication results.

4.5.1 Maximum Matching

The tendency of the savings approach to make early merges that lead to poor solutions has been independently recognized by several researchers (Cordeau et al., 2002). To remedy this, and to simultaneously make the estimation of the merged route quality more accurate, Desrochers and Verhoog (1989) proposed a new savings heuristic MBSA (**DV89-MBSA**), where the merges are chosen by repeatedly solving a maximum weighted matching problem. The matching problem involves finding a set of edges in a graph so that no two edges share a node. When each edge is given a weight w_{ij}, maximum matching is the set of edges with this condition that has the largest sum of weights. Written as a mathematical programming model it becomes:

$$\begin{aligned} \max \quad & \sum_{i=1,j=1}^{K} w_{ij} x_{ij} & & \\ \text{s.t.} \quad & \sum_{j=1}^{K} x_{ij} \;=\; 1, & & i = 1, \ldots, K \\ & x_{ij} \;=\; 0 \text{ or } 1, & & \substack{i=1,\ldots,K \\ j=1,\ldots,K} \end{aligned}$$

The Figure 13 illustrates how the emerging solution is initialized similarly to the parallel savings algorithm. That is, there are K routes each serving a single customer. Each route forms a node in the matching problem graph and the edge weights represent the improvements of the solution if the corresponding routes are merged. Desrochers and Verhoog (1989) proposed using a TSP-algorithm to get the exact merged route cost, which lifts the requirement of the routes needing to be joined at the route ends. Thus, the savings value doubling as the matching graph weight is calculated as follows:

$$w(i,j) = \mathrm{TSP}(R_i) + \mathrm{TSP}(R_j) - \mathrm{TSP}(R_i \cup R_i), \tag{9}$$

where R_i is the set of the customers and the depot of the route i, TSP(R_i) is the cost of a TSP-optimized routing through these locations, and the savings value $w(i, j)$ is the saving in cost achieved my merging routes i and j to for a new TSP-optimized route. Our implementation uses Gurobi (2018) in both solving the maximum matching problem and the TSP, although the implementation optionally supports using a local search TSP algorithm for very large problem instances.

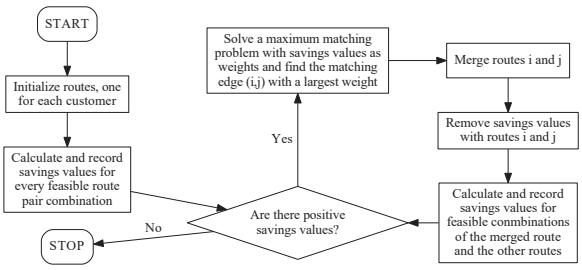


Figure 13: Operating principle of a maximum matching based savings heuristic.

If a maximum matching solution is found, then the routes with a connecting matching edge with the largest weight are merged and the savings weights updated accordingly. This means removing all weights that were associated to the routes prior to the merge and calculating the new weights for connecting the merged route with the other routes.

Our early experiments revealed that it is possible that two decision variables have the same largest savings value when solving the maximum matching problem. With our implementation this happened on 12 of the 18 problem instances. Unfortunately, Desrochers and Verhoog (1989) did not specify how these ties should be resolved. We decided to use a secondary criteria that allows choosing the merge that has the poorest alternative merges. To be more specific, this is selecting the merge with the smallest sum of savings values for the other optional merges:

$$w_{\text{secondary}}(i,j) = -2w_{ij} + \sum_{k=1,l=1}^{K} \begin{cases} w_{lk}, & \text{if } \substack{k=i \vee k=j \vee \\ l=i \vee l=j}, \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

Despite its good accuracy, especially for the iterative variant from Wark and Holt (1994), Cordeau et al. (2002) score the matching based approach low on flexibility, speed, and criticize it for its high complexity. Our replication efforts somewhat validate the critique on the complexity but the bigger issue in our opinion is consistency: while the heuristic is reasonably simple to implement if one has access to MIP and TSP solvers, the algorithm seems to be very sensitive to the implementation details. In Table 14 this can be seen as a large average replication gap of 0.81%, and in a large instance to instance variation as indicated by the standard deviation of 2.3 percentage points. Our implementation produces clearly worse solutions especially on the problems 10 and 12 when compared to the results given in (Desrochers and Verhoog, 1989). If we omit the outliers (marked in bold typeface in Table 14) the average replication gap shrinks to 0.4%. Also, looking closely the results, it seems that the replication difficulties concentrate on the problem instances that have the maximum route length constraint L. If we calculate the average replication gap only for the CVRP instances 5-9, 15 and 16, our implementation is actually 0.7% better

than that of Desrochers and Verhoog (1989) together with much more reasonable variation in instance to instance gap.

Table 14: Replicated results of the Desrochers and Verhoog (1989) maximum matching based savings heuristic.

Problem				DV89-MBSA			
no.	source	size	type	f_{ref}	f	$\text{gap}_C(\%)$	$\text{gap}_L(\%)$
1	Ga67#4	21	I^{CD}	587	598		1.87
2	Ga67#6	22	I^{CD}	970	958		-1.24
3	Ga67#5	29	I^{CD}	939	962		2.45
4	Ga67#3	32	I^{CD}	833	839		0.72
5	CMT79#01	50	R^{C}	586	580	-1.02	
6	CMT79#02	75	R^{C}	885	889	0.45	
7	CMT79#03	100	R^{C}	889	896	0.79	
8	CMT79#04	150	R^{C}	1133	1122	-0.97	
9	CMT79#05	199	R^{C}	1424	**1383**	**-2.88**	
10	CMT79#06	50	R^{DC}	593	**617**		**4.05**
11	CMT79#07	75	R^{DC}	963	973		1.04
12	CMT79#08	100	R^{DC}	914	**982**		**7.44**
13	CMT79#09	150	R^{DC}	1292	1314		1.70
14	CMT79#10	199	R^{DC}	1559	1581		1.41
15	CMT79#11	120	R^{C}	1058	1046	-1.13	
16	CMT79#12	100	R^{C}	828	828	0.00	
17	CMT79#13	120	R^{DC}	1562	1580		1.15
18	CMT79#14	100	R^{DC}	882	871		-1.25
			average			-0.68	1.76
			st.dev.			(1.14)	(2.30)
					total average		**0.81**
					total st.dev.		(**2.27**)

Ga67 = Gaskell (1967), CMT79 = Christofides et al. (1979)
Results with a z-score over 1.0 are in bold typeface.

There are at least two possible causes for the difficulties in replicating the results. In addition to the method that is used to resolve equally good matching results, a potential source of variation is in the calculation accuracy of the real distances. Recalculating the solutions and route lengths for the two new best solutions in Appendix A2 of Desrochers and Verhoog (1989) (and correcting the multiple misprints in the solutions) reveals inconsistencies: On average, the values given for the route length in the paper are 0.04 units off to our calculations with 32-bits of floating point accuracy, and worst per route difference is almost 0.1% off. This is probably due to differences in real distance representation and arithmetic. Unfortunately, the slight differences in route length calculations can change the results of the maximum matching, ultimately leading to a completely different solution.

Summarizing the replication results regarding DV89-MBSA, we cannot be certain if the main reason for not being able to replicate the general solution quality level is due to 1) differences in resolving the ties in choosing the maximum matching edge or 2) in floating point representation accuracy. The fact that we were able to match

and even surpass the quality of the solutions reported in (Desrochers and Verhoog, 1989) on the capacity constrained instances suggests that our implementation works correctly and is able to solve the problem instances effectively. As Desrochers and Verhoog (1989) did not publish all solutions of their experiments, it is hard to troubleshoot why our implementation fails to find satisfactory solutions for the few remaining instances with a maximum route length constraint.

4.5.2 Local Search with Lagrangian Relaxation

The basic idea of this heuristic is to start from an infeasible solution and use local search to move this initial solution towards better and, eventually, feasible solutions. The constraint checks in the local search are removed and replaced with a penalty in the objective function that is dependent on the amount of the constraint violation in the incumbent solution. The heuristic is discussed here under relaxed mathematical programming methods because the idea is borrowed from the Lagrangian relaxation technique used in mathematical optimization.

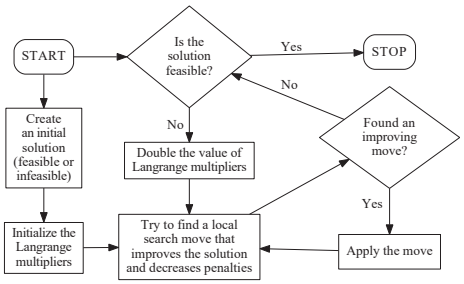


Figure 14: Operating principle of the Stewart and Golden (1984) heuristic.

For solving CVRPs, this technique was proposed in 1984 when Stewart and Golden introduced their LR3OPT heuristic that combines 3-opt* local search with Lagrangian relaxation. The operating principle of the heuristic is illustrated in Figure 14. They proposed the following objective function to model the penalties:

$$\sum_{i,j,k} c_{ij}x_{ijk} + \lambda_{1_C} \sum_{k \in K^*} (\sum_{i} d_i y_{ik}). \tag{11}$$

The decision variables x_{ijk} and y_{ik} are the same as in three-index vehicle-flow formulation (Golden et al., 1977) and dictate if an edge is traversed by a vehicle k but also if the customer i is served by that vehicle. Here, c_{ij} and d_i are the edge weight and the customer demand, K^* is the set of routes that would violate the capacity constraint, and λ_{1_C} is the Lagrange multiplier that is iteratively increased to steer the search towards feasible solution.

To extend the algorithm for solving problem instances with maximum route duration/length constraint, we propose the following modification to (11):

$$\sum_{i,j,k} c_{ij}x_{ijk} + \sum_{k \in K^*} (\lambda_{1_C} \sum_{i} (d_i y_{ik} - C) + \lambda_{1_L} \sum_{i,j} (c_{ij}x_{ijk} - L)).$$

Regarding the multiplier values, Stewart and Golden (1984) propose that the first multiplier is initially set to value $\lambda^1_{1_C} = \bar{d}/(20 \max c_{ij})$, where $\bar{d}$ is the average

customer demand. For exceeding the maximum route duration/length we propose using a similar initial value for the multiplier: $\lambda^1_{1_L} = \sum_i (\min_{j \neq i} c_{ij})/(n10 \max c_{ij})$.

In the implemented **SG84-LR3OPT** algorithm the constraint checks of the vanilla 3-opt* local search heuristic were replaced with penalty calculations. Performance considerations became apparent after preliminary experiments, and we decided to use the 3-opt* version with first-accept strategy that operates on the entire solution (as opposed to individual routes). This version checks both intra- and inter-route moves on one pass and we assumed that this is also the way which it was implemented in (Stewart and Golden, 1984). As the search progresses, only the modified edges are checked and the objective function update for the move has a constant time complexity. We made sure there is always an empty route available for the 3-opt* to link solution segments to, as this allows the algorithm to increase the number of routes when it needs to reach feasibility.

Whenever the local search procedure ceases to find improving moves, we check if the incumbent is a feasible solution. If it is, the incumbent is returned as the solution to the problem. However, if the solution remains infeasible, we increase the penalties by doubling the multiplier values as proposed by Stewart and Golden (1984) and the local search continues.

Stewart and Golden (1984, p. 88) propose initializing and repeating the heuristic with multiple random solutions. To create a deterministic variant, we propose initializing the algorithm with a LKH computed TSP tour through all customers and the depot. Consequently this also somewhat mitigates the 3-opt*'s high computational time complexity of $\mathcal{O}((n+k)^3)$, as the LR3OPT procedure is run only once.

Stewart and Golden (1984) did not report CPU times of their computational experiments, but omitting the 250 customer problem from the Gillett and Miller (1974) problem set suggests that the computation times for that specific instance became prohibitively long. As a comparison, our implementation solves a 100 customer problem in just under 5 minutes on a Intel Core i5 M460 CPU. This, together with the relatively poor computational performance of our pure Python 3-opt* implementation, limits our implementation to small and medium sized problem instances. To illustrate the point, if a third order polynomial is fitted to the performance data of LR3OPT, it can be estimated that the CPU time for solving a 1000 customer problem would be around 40 CPU days.

Unfortunately, the extensive computational effort of LR3OPT became apparent only after we had implemented the heuristic. Still, because it is an interesting and effective approach for solving smaller problems, we decided to include it to VeRyPy and implemented a greedy backup heuristic to make the incumbent solution feasible if the heuristic is interrupted.

Regarding replication of the results in (Stewart and Golden, 1984), we have already discussed the potential issues in replicating the result of heuristics that rely on local search in Section 2. In addition to those, reproducing the results of a stochastic algorithm can be problematic and this is also the case with LR3OPT: Stewart and Golden (1984) Table 1 only gives the best solutions and there are no descriptive statistics for the distribution of the solution quality. Another source of variation is in the initial solutions. Stewart and Golden (1984, p.88) started the heuristic 'several times from different random starting solutions'. However, the

Table 15: Replicated results ofthe Stewart and Golden (1984) 3-opt* heuristic with Lagrangian relaxation (SG84-LR3OPT).

Problem					LR3OPT$_{\text{STO}}$					LR3OPT$_{\text{DET}}$	
no.	source	size	type	f_{ref}	trials	f	gap(%)	f^*	gap(%)	f	gap(%)
1	CW64	30	CI	1212	10	1246.3(11.1)	2.8(0.9)	1213	0.1	1250	3.1
2	CE69	50	CR	521	10	543.2(11.0)	4.3(2.1)	525	0.8	525	0.8
3	GM74	75	CR	1058	7	1057.7(9.5)	-0.0(0.9)	1043	-1.4	1074	1.5
4	CE69	75	CR	847	28[a]	870.8(11.8)	2.8(1.4)	847[a]	0.0[a]	867	2.4
5	GM74	75	CR	751	10	769.5(11.6)	2.5(1.5)	748	-0.4	756	0.7
6	GM74	75	CR	692	20[a]	711.8(7.8)	2.9(1.1)	696[a]	0.6[a]	697	0.7
7	GM74	100	R$^{\text{C}}$	1117	3	1123.7(6.6)	0.6(0.6)	1119	0.2	1150	3.0
8	CE69	100	R$^{\text{C}}$	829	12[a]	852.9(7.7)	2.9(0.9)	838[a]	1.1[a]	851	2.7
			average				2.33		0.1		1.9
			st.dev.						(0.7)		(1.0)

CW64 = Clarke and Wright (1964), CE69 = Christofides and Eilon (1969),
GM74 = Gillett and Miller (1974)
[a] = similar quality level reached only after additional trials

exact method used to generate these random solutions was not specified. In our replication experiments we assumed a random tour through all of the customers and the depot.

We initially used the same number of trials as Stewart and Golden (1984), but were unable to find as good solutions for problem instances 4, 6, and 8. We would like to point out that a stochastic algorithm can produce a good solution after just few trials by pure chance, even if the general level of quality of solutions is much lower. This is the reason why rigorous statistical testing is so important when designing experiments for stochastic heuristics (Barr et al., 1995). Considering this, we decided to increase the number of trials for the three remaining instances, which allowed our implementation to produce the remaining good best solutions. The number of initial solutions and the final results of our replication efforts are reported in Table 15.

As can be seen from the table, our implementation has an average accuracy that is very close to the original results (within 0.11%). The low standard deviation value suggests that when enough initial solutions are explored, the algorithm can consistently produce the reported level of quality of solutions. However, as can be seen from the per instance standard deviations of Table 15, the LR3OPT heuristic seems to be sensitive to how it is initialized. We can also see that if the heuristic is initialized with a TSP tour, then the results are clearly inferior to the best results of the stochastic version. However, the average results produced by the deterministic version are better than the average performance of the stochastic one, which validates that the TSP tour is generally a good starting point.

Taken together, the SG84-LR3OPT from Stewart and Golden (1984) is an interesting approach for solving vehicle routing problems. Implementing the 3-opt* local search for VRP can be tricky, but otherwise the algorithm is fairly simple. It is also flexible as most constraints can be relaxed using suitable penalty scheme. However, balancing the more complex objectives, such as primarily minimizing the number of vehicles, can be difficult. This is because finding correct weights for the

different penalties during the search is hard. Consequently, minimizing the number of vehicles was not considered by Stewart and Golden (1984) and it is also missing from our implementation. According to our preliminary experiments, the algorithm is accurate but slow, and it also seems that the robustness of the heuristic is rather poor as the accuracy is sensitive to the initial solution. Thus, the method is limited to solving instances of at most few hundred customers.

4.5.3 Generalized Assignment

As one of the best known (Laporte and Semet, 2002) cluster-first, route-second algorithms, the Fisher and Jaikumar heuristic **FJ81-GAP** relies on solving a Generalized Assignment Problem (GAP) relaxation of the VRP to form clusters for the routes. The algorithm has three phases: First, a seed point is selected or generated for each proto-route (cluster). Then, in the second phase, a generalized assignment problem (GAP) is solved. In the GAP model, the seeds from the first phase are used to approximate the customer node service costs. Finally, in the third phase, the customer assignments of the GAP solution are routed to create a VRP solution. Please refer to Figure 15 for a high level description of the process.

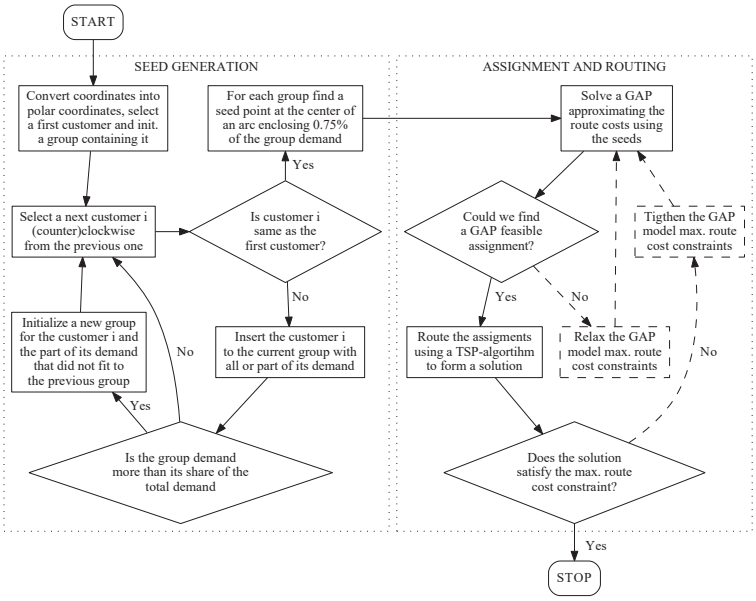


Figure 15: Operating principle of the Fisher and Jaikumar heuristic. The dashed part is our extension with the repeated self-adjustment for solving problems with a maximum route length/duration constraint.

For the seed generation, Fisher and Jaikumar (1981) proposed using a sweep-like process to divide customers into K groups, that is, one for each vehicle. The major differences to sweep are that K is predefined (or precalculated using demands and capacity) and that the customer demands can be split between adjacent groups. Each group is formed between two rays leaving from the depot. The rays are not required to exactly bisect the angle between two consecutive customers. Instead, these rays are selected in a way that they create a cone that captures exactly $\sum d_i/(KC)$ of the total demand for each group. Then, the angle between the rays is bisected

and a seed point is set on this bisection. The distance from the depot to the seed point is determined using the sector area. More specifically, the distance is the length of the radius lines of the sector when the sector captures a demand equal to $0.75 \sum d_i/K$. Similar *cone covering* seed generation method has also been used by Baker and Sheasby (1999) and one can refer to Appendix A of that paper for details.

The second phase concerns finding a good assignment of customers to the vehicles. This is achieved by solving a linear generalized assignment problem with an objective function that approximates the routing cost. Here, the seed points i_k generated in the previous phase are used to calculate the approximate costs $a_{ik} = c_{0i} + c_{ii_k} - c_{i_k0}$ of including a customer i on the route k. Thus, if each y_{ik} is a decision variable assigning customer i on the route k, d_i is the demand of customer i, and C is the vehicle capacity, the assignment problem relaxation of VRP can be written as:

$$
\begin{aligned}
\min \quad & \sum_{k=1}^{K} \sum_{i=1}^{n} a_{ik} y_{ik} & & & \\
\text{s.t.} \quad & \sum_{i} d_i y_{ik} & \leq & C, & k = 1, \ldots, K \\
& \sum_{k} y_{0k} & = & K, & \\
& \sum_{k} y_{ik} & = & 1, & i = 1, \ldots, n \\
& y_{ik} & = & 0 \text{ or } 1. & \substack{i=1,\ldots,n \\ k=1,\ldots,K}
\end{aligned}
$$

Additionally, while not formally expressed in the paper, the wording in Fisher and Jaikumar (1981, p. 121) suggests that constraints for the maximum route length or distance L can be modeled using the approximation of the objective function. We assumed that this means including following constraints to the GAP:

$$
\sum_{i} a_{ik} y_{ik} \leq L, \qquad k = 1, \ldots, K. \tag{12}
$$

After solving the GAP, the assignments in the solution are used to create the clusters of customers, each served by a single route. This is followed by the final third phase where the clusters are routed using a TSP algorithm. The result is K routes that form a solution of the original VRP.

The implementation of the seed generation of the first phase is problematic since the description of the cone covering seed generation is somewhat vague. The first part that uses a variant of the sweep approach (Wren and Holliday, 1972; Gillett and Miller, 1974) is quite straightforward. However, it is unclear how the partial cones are considered to contribute to the total demand of the cone. Fisher and Jaikumar (1981) give one example on seed generation for an unspecified problem instance with 10 customers and a vehicle capacity of 30. Unfortunately, only a figure illustrating the result is given (Figure 4 in Fisher and Jaikumar, 1981, p. 119) and no specifics or numerical representation of the problem or the seed points are given

in the paper. To test our seed generation implementation, we converted the problem into a TSPLIB compatible `.vrp` file using the blob detection of image processing tool ImageJ (Schneider et al., 2012). ImageJ allowed us to extract the coordinates of the blob centers for the depot, customers, and the seed points w_1, w_2, and w_3. We interpreted the numbers printed on the figure as the demands of the customers (the number 2 was ignored as this seems to be misprint). Please note that the distances of the illustration may be inaccurate as it originates from an earlier published report where the illustration appears to be hand drawn. Despite these potential sources of error, our implementation is able to generate the seed positions with 3.5% accuracy in this example (relative to the depot-seed distance, see Figure 16).

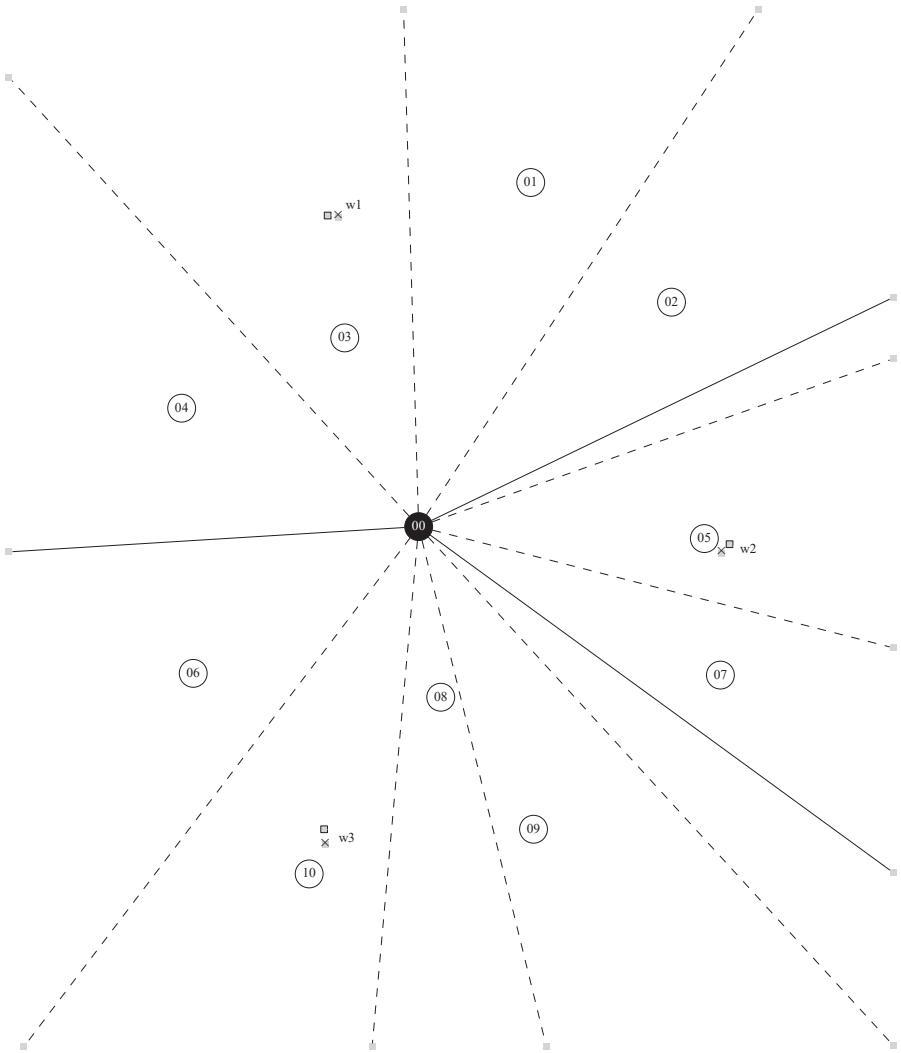


Figure 16: Replication of the Fisher and Jaikumar (1981) example seed point generation. where □ = the reference seed location, × = the seed generated by our implementation.

As a side note on the seed generation, our implementation also allows generating them using k-means or the two other selection criteria suggested by Fisher and Jaikumar (1981): the customers with large demands and customers at the end

of thoroughfares. However, as no experimental results were given for these seed selection methods, replication of results is not possible.

Moving on to second phase of the algorithm, the distances from all customers to the generated seeds are calculated and appended to the distance matrix. Then, the GAP is solved using Gurobi (2018). We used the default settings of Gurobi and let it select the most convenient solving approach. This was chosen instead of relying on the specialized GAP solving technique referred in the paper and later described in (Fisher et al., 1986). Then, the customers that are assigned to the clusters are routed by solving the set of corresponding TSPs with Gurobi.

Regarding replication the results, there seems to be multiple discrepancies with the CMT (Christofides et al., 1979) problem instances in (Fisher and Jaikumar, 1981): they report a result value 1014 for CMT4 which is better than the optimal solution of 1028.48 listed in (Uchoa et al., 2017). This suggests that either the instances are different, the solution costs are calculated incorrectly, it is infeasible, or that they truncate the travel distances. Similar issue exists with the Fisher and Jaikumar (1981) problem 12 (CMT14) as the solution with total route length of 848 is better than the best known solution of 866.37 for that problem. Unfortunately, Fisher and Jaikumar (1981) do not specify the distance rounding convention they had used, which would have allowed to rule out or confirm it as the reason for these discrepancies. Please note that the issues caused by the use of varying rounding conventions has been recognized earlier, for example, by Laporte (2009, p. 412). These, and other issues related to the problem instances outlined below, make replicating the results of Fisher and Jaikumar (1981) algorithm very challenging.

Furthermore, it is not always clear which CMT instance the numbering used by Fisher and Jaikumar (1981) refers to. Fisher and Jaikumar (Table I in 1981, p. 120) lists two problems with 100 customers and tightness $T = \sum_i d_i/(KC)$ of 0.91, which one would expect correspond to CMT12 and CMT14 of (Christofides et al., 1979; Uchoa et al., 2017). However, the later details in (Fisher and Jaikumar, 1981, p.121) do not mention a restriction to route time for problem 12. As the given solution value is close to the one of CMT14 we assumed this information was omitted by accident and in our replication experiments the CMT12 is used as the problem 11 and CMT14 as the problem 12.

There are also inconsistencies regarding the tightness scores in the problem characteristics Table I in Fisher and Jaikumar (1981). For example, tightness score of 0.94 for CMT4 differs from the 0.93 calculated from the problem instance of (Christofides et al., 1979). There are similar discrepancies in the given tightness for problems 3, 4, 5, 8, 9, and 10, where the differences are between 0.1 and 0.3. However, for the problem 10 the difference is 0.7, which is puzzling, since the instance should be the same as problem 5, only with an additional maximum route duration constraint. However, the tightness value of 0.77 for CMT10 in (Table I in Fisher and Jaikumar, 1981, p. 120) might also be a misprint instead of an issue with the problem instance data.

Regarding replication of the results there are other issues as well: firstly, Fisher and Jaikumar (1981) does not specify how the first customer of the sweep is selected or if the algorithm is repeated from multiple sweep start positions. Based on our experiments the quality of the solutions produced by the Fisher and Jaikumar heuristic is very sensitive to small changes in the seed positions, which is an impor-

tant shortcoming of the algorithm that they fail to discuss in the paper. Also, in our replication experiments, we were unable to reach similar accuracy without an extensive search of optimal seeds (that is, applying the sweep heuristic variant for determining the seed points from all N possible starting positions). In practice, this means repeating the procedure (Figure 15) N times.

A more serious issue for the replication was that of handling maximum route duration constraint. Fisher and Jaikumar acknowledge that because the route duration is approximated, a final feasibility check on the maximum route duration/length constraint is required. In practice, this check fails frequently, but unfortunately they did not specify what to do if that happens. We are not first who are puzzled on how the maximum route duration/length constraint is handled by the algorithm and it seems that the previous replication efforts have been unsuccessful (see, e.g., Laporte, 2009, p. 412).

Our experiments revealed that the approximated route cost can be significantly different to the TSP optimized actual route cost, which causes the GAP assignments to be infeasible because of the VRP maximum route duration constraint. To counter this, we devised a self-adjusting correction factor to the constraints (12), which is tightened if an infeasibility is detected. If the GAP becomes unsolvable, then the maximum route duration constraints are relaxed and violations penalized quadratically with Gurobi `feasrelax` procedure. We optionally also allow the number of vehicles K to be temporarily increased by, for example, 10% if relaxing the maximum route duration constraint does not lead to a feasible solution. This procedure is illustrated with the dashed parts of Figure 15. While this does not guarantee that a feasible solution is found, it allows solving problems with the maximum route duration constraint. We also had to impose a time limit to Gurobi because solving GAP iterations for some pathological seed configurations took excessively long time in our replication experiments. It is possible that a more specialized Lagrangian relaxation multiplier adjustment method that was published few years later by Fisher et al. (1986) would help to resolve this issue. It could also allow us to more closely replicate the original results. However, implementing it would not have been possible given the details of the 1981 paper. Also, implementing a specialized MIP method for a single heuristic was deemed too laborious for this study.

The results of our replication efforts are documented in Table 16. We computed the results with the exact (real) distances and with a truncated distance matrix (rounded down to the closest integer). The distances of CMT instances are usually calculated using real distances, but the fact that some of the results reported by Fisher and Jaikumar (1981) were better than the optimal or best known results led us to assume that truncated distances were used.

As can be seen from the table, we were unable replicate the results when using exact distances. However, for the truncated distances, the quality of the solutions produced by our implementation is reasonably close to those reported in (Fisher and Jaikumar, 1981, p. 122, Table II). We suspect that the discrepancies are mostly due to differences in problem instance data and in how the seed positions are determined. Also, incorporating the maximum route length constraint to the model seems to have a significant effect to the replication results as the largest differences happen on the problem instances with this side constraint.

Our experiments recognized a failure mode where the algorithm may be unable

Table 16: Replicated results of the Fisher and Jaikumar (1981) Generalized Assignment Problem VRP approximation algorithm (FJ81-GAP).

Problem					Exact distances		Truncated distances	
no.	source	size	type	f_{ref}	f_{R}	gap(%)	f_{I}	gap(%)
1	CMT1	50	C	524	536	2.29	522	-0.38
2	CMT2	75	C	857	860	0.35	820	-4.32
3	CMT3†	100	C	833	860	3.24	824	-1.08
4	CMT4†	150	C	1014	1067	5.23	1021	0.69
5	CMT5†	199	C	1420	1338	-5.77	1297	-8.66
6	CMT6	50	CD	560	570	1.79	549	-1.96
7	CMT7	75	CD	916	1011	10.37	992	8.30
8	CMT8†	100	CD	885	962*	8.70	891	0.68
9	CMT9†	150	CD	1230	1300*	5.69	1241	0.89
10	CMT10†	199	CD	1518	1634*	7.64	1556*	2.50
11	CMT12	100	C	824	872	5.83	845	2.55
12	CMT14	100	CD	848	912*	7.55	886*	4.48
			average			4.41		0.31
			st.dev.			(4.20)		(4.09)

CMT = Christofides et al. (1979), † = tightness score discrepancy, $*$ = our solution uses one additional vehicle.

to produce a feasible solution: According to the algorithm description in (Fisher and Jaikumar, 1981) the number of vehicles K is calculated with $K = \lceil \sum_i d_i / C \rceil$, which usually works for synthetic cases. However, real world demands may be distributed in a way that there is no feasible assignment for K vehicles. To overcome this, our implementation restarts the procedure with a larger K value if finding a feasible solution fails. Also the option to increase K temporarily, as described above, can be used on time constrained cases and when it is important to find a feasible solution early in the search process.

Based on the quality of the solutions reported in Fisher and Jaikumar, the accuracy of the method seems good. However, issues with the reported solution values have been previously recognized (e.g., Wark and Holt, 1994; Cordeau et al., 2002; Laporte, 2009). Our replication efforts suggest that the reason of these discrepancies may be in the use of the truncated distances. Also, because the seed generation procedure is hard to replicate accurately due to proper reference results, and because the heuristic requires additional tweaks and modifications to be more robust, we side with Cordeau et al. (2002) on that the algorithm scores low on simplicity and flexibility. Thus, our replication efforts validate some of the criticism towards the Fisher and Jaikumar algorithm.

Regarding the CPU time requirements of the algorithm, we recognized that the GAP solve time varies greatly depending on the positions of the seeds or particularities of the GAP model. Additionally, a maximum route duration constraint can cause the runtime of the algorithm to increase significantly and become very hard to predict. Usually the GAP iterations are solved in a few seconds or less but there seems to be some pathological configurations, especially for instances the

maximum route cost constraints, that Gurobi is unable to solve in a reasonable time. In our replication tests, we used an one minute timeout for the GAP solver to allow the algorithm to move on to the next seed configuration if a pathological GAP model was encountered. The MIP solver timeout is not the only free parameter that strongly affects the quality of the solutions. Similarly, the maximum route duration constraint adjustment factor has to be set for the problem instances with such constraint. We used the base value of 0.85 incremented by at most 10 steps of 0.12 until a feasible solution for given set of seed points was found or the current seeds were rejected. Furthermore, regarding seeds, the method seems to rely heavily on generation and exploration of multiple seeds which makes it computationally more demanding than Tables II and III in Fisher and Jaikumar (1981) suggest. For example, solving the maximum route cost constrained 199 customer problem (CMT10) with our implementation takes almost 20 hours. This involves checking all possible seed configurations, which is further amplified by repeated solving attempts with the self-adapting constraint multiplier and frequently hitting the MIP timeout. Hence, we must assume that the values given in the original tables report only the runtime of a single trial on a single seed configuration. This is misleading since checking only one seed is typically inadequate for finding good solutions.

In our further experiments, we found out that there can be order of magnitude variation in GAP solve time from similar sized instances. This inherent variability of modern MIP solver performance is a well known issue (see, e.g., Koch et al., 2011, Section 5). This seems to be an undesirable property for a VRP heuristic. To illustrate the point, checking a single seed configuration for the 385 customer problem instance from Taillard (1993) with a 8 core Intel Xeon E5-2673 takes 9 hours. Computation times such as this makes trying all possible seed configurations for large problem instances infeasible. Therefore, the method does fully meet our requirements as it is not able to fully solve instances with 1000 customers.

Taken together, we were unable to reach the reported level in the quality of the solutions using exact distances. While the results of Fisher and Jaikumar (1981) algorithm were originally compared against those calculated with other algorithms that use rounded or exact distances in (Fisher and Jaikumar, 1981, p. 122, Table II), it seems probable that unconventional truncation of travel costs was used. With this assumption, our implementation seems able to replicate the general level of quality of the solutions, albeit with a large instance to instance variation. Also the way algorithm runtime is reported in (Fisher and Jaikumar, 1981) seems misleading. However, we cannot be sure if the source of the discrepancies is in the problem instances, in misunderstanding some part of the algorithm, or in an implementation error from our part.

4.5.4 Set Covering

The basic operating principle of the Petal heuristic **FR76-1PTL** from Foster and Ryan (1976) makes solving CVRPs simple. First, a large set of routes L, called petals, are generated. These routes can overlap and contain shared customers, but every customer should be served at least by one route. Then, a solution can be found by solving a weighted set covering problem:

$$\min \quad \sum_{l \in L} \bar{c}_l x_l \tag{13a}$$

$$\text{s.t.} \quad \sum_{l \in L} a_{il} x_l = 1, \qquad i = 1, \ldots, n \tag{13b}$$

$$\sum_{l \in L} x_l \leq K, \tag{13c}$$

$$x_l = 0 \text{ or } 1, \qquad l \in L. \tag{13d}$$

where the decision variable x_l selects the route l as the part of the solution, c_l is the cost of petal (route) l. In the constraints, the constant a_{il} equals 1 if customer i is served by the route l and 0 otherwise. Depending on how the routes in L are generated, the problem may be unsolvable. It is, however, possible to generate more petals or allow a customer to be served multiple times. Any overlapping routes in the solution can be improved by removing the duplicate customers in the order that improves the solution the most until all customers are served exactly once.

Foster and Ryan proposed to use a method similar to the sweep heuristic (see Figure 8 on page 45) to generate all feasible routes of radially consecutive customers and optimize them using a TSP-algorithm. They also proposed that these petals are divided into three petal sets with increasing number of petals. The first set is the *restricted* set, where the total demand of each petal route is larger than $0.75C$. The second is the *reduced* set, where the total demand of each route is larger than a lower bound C_l which is determined as follows:

$$C_l = \sum_{i=1}^{n} d_i - (K_{\text{LB}} - 1)C,$$

where the number of required vehicles K_{LB} is estimated with:

$$K_{\text{LB}} = \left\lceil \frac{\sum_{i=1}^{n} d_i}{C} + \epsilon \right\rceil.$$

The third *extended* petal set contains all of the generated routes, even those serving only a single customer. Between these, the heuristic moves progressively to larger petal sets if set covering is unable to find feasible solutions, or if growing the petal set is expected to lead to improved solutions. Also, the extended set is used if the number of petals (routes) in the SCP solution with the reduced set is more than K_{LB}. This entire procedure is illustrated in Figure 17.

For solving the SCP the current petal set is always complemented with a set of *relaxed petals*. Initially this set is empty, but new petals are generated on each iteration from the current SCP solution using a custom local search operator. Foster and Ryan (1976, p.374) call this step a relaxation of the over-constrained petals. The operation involves trying to move a customer from one route to another. If the move would improve the total quality of the solutions, and a constraint is broken, existing customers on the receiving route are redistributed if it is necessary to do so to regain feasibility. It is required that after redistribution the total quality of the solution will still be improved from the original. All possible moves are tried and improving and feasible solutions are added to the petal set.

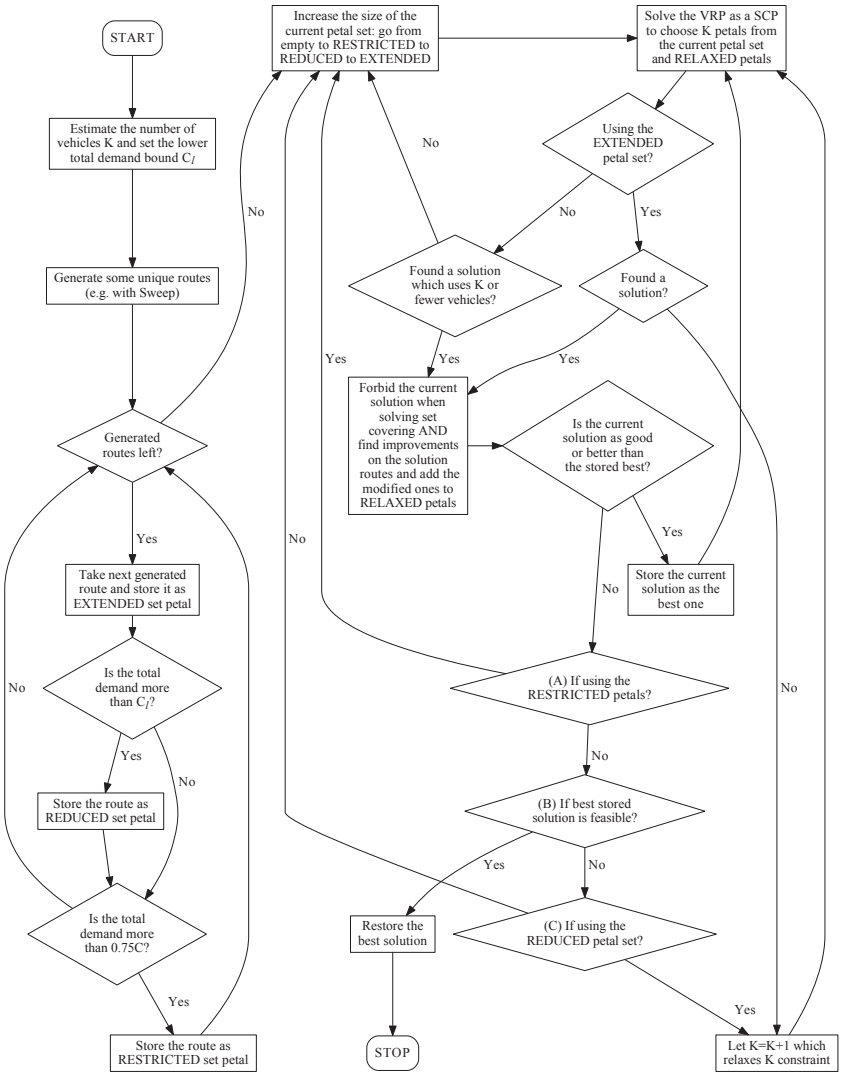


Figure 17: Operating principle of the Foster and Ryan (1976) Petal heuristic.

To allow the SCP solver to generate previously unseen solutions, the heuristic adds constraints forbidding previously seen petal combinations (solutions) to the set covering problem formulation (13a)-(13d). The constraints are of the form:

$$\sum_{l \in F_j} x_l - \sum_{l \in L \setminus F_j} x_l < |F_j|, \qquad \forall F_j, \tag{13e}$$

where F_j is the set of petals forming the solution to forbid. The inequality forbids that exact petal configuration to be returned as a solution to the SCP and a new constraint row is generated for each new SCP solution.

Unfortunately, Foster and Ryan (1976) do not explicitly state the termination criteria. This makes replicating the results difficult (Barr et al., 1995). From the description, we assumed that new solutions are generated until the quality of the solutions from the SCP solver starts to degrade. This is supported by the mention of 'a sequence of feasible solutions are produced with mileages decreasing to the final solution' where the numerical results are discussed (Foster and Ryan, 1976, p.379), even though such requirement is not given in the main algorithm description.

Foster and Ryan allow loosening the constraint specifying the number of vehicles (13c) and relaxing from restricted to 'complete' petal set when no further improvements are found (Foster and Ryan, 1976, p. 375, first paragraph). We assumed that by 'complete' petal set they mean the restricted set as opposed to the extended set, but the terminology is not very accurate here. Also, based on the experimental results (Table 2, Foster and Ryan, 1976), the SCP solver they used was allowed to return infeasible solutions where one or more customers are left unserved. It is difficult to see from the algorithm description how these relaxations came to be, but we decided to rely on Gurobi's built-in relaxation routine to lift minimum number of (13b) constraints to get a solution. This solution will be infeasible per original constraints, but since relaxations that improve the solution can be found also among the infeasible solutions, this can eventually lead to better feasible solutions. However, our procedure does not allow infeasible solutions after the first feasible solution is found.

Foster and Ryan have attached all solutions produced by their implementation as an Appendix to their paper. This allowed us to notice that the Clarke and Wright (1964) instance used in Foster and Ryan (1976) seems to have slightly different distance matrix than the original. If the mileage of the solution given in Appendix (Foster and Ryan, 1976) is recalculated with the correct data, we get a result 1419 as opposed to 1377 given in the Appendix and Table 2 of (Foster and Ryan, 1976). Please note that the original problem definition gives place names but does not specify their coordinates. These would have to be guessed either using geocoding or some multidimensional scaling technique, neither of which does guarantee that the locations are same as those used in (Foster and Ryan, 1976). Therefore, we decided to omit it from our replication problem set.

As can be seen from our replicated results (Table 17a), we are able to get reasonably close to the results of petal MIP given in Foster and Ryan (1976, Table 2) with the average quality gap of 0.4%. This can be considered to be a good replication result considering that the order in which the customer serving constraints

Table 17: Replicated results of the Foster and Ryan (1976) Petal weighted set covering problem (FR76-1PTL) heuristic.

Problem				(a) Petal MIP				
no.	source	size	type	$\|L_{ref}\|$	$\|L\|$	f_{ref}	f	gap(%)
1	Ga67#4	21	I^{DC}	30	31	607	607	0.00
2	Ga67#6	22	I^{DC}	97	97	863*	877*	1.62
3	Ga67#5	29	I^{DC}	188	188	813*	**833***	**2.46**
5	Ga67#3	32	I^{DC}	67	67	792*	805*	1.64
6	Ga67#1	36	I^{D}	286	285	841	841	0.00
7	CE69#8	50	I^{C}	79	79	523	528	0.96
8	GM74#6	75	I^{C}	150	152	1084*	1089*	0.46
9	CE69#9	75	I^{C}	157	153	864*	860*	-0.46
10	GM74#8	75	I^{C}	495	315	768	**753**	**-1.95**
11	GM74#9	75	I^{C}	731	729	692	692	0.00
12	FR76#12	75	I^{DC}	152	148	852*	856*	0.47
13	GM74#10	100	I^{C}	733	730	1174	1174	0.00
14	CE69#10	100	I^{C}	964	967	825	827	0.24
15	FR76#15	100	I^{CD}	921	925	827	833	0.73
			average					0.44
			st.dev.					(1.02)

Problem				(b) With relaxations, terminate by rules			iteration cap	
no.	source	size	type	f_{ref}	f	Gap (%)	f_{50}	gap(%)
1	Ga67#4	21	I^{CD}	585	592	1.20	585	0.00
2	Ga67#6	22	I^{CD}	953	958	0.52	958	0.52
3	Ga67#5	29	I^{CD}	873	**936**	**7.22**	888	1.72
5	Ga67#3	32	I^{CD}	809	809	0.00	809	0.00
6	Ga67#1	36	I^{D}	838	841	0.36	841	0.36
7	CE69#8	50	I^{C}	521	524	0.58	524	0.58
8	GM74#6	75	I^{C}	1081	1101	1.85	1087	0.56
9	CE69#9	75	I^{C}	852	853	0.12	853	0.12
10	GM74#8	75	I^{C}	760	751	-1.18	**744**	**-2.11**
11	GM74#9	75	I^{C}	692	692	0.00	692	0.00
12	FR76#12	75	I^{CD}	865	861	-0.46	859	-0.69
13	GM74#10	100	I^{C}	1116	1112	-0.36	1107	-0.81
14	CE69#10	100	I^{C}	825	825	0.00	825	0.00
15	FR76#15	100	I^{CD}	826	831	0.61	830	0.48
			average			0.75		0.05
			st.dev.			(1.93)		(0.84)

Ga67 = Gaskell (1967), CE69 = Christofides and Eilon (1969)
GM74 = Gillett and Miller (1974), FR76 = Foster and Ryan (1976)
$*$ = infeasible solution, outliers (z-score > 1.5) are in bold typeface

(13b) are relaxed is not specified in the paper. To control this, we can omit the instances with infeasible solutions and the average quality gap becomes 0.00%. That is, our implementation almost perfectly replicates the general quality of solutions

after first iteration of Foster and Ryan (1976) implementation. The number of generated petals ($|L_{ref}|$ vs. $|L|$) is also very similar. The minor differences may be attributed to different floating point accuracy in polar coordinate transformation and comparison. Only on the problem instance 10 our implementation, for some unknown reason, includes significantly fewer petals in the reduced petal set (315 vs. 495).

The other set of results given in (Foster and Ryan, 1976, Table 2) titled *'Relaxations'* have been produced by solving the SCP iteratively and generating relaxed petals for each solution as described above. Unfortunately, replicating these results proved tricky. The usual replication issues with local search components (see Section 2) can manifest also in the solution relaxation procedure, especially since the description of the node relocation with a redistribution heuristic that is used in the paper is not very accurate. Trying all possible combinations for each potential relocated candidate would be impossible, as this would mean removing and re-inserting all customers on the receiving route, in all possible insertion orderings, on all possible receiving routes, in all possible receiving route orderings. As one can expect, this leads to combinatorial explosion, which makes it unlikely that Foster and Ryan (1976) did an exhaustive search and instead used an unspecified greedy heuristic. However, the specifics are not given.

Our local search implementation checks all combinations of removing customers from the receiving route that free enough carrying capacity or travel time slack to fit the relocated customer. If a combination of removed customers frees enough capacity and cost, further removals on top of that combination are not considered. This forms a breadth-first search pattern, where the branches are pruned if a combination frees enough capacity or cost. The removed customers are redistributed in the order they were on the receiving route and the routes that are candidates for accepting the redistributed customers are in the petal generation order, followed by the route where the relocated customer was removed and an empty route. Even with these assumptions, a combinatorial explosion may make it infeasible to find relaxed petals for larger instances. Therefore, as the number of customers of the longest route grows, we limit the maximum number of removed customers from the receiving route. This was necessary to allow solving also the larger problems.

As can be seen from the Table 17b, our implementation produces worse results (the average gap is 0.75%) if the rule based termination is used. In these results, there is a single clear outlier, namely the problem 3, where our implementation is unable to find the good feasible solutions. The difference in the quality of solutions for this instance is large if it is compared to that reported by Foster and Ryan (1976) (7.22% difference). Omitting this outlier brings the average replication error on our relaxed Petal implementation down to the acceptable level of 0.25%. Furthermore, based on our results and extensive experimentation, it seems the our termination criteria differs from the one of Foster and Ryan (1976). Specifying lower bound for the number of iterations allows our implementation to explore a larger number of solutions, which in turn allows discovery of relaxed petals ultimately leading to better final solutions at the expense of longer computation time. Through experimentation, we set the minimum number of iterations in the final set of replication experiments to 50, which allowed us to almost perfectly replicate the *'Relaxations'* results reported in (Foster and Ryan, 1976, Table 2). These results are reported in

the rightmost columns of Table 17b.

Taken together, while it seems that our implementation is able match the quality of the solutions reported by Foster and Ryan (1976), some open questions remain especially regarding the termination criteria, where we had to parametrize and manually set the termination criteria in order to reach our replication target of 0.1%. Despite extensive experimentation and troubleshooting, we were unable to find a rule-based criterion that would terminate the algorithm exactly like in (Foster and Ryan, 1976). However, another possible source of replication errors is in local search and solving TSPs. The local search is not described in sufficient detail and the solver we used for solving TSPs was LKH (Helsgaun, 2000, 2009) instead of the obsolete heuristic described in (Foster and Ryan, 1976).

Please note that our implementation can use the first of the secondary additional relaxations presented in the end of (Foster and Ryan, 1976). This is done by specifying a desired number of iterations, which forces the code to iteratively forbid feasible solutions to the set covering problem. This may allow finding relaxed petals that together with the extended petal set can produce an improved solution. However, the second secondary relaxation involving movement of consecutive customers from a route to another in the relaxed petal generation was not implemented.

Cordeau et al. (2002) score the Petal high on accuracy, at least compared to other classical heuristics; high on simplicity, given that the petal generation is kept straightforward; and high on flexibility as different constraints can easily be accommodated to the petal generation phase. The weakness of the method is in that good results depend on a comprehensive collection of suitable petals and for larger problem instances the petal generation may become computationally infeasible.

5 Technical and User Documentation

One of the main goals of our work was to provide and publish the software artifact, that is, the classical algorithms, in a readily available, permissively licensed package that is easy to extend. The source code and the related documentation can be found and downloaded from Github [10]. During the development, we tried to minimize the internal and external dependencies. This allows researchers and practitioners select only those parts that are relevant to their work. This section was written to clarify decisions behind including the dependencies, and to help the reader to gain understanding of the overall performance considerations, structure, and the use of the library.

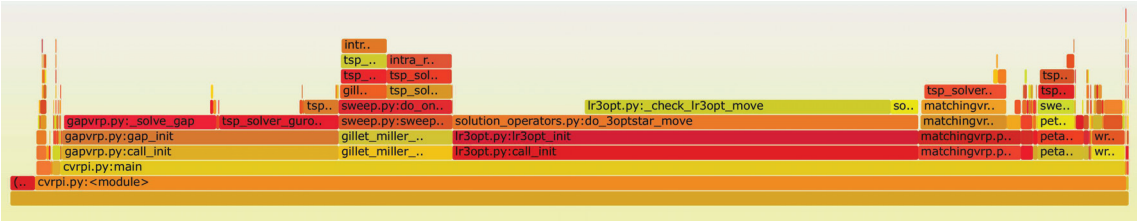
5.1 General Performance Considerations

Our main concern in implementing the algorithms for VeRyPy was the replicating of the results. However, we also made sure the code remained easy to read, understand, and extend. On the other hand, while appropriate data structures and low level algorithms were used when implementing the performance critical parts of the algorithms, reaching state-of-the-art performance nor computation speed was not our top priority. This design choice allowed us to fully utilize the strengths of

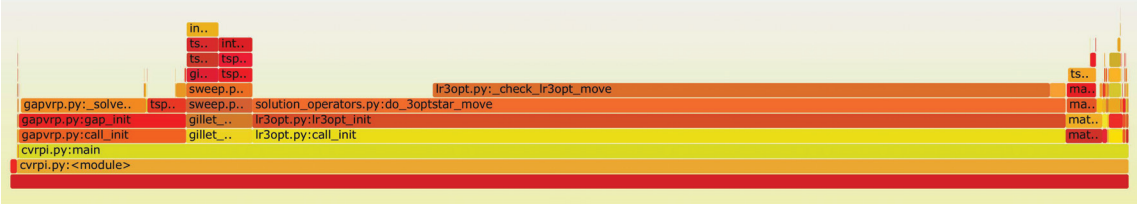
[10] `https://github.com/yorak/VeRyPy`

Figure 18: Flame graphs of the profiling data on three runs with all algorithms

(a) Gaskell (1967) instance number 4, total time 8.3s



(b) Eilon et al. (1971) instance number 8 (eil51), total time 20.8s



(c) Eilon et al. (1971) instance number 10 (eil101), total time 5m 34s

Python as an expressive, high level, multi-paradigm language (Perez et al., 2011). Also, Python standard library comes with optimized implementations of usual data structures and low level algorithms which saved us effort to implement them or including them from extension modules, which, in turn, made it possible to keep the number of external dependencies small.

We acknowledge that Python as an interpreted and dynamically typed language can cause significant performance overhead compared to, say, algorithms implemented in C or Fortran (Cai et al., 2005). Also, utilizing multiple processor cores effectively to speed up the computations is difficult due to the global interpreter lock (GIL) of the CPython interpreter. Despite these considerations, our a Python implementation still allows several alternative possibilities of improving the performance if the need arises, which are explored in detail below.

When doing performance analysis and code optimization, it is critical to recognize and concentrate on those parts of the algorithms where most of the computing time is spent (Graham et al., 1982). This is typically done using a profiler tool that gathers information from a running program. We used the PyFlame tracing profiler (Klitzke, 2016) to gather the necessary profiling data and flame graphs from Gregg (2016) to visualize it. Each box in the flame graph represents the time spent in their corresponding functions, the call stack depth is shown by the vertical stacking of the boxes, and the horizontal span roughly corresponds to the time spent in those functions. Colors do not have a significant meaning but help to separate the layers from each other.

The profiling data for solving three classical problem instances using all of the 15 classical algorithms is visualized as a flame graphs in Figure 18. The first CVRP problem instance is from Gaskell (1967) and has 32 customers with the maximum route duration constraint in addition to the capacity constraint. The other two are the 50 and 100 customer capacitated problem instances from Eilon et al. (1971). Few things quickly become apparent when observing the flame graphs. Firstly, the poor quality of the route duration approximation in FJ81-GAP, especially when combined with the trial and error approach required to find feasible solutions, make that algorithm to use disproportionate share of the computing time (see Figure 18a). Almost two-thirds of the total computing time is spent in function `_solve_gap` which is called to repeatedly solve Generalized Assignment Problems with Gurobi. However, the share of total computing time used by FJ81-GAP is smaller for the problems without the maximum route duration constraint, and as the number of customers increases the SG84-LR3OPT starts to use a greater share of the total time (see Figures 18b&c). This is mostly due to the time complexity of 3-opt* and amplified by the pure Python implementation that was used to implement SG84-LR3OPT. If FJ81-GAP and SG84-LR3OPT are ignored, one can see how a significant part of the CPU time is spent solving TSPs. Therefore, we can recognize these three as bottlenecks and promising future targets for performance improvement efforts.

The prominence of TSP solving in the profiling data is not a surprise, as many classical algorithms make the individual routes r-optimal by solving the route as a TSP with 2-opt, 3-opt, or Lin-Kernighan algorithms. If the maximum route distance constraint L is set, then TSP is solved repeatedly in some algorithms to make sure that the emerging route stays feasible. For example, the sweep algorithm from Gillett and Miller (1974) uses the following heuristic to fix a maximum route distance constraint violation: the customers are removed from the emerging route in the reverse insertion order until feasibility is regained. This involves solving a TSP each time after a customer is removed.

For state-of-the art heuristic solving of TSPs, we used the Lin-Kernighan implementation of Helsgaun (2000, 2009). We would like to point out that there is a slight overhead in the way LKH is invoked in VeRyPy: Whenever there is a need to solve a TSP, we use Python's `popen` to fork a new process for the LKH executable. This method also requires writing a parameter file for the executable and processing the LKH output from the `stdout` and output file. A more efficient approach would be to wrap the LKH code as a Python module, where it would not require disk access. This is possible as the authors of LKH have made the source code publicly available. Unfortunately, the LKH code heavily relies on the use of global variables, which makes it hard to use or extend to a callable extension library. Also, the license of LKH source code is not very permissive as it distributed only for research use and the original authors reserves all rights to the code. This makes contributing to the code base complicated. Therefore, we decided to use the unmodified LKH executable and call it from the Python. We have also included an option to use ACOTSP (Stützle, 2002) as a TSP solver, and the standalone `acotsp` executable can be called similarly to LKH. Our slightly modified ACOTSP (available from `https://github.com/juherask/ACOTSP`) allows disabling the ant systems, disabling randomization of route order, and initializing routes with the

nearest neighbor procedure. This facilitates the use the extremely fast pure C language implementations of the 2-opt and 3-opt local search that comes with ACOTSP and allows it to be deterministic.

For exact solving of TSPs, we use Gurobi and a modified version of the TSP example model provided as a part of the online reference documentation[11]. A more specialized exact TSP solver such as the Concorde TSP Solver (Applegate et al., 2006) could be faster on large problems but the TSP instances we faced were small enough to be solved with a generic MIP solver.

If the overhead of calling the LKH executable begun to dominate (when the number of nodes to solve the TSP was very small), we also had the option to use the internal intra-route local search implementations. Please see Table 18 for our experimental results comparing our Python implementation of 3-opt, LKH with default parameters, our modified ACOTSP with ants disabled and with deterministic local search, and the Gurobi TSP solver. Problems of two small classical VRP instances were interpreted as TSPs and also the nine of the TSPLIB (Reinelt, 1991) instances with less than 100 customers were included in this comparison. Solvers were run single-threaded on a machine with Intel Xeon Platinum 8168 CPU. The results suggest that after the route size grows over around 10 customers long, the overhead of calling LKH becomes negligible, and LKH becomes the recommended method of solving TSPs if the accuracy is the main concern. However, there is a tradeoff to be made between speed and accuracy: if the routes are long with many customers, one has the option to use the ACOTSP based local search solver. Please note that these results apply to these specific implementations and, for example, the built-in 3-opt could be made significantly faster by using the well-known TSP r-opt search acceleration methods (see, e.g., Helsgaun, 2000; Funke et al., 2005).

Also, we would like to point out the differences in resulting TSP tour length between ACOTSP and the VeRyPy built-in 3-opt heuristic. This variation in the quality of the solutions illustrates the point made in Section 2 that the search order and move acceptance decisions can have a major impact on the result, even in cases where both of the solvers use the same heuristic optimization method.

Besides algorithm choice, using upper and lower bounds is another option to improve the performance for cases with a maximum route length constraint. As noted above, solving TSPs may take a significant portion of the total CPU time of an algorithm, and the effect becomes more pronounced in cases where the maximum route length constraint is tight compared to the capacity constraint. Calculating and updating a TSP lower bound (e.g., with Held-Karp algorithm Held and Karp, 1970, 1971) for the emerging routes would allow avoiding costly re-computation of the cheapest TSP tour. This re-computation is required when algorithms check if inserting a customer to the route would break the maximum route length constraint. In some cases our implementations already utilizes upper bound: the inserted customer is naively appended to the route, and the actual TSP solution is checked only after the upper bound violates the constraint. A tight lower bound would allow more accurately to determine when to check the possibility to insert the customer without the computational expense of solving the actual TSP. The issue of excessive TSP constraint checks is apparent, for example, in Christofides et al. (1979)

[11] `http://examples.gurobi.com/traveling-salesman-problem/`

Table 18: Comparison of the TSP solvers used in VeRyPy. Results in bold typeface are optimal (length) or fastest (time).

problem		length, time (s)							
file	size	ACOTSP		Gurobi		LKH		built-in 3-opt	
eil7.tsp	7	**66**	0.00	**66**	0.00	**66**	0.00	**66**	**0.00**
eil13.tsp	13	**142**	**0.00**	**142**	0.00	**142**	0.01	**142**	0.01
ulysses16.tsp	16	**8150.7**	**0.00**	**8150.7**	0.01	**8150.7**	0.02	8155.4	0.01
ulysses22.tsp	22	8362.1	**0.00**	**8300.5**	0.02	**8300.5**	0.02	**8300.5**	0.01
att48.tsp	48	3404.0	**0.00**	**3377.0**	0.10	**3377.0**	0.03	3408.0	0.47
eil51.tsp	51	440	**0.01**	**426**	0.07	**426**	0.06	435	0.92
st70.tsp	70	684	**0.01**	**675**	0.20	**675**	0.12	694	2.76
eil76.tsp	76	560	**0.01**	**538**	0.09	**538**	0.10	543	5.77
pr76.tsp	76	112866	**0.01**	**108159**	2.42	**108159**	0.39	111548	4.77
gr96.tsp	96	55626.3	**0.01**	**53669.6**	2.11	**53669.6**	0.23	54600.2	3.47
rat99.tsp	99	1217	**0.01**	**1211**	0.41	**1211**	0.09	1264	6.74

2-phase heuristic CMT79-2P, where every non-routed customer is tested for feasible insertion for each emerging route. This leads to excessive number of expensive feasibility checks. A performance improvement would be keeping track of the current smallest available demand, as this would make it possible to terminate the search for capacity-feasible insertions early. Similar modification would be possible for the minimum route length checks via calculation of a lower bound for the TSP-optimized route length. However, these modifications would involve keeping track and updating several bounds, which would add significant complexity to the implementations. As the performance was not the main concern, these changes were not implemented but is documented here as part of the performance considerations.

Still, optimizing the built-in local search algorithms would be a good target to increase the overall performance of the library and to remove external dependencies. Especially the algorithms that heavily rely on using the internal local search module to solve TSP or improve VRP solutions (that is, most of them) would benefit from faster and more efficient local search implementations. Some easy options to improve the performance of the current implementation is to utilize Numba (Lam et al., 2015), which allows compiling the low level local search Python code to native machine instructions. Similar enhancement could be achieved by porting the local search module code to Cython (Behnel et al., 2011) which should allow achieving C-level performance with minor changes to the local search module code. Also, parallelization with multithreading

Further performance could be gained with the cost of additional code complexity, for example, by adapting the ideas from *sequential search* (Irnich et al., 2006), in which the move operators are systematically decomposed and a move can be rejected already after a partial evaluation. Another typical improvement technique is *pruning* those parts of the neighborhood that are less important (Funke et al., 2005) by considering only those customers that are close to each other. This approach can be parametrized as proposed in *granular neighborhoods* of Toth and Vigo (2003a). Further speed-up techniques such as dynamic programming, branch-and-bound, utilizing network optimization algorithms for shortest paths or cycles,

matchings, and keeping auxiliary data structures that allow faster evaluation of the objective function and feasibility checking are also possible, again with the cost of added complexity. Furthermore, techniques such as *fixing edges*, *candidate lists*, and *combinatorial structure approaches* for accelerating VRP local search, as listed in Funke et al. (2005, Section 3). All these could be adapted to further accelerate the local search implementations of VeRyPy.

While we do not use the more sophisticated and complex techniques of the previous paragraph, our implementation includes some performance optimizations. Currently, the internal local search module has been implemented in a way that makes constraint checks and objective calculations constant time (Savelsbergh, 1992, see, e.g.,). The `do_local_search` function also keeps track of the routes and operator combinations that have already reached the local optima and it re-applies the operators only when the situation changes. However, it should be acknowledged that this could be further improved by extending this to individual moves similarly to static move descriptors (SMD) of Zachariadis and Kiranoudis (2010).

All local search operators of VeRyPy can optionally take a required level of improvement as an argument, which allows terminating the search if it becomes clear that a good enough improvement cannot be made (similarly to sequential search). This can reduce search effort, for example, when multiple operators are compared and only the best move is applied. However, after the `do_local_search` function terminates, this auxiliary information is lost. Thus, warm-starting the local search and TSP solver could be another interesting improvement. Currently this is limited to passing the existing solution from the previous checks as the base solution for the modified problem.

There are also some further potential performance improvements in relation to how Gurobi is used. Most of the relaxed mathematical programming methods solve the same mixed integer problem repeatedly, where our implementation instantiates a new model every time. In some cases it would be possible to just update the constraint coefficient matrix values, or modify the model using column generation functionality of Gurobi. This would sometimes even allow a warm start for the solver.

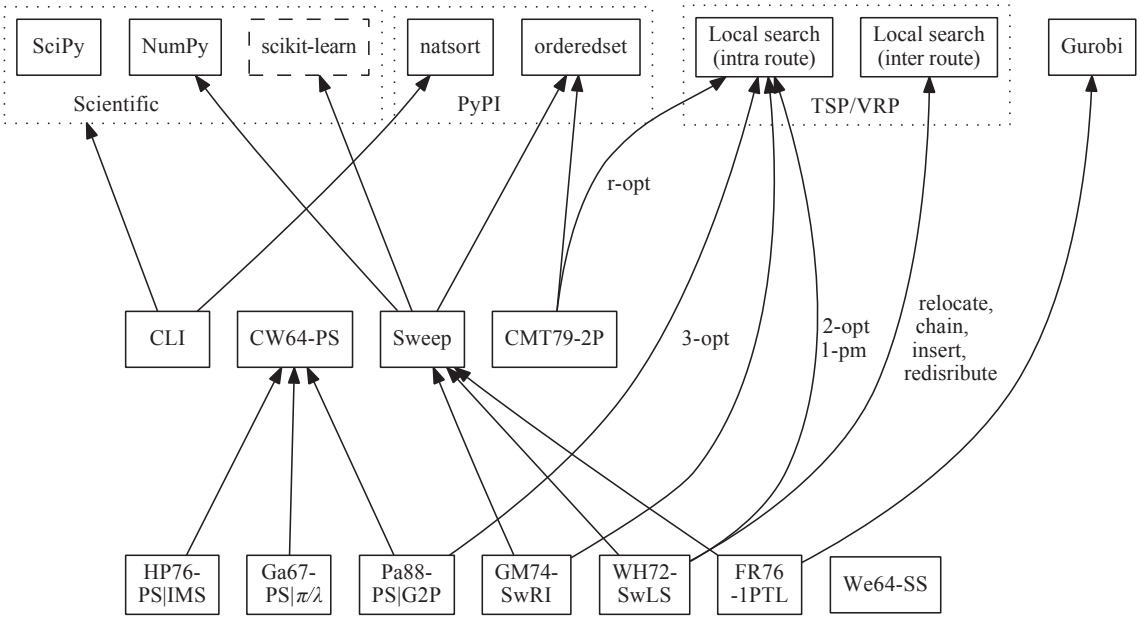
The VeRyPy software library comes with unit, integration, and result replication tests. The unit tests mostly concentrate on the local search component, which, as a result, has good test coverage. Also few of the functions implementing some part of the classical algorithms have unit tests if the functionality was easy to separate from the main algorithm code and the implementation had features that made it error prone. The integration tests are mostly smoke tests that do not throughly test the functionality of the algorithms, but recognize failures in the most typical code paths. Finally, the replication tests are used to produce the data presented in Section 4. These test the algorithms on a limited amount of problem instances, but usually quickly catch incorrect operation or regressions after code modifications.

5.2 Code Structure and Dependencies

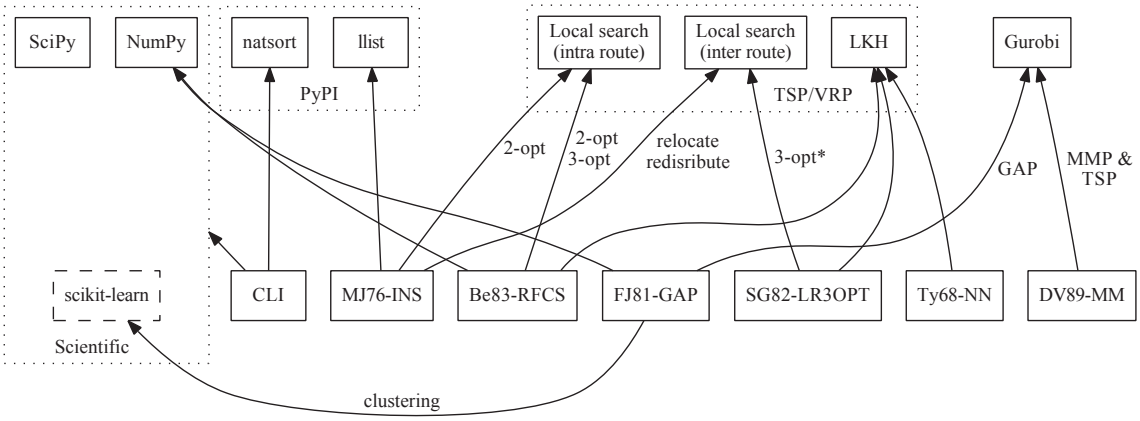
We tried to keep the dependencies to external code and libraries minimal (they are illustrated Figure 19). However, for some operations, such as calculating the distance matrix D, VeRyPy uses NumPy to make vector and matrix operators fast

and convenient. Numpy is also used by the sweep algorithms WH72-SwLS, GM74-SwRI, and consequently in FR76-PTL and FJ81-GAP, which borrow some of the functionality from the sweep implementation. In addition, CMT79-2P two-phase heuristic and Floyd's algorithm in Be83-RFCS rely on NumPy to implement the algorithms.

Figure 19: Algorithm dependencies to the external modules, executables and local search



(a) Savings and sweep algorithms



(b) Insertion, nearest neighbor and relaxed mathematical programming algorithms

Scipy's spatial module is only used to make distance matrix calculation more convenient but this dependency would be trivial to remove by implementing it in pure Python. The dependency can also be ignored if the distance matrix is given as part of the problem description. Also, please note that algorithms WH72-SWLS, GM74-SwRI, FR76-PTL, and FJ81-GAP require the coordinates of the depot and the customers to be known. Scikit-learn is used to generate missing coordinate points with SMACOF if only the distance matrix is given. Additionally, Scikit-learn's

clustering algorithms are used in FJ81-GAP, where they are used to implement optional GAP seed generation methods.

Omitting the few exceptions mentioned above, the use of NumPy, SciPy, and Scikit-learn are well contained in the CLI (command line user interface) code and removing these dependencies is relatively straightforward task, if the user so wishes. The algorithms themselves do not depend on NumPy, SciPy, or Scikit-learn, unless otherwise explicitly stated above. Thus, by design, the algorithms can easily be used as stand-alone solvers: while the CLI requires those libraries in order to read the TSPLIB formatted problem files and prepare the data, these dependencies can be ignored if the necessary problem details can be provided to the algorithm directly.

Regarding the third-party Python modules, MJ76-SI uses the *llist* module[12] and its doubly linked list `dllist` data structure to make insertions constant time operations. Additionally, WH72-SwLS, GM74-SwRI, and CMT79-2P require the `OrderedSet` module[13] to maintain the order of the non-routed customers or emerging routes. Finally, the CLI script uses `natsort` module[14] to solve the `.vrp` files in a directory in a lexicographical order. All of these packages can be installed from the PyPI Python package index using the Python package manager `pip`.

VeRyPy uses several external applications and libraries. Gurobi (2018) is used to solve mixed integer programming models in DV89-MBSA, FR76-1PTL, and FJ81-GAP. Furthermore, the TSP solver LKH (Helsgaun, 2000, 2009) is used by SG84-LR3OPT and Be83-RFCS to quickly generate an optimal or nearly optimal TSP route through all customers. LKH is also used in Ty68-NN to route the customers clustered in the first phase of the heuristic.

As can be seen from Figure 19, many of the algorithms use the built-in local search implementations. As outlined in the previous section, there are potentially quite many optimization opportunities for the local search implementation. Optionally to these improvements, the built-in local search can often be switched to an external TSP solver with the expense of adding another dependency.

As a final note on the topic of dependencies: to make the code loosely coupled to the dependencies, we have preferred to use the Python "`from <here> import <this>`" convention. This has allowed us to more explicitly state which parts of the imported module or library was used. This, in turn, makes it easier to swap the dependency to another implementation if needed. For the same reason we have used import aliasing, for example, to make changing the TSP algorithm for a heuristic a simple one import line edit.

5.3 Usage

From the user's perspective, there are three alternative ways of using VeRyPy. The first is to use the provided vehicle routing algorithms through the main `VeRyPy.py` script, which acts as an executable command line interface (CLI) to the library. One can get help by executing it with an argument:

[12] `https://pypi.org/project/llist/`
[13] `https://pypi.org/project/orderedset/`
[14] `https://pypi.org/project/natsort/`

```
$ ./VeRyPy.py --help
```

The most central command line arguments are `-a <algoname>` for selecting the classical algorithms to use and `-v <verbosity_level>` for adjusting how much output is provided. Please note that one can use `-a all` to use every algorithm provided by the library or `-a classic` to use only the classical algorithms presented in Section 4. The CLI uses shorthands for the algorithm names but also the two part names used through this study can be used. One can get a list of supported algorithms with their shorthands by invoking the CLI with the `-l` argument:

```
$ ./VeRyPy.py -l
```

The last argument of the command line invocation should always be a TSPLIB formatted (Reinelt, 1991) `.vrp` file, a folder containing `.vrp` files, or a text file with full paths to `.vrp` files. Given that the algorithm(s) and the problem(s) to solve have been specified, the program proceeds to solve all given `.vrp` files with the specified algorithm(s). However, typical command line use case would be to solve a single problem with a specific algorithm and objective. In the following example, the Clarke and Wright (1964) heuristic with a minimal amount of logging is used to minimize the number of routes K and route cost on the 50 customer problem instance from Eilon et al. (1971):

```
$ ./VeRyPy.py -v 0 -o K -a CW64-PS E-n51-k5.vrp

Solving E-n51-k5 with CW64-PS

SIZE: 51
CAPACITY: 160
DISTANCE: None
SERVICE_TIME: None

SOLUTION: [0, 6, 24, 43, 7, 23, 48, 1, 32, 27, 0, 8, 26, 31, 28, 3,
   ↪  36, 35, 20, 2, 22, 0, 10, 49, 9, 50, 29, 21, 34, 30, 39, 33,
   ↪  45, 15, 0, 12, 5, 38, 16, 11, 46, 0, 14, 25, 13, 41, 40, 19,
   ↪  42, 44, 37, 17, 0, 18, 4, 47, 0]
FEASIBLE: True
SOLUTION K: 6
SOLUTION LENGTH: 580
```

Please note that if one is solving larger problems and wishes to gain extra performance, or the CLI still outputs too much even with minimal verbosity `-v 0`, turning on the basic optimizations of the Python interpreter will disable logging altogether:

```
$ python -O VeRyPy.py ...
```

The algorithm may return an infeasible solution. Usually this happens when the algorithm is interrupted with `Ctrl+C` or with `SIGINT`. This will only happen for some algorithms, but it is recommended to check the solution feasibility after the algorithm terminates.

If one is more interested in using specific algorithms, the Python files in the `classic_heuristics` folder contain the implemented algorithms and offer a similar CLI to the main script. This, together with the minimal architecture and loose dependencies makes it possible to select only the implementation of an individual algorithm and use it independently of the CVPRI library. Below is an equivalent example to the earlier one for solving the 50 customer Eilon et al. (1971) instance with Clarke and Wright (1964) heuristic illustrating this use case:

```
$ ./classic_heuristics/parallel_savings.py -v 0 -o K E-n51-k5.vrp
```

The third option is to import the library as Python modules. This allows fine-grained access to the implementations of local search (with 2-opt, 3-opt, one-point-move, two-point-move, 2-opt*, 3-opt*, relocate, exchange, chain, insert, and redistribute operators), three TSP solvers (3-opt, LKH, and Gurobi), 20 CVRP heuristics, and functionality to reading, writing and processing vehicle routing problem files and models. An interactive Python session example that reads a TSPLIB formatted file, solves it with Clarke and Wright (1964) heuristic, and prints the resulting solution route-by-route is given below:

```
>>> from VeRyPy import cvrp_io
>>> from VeRyPy.classic_heuristics.parallel_savings import
   ↪ parallel_savings_init
>>> from VeRyPy.util import sol2routes

>>> problem = cvrp_io.read_TSPLIB_CVRP("E-n51-k5.vrp")
>>> solution = parallel_savings_init(
...   D=problem.distance_matrix,
...   d=problem.customer_demands,
...   C=problem.capacity_constraint)
>>> for route_idx, route in enumerate(sol2routes(solution)):
...   print("Route %d : %s"%(route_idx+1, route))

Route 1 : [0, 8, 26, 31, 28, 3, 36, 35, 20, 2, 22, 0]
Route 2 : [0, 18, 4, 47, 0]
Route 3 : [0, 12, 5, 38, 16, 11, 46, 0]
Route 4 : [0, 15, 45, 33, 39, 30, 34, 21, 29, 50, 9, 49, 10, 0]
Route 5 : [0, 17, 37, 44, 42, 19, 40, 41, 13, 25, 14, 0]
Route 6 : [0, 27, 32, 1, 48, 23, 7, 43, 24, 6, 0]
```

The application programming interfaces (APIs) are documented with docstrings interleaved with the code, and, as demonstrated with the example above, they should be straightforward to use. This API documentation complements this document with additional implementation details and role descriptions of different components in the VeRyPy library.

6 Computational Experiments

In addition to the reproduction of the results, we were also curious about the differences between the accuracy and performance of the implemented algorithms. It is hard to gain a clear view on the strengths and weaknesses of the classical algorithms from the literature alone, as they are rarely tested on the same instances, and even if they are, only some selected results are typically reported. Unfortunately, this has also meant that the classical algorithms have not always been tested against the best competition of the time, which is the recommended practice (Barr et al., 1995). Furthermore, the computing environments and implementation techniques have varied greatly from a publication and decade to another, which has made most computation time comparisons meaningless.

We have already touched the topics of algorithm *simplicity* (how easy it was to implement), *speed*, *accuracy*, and *robustness* in Section 4. In addition to these features, Barr et al. (1995) also lists *innovativeness* and which kind of *revealing* insight into VRP or general heuristics they offer. This all contributes to the *impact* of the research. Also relevant is the ability to *generalize* the performance over a broad range of problems. These all are desirable and important properties for a good heuristic algorithm. However, it is hard to estimate these without extensive experimentation on a wide range of different problem instances. Hence, we set out to design an experimental setup to thoroughly measure the implementations of the classical CVRP algorithms.

While the decision to use a heuristic instead of exact methods in a specific application is usually based on requirements of fast solution time, in our experiments we were mostly interested in the accuracy of an algorithm (i.e., the quality of the solutions). This aim is in line to, what is according to Barr et al. (1995), the ultimate goal of heuristic methods: providing *high-quality solutions* to important problems with small to moderate computational effort.

6.1 Experiment Design

It is important to control the factors of a computational study, especially when algorithm comparisons are made (Barr et al., 1995). In this section, we document our experiment design and setup. In designing the experiments, we followed the recommendations from Barr et al. (1995) on empirical testing of heuristic methods, because it is critical to ensure that the results given by individual algorithms are objective and comparable with each other. The selection criteria for the algorithms to be implemented and compared ensured their determinism (i.e., they return the same result for the same problem instance each time they are run) and had a well-defined termination rules, which are both aspects Barr et al. (1995) recommends considering carefully.

The primary optimization objective varies in the literature. Fixing the number of vehicles K is typical in the literature on exact methods, but the later CVRP problem sets usually allow the K to be chosen freely (Uchoa et al., 2017). The surveyed works proposing classical algorithms use two alternative optimization objectives: in some the primary objective is to minimize the number of vehicles, while others have chosen the simpler objective of minimizing the total route length. The

algorithms of VeRyPy were implemented as per their original descriptions, but both primary objectives were supported if the algorithmic procedure allowed this. For some algorithms this was trivial. For example, with CW64-PS where allowing negative savings will minimize the number of vehicles, even with the expense of total route length, or for FR67-1PLT where omitting the constraint for the number of vehicles will allow finding the combination of petals with minimum cost. However, some algorithms had been geared more towards minimizing the number of vehicles such as the FJ81-GAP, where the initial number of vehicles is determined by the to constraints. The algorithm was modified to minimize cost by allowing it to continue increasing the current available number of vehicles until no improvements were found. Of the algorithms in VeRyPy, only SG84-LR3OPT and the nearest neighbor heuristics such as the Ty68-NN do not support minimizing the number of vehicles. In Ty68-NN, minimization of the number of vehicles is built in feature of the algorithm. Modifying SG84-LR3OPT would require balancing between two parametrized penalty functions for constraint violations and increasing the number of routes. Hence, because all algorithms can be modified to support minimizing the route cost, and because adding the support to minimize the number of vehicles is problematic for the two aforementioned algorithms, in our experiments we decided to minimize the route length. According to Uchoa et al. (2017), not minimizing or fixing the number of routes has become the usual practice with the later problem sets. This, together with their rationale of doing multiple trips with the same vehicle, and the fact that fixing the number of routes was not in the original CVRP definition (Dantzig and Ramser, 1959), further supports our decision.

Regarding runtime, each algorithm was given at most one hour of wall clock time to solve each problem instance. This is more than enough for most greedy heuristics, and for the more sophisticated algorithms to solve small to medium sized problem instances. However, the computational requirements for algorithms DV89-MBSA, FR76-1PTL, and especially for FJ81-GAP and SG84-LR3OPT, can grow prohibitively large for the larger instances. In those cases, the computation was interrupted after the one hour limit and the current emerging solution (or the best solution so far) was returned. Note that reading the problem instance from the file and computing the distance matrix was included in the runtime of the algorithm for the termination criteria, but not in the algorithm wall clock (real) time of the reported results. Also, note that Gurobi MIP solver timeout was set to 10 minutes, which is separate to the overall algorithm time limit of one hour.

For some algorithms, it did not make sense to evaluate all optional variants separately. Ga67-PS$|\pi\lambda$ and MJ76-INS are very fast, and for them the solution was produced using all savings/stress functions available for the algorithm and the best result was returned as the solution. This allows more balanced and fairer comparison with the more sophisticated heuristics (Rardin and Uzsoy, 2001, p. 268).

In case the algorithm had parameters they were left to the values specified in the original publications or in Section 4. For example, the deterministic CMT79-2P was run with the default parameters $\lambda = 2.0$ and $\mu = 1.0$, and Pa88-G2P used the parameter search strategy M4.

According to Barr et al. (1995), to keep the results comparable the algorithms should not only be run on the same computer, but also written in the same programming language by the same expert programmer. The first author of this study,

who also implemented all the algorithms, has over 15 years of professional software engineering experience. He had been using Python as his primary programming language for six years prior to starting implementation work on the algorithms. Thus, the effect of these experiment factors were controlled.

The computations were run on a computing server with Python 2.7.12, NumPy 1.11, SciPy 0.17, Scikit-learn 0.17, Gurobi 7.5.2, and LKH 2.0.7. We also had an option to use a slightly modified version of ACOTSP v.1.03. Following additional packages were used from PyPI: natsort 5.3.2, orderedset 2.0.1, and llist 0.6. The server had Intel Xeon Platinum 8168 CPUs with Ubuntu 16.04.4 LTS operating system using Linux kernel 4.15.0. For orchestrating the parallel computations, we used GNU parallel 20141022 (Tange, 2011).

Considering the factors described earlier, together with the requirements of determinism and being parameter free, we were able to overcome most of the objections on competitive testing made by Hooker (1995). Next, we will continue by describing how we addressed some of the concerns in choosing the test problems (Hooker, 1995).

6.2 Problem instances

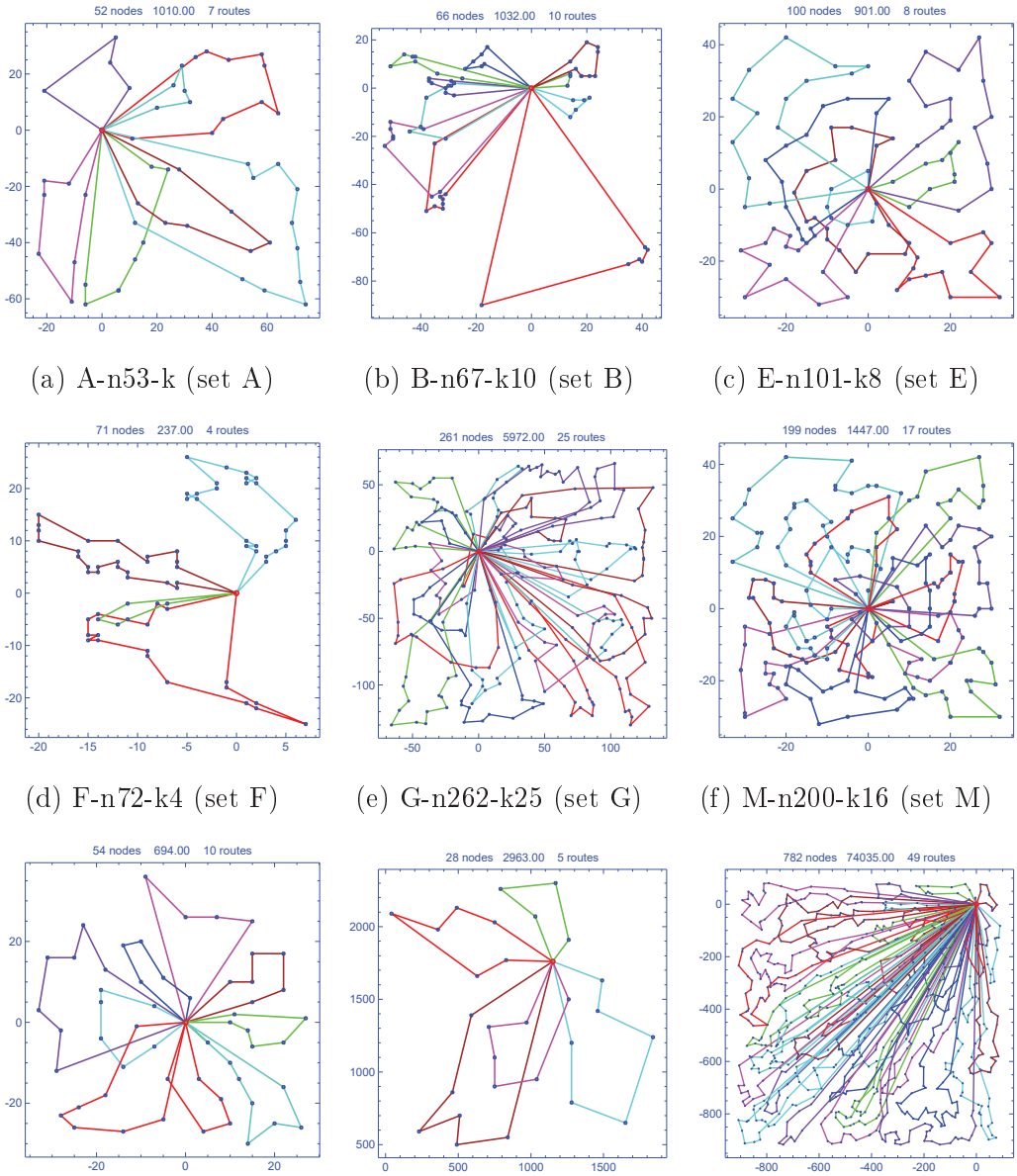
The performance of the algorithms is typically measured against a set of well known problem instances. Sometimes these instances are modified, for example, to include necessary information for additional constraints, but for a specific variant an established benchmark set can usually be recognized.

To give a comprehensive view on experimental performance and accuracy of classical CVRP heuristics, and to lessen the effect of selective advantage for any particular algorithm due to limited set of benchmark problems (Hooker, 1995), we decided to consider all instances listed by Uchoa et al. (2017) complemented with several other benchmark sets. The benchmark instances cover a large variety in problem size, generation method, and variation in how realistic they are, thus satisfying the recommendations made by Rardin and Uzsoy (2001) concerning sourcing the test instances. Understanding where the instances come from, how they are generated if they are artificial, and if there are some special remarks is relevant to the goals of this study. Therefore, short descriptions of the benchmark sets are given below, but for a more complete review of the CVRP problem instances please refer to (Uchoa et al., 2017).

Sets A,B that were proposed in (Augerat, 1995). The instances are randomly generated to have sizes (N) between 30 and 80 customers, capacity Q of 100, and average demand of 15.0 with 10% of the demands multiplied by 3. The customers in set A (27 instances) are uniformly located inside a 100×100 square. The customers in set B (23 instances) are distributed in a more clustered fashion.

Set E was collected by Christofides and Eilon (1969). It includes 6 earlier small instances from the literature and three new larger ones, with 50, 75, and 100 customers, respectively. Later, Gillett and Miller (1974) added four instances to the set by varying the capacity in the 75 and 100 customer instances.

Figure 20: Example instances from the ABEFGMPVX benchmark problem sets



(a) A-n53-k (set A) (b) B-n67-k10 (set B) (c) E-n101-k8 (set E)

(d) F-n72-k4 (set F) (e) G-n262-k25 (set G) (f) M-n200-k16 (set M)

(g) P-n55-k10 (set P) (h) bays-n29-k5 (set V) (i) X-n783-k48 (set X)

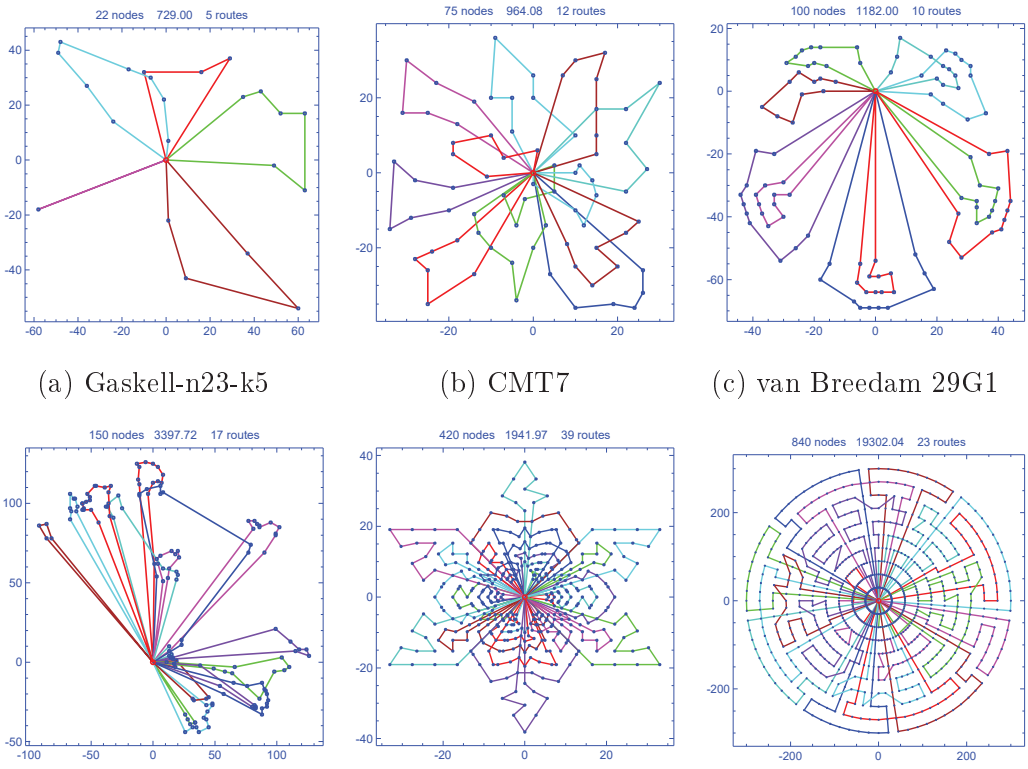
Set F contains three instances based on real world routing scenarios as reported in (Fisher, 1994) with 44, 71, and 134 customers.

G is a set with only one instance with 261 customers adapted from Gillett and Johnson (1976).

Set P also comes from Augerat (1995). The 24 instances are based on selected ones from A, B, E, and M sets, and are created by decreasing the capacity of the original problem instances.

Set M contains five instances from 100 to 199 customers. The instances were introduced by Christofides et al. (1979) and only very recently the two largest

Figure 21: Example instances from the other benchmark problem sets



(a) Gaskell-n23-k5 (b) CMT7 (c) van Breedam 29G1

(d) Taillard_150A (e) Golden20 (f) Li_27

instances with 199 customers in this set have been solved to optimum (Pecin et al., 2017). These two largest instances are interesting also because, according to Uchoa et al. (2017), the 17 route version of these problems is the only instance in the entire collection where the fixed number of routes is less than the optimal number of routes.

Set V contains 13 instances that were converted from the TSPLIB instances (Reinelt, 1991). They are interesting as they vary greatly in size and origins.

Set X Uchoa et al. recognized in 2017 that existing CVRP benchmarks were too artificial, too homogeneous, and had become too easy for the state-of-the-art algorithms. They proposed a new problem set of 100 instances. The instances have sizes from 100 up to 1000 customers with varied attributes in depot positioning, random customer and demand distribution, and average route customer count.

All distances above are given as integers or rounded to the nearest integer. The ABEFGMPVX collection was further completed with six other benchmark instance sets from the literature:

Gaskell instances were popular in papers that introduced early heuristics. The size of the problems is quite small, as all have under 36 customers, but five of the six problem instances have maximum route duration constraint, and only the

largest of the instances are synthetic, with the rest being derived from a set of real world transportation problems (Gaskell, 1967). These properties make the problem set interesting. The problem set also contains the real world problem instance from Clarke and Wright (1964), with the corrected value for the one distance that had been misprinted in the original publication. The value in (Gaskell, 1967, Table A4) seems to be a misprint as the 210 is also given along with the sample solution in the paper. Thus, for Case 4 we used the maximum route cost value of 210. All distances are rounded to the nearest integer.

CMT is another well-known set of 14 problem instances (Christofides et al., 1979), on which the sets E and M were based on. While the demands and node locations are identical, the distances between nodes in the CMT set are calculated using real floating point accuracy instead of rounding them to the nearest integer. Therefore, despite the same origin, the corresponding instances in E and M have different solutions. Also, half of the instances in this set include a maximum route duration constraint. Thus, despite their similarities, including CMT set in this study is well justified.

van Breedam introduced a synthetic set of 20 geometric problem prototypes (with spread patters: uniform, concentric, 50% central, and compressed which were combined with grouping patterns: singleton, clusters, 50% clusters, cones, 50% cones), which after combinations with different depot locations (central, inside, and outside) and capacities (50, 100, and 200) produce a benchmark set of 180 instances. Size of each instance in this set is 100 customers and demand of each customer is 10. He used this problem set to analyze the behavior of VRP heuristics and the effects of their parameters, which makes it an ideal addition to this study (Van Breedam, 1994). The set uses real distances.

RT contains 13 instances with real distances. Only one of the instances (`tai385`) is based on real geographical data, but others are generated as described by Rochat and Taillard (1995). Their generator produces real-world-like non-uniform routing instances with sizes from 75 to 150 and variable number of clusters. Customer demands follow exponential distribution.

Golden set was proposed in Golden et al. (1998) and it contains 20 instances where customers (N ranging from 240 to 420) are placed in concentric geometric figures: stars, squares, circles, and rays. The demands of the customers also follow this symmetry. Instances use real distances, are homogeneous, and difficult to solve effectively using heuristics (Uchoa et al., 2017).

Li is a set of 12 large-scale duration-constrained instances from Li et al. (2005). The instance size goes from 560 up to 1200 customers and can be considered to be an extension of the Golden problem set. Like in Golden set, the instances are geometric (synthetic) with the clients placed on a sector like symmetric structures with real distances.

Figures 20 and 21 show one instance from each of the benchmark sets. From the figures reader can get an impression the differences and similarities between the CVRP benchmark instance sets. However, the instance to instance variation

within a problem set can be large, and it should not be assumed that all of the instances in the set are exactly similar to the one presented in the figures. The problem instances were plotted using the visualization tool that comes with VRPH (Groër et al., 2010). For those readers looking for a more in-depth analysis on the differences between classic CVRP instances, please see (Steinhaus, 2015, p.84-) and (Rasku et al., 2016), which both include a look into problem characteristics and the diversity in benchmark problems.

The real valued distance matrix is usually calculated using the native floating point accuracy of the computation platform. However, during our replication efforts we made the same observation as Fisher (1995) that some researchers have rounded the distances to the nearest integer, or in some cases even truncated to the next smaller integer. The rounding convention used can have a significant effect on the solution cost (Fisher, 1995). Additionally, there seems to be some confusion on how to calculate the geographical `GEO` distances in TSPLIB files. We used an implementation for the `GEO` distance calculation that follows the specification in Reinelt (1991) and reproduces the optimal solution cost for the original TSPLIB instances `ulysses16.tsp` and `ulysses22.tsp`. After changing the SYMPHONY geographical distance calculation to follow the specification of TSPLIB, we got the optimal values 8077 and 9291 for these two instances.

For the other problem instances, to get the optimal or best known solution values we relied on (Uchoa et al., 2017) and also consulted other literature sources. Please note that because we optimized for the route length, and did not minimize the number of vehicles, some found results may be better than the optimal solution for a fixed K. This happened for B-n51-k7, B-n57-k7, E-n30-k3, P-n22-k8, P-n55-k15, and P-n55-k8, where we used the SYMPHONY to find the optimal solution for a larger number of vehicles than the minimal one. Here, VRPH was used to provide an upper bound for SYMPHONY. If SYMPHONY failed, we used VRPH to find a best known solution. Additionally, only few best known solutions seem to have been published for the van Breedam instances (Alba and Dorronsoro, 2006). We made an attempt to solve these problem instances exactly with SYMPHONY and succeeded to find new optimal solutions for 65 of the 180 van Breedam CVRP instances. Again, if the optimal solution was not found, the heuristic solution value from VRPH was used as the best known solution value for that instance.

If the coordinates for the customers and the depot were not specified in the problem instance file, our implementation generates this missing data from the distance matrix using multidimensional scaling algorithm SMACOF. While SMACOF is a stochastic algorithm, we use a fixed seed for the random number generator to get reproducible results. However, please note that the generated coordinates are not necessarily the same as those that are used in the computational experiments in the earlier literature.

Usually the route length stayed quite short in the CVRP academic benchmarks and each route serves around 10 customers. Of the benchmark sets presented above, only instances in the Golden et al. (1998) and Li et al. (2005) sets have routes with significantly longer routes. As an extreme example, some Li et al. (2005) problem instances require that a route can serve over 100 customers. Finding a solution for these instances with the maximum route duration constraint required fast and effective TSP solvers.

During collecting the problem instances for the experimental study, we observed some inconsistencies in the publicly available problem instance data compared to the original data tables in the literature. The 30 customer instance in the set E is subtly different from (Clarke and Wright, 1964) where it originates from. As can be seen from (Clarke and Wright, 1964, Appendix II, Midlands II table) "Birmingham", or the customer 19, is listed twice. Taking into account that there is 20 hundredweight (Cwt) in a ton (T), the first has a demand of 33 and the other demand of 140. However, the one with a demand of 140 is missing from the problem instance `eil31/E-n31-k7`. This second customer in "Birmingham" with the demand of 140 must be served with a separate one customer route due to the demand being exactly equal to the vehicle capacity of 140. Thus, it can be omitted from the optimization problem with the cost of 164 miles induced by this single delivery added after the problem has been optimized. However, the TSPLIB file format does not have a support for adding constant terms to the objective function. If the customer with the demand of 140 is modeled as an additional customer, then all results are no longer replicated because some heuristics, such as the sweep heuristic from (Gillett and Miller, 1974), rely on customers forming consecutive chains. The issue with the Clarke and Wright (1964) problem instance has been noted earlier by Beasley (1983). Furthermore, while replicating the results we noticed that there was an error in the problem set E data: `E-n33-k4.vrp` has the depot at $(292, 495)$, whereas the coordinates given in (Gaskell, 1967) are $(292, 425)$. However, as the instances are different due to CVRPLIB version lacking also the maximum route duration constrain, we decided to include `E-n33-k4.vrp` without correcting the depot coordinate and include the original 32 customer Gaskell (1967) instance as part of the Gaskell problem set.

To summarize, a total of 454 well-known benchmark instances were collected from the literature to carry out our experimental study. To our knowledge, classical algorithms have not been previously tested on such a variety of problem instances. We acknowledge that using a problem generator would have been another option to make an extensive experimental study, but randomly generated problems from a single source are often quite similar to one another, and typically produce only slight variations of the same patterns (Rardin and Uzsoy, 2001). We expected evaluating the performance and accuracy of the implemented algorithms on instances from varied sources to better reveal the algorithms' respective strengths and weaknesses.

6.3 Results

After an initial run, it turned out that FR76-1PTL, CMT79-2P, and FJ81-GAP are not able to produce feasible solutions for all of the problems in our problem sets with their default settings. Thus, some modifications were needed to the algorithms and their parameters to solve some of the renaming problem instances. A more generous Gurobi time limit of 1 hour was attempted to help FR76-1PTL and FJ81-GAP to find feasible solutions on the remaining instances. The rationale was that the more generous timeout will allow Gurobi to solve the models that eventually will converge to optimum. Also, for FJ81-GAP, a stricter timeout of 1 minute should allow wider exploration of several iterations in case some of them have abnormally long MIP solution time (which is to be expected. See, e.g., Koch et al., 2011).

These changes allowed us to produce feasible solutions for some of these problematic problem instances.

In its default configuration (as described in Section 4.5.4), FR76-1PTL is unable to produce results for 8 of the 12 Li instances and for some of the larger ones in the X problem set. For some, it seemed that TSP procedure took too long to generate all petals, and for others, the default Gurobi time limit of 10 minute was too strict. After generating the petals using LKH (with only one iteration), increasing the Gurobi time limit to one hour, and starting straight from the extended petal set, we allowed us to get a feasible solution for all but the four largest problems in Li et al. (2005) problem set. Note that all problem instances in the Li set have the maximum duration constraint, and three out of these four problem instances have over 1000 customers. Thus, FR76-1PTL only partially manages to fulfill the criteria on the problem size as it is able to solve the 1000 customer instances in the X set, but not those in the Li set. Ehen considering the results, one should keep in mind that the algorithm is interrupted on many of the large instances. This is because the algorithm did not have the time to trigger the termination rule. Also, related to these issues, our implementation does not allow postponing the TSP routing of the complete petal set, which, due to the large number of petals to be generated for the larger instances, may be the reason the last few Li problems remain unsolved with our implementation.

The CMT79-2P has a similar issue as FR76-1PTL with the Li et al. (2005) problem set. When a problem instances has the maximum route constraint, rejecting the remaining customers from the emerging route can only be done *after* solving the TSP with that customer included on the route. When the routes become sufficiently long, the time spent by this rejection procedure starts to dominate. Christofides et al. (1979) did not explicitly specify the method used for the feasibility check, but we assumed that their method does not include calculating and updating lower bounds for the insertions. We use a lower bound, but it only saves computational effort when there is still room in the emerging route. Hence, using a lower bound does not allow rejecting customers in this case. To make CMT79-2P solve these instances, we changed it to use the faster ACOTSP solver, which allowed it to solve all the remaining large instances.

Compared to FR76-1PTL and CMT79-2P, the failures with FJ81-GAP are more common and less systematic. Especially FJ81-GAP seems to have difficulties with the large problems and those instances with maximum route duration constraint. When replicating the results from Fisher and Jaikumar (1981) (see Section 4.5.3) we discovered that it may take hours for FJ81-GAP to find the first feasible solution for the larger CMT instances with maximum route duration constraint. Our later computational experiments revealed that the same applies to most of the maximum route duration constrained Golden and Li instances. Furthermore, many problem instances in the X set remained unsolved for FJ81-GAP. Enabling an option to allow at most 10% temporary increase in vehicle count K for seed configurations and varying the Gurobi time limit first from 10 minutes to 1 minute and then to 1 hour allowed our FJ81-GAP implementation to find feasible solutions for further 64 cases that it was unable to solve with the default parameters. Temporary increasing seems to have the biggest effect as it allows the algorithm to increase the number of vehicles earlier in the search process, an important variation to the search scheme in

time constrained runs. Further increasing the K up to 50% of its initial value allowed to solve additional 21 of the remaining problem instances. After these additional experiments, our FJ81-GAP implementation is able to find a feasible solution for all of the Golden instances, for 11 of the 14 CMT instances, and for 9 of the 12 Li instances. However, ten of the X instances remain unsolved. These seem to be those with a large number of routes in relation to the number or customers. It also seems some of the solving attempts on the remaining problem instances lead to pathological GAP models, which are difficult for the Gurobi MIP solver. We tried to use the Gurobi `Model.tune()` procedure to automatically configure its parameters and after tuning Gurobi for 40 CPU days on a single GAP model we got a result that recommends the use of `MIPFocus 3` to make Gurobi concentrate on the bounds and somewhat ignore the objective function. Unfortunately, this did not help us to solve the remaining instances, and 16 of the 454 problem instances remain unsolved for our FJ81-GAP implementation given the one hour time limit.

It can be argued that tuning the parameters for the few problematic cases for CMT79-2P, FR76-1PTL, and especially FJ81-GAP gives these algorithms some unfounded advantage. However, we argue that the issues with these algorithms were to some extent implementation specific, and we decided to make an effort to allow them to produce a feasible solution if it was possible without deviating too far from their original descriptions.

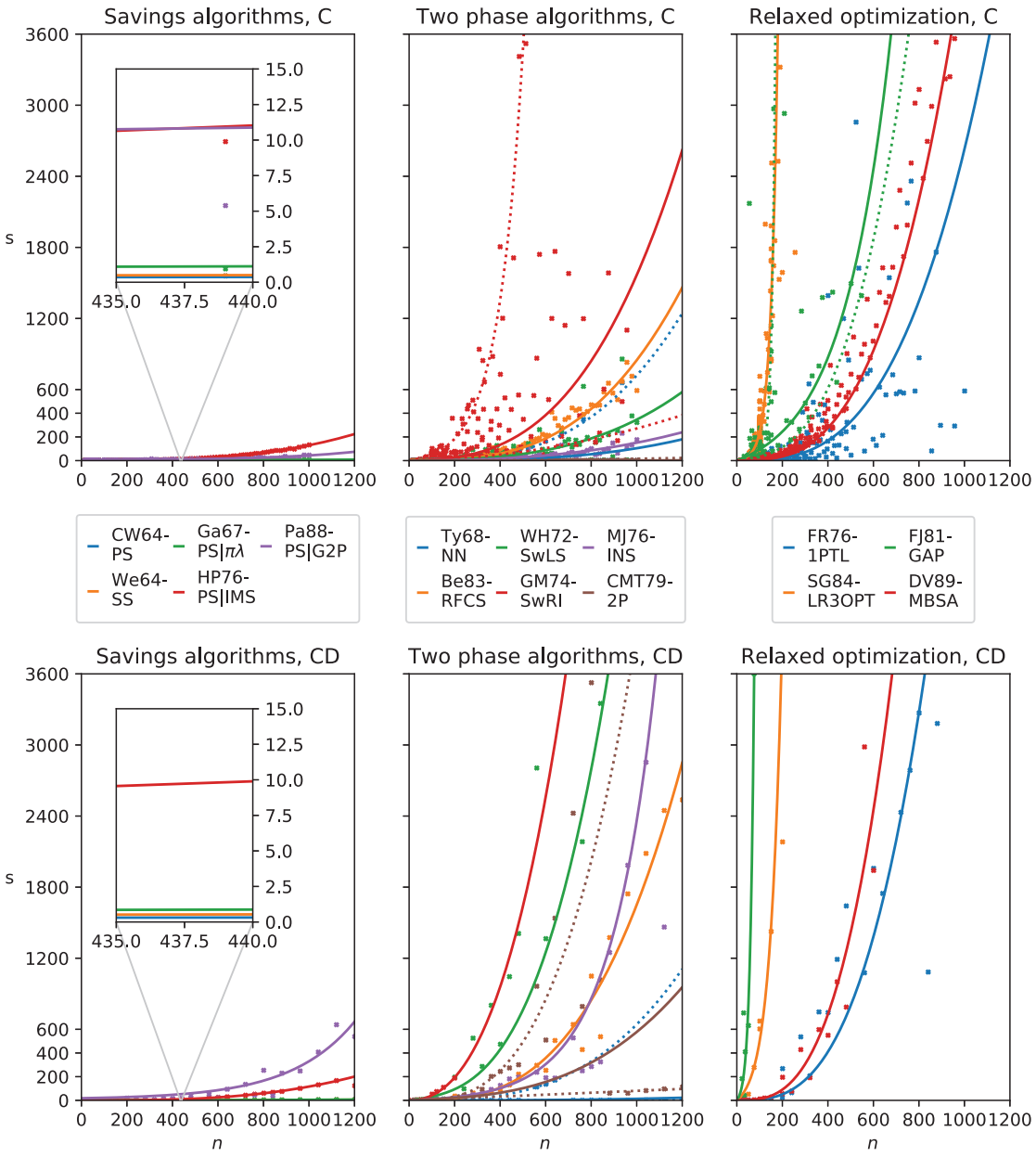
Moving on to the performance of the heuristics: Computation times of different classical heuristics are hard to compare because the variety of computational setups used in the original experiments (Laporte, 2009). Our experimental setup allowed us to control the factors from all thee categories: problem, algorithm, and environment (Barr et al., 1995). Thus, we can present the first controlled analysis on the computation time differences between the 15 classical heuristics. We acknowledge that some of the differences in algorithm runtimes are still implementation specific, but a some general remarks can be made.

To best illustrate the relative differences in the speed of the algorithms, an attempt was made to fit the algorithm runtimes to a set of time complexity curves. The most typical curves (linear, quasilinear, quadratic, cubic, several different exponential, and factorial time) were tried for the results of each algorithm and the best fit was kept and plotted. Fitting was done using Scikit-learn's `curve_fit`, or when there were clear outliers or multimodality in the algorithm performance data, we used `xcrvfit`, which is an interactive program to fitting the parameters of a function to a given set of scientific measurement data (Boyko and Sykes, 2012). The results where the time cap was reached were removed from this analysis and the curve fitting was done separately for capacity constrained problem instances (C) and for capacity and maximum route duration constrained problem instances (CD).

As can be seen from Figure 22, the savings algorithms CW64-PS, We64-SS, and Ga67-PS|$\pi\lambda$ are among the fastest algorithms in this study. They all have a very similar quadratic $\mathcal{O}(n^2)$ time complexity with very small coefficients. The more sophisticated savings algorithms HP76-PS|IMS and Pa88-PS|G2P seem to have a higher time complexity, which based on the curve fit is cubic $\mathcal{O}(n^3)$ or higher. However, also these methods are able to solve all of the problem instances in a reasonable time.

Of the two-phase algorithms, CMT79-2P can be very fast and its best case per-

Figure 22: Time complexities for the algorithms derived from the recorded runtimes. Dashed lines are worst and best case complexities for those algorithms that have a bimodal runtime distribution.



formance seems to be very close to linear time complexity $\mathcal{O}(n)$. However, especially for problems with the maximum route duration constraint, its runtime grows significantly as a TSP has to be solved before a non-routed customer can be rejected. Same happens for MJ76-INS, where the insertion feasibility checks for the route duration constraints starts to dominate as the problem size grows: when the maximum duration is more constraining than the capacity, rejecting the insertions is costly due to solving TSPs with 2-opt for each candidate. This causes MJ76-INS runtime distribution to appear bimodal. The Ty68-NN algorithm has a similar bimodal runtime disribution but for a different reason. We remind that in Ty68-NN

the nearest neighbor greedy heuristic can be run several times if a single customer route is formed during the initial route building step and solving such problem instances takes significantly longer. Therefore, while the typical time complexity of the Tyagi heuristic seems to be quadratic $\mathcal{O}(n^2)$ with a small coefficient, our best curve fit becomes cubic $\mathcal{O}(n^3)$ for problem instances with a single customer route.

The runtime of the GM74-SwRI sweep algorithm varies a lot depending on the structure of the particular problem instance. The reason can be found from the details of the improvement procedure. If a potential improving move is not rejected early, then the TSP algorithm may have to be called multiple times, which, in turn, causes the runtime of GM74-SwRI to be erratic. This is in contrast with another sweep algorithm WH72-SwLS, which uses local search only after clustering the customers. As a result, the algorithm shows quite predictable computational effort with a time complexity that seems to be cubic $\mathcal{O}(n^3)$. However, introducing a maximum route duration constraint makes the local search in WH72-SwLS significantly slower due to the TSP check made in when checking for redistribute moves. Thus, when the maximum route duration is introduced, we can see how WH72-SwLS changes places with another two-phase algorithm (Be83-RFCS) in the execution time comparision. Please note, however, that Be83-RFCS is still empirically assigned to the same cubic $\mathcal{O}(n^3)$ time complexity class as WH72-SwLS.

The runtime of the relaxed optimization algorithms FR76-1PTL and FJ81-GAP are hard to predict. In FR76-1PTL, where the relaxed routes are used to find even better solutions on the next iteration, the search is terminated sometimes early (where further improvements cannot be found) or later (when relaxed petals allow exploring new combinations which in turn allow more relaxed petals to be generated). Based on our experimentation, the erratic runtime of FJ81-GAP is caused by the varying solution time of GAPs. Predicting the runtime of a MIP solver, such as Gurobi, using only the size of the problem is known to be very difficult (Koch et al., 2011). This, together with its issues in solving maximum route duration constrained problem instances with FJ81-GAP, means that the method is unable to consistently solve problems with around 1000 customers in a reasonable time. In fact, of the relaxed optimization algorithms, only DV89-MBSA and FR76-1PTL have a chance of solving the larger problems within the given one hour time budget. SG84-LR3OPT has the longest runtime of the compared algorithms and is able to solve problems up to around 200 customers in a reasonable time. The high computational requirements of the SG84-LR3OPT and FJ81-GAP can clearly be seen from the best fit computational complexity curves which are exponential $\mathcal{O}(e^n)$ as opposed to the empirical (best fit) time complexity of $\mathcal{O}(n^3)$ of FR76-1PTL and DV89-MBSA.

Now we turn to examine the quality of the solutions produced by the algorithms, which is usually measured using the optimality gap. The gap is the deviation from the best known solution and is often given as a percentage difference to the best known (or optimal, if proven so) solution. The average algorithm performance gaps between the solved instances grouped by the benchmark sets are given in Table 19. To find the algorithms that were statistically significantly better than the rest, we used the non-parametric paired samples Wilcoxon signed-rank test with Bonferroni correction. The required significance level for the tests was $p = 0.05$. The advantage of this statistical method is that it is not sensitive to outliers, which exist in our

Table 19: Average optimality gap and the wall clock times on the problem sets. Best algorithms in bold typeface.

Problem set	A	B	E	F	G	M	P	V	X	Gask.	CMT	vanB.	RT	Gold.	Li	Total
Count	27	23	13	3	1	5	24	13	100	6	14	180	13	20	12	454
Types	I^C	I^C	I^C	I^C	I^C	I^C	I^C	I^C	I^C	I^C_{CD}	R^C_{CD}	I^C	R^C	R^C_{CD}	R^{CD}	-
CW64-PS	4.6%	4.1%	11.6%	3.8%	6.7%	4.9%	6.9%	5.9%	5.9%	8.0%	7.5%	5.6%	4.7%	11.9%	11.9%	6.3%
	0.0s	0.0s	0.0s	0.0s	0.1s	0.1s	0.0s	0.0s	0.6s	0.0s	0.0s	0.0s	0.0s	0.3s	1.6s	0.2s
We64-SS	34.1%	39.7%	34.6%	47.8%	63.2%	40.5%	24.8%	17.0%	30.0%	20.9%	31.2%	23.1%	57.5%	30.5%	52.0%	29.1%
	0.0s	0.0s	0.0s	0.0s	0.2s	0.1s	0.0s	0.0s	0.6s	0.0s	0.0s	0.0s	0.1s	0.3s	2.7s	0.2s
Ga67-PS|$\pi\lambda$	6.0%	5.8%	8.1%	6.9%	5.6%	7.0%	6.1%	7.2%	7.2%	6.0%	7.9%	5.8%	7.7%	11.1%	15.6%	6.8%
	0.0s	0.0s	0.0s	0.0s	0.3s	0.1s	0.0s	0.0s	1.3s	0.0s	0.1s	0.0s	0.1s	0.6s	3.4s	0.4s
Ty68-NN	21.4%	20.1%	29.4%	27.5%	22.8%	24.8%	20.8%	15.6%	14.4%	33.6%	29.9%	8.0%	25.0%	21.8%	29.5%	15.3%
	0.1s	0.1s	0.1s	0.2s	0.2s	0.5s	0.1s	0.0s	17.5s	0.0s	0.3s	0.2s	3.4s	3.9s	36.5s	5.2s
WH72-SwLS	5.4%	2.8%	5.4%	4.6%	14.3%	11.3%	2.8%	3.3%	12.6%	3.9%	6.7%	8.0%	7.4%	11.8%	11.3%	8.3%
	1.3s	1.5s	1.2s	5.4s	12.0s	1.5s	1.1s	0.7s	65.2s	0.4s	5.2s	0.2s	17.1s	251s	3209s	111s
GM74-SwRI	9.7%	7.6%	12.5%	10.9%	17.4%	11.8%	6.2%	6.2%	22.8%	4.1%	7.0%	7.1%	23.0%	12.8%	5.5%	11.5%
	1.5s	1.3s	3.6s	50.2s	95.1s	57.2s	7.4s	3.3s	551s	1.5s	51.0s	25.3s	58.4s	1883s	3601s	315s
HP76-PS|IMS	4.6%	4.0%	10.0%	3.8%	6.7%	4.9%	6.7%	5.5%	5.6%	7.3%	7.4%	5.0%	4.6%	10.4%	11.7%	5.8%
	0.0s	0.0s	0.0s	0.1s	1.9s	0.5s	0.0s	0.0s	20.2s	0.0s	0.3s	0.1s	0.8s	6.0s	74.9s	6.8s
MJ76-INS	3.7%	3.6%	7.1%	4.9%	6.8%	9.3%	5.1%	2.5%	6.2%	4.4%	6.7%	4.2%	5.4%	10.8%	12.5%	5.4%
	0.2s	0.2s	0.4s	1.4s	7.8s	2.6s	0.3s	0.2s	29.6s	0.2s	2.9s	1.0s	3.1s	46.1s	1096s	38.2s
FR76-1PTL	5.0%	3.3%	5.3%	3.8%	15.6%	8.5%	2.7%	3.0%	10.2%	3.6%	5.5%	3.1%	9.6%	7.9%	0.9%	5.4%
	1.5s	1.2s	1.6s	29.7s	32.5s	9.0s	2.1s	1.2s	531s	0.5s	9.2s	5.9s	50.3s	325s	2192s	176s
CMT79-2P	8.8%	7.6%	14.8%	4.8%	8.6%	14.5%	7.5%	5.6%	7.9%	6.8%	6.2%	4.7%	19.3%	7.7%	10.0%	7.2%
	0.0s	0.0s	0.0s	0.3s	0.2s	0.1s	0.0s	0.0s	0.5s	0.0s	0.8s	0.1s	0.1s	60.5s	932s	27.5s
FJ81-GAP	2.3%	1.2%	5.7%	3.5%	6.4%	5.2%	1.4%	1.7%	6.6%	2.3%	7.2%	**1.6%**	3.9%	8.0%	16.7%	**3.7%**
	24.6s	46.6s	29.5s	89.5s	3600s	1505s	103s	1.6s	2943s	236s	1456s	24.0s	1060s	3032s	3602s	935s
Be83-RFCS	5.1%	4.5%	8.1%	6.0%	9.5%	5.8%	5.6%	2.3%	8.5%	6.7%	6.4%	3.3%	9.5%	11.1%	11.6%	5.8%
	0.2s	0.3s	0.4s	3.3s	23.7s	6.1s	0.4s	0.2s	147s	0.0s	2.7s	3.0s	6.8s	68.3s	1158s	67.6s
SG84-LR3OPT	2.8%	1.5%	5.5%	5.5%	22.1%	7.7%	2.3%	1.8%	13.7%	2.4%	5.5%	2.8%	4.8%	22.3%	20.9%	6.8%
	32.9s	39.2s	84.4s	254s	3602s	1774s	35.4s	11.7s	3194s	4.8s	987s	219s	975s	3438s	3601s	1133s
Pa88-PS|G2P	2.8%	2.5%	5.5%	2.5%	3.8%	4.2%	4.1%	2.6%	**4.2%**	3.7%	4.9%	3.0%	3.4%	6.3%	6.6%	**3.7%**
	0.1s	0.1s	0.2s	1.0s	1.9s	1.1s	0.2s	0.2s	8.0s	0.0s	0.6s	0.7s	1.1s	9.6s	226s	8.5s
DV89-MBSA	4.3%	3.5%	9.6%	2.4%	4.9%	5.5%	6.5%	4.8%	5.7%	8.0%	7.4%	4.9%	3.7%	10.2%	33.3%	6.2%
	1.7s	1.8s	3.2s	21.7s	107s	34.6s	2.7s	1.0s	776s	0.4s	18.7s	10.2s	46.3s	368s	3409s	284s

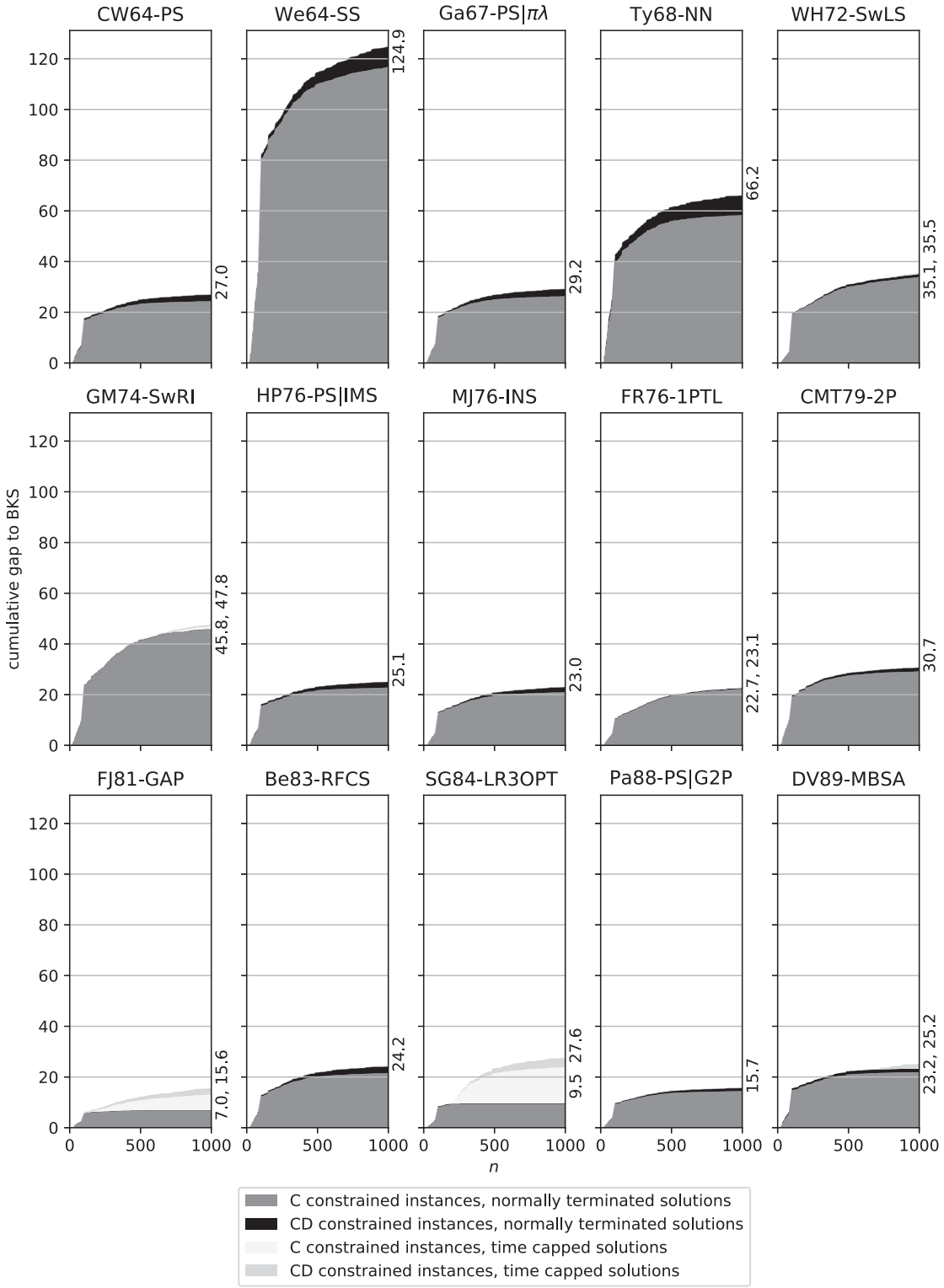
data for some methods due to reaching the time limit. However, if a result was completely missing, it was removed from all pairwise comparisons. Also, while all algorithms have well defined termination rules due to being deterministic, they do *not* consume an equal amount of computational resources, which is typically recommended practice when doing computational comparisons (Rardin and Uzsoy, 2001). Thus, we also give the average CPU wall times in Table 19 to allow estimating the effort-accuracy tradeoff.

In the benchmark set-wise comparison, the null hypothesis of no single best algorithm can be rejected only for the two largest problem sets (X and van Breedam) where two algorithms, the Paessens (1988) savings algorithm Pa88-PS|G2P and the Fisher and Jaikumar (1981) Generalized Assingment Problem relaxation heuristic FJ81-GAP, are statistically significantly better than their competitors. There is insufficient statistical evidence for choosing the best algorithm for the other benchmark sets. This is mostly due to the conservative multiple comparison correction and the small number of problem instances in those other sets. Despite this, in many cases there are clear statistically significant differences in the quality of the solutions between the pair-wise comparisons. For example, while FJ81-GAP is the best in solving van Breedam instances, also SG84-LR3OPT shows statistically good performance on the benchmark sets together with Be83-RFCS and Pa88-PS|G2P. FJ81-GAP is also a good choice to solve the problems in the set P followed by SG84-LR3OPT. Overall, SG84-LR3OPT seems to be a good choice when solving small problem instances. However, its accuracy is poor on the larger instances due the time cap that interrupts the algorithm before it reaches the inter-route 3-optimality. Also, FJ81-GAP, Pa88-PS|G2P, and SG84-LR3OPT are statistically significantly better than the other algorithms on sets A and B, and CW64-PS and its extension HP76-PS|IMS show almost as good statistically significant performance as Pa88-PS|G2P on the set X. Hence, we can make observations such as the savings approach seems to be well suited for solving the problems proposed by Uchoa et al. (2017).

If we consider the overall accuracy of the methods over all problem instances, our Pa88-PS|G2P and FJ81-GAP implementations are statistically significantly better than the others according to the Wilcoxon signed-rank test. Please note that even if their average accuracy is similar, the Pa88-PS|G2P is significantly faster than FJ81-GAP on all tested problem instances. Actually, FJ81-GAP is unable to search all seed configurations due to being interrupted because of the time limit in around one third of the problem instances and some of the larger problem instances in X and Li sets in addition to two problem instances in CMT set are not solved at all. Thus, the average accuracy of FJ81-GAP would be expected to improve with a more generous maximum runtime limit.

Further observations on the differences of the algorithm accuracy can be made from Figure 23 if we consider how the problem size and constraints are varied. This is because the figure illustrates how the cumulative gap to the best known solutions grows as the function of the problem size. The shape and composition of the cumulative curve should reveal any major systematic advantages or disadvantages of the methods in regard to problem size or presence of maximum route duration constraints. Please note that to keep the cumulative curves comparable, all results for the 18 problems instances for which FJ81-GAP and FR76-PTL were unable to produce feasible solutions are removed from this analysis. Also, the different shadings

Figure 23: Cumulative optimality gap of the algorithms over the 454 problem instances with the problem size increasing from left to right.



of Figure 23 allow observing the differences in the cumulative quality gap between those instances with capacity constraint (C) and with those that have the additional maximum route duration constraint (CD).

Because the plots are arranged chronologically from left to right and top to bottom we can see how the research on CVRP heuristics has allowed closing the optimality gap through the years. For example, it is interesting to see that the original savings heuristic from Clarke and Wright (1964) (CW64-PS) still holds its ground against the later classical methods, and the latest of its tested derivatives Pa88-PS|G2P has one of the smallest cumulative gaps in our study. When considering the chronology, please note that the basic principle of Be83-RFCS had been introduced already in 1969 and would compare favorably to the contemporary methods of the early 1970s. Furthermore, similarly to CMT79-2P, the original Be83-RFCS heuristic also relied on stochasticity and our deterministic implementation does not quite reach the level of accuracy and robustness of the original.

Comparing the different savings algorithm variants allows us to conclude which extensions are the most beneficial. The advantages of HP76-PS|IMS and DV89-MBSA over the original savings algorithm CW64-PS are modest, but clearly apparent. However, the same does not apply to Ga67-PS|$\pi\lambda$. While our experiments confirmed that the savings functions s_π and s_λ of Ga67-PS|$\pi\lambda$ are able to find better solutions on the problem instances in the Gaskell set, as shown in (Gaskell, 1967), the average and cumulative quality of solutions of Ga67-PS|$\pi\lambda$ is worse than that of CW64-PS. This result illustrates how benchmarking the proposed algorithm on only a handful of problems can lead to overfitting and give a misleading impression on the performance of the algorithm. Of the savings variants, the Pa88-PS|G2P with its parametrized savings function, parameter search strategy, and the 3-opt route post-optimization step clearly outperforms other savings variants. The good accuracy of Pa88-PS|G2P is probably related to the observations from McDonald (1972) and Webb (1972) that even a minor change in the merge order can have a significant effect on the route structures, and, thus, on the final quality of the solutions. The parametrized savings function and parameter value search strategy allows the Pa88-PS|G2P heuristic to explore several different route structures, which eventually leads to good solutions on the tested CVRP instances.

Compared to their overall accuracy, the sweep algorithm GM74-SwRI is able to find good solutions for the CD instances in this study. From the figure we can also see that the CD instances contribute very little to the overall cumulative gap on FR76-PTL. In fact, FR76-PTL has better accuracy than many metaheuristics published almost 30 years later (Prins, 2004; Toth and Vigo, 2003b; Li et al., 2005; Kytöjoki et al., 2007) on the CD instances of the Golden problem set. It reaches a similar level of accuracy as D-Ants metaheuristic from (Reimann et al., 2004), and is only slightly worse than the guided evolution strategies of Mester and Bräysy (2005). Thus, FR76-PTL seems to be a good choice for solving large maximum route duration constrained instances. However, this performance does not generalize to the C instances where FR76-PTL is clearly inferior to the aforementioned metaheuristics.

The two lighter colors on Figure 23 illustrate the quality gap accumulated from solution attempts that were interrupted due to reaching the time cap. We can see that FJ81-GAP is affected by the time capping quite early and solving attempts after 100 customers are consistently interrupted. Based on its good accuracy on the small instances, it can be expected that further performance optimizations or a more generous runtime limit could allow FJ81-GAP to produce even better solutions for the larger problems.

Similarly to FJ81-GAP, also GM74-SwRI, SG84-LR3OPT, and DV89-MBSA are affected by the one hour time limit. SG84-LR3OPT shows promising performance on the smaller problem instances, but after around 200 customers the solution attempts start to be terminated early and the resulting quality degrades significantly. A faster intra-route 3-opt procedure preferably with acceleration techniques (see Section 5.1) would allow one to confirm whether the good performance generalizes for the larger problem instances with over 200 customers. For DV89-MBSA the time cap starts to affect quite late and the quality of the solutions is impacted only slightly due to an separate greedy savings heuristic we used in the interrupt handler. With the one hour time limit, this happens around 650 customers on the CD instances and around 950 customers on the C instances.

From the overall height of the cumulative curves (Figure 23), we can see that the accuracy of Ty68-NN and We64-SS are clearly worse than their competitors, which verifies the earlier observations (e.g., Gaskell, 1967; Kirby and McDonald, 1973; Golden et al., 1977; Cordeau et al., 2002). Furthermore, from the shapes of the curves, we can observe that Pa88-PS|G2P seems to work very well on the larger instances. It cumulates only modest gap over the best known solutions on the right-hand side of the curve, whereas WH72-SwLS shows promisingly good accuracy on small instances, but is inferior to savings algorithms on larger instances. Thus, our results allow only partial confirmation of the observation by Cordeau et al. (2002) that sweep does not seem to be superior to savings approach.

Because our implemented algorithms are deterministic, we can analyze their accuracy by counting how many times the implementation was the single best algorithm over all problem instances. In case there are ties, i.e., the resulting quality of the solutions is equal, the faster algorithm was marked as the winner. The results of this analysis are presented in Table 20. As we can see, the algorithms FJ81-GAP and Pa88-PS|G2P are most capable of finding good solutions for the 454 well known academic problem instances from the literature. Together, these two are responsible of finding 42% of the best solutions in the study. FR76-PTL and SG84-LR3OPT are also often able to find good solutions (together, these four algorithms find 72% of the best solutions). Additionally, FR76-PTL seems to show its best performance on the problem sets with maximum route duration constraint (sets CMT, Golden, and Li) and SG84-LR3OPT on instances originating from Augerat (1995) (sets A, B, and P).

Another interesting observation from Table 20 is that all algorithms except We64-SS are able to be the single best algorithm for at least one problem instance. That is, we have managed to verify empirically that the no free lunch theorem (Wolpert and Macready, 1997) applies also on classical vehicle routing heuristics. Our results also verify the observation of Cordeau et al. (2002) that the parallel version of the savings algorithm is in practice much better than the sequential approach. If we make a pairwise comparison between the our CW64-PS and We64-SS implementations, We64-SS is better only for three problem instances out of 454.

Robustness (i.e., consistency) was one of the attributes mentioned in (Cordeau et al., 2002). Generally, robust heuristics are superior to those that are sensitive to small changes in the parameter values (Barr et al., 1995) or to the variation between problem instances. All of the implemented algorithms are parameter free, use defaults, or an automatic strategy to set them, and, thus, we can on disregard

Table 20: Best algorithm counts for each problem set. The number indicates the number of problem instances for which the algorithm was quickest to produce the best solution.

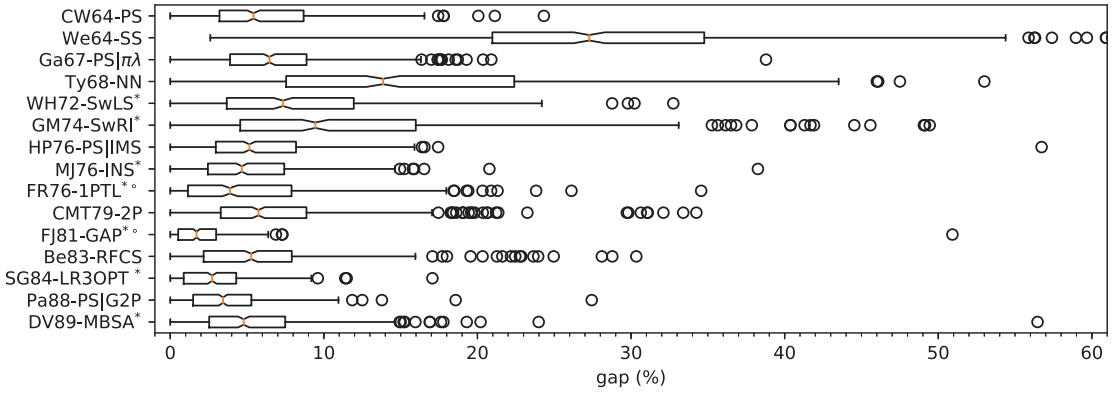
Set	Cnt.	CW64-PS	We64-SS	Ga67-PS|$\pi\lambda$	Ty68-NN	WH72-SwLS	GM74-SwRI	HP76-PS|IMS	MJ76-INS	FR76-1PTL	CMT79-2P	FJ81-GAP	Be83-RFCS	SG84-LR3OPT	Pa88-PS|G2P	DV89-MBSA
A	27	1	0	0	0	1	0	0	1	0	0	**9**	3	*8*	4	0
B	23	1	0	1	0	2	0	0	0	2	0	6	1	**8**	2	0
E	13	1	0	0	0	0	0	0	0	3	2	**4**	1	1	1	0
F	3	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1
G	1	0	0	0	0	0	0	0	0	0	0	**0*	0	*0*	**1**	0
M	5	0	0	0	0	0	0	0	0	2	0	**0*	0	*0*	2	1
P	24	0	0	0	0	3	0	0	0	4	2	5	0	**8**	2	0
V	13	2	0	0	0	1	0	0	1	2	1	2	1	**3**	0	0
X	100	0	0	0	0	1	*0*	1	5	**3*	2	**°17*	5	*6*	**54**	*6*
Gask.	6	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0
CMT	14	0	0	0	0	1	0	0	0	**5**	1	**°1*	1	*2*	2	1
vanB.	180	5	0	3	7	4	0	0	3	**43**	24	38	14	14	20	5
RT	13	0	0	0	0	1	0	0	0	0	0	**2*	0	*4*	**5**	1
Gold.	20	0	0	0	0	0	*0*	0	0	**8**	2	**7*	0	*0*	3	0
Li	12	0	0	0	0	*0*	*3*	0	*0*	**°**7***	**0	**°0*	1	*0*	1	*0*
Total	454	10	0	4	7	14	3	1	11	81	35	92	28	55	**98**	15
%		2.2	0.0	0.9	1.5	3.1	0.7	0.2	2.4	17.8	7.7	20.3	6.2	12.1	21.6	3.3

The results where the algorithm was interrupted are marked in *italics*.

The results where manual changes to the algorithm parameters were required to allow solving some of the remaining instances are marked with an asterisk(*).

Results where some of the problem instances remain unsolved are marked with a small circle (°).

Figure 24: Boxplots of the accuracy of the 15 algorithms for illustrating the quality of the solutions and the robustness of the algorithms. Missing results (°) and those that hit the time cap (*) have been removed from this analysis.



the sensitivity to parameter values when comparing the robustness between the algorithms.

The differences in the algorithm robustness can be observed from the boxplot of the quality of the solutions in Figure 24. Please note that the results where the algorithm was interrupted before it was able to reach the termination criteria were omitted from the data in this analysis. The box of each algorithm extends from the lower to upper quartile and the whiskers show the range of these values. There are some outliers that are situated beyond the right side of the plot as the optimality gap range of the figure was adjusted to allow better resolution when comparing the median behavior.

From Figure 24 it appears that, when ignoring the interrupted results, FJ81-GAP is the most robust algorithm in our study followed by SG84-LR3OPT, and Pa88-PS|G2. All three are able to consistently produce solutions with a good quality for the problem instances in our study. However, SG84-LR3OPT and FJ81-GAP were not able to solve all of the problem instances within the given time. Of the methods with large variation in the quality of the solutions, We64-SS, Ty68-NN, and GM74-SwRI all use rather simplistic approach and their accuracy seems to be quite unpredictable. Another observation can be made using the outliers in Figure 24: the quality value distribution of Ga67-PS|$\pi\lambda$, CMT79-2P, Be83-RFCS, and DV89-MBSA appears to be multimodal. This indicates that the approach of these algorithms works well on some problem instances but fails on the others. The multimodality is most apparent for CMT79-2P, where the failure of the second phase to find a feasible solution is the most probable explanation for the apparent bimodality of the quality value distribution, whereas Be83-RFCS is expected to perform poorly on problem instances where the optimal combination of customers according to their demands is more important than the shortest path routing.

It is very difficult to objectively estimate the simplicity of an algorithm. To sidestep this issue, we used a derivative of the source code maintainability index (see, e.g., Coleman et al., 1994). It was calculated by the Python code analysis tool Radon[15] and used as a proxy of the implementation complexity, which should

[15] `https://pypi.org/project/radon/`

strongly correlate on how simple the algorithm is to implement. This is verified by the fact that the rankings given in the *Simplicity* column of Table 21 correspond quite well to our objective intuition of the simplicity of the algorithms.

Table 21: Summary of replication results and measuring heuristic according to four attributes.

Algorithm	Replication level	Replication results	# of timeouts	# of missing	Maintainability index	Simplicity	Accuracy	Consistency	Speed
CW64-PS	A	p. 35, 37			61.3	1	5	5	1
We64-SS	B	p. 35			56.5	2	11	9	2
Ga67-PS|$\pi\lambda$	A	p. 36			51.7	3	7	4	3
Ty68-NN	D[1]	p. 44			24.4	9	10	9	4
WH72-SwLS	B	p. 47	8		0.0	14	8	6	10
GM74-SwRI	C	p. 51	26		23.2	10	9	8	13
HP76-PS|IMS	B	p. 38			50.7	4	3	5	6
MJ76-INS	C/B[2]	p. 41	1		11.3	13	2	4	7
FR76-1PTL	C/A[3]	p. 77	6	4*	0.0	15	2	4	11
CMT79-2P	D/B[4]	p. 56, 57		0*	27.5	8	7	7	8
FJ81-GAP	C	p. 72	104	16*	12.5	12	1	3	14
Be83-RFCS	A/C[4]	p. 59, 60			36.2	6	2	4	9
SG84-LR3OPT	A	p. 66	118		22.0	11	6	1	15
Pa88-PS|G2P	A	p. 37			31.2	7	1	2	5
DV89-MBSA	C/B[5]	p. 63	13		41.7	5	4	5	12

* = After manual adjustment of the algorithm parameters and timeouts and using a faster TSP solver.

Replication success levels:
A nearly perfect replication of the algorithm accuracy results
B difference in accuracy is small, but the standard deviation is slightly elevated
C as with B, but larger instance to instance accuracy variation
D significantly worse accuracy **or** other major uncertainties

[1] There is only one published result for a single problem instance and we were able to replicate the result only after developing extensions to the algorithm.
[2] The replication of individual stress functions is at level C, but the overall accuracy when using the best function for each instance is at level B.
[3] We were unable to recreate the exact termination rule used in (Foster and Ryan, 1976). After manually setting the number of iterations for our implementation, we are able to replicate the results.
[4] The original algorithms use stochasticity and repeated runs. The first level shows the replication with a stochastic and the second with a deterministic version.
[5] There are outliers in replication results of DV89-MBSA, and without these the replication level would be B.

The score, and our experience in implementing the algorithm, agrees with the literature in that the savings algorithms are simple to implement and extend. The

same applies to the route-first, cluster-second heuristic Be83-RFCS from Beasley (1983). The basic principle of the nearest neighbor heuristic is also very simple, but the improvement scheme together with the extensions that were needed to replicate the results of Tyagi (1968) add a layer of complexity to Ty68-NN. Similarly, the basic idea of sweep heuristic is simple, but the improvement procedures of WH72-SwLS and GM74-SwRI require extensive implementation effort. Overall, local search in a heuristic algorithm makes it more complex to implement as implementing efficient local search operators can be tricky. The heuristics inspired by relaxed mathematical programming rank low on simplicity because they usually require modeling a MIP in addition to a local search or other heuristic components.

We have already provided a detailed discussed on the differences in the accuracy of the algorithms, but Table 21 summarizes this by ranking the algorithms by average accuracy. The ranking is statistically significant according to the Wilcoxon signed-rank test ($p = 0.05$). Please note that results that were interrupted due to the time limit of one hour are included in these comparisons, which to some extent penalizes GM74-SwRI, FJ81-GAP, SG84-LR3OPT, and DV89-MBSA. The Pa88-PS|G2P heuristic and the generalized assignment problem solving heuristic FJ81-GAP are ranked as the most accurate algorithms in this study. MJ76-INS, FR76-PTL, and, somewhat surprisingly, Be83-RFCS share the second place. The sequential savings algorithm We64-SS and nearest neighbor algorithm Ty68-NN are clearly the two least accurate of the compared classical heuristics. Please note that the We64-SS algorithm (Webb, 1964) was implemented using secondary sources as we were unable to gain access to the original paper. Some details of the algorithm such as how the emerging routes are initialized may be different than in the implementation of Webb. Furthermore, according to Cordeau et al. (2002) matching significantly improves upon the Clarke and Wright (1964), but our results are more inconclusive, at least when the accuracy of CW64-PS is compared to our implementation of matching heuristic DV89-MBSA of Desrochers and Verhoog (1989).

The ranking of Be83-RFCS can also address the knowledge gap identified by Laporte and Semet (2002) regarding computational evidence on showing that route-first, cluster-second heuristics are competitive with other approaches. In our experiments, Be83-RFCS was the best algorithm for 28 of the 454 problem instances (Table 20), and even if we rule out the cases where it was selected based on a faster solution time, it was the single best algorithm on 21 problem instances.

The ranking by consistency (robustness) is also included. In this comparison, we ignored the results where any algorithm was interrupted by the time limit as it would make the comparison unreliable. Thus, a subset of the results with 319 problem instances which all algorithms were able to solve within the given time limit were used. We used Levene's test ($p = 0.05$) to rank the algorithms by their standard deviation of the optimality gaps. The SG84-LR3OPT heuristic seems to be very robust on the problem instances that it is able to solve without being interrupted due to the time limit. Also, Pa88-PS|G2P and FJ81-GAP seem to be stable and both are consistently able to produce good solutions. The least accurate algorithms are also the least robust as there is large variation in the quality of solutions for the We64-SS and Ty68-NN heuristics.

Table 21 also summarizes the replication results and some reproduction and implementation considerations. We were able to replicate the results of CW64-PS,

We64-SS, Ga67-PS|$\pi\lambda$, WH72-SwLS, HP76-PS|IMS, Be83-RFCS, SG84-LR3OPT, and Pa88-PS|G2P with high confidence. However, while any publication proposing new algorithms should include sufficiently detailed information to allow reimplementation of the algorithm and replication of results (Cordeau et al., 2002), this was not always the case and some of the classical algorithms were not described in sufficient detail to allow the results to be fully replicated. In our replication results this was, at least so some extent, the case with Ty68-NN, GM74-SwRI, MJ76-INS, FR76-1PTL, CMT79-2P, FJ81-GAP, and DV89-MBSA. As described in Section 4, we had to make many design choices when implementing the heuristics and local search operators. Those that we believe had the greatest effect on our replication results are listed in the notes of Table 21 and below:

- The insertion heuristic MJ76-INS from Mole and Jameson (1976) uses a redistribute local search operator, but the details concerning the combinations that are tried in this operation were missing from the paper. We decided to use a greedy search, where the redistribution is done in the order the customers are on the dismantled route.

- For WH72-SwLS we had to make many design decisions concerning seed customer generation and local search because description of these details is in many places ambiguous in (Wren and Holliday, 1972). However, despite these efforts, our replication results contain two outlier problem instances, where the quality of the solutions is very different from the original results.

- The description of GM74-SwRI in (Gillett and Miller, 1974) has several ambiguities and contradictions concerning the emerging route customer swap candidate selection. We tried to find the correct implementation through extensive experimentation while staying true to the description of the algorithm, but, despite being close, we were unable to perfectly reproduce the results.

- The experimental setup for measuring the performance of the stochastic two-phase algorithm in Christofides et al. (1979) is not explained in sufficient detail to reproduce the computational study. Therefore, major uncertainties remain in the replication study of this algorithm. Furthermore, our implementation CMT79-2P is deterministic, and while after parameter tuning it replicates the results of Christofides et al. (1979) fairly well, it remains undecided if the basic operating principle of our implementation is equivalent to the original two-phase algorithm.

The replication results in Table 21 together with the simplicity ranking in the same table can be used to determine which classical heuristics are complicated to understand and implement: It seems impossible that the example given in Tyagi (1968) is produced if the algorithm is implemented as described in the paper. To replicate the result changes were made to the Ty68-NN heuristic, which consequently made the algorithm more complicated. While the improvement procedure for GM74-SwRI is given in a pseudocode format, but in addition to being quite complicated, there are some ambiguities and contradictions between the text, formulas, and the pseudocode. As a result, we were unable to fully reproduce the results given in Gillett

and Miller (1974) and the source code of the heuristic has become quite complex. The complexity for WH72-SwLS is very high with the poorest maintainability of all the implementations. This is mostly due to the many local search operators used by this heuristic which adds significant implementation effort and complexity. Furthermore, the process orchestrating the local search is not as straightforward as in many later local search metaheuristics.

There is always some uncertainty in replication of results of stochastic methods. This is especially true if the computational experiment is not described in sufficient detail to allow reproducing the results. This was the case with the 2-phase algorithm (CMT79-2P), where the number of repeated tries nor used parameter value ranges to produce the results in Christofides et al. (1979) were not given. To amplify the issue, our experiments exposed that the performance and resulting quality of solutions for this algorithm are very sensitive to the selection of the route seed customers, which is apparent from the poor consistency ranking of the deterministic CMT79-2P.

Implementation of the relaxed optimization heuristics is significantly more complicated than that of the simple heuristics, and as a result, their implementations become difficult to manage and the results hard to replicate. This is especially true for FR76-1PTL and FJ81-GAP. DV89-MBSA is simpler than aforementioned heuristics as it is built on top of the savings framework, but added complexity of the MIP solver still made the results harder to replicate. SG84-LR3OPT does not use a MIP solver, which is probably the reason why we were able to reproduce the results of Stewart and Golden (1984) almost perfectly.

7 Conclusions

In this study, we recognized and implemented 15 classical capacitated vehicle routing problem (CVRP) heuristics from the literature. We sought to replicate the original results and provide a comprehensive comparison of these classical heuristics based on their *accuracy*, *consistency* (or robustness), *speed*, and *simplicity* (for definitions, see Cordeau et al., 2002) on 454 well-known CVRP problem instances from the literature.

The implemented 15 classical heuristics are: the influential parallel savings algorithm from Clarke and Wright (1964, CW64-PS), sequential savings algorithm from Webb (1964, We64-SS), savings variant by Gaskell (1967, Ga67-PS|$\pi\lambda$), nearest neighbor algorithm with a capacity utilization improvement step from Tyagi (1968, Ty68-NN), sweep algorithms from Wren and Holliday (1972, WH72-SwLS) and Gillett and Miller (1974, GM74-SwRI), parallel savings algorithm with a merge suppression scheme from Holmes and Parker (1976, HP76-PS|IMS), sequential insertion algorithm from Mole and Jameson (1976, MJ76-INS), Petal algorithm from Foster and Ryan (1976, FR76-PTL) that iteratively solves VRP as a weighted set covering problem, Christofides et al. (1979) 2-phase heuristic (CMT79-2P), heuristic based on solving generalized assignment problems from Fisher and Jaikumar (1981, FJ81-GAP), Lagrangian relaxation 3-opt heuristic from Stewart and Golden (1984, SG84-LR3OPT), route-first, cluster-second heuristic as proposed by Beasley (1983, Be83-RFCS), extension to the generalized parallel savings algorithm from Paessens (1988, Pa88-PS|G2P), and parallel savings heuristic with maximum matching based

merge scheme from Desrochers and Verhoog (1989, DV89-MBSA).

These classical algorithms were introduced between 1964 and 1989. During those early years the research of different VRP variants had been somewhat intertwined, but we were able to recognize the aforementioned CVRP heuristics that have been widely cited in the previous surveys on the topic. In addition to being well-cited, the implemented methods were required to be deterministic (or to be easily converted to be such), relatively parameter free, and expected to be capable of solving instances up to 1000 customers.

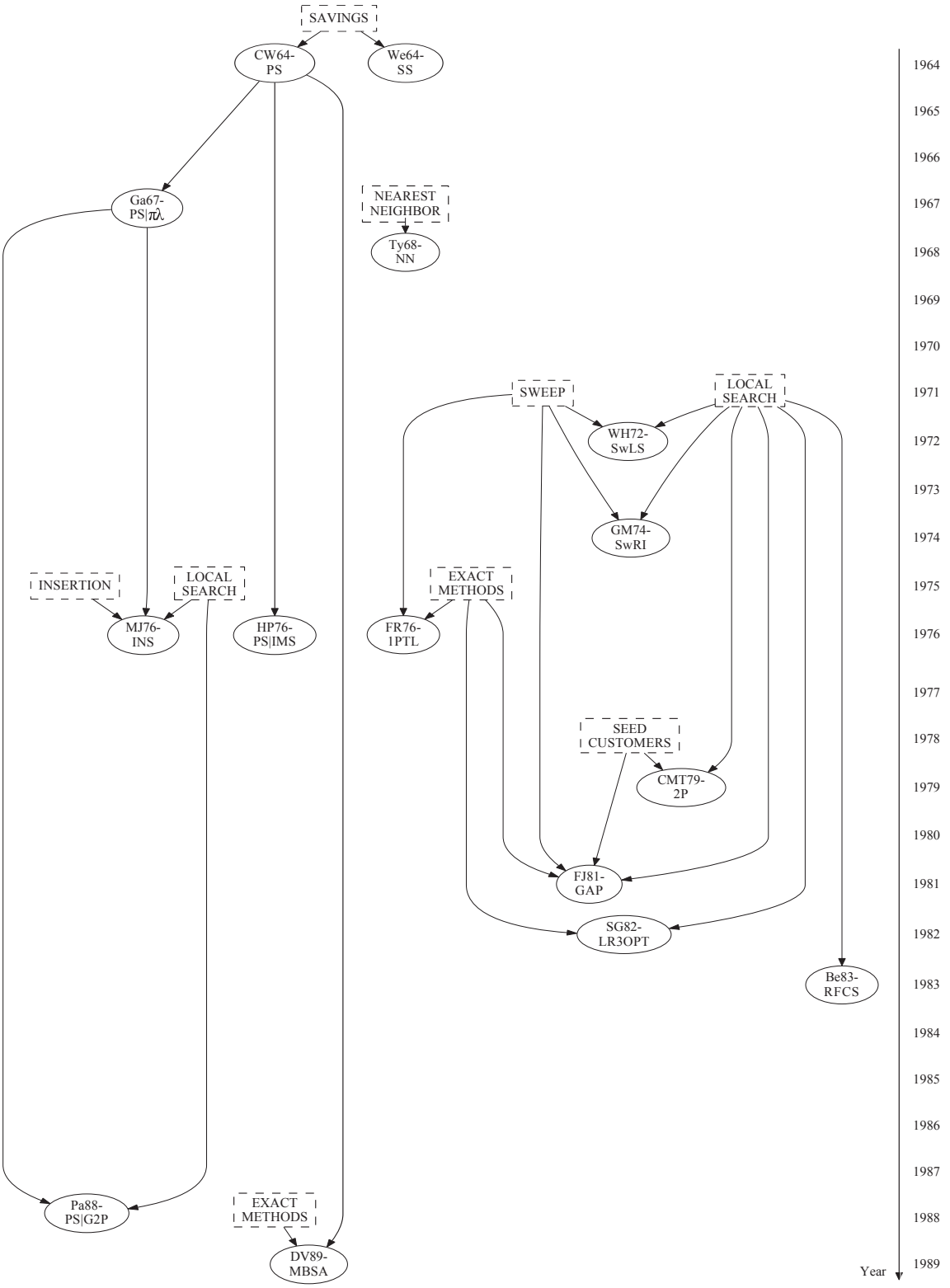
The timeline of the classical heuristics is illustrated in Figure 25, which also shows the shared ancestry and the flow of ideas between the different heuristics. The dashed rectangles represent general ideas, which were usually borrowed from the literature on solving the traveling salesman problem. Please note that their positions in the figure do not indicate the exact years the ideas were originally introduced. Instead, they are used to illustrate the shared foundation of different classical algorithms. One can, for example, see how the idea of savings, or the idea of radially sweeping the customers when looking from the depot, are the two most influential ideas among the classical CVRP heuristics. Also, the idea of local search, that is, improving an initial solution via heuristic perturbation operators (or moves), has been used to improve the accuracy of many classical methods.

All included algorithms have been implemented in Python following the functional paradigm with an emphasis on code readability and correct operation. Furthermore, there are very few external dependencies that should allow easy portability, reusability, and extendability. The most notable dependencies are Numpy, used for doing linear algebra; Gurobi (2018), which is mainly used to solve the mixed integer linear programming relaxations of VRP; and LKH (Helsgaun, 2000, 2009), a high-performance implementation of the Lin-Kernighan algorithm for solving TSPs (Lin and Kernighan, 1973).

Our work is complementary to the existing open source VRP algorithm software. The closest counterpart to VeRyPy is probably the VRPH local search library from Groër et al. (2010). However, the focus of existing libraries is in high-performance local search and metaheuristics (Groër et al., 2010; OscaR team, 2012) or in real-world delivery routing (De Smet et al., 2016; Schröder and GraphHopper team, 2018; Google, 2018), whereas our software library is aimed to be more comprehensive (breadth instead of depth) and focused on understanding and replication of the existing extensive academic research on the topic. Furthermore, we believe that our implementation with its functional programming style allows easier extendability and experimentation than the existing libraries.

A major contribution, in addition to the open source implementations themselves, are the detailed descriptions of the algorithms and the implementation notes included in this report. These document many of the important design decisions that had been left out from the original papers. Thus, our work addresses the concerns raised by Cordeau et al. (2002); Several of the heuristics they surveyed are rarely implemented either because of their inherent complexity, due to missing necessary details, or because the details are hidden in pseudocode or antiquated program listings. In this study, we have made a serious replication effort also on the more complex classical CVRP heuristics. Our implementations and documentation clarify some of the finer aspects of these well-known algorithms and should allow

Figure 25: Timeline of the implemented classical vehicle routing algorithms



researchers to effortlessly utilize the existing extensive literature on how to solve vehicle routing problems. We agree with Barr et al. (1995) and Sörensen et al. (2019) in that there is significant scientific merit in giving detailed description of all factors

in a computational study and in making the code available to other researchers of the field. Freely available and well documented implementations makes the algorithmic experimentation more productive and approachable. According to Hooker (1995), this is the frontier where valuable insights and deep knowledge that advance the field are created.

Regarding correctness of our implementations, we were, in most cases, able to replicate the results of the classical papers. Hence, we can be reasonably certain that most of our implementation closely correspond to the algorithms described by their original authors. However, our reproduction efforts also confirmed the observation of Cordeau et al. (2002, p. 514) that many of the heuristics are very sensitive to minor implementation details. Our reproduction work revealed that implementing Tyagi (1968), Gillett and Miller (1974), Mole and Jameson (1976), and Fisher and Jaikumar (1981) heuristics is difficult due to insufficient and contradicting algorithm details. For the algorithms from Foster and Ryan (1976) and Desrochers and Verhoog (1989), the reason why the results are not perfectly reproduced remained more elusive. However, even for the aforementioned heuristics, we were able to reproduce some of the results, and the overall quality of solutions is, in most cases, very similar to that of the original publications. Based on our replication experiences, it is easy to agree with Cordeau et al. (2002) in that heuristics should be reasonably robust to different minor implementation decisions. This is because otherwise replication of their results becomes extremely difficult.

When discussing reproducibility it is always important to address stochasticity, which is also used in many classical VRP algorithms. Even the original savings algorithm from Clarke and Wright (1964) used randomness to resolve ties between merges with the same savings value. Introducing stochasticity can improve the accuracy of any algorithm as it allows wider exploration with the cost of an increased execution time. However, it also makes it harder to define a termination criteria because each successive iteration has the potential to improve the best found solution. Furthermore, with stochastic methods, reproducing the results requires the use of statistical testing. To sidestep these issues in this study, we modified savings algorithms to resolve the ties similarly to Gaskell (1967) by using a secondary savings criteria, and proposed deterministic variants for the CMT79-2P (Christofides et al., 1979) and Be83-RFCS (Beasley, 1983) algorithms. We showed that our deterministic CMT79-2P implementation has similar performance to the stochastic version but uses only fraction of the computational resources.

Based on our extensive computational testing, where the 15 classical heuristics were used to solve 454 CVRP problem instances from the literature, we could rank the heuristics according to the following attributes: *accuracy*, *consistency* (or robustness), *speed*, and *simplicity*. This made it possible to reliably compare for the first time them along these different qualities. These results validated many earlier observations in the literature (e.g., Cordeau et al., 2002). Overall, our comprehensive experiments show that the first generation of VRP heuristics has better accuracy than previously thought: according to Renaud et al. (1996) one can expect classical heuristics to be within 10-15% of the best known solution. Based on our experiments, one can expect a better accuracy, especially when selecting the heuristic carefully considering the idiosyncrasies of a problem instance. The average gap over all of our experiments is 8.3% (with infeasible and interrupted results removed), and

most heuristics have an average optimality gap of around 5%. If only the best algorithm for each of the 454 CVRP problem instances is considered, then the optimality gap shrinks down to 1.7%.

Our results also allow making interesting observations about the relative speed and accuracy trade-offs of the heuristics, and offer insights into the relative strengths and weaknesses of different classical algorithms. For example, GM74-SwRI and FR76-PTL have excellent accuracy on the problem instances with a maximum route duration constraint, and Pa88-PS|G2P is well suited for solving larger problem instances. All three: FR76-1PTL, FJ81-GAP, and Pa88-PS|G2P all show good general performance over all of the problem instances. Pa88-PS|G2P is significantly faster than the other two accurate heuristics, and, in fact, FJ81-GAP and FR76-1PTL were unable to finish before the one-hour time limit when solving the largest problem instances. Hence, our results can help to select a suitable construction heuristic for particular use case depending on individual requirements on algorithm simplicity, accuracy, consistency, or speed.

There are several aspects to consider regarding the validity of our conclusions: reproduction of the results, stochasticity, composition of the problem set, and the algorithm configuration effort. As described earlier, many design decisions were required to successfully implement the 15 classical heuristics included in this study. While this we were able to mostly replicate the earlier results from the literature, some uncertainty remains whether Ty68-NN, GM74-SwRI, FR76-1PTL, FJ81-GAP, DV89-MBSA, and the stochastic version of CMT79-2P correspond to the algorithms proposed by their original authors. Some of the reproduction and replication issues that concern missing details or the specifics of the original experimental setup are well known in the literature, but we have also discovered new issues and have done our best to address them.

We acknowledge that removing stochastic elements from a heuristic can have significant effect on its general level of accuracy and robustness. The original versions of the CMT79-2P and Be83-RFCS algorithms relied heavily on stochasticity, but our implementations disable stochasticity by default. However, we have proposed extensions to these two algorithms to compensate this.

In our experiments we allowed at most one hour for each heuristic to solve each of the 454 problem instances. This has an effect on the reproducibility of the results and causes some results to be missing or being finished by a complementary greedy heuristic. This has an effect on validity of the average accuracy and runtime results, but the impact is limited to the results of GM74-SwRI, FJ81-GAP, SG84-LR3OPT, and DV89-MBSA on the largest problem instances. However, we emphasize that the results and conclusions of our computational study strictly apply only to these specific implementations, and complementary replication efforts would be welcomed to see how well our observations generalize.

One should also keep in mind that many of the 454 problem instances from the literature are quite artificial, and some are variations of the earlier problem instances. However, together the instances create a fairly heterogeneous problem set, and by comparing the provided results on a problem set basis can allow reader to concentrate on those features the reader one is most interested in.

Although most of the classical algorithms are parameter free or come with sensible defaults, it is normally advisable to configure any free parameters, for example,

by using some suitable configuration tool such as SMAC from Hutter et al. (2011). Due to the scope of our study, such automatic algorithm configuration was omitted from the experiments. However, some manual configuration was required to make FJ81-GAP, CMT79-2P, and FR76-1PTL solve some of the larger instances, and this may give them some unfounded advantage. For a thorough introduction on the topic of automatic algorithm configuration in the context of vehicle routing problems, please refer to (Rasku et al., 2019b), where we compared the performance of several methods in a task of configuring vehicle routing metaheuristics.

As a final point regarding validity, please note that the performance measures are always dependent on the problem instances, algorithms, and test environment factors (Barr et al., 1995). In this study, we carefully controlled the factors and conscientiously documented them and our underlying assumptions. Thus, we argue that the we have addressed the main concerns regarding validity and, despite the aforementioned issues, convincing and repeatable observations can be made.

Proposing and implementing new enhancements to the existing classical algorithms was outside of the scope of this study. We acknowledge that many of the issues we had with the performance of FR76-1PTL, DV89-MBSA and SG84-LR3OPT could be resolved with faster and more modern local search procedures. For example, implementing accelerated 2-opt, 3-opt and 3-opt* search procedures using the methods suggested by Funke et al. (2005) would allow VeRyPy implementations to solve larger instances in a more reasonable time. Furthermore, implementing a procedure that can quickly calculate a TSP lower bound for a route, and update it after insertions are made, would make some algorithms, such as CMT79-2P, faster on route distance or duration constrained problems. Additionally, using programming tools and techniques that are better suited for array manipulation, such as Numba (Lam et al., 2015), Cython (Behnel et al., 2011) or pure C extensions, would significantly improve the performance of our local search implementations.

Regarding extending the VeRyPy library with new heuristics, there were many interesting algorithms that could be considered to be classical, but which were not included in VeRyPy as they did not meet all or some of the criteria used in this study. These include the parametrized characteristics based heuristic of Hayes (1967), r-optimal improvement approach as proposed by Christofides and Eilon (1969), M-TOUR heuristic from Russell (1977), and capacitated concentrator location problem heuristic from Bramel and Simchi-Levi (1995). Other interesting variants to the already implemented methods include the savings maximum matching extensions from Altinkemer and Gavish (1991) and Wark and Holt (1994), and the 2-Petal heuristic from Renaud et al. (1996). Implementing them using the existing framework is possible, and in some cases trivial. VeRyPy could also be extended to implement several of the popular metaheuristics. The scaffolding in the form of input and output, local search implementations, feasibility checkers, and objective evaluations are already implemented, which would make implementing some of the simpler metaheuristics straightforward.

As a result of our reimplementation efforts, our comprehensive and easy-to-extend VRP algorithm library can open up new possibilities for future research. For example, as a future research topic one could see which heuristics have been underutilized in solving specific VRP variants and which heuristic-problem variant combinations could be promising. As an example of such work, Prins (2002) sought

to implement an effective heuristic for a real world heterogeneous fleet multitrip VRP (HFMVRP) based on several classical methods. Additionally, the implementations and the results of our work could allow testing different hypothesis about algorithmic components and how problem characteristics influence the behavior of different heuristics (see, e.g., Hooker, 1995; Rardin and Uzsoy, 2001). Our study already contains some insights on this, and we have published a study on VRP algorithm selection (Rasku et al., 2019a).

Besides making aforementioned experimental work more accessible, we believe that sharing code will benefit the community as a whole. Sharing the source code "relieves the community from the burden of programming everything from the scratch, time and time again" (Sörensen et al., 2019). The presented work of reimplementing and reproducing the results of 15 classical CVRP heuristics provides a practical, comprehensive, and approachable introduction to the topic With it, a new or a seasoned researcher alike are able to understand and leverage the extensive research on classical vehicle routing heuristics. According to the recent review from Braekers et al. (2016), the classical methods had still been applied in around 10% of the surveyed papers published between 2009 and 2015. Thus, the classical heuristics seem to remain relevant both to the research and practice, and our study can be used to recognize and utilize the different strengths and advantages of the classical vehicle routing heuristics.

References

Alba, E. and Dorronsoro, B. (2006). Computing nine new best-so-far solutions for capacitated VRP with a cellular genetic algorithm. *Information Processing Letters*, 98(6):225–230.

Altinkemer, K. and Gavish, B. (1991). Parallel savings based heuristics for the delivery problem. *Operations Research*, 39(3):456–469.

Applegate, D., Bixby, R., Chvatal, V., and Cook, W. (2006). Concorde TSP solver. Accessed: 2017-06-27.

Archetti, C. and Speranza, M. G. (2014). A survey on matheuristics for routing problems. *EURO Journal on Computational Optimization*, 2(4):223–246.

Augerat, P. (1995). *Approche polyèdrale du problème de tournées de véhicules*. PhD thesis, Institut National Polytechnique de Grenoble-INPG.

Baker, B. M. and Sheasby, J. (1999). Extensions to the generalised assignment heuristic for vehicle routing. *European Journal of Operational Research*, 119(1):147–157.

Barnhart, C. and Laporte, G., editors (1993). *Transportation*, volume 14 of *Handbooks in Operations Research and Management Science*. Elsevier.

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

Beasley, J. E. (1983). Route first—cluster second methods for vehicle routing. *Omega*, 11(4):403–408.

Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D. S., and Smith, K. (2011). Cython: The best of both worlds. *Computing in Science & Engineering*, 13(2):31–39.

Beltrami, E. J. and Bodin, L. D. (1974). Networks and vehicle routing for municipal waste collection. *Networks*, 4(1):65–94.

Bertsimas, D. J. and Simchi-Levi, D. (1996). A new generation of vehicle routing research: robust algorithms, addressing uncertainty. *Operations Research*, 44(2):286–304.

Bodin, L., Golden, B. L., Assad, A. A., and Ball, M. O. (1983). Special issue on routing and scheduling of vehicles and crews: The state of the art. *Computers & Operations Research*, 10(2).

Booch, G. (1998). *Object oriented analysis & design with applications*. Addison Wesley Longman.

Boyko, R. and Sykes, B. D. (2012). xcrvfit: a graphical X-windows program for binding curve studies and NMR spectroscopic analysis. `http://www.bionmr.ualberta.ca/bds/software/xcrvfit`. University of Alberta.

Braekers, K., Ramaekers, K., and Van Nieuwenhuyse, I. (2016). The vehicle routing problem: State of the art classification and review. *Computers & Industrial Engineering*, 99:300–313.

Bramel, J. and Simchi-Levi, D. (1995). A location based heuristic for general routing problems. *Operations research*, 43(4):649–660.

Bräysy, O. and Gendreau, M. (2005). Vehicle routing problem with time windows, part i: Route construction and local search algorithms. *Transportation Science*, 39(1):104–118.

Buxey, G. M. (1979). The vehicle scheduling problem and monte carlo simulation. *Journal of the Operational Research Society*, 30(6):563–573.

Cai, X., Langtangen, H. P., and Moe, H. (2005). On the performance of the Python programming language for serial and parallel scientific computations. *Scientific Programming*, 13(1):31–56.

Christofides, N. and Eilon, S. (1969). An algorithm for the vehicle-dispatching problem. *Journal of the Operational Research Society*, 20(3):309–318.

Christofides, N., Mingozzi, A., and Toth, P. (1979). The vehicle routing problem. In Christofides, N., Mingozzi, A., Toth, P., and Sandi, C., editors, *Combinatorial Optimization*, chapter 11, pages 315–338. Wiley.

Clarke, G. and Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581.

Coleman, D., Ash, D., Lowther, B., and Oman, P. (1994). Using metrics to evaluate software system maintainability. *Computer*, 27(8):44–49.

Cordeau, J.-F., Gendreau, M., Laporte, G., Potvin, J.-Y., and Semet, F. (2002). A guide to vehicle routing heuristics. *Journal of the Operational Research Society*, 53(5):512–522.

Cordeau, J.-F., Laporte, G., Savelsbergh, M. W., and Vigo, D. (2007). Vehicle routing. In *Handbook in OR & MS*, volume 14, chapter 6, pages 366–427. Elsevier.

Coupey, J. (2018). VROOM - vehicle routing open-source optimization machine. `http://vroom-project.org/`. Accessed: 2018-04-10.

Črepinšek, M., Liu, S.-H., and Mernik, M. (2014). Replication and comparison of computational experiments in applied evolutionary computing: common pitfalls and guidelines to avoid them. *Applied Soft Computing*, 19:161–170.

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

De Smet, G. et al. (2016). OptaPlanner user guide.

Desrochers, M. and Verhoog, T. W. (1989). G-89-04 : A matching based savings algorithm for the vehicle routing problem. Technical report, GERAD, Montreal, Canada.

Di Gaspero, L. and Schaerf, A. (2003). EasyLocal++: an object-oriented framework for the flexible design of local-search algorithms. *Software: Practice and Experience*, 33(8):733–765.

Eilon, S., Watson-Gandy, C., and Christofides, N. (1971). *Distribution management*. Griffin.

Eksioglu, B., Vural, A. V., and Reisman, A. (2009). The vehicle routing problem: A taxonomic review. *Computers & Industrial Engineering*, 57(4):1472–1483.

Erdoğan, G. (2017). An open source spreadsheet solver for vehicle routing problems. *Computers & Operations Research*, 84:62–72.

Fahrion, R. and Wrede, M. (1990). On a principle of chain-exchange for vehicle-routeing problems (1-VRP). *Journal of the Operational Research Society*, 41(9):821–827.

Fisher, M. (1995). Vehicle routing. In *Handbooks in operations research and management science*, volume 8, pages 1–33. Elsevier.

Fisher, M. L. (1994). Optimal solution of vehicle routing problems using minimum k-trees. *Operations Research*, 42(4):626–642.

Fisher, M. L. and Jaikumar, R. (1981). A generalized assignment heuristic for vehicle routing. *Networks*, 11(2):109–124.

Fisher, M. L., Jaikumar, R., and Van Wassenhove, L. N. (1986). A multiplier adjustment method for the generalized assignment problem. *Management Science*, 32(9):1095–1103.

Foster, B. A. and Ryan, D. M. (1976). An integer programming approach to the vehicle scheduling problem. *Journal of the Operational Research Society*, 27(2):367–384.

Funke, B., Grünert, T., and Irnich, S. (2005). Local search for vehicle routing and scheduling problems: Review and conceptual integration. *Journal of Heuristics*, 11(4):267–306.

Gaskell, T. (1967). Bases for vehicle fleet scheduling. *Journal of the Operational Research Society*, 18(3):281–295.

Gendreau, M., Hertz, A., and Laporte, G. (1994). A tabu search heuristic for the vehicle routing problem. *Management Science*, 40(10):1276–1290.

Gendreau, M. and Potvin, J.-Y., editors (2010). *Handbook of metaheuristics*. Springer, 2nd edition.

Gillett, B. E. and Johnson, J. G. (1976). Multi-terminal vehicle-dispatch algorithm. *Omega*, 4(6):711–718.

Gillett, B. E. and Miller, L. R. (1974). A heuristic algorithm for the vehicle-dispatch problem. *Operations Research*, 22(2):340–349.

Golden, B. L. (1978). Recent developments in vehicle routing. In White, W. W., editor, *Computers and Mathematical Programming*, pages 233–240. US Department of Commerce.

Golden, B. L., Magnanti, T. L., and Nguyen, H. Q. (1977). Implementing vehicle routing algorithms. *Networks*, 7(2):113–148.

Golden, B. L., Wasil, E. A., Kelly, J. P., and Chao, I.-M. (1998). The impact of metaheuristics on solving the vehicle routing problem: algorithms, problem sets, and computational results. In *Fleet management and logistics*, pages 33–56. Springer.

Google (2018). Google optimization tools (OR-Tools) software suite. `https://developers.google.com/optimization/`. Accessed: 2018-04-10.

Graham, S. L., Kessler, P. B., and Mckusick, M. K. (1982). Gprof: A call graph execution profiler. In *ACM Sigplan Notices*, volume 17, pages 120–126. ACM.

Gregg, B. (2016). The flame graph. *Communications of the ACM*, 59(6):48–57.

Groër, C., Golden, B. L., and Wasil, E. A. (2010). A library of local search heuristics for the vehicle routing problem. *Mathematical Programming Computation*, 2(2):79–101.

Gurobi (2018). Gurobi optimizer reference manual.

Haimovich, M. and Rinnooy Kan, A. (1985). Bounds and heuristics for capacitated routing problems. *Mathematics of Operations Research*, 10(4):527–542.

Hallberg, M. and Kriebel, W. (1979). A routing algorithm using the nearest-neighbor concept. *American Journal of Agricultural Economics*, 61(1):87–90.

Hansen, P. and Mladenović, N. (1999). An introduction to variable neighborhood search. In *Meta-heuristics*, pages 433–458. Springer.

Hayes, R. L. (1967). The delivery problem. Technical report, Carnegie Institute of Technology, Management Sciences Research Group Report No. 106, Pittsburgh, PA.

Heins, H. (1981). Ein 2-phasen-algorithmus für das ein-depot-tourenplanungsproblem. In *6. Symposium uber Operations Research: proceedings*, volume 43, page 219. Verlagsgruppe Athenäum/Hain/Scriptor/Hanstein.

Held, M. and Karp, R. M. (1970). The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162.

Held, M. and Karp, R. M. (1971). The traveling-salesman problem and minimum spanning trees: Part ii. *Mathematical Programming*, 1(1):6–25.

Helsgaun, K. (2000). An effective implementation of the Lin–Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130.

Helsgaun, K. (2009). General k-opt submoves for the lin–kernighan TSP heuristic. *Mathematical Programming Computation*, 1(2-3):119–163.

Holmes, R. and Parker, R. (1976). A vehicle scheduling procedure based upon savings and a solution perturbation scheme. *Journal of the Operational Research Society*, 27(1):83–92.

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

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pages 507–523. Springer.

Irnich, S., Funke, B., and Grünert, T. (2006). Sequential search and its application to vehicle-routing problems. *Computers & Operations Research*, 33(8):2405–2429.

Kirby, R. F. and McDonald, J. J. (1973). The savings method for vehicle scheduling. *Journal of the Operational Research Society*, 24(2):305–307.

Klincewicz, J. (1975). The Tyagi algorithm for truck dispatching. UROP Final Project Report, MIT, MA, USA.

Klitzke, E. (2016). Pyflame: Uber Engineering's ptracing profiler for Python. `http://eng.uber.com/pyflame/`.

Knowles, K. (1967). The use of a heuristic tree-search algorithm for vehicle routing and scheduling. Lecture at Operational Research Conference, Exeter, England.

Koch, T., Achterberg, T., Andersen, E., Bastert, O., et al. (2011). Miplib 2010. *Mathematical Programming Computation*, 3(2):103.

Krolak, P., Felts, W., and Nelson, J. (1972). A man-machine approach toward solving the generalized truck-dispatching problem. *Transportation Science*, 6(2):149–170.

Kytöjoki, J., Nuortio, T., Bräysy, O., and Gendreau, M. (2007). An efficient variable neighborhood search heuristic for very large scale vehicle routing problems. *Computers & Operations Research*, 34(9):2743–2757.

Lam, S. K., Pitrou, A., and Seibert, S. (2015). Numba: A LLVM-based python JIT compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, page 7. ACM.

Laporte, G. (1992). The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(3):345–358.

Laporte, G. (2007). What you should know about the vehicle routing problem. *Naval Research Logistics*, 54(8):811–819.

Laporte, G. (2009). Fifty years of vehicle routing. *Transportation Science*, 43(4):408–416.

Laporte, G., Gendreau, M., Potvin, J.-Y., and Semet, F. (2000). Classical and modern heuristics for the vehicle routing problem. *International Transactions in Operational Research*, 7(4-5):285–300.

Laporte, G. and Nobert, Y. (1987). Exact algorithms for the vehicle routing problem. In *North-Holland Mathematics Studies*, volume 132, pages 147–184. Elsevier.

Laporte, G., Ropke, S., and Vidal, T. (2014). Heuristics for the vehicle routing problem. In *Vehicle Routing: Problems, Methods, and Applications*, chapter 4, pages 87–116. SIAM, second edition.

Laporte, G. and Semet, F. (2002). Classical heuristics for the capacitated VRP. In *The vehicle routing problem*, pages 109–128. SIAM.

Li, F., Golden, B. L., and Wasil, E. A. (2005). Very large-scale vehicle routing: new test problems, algorithms, and results. *Computers & Operations Research*, 32(5):1165–1179.

Lin, S. and Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516.

Lodi, A. and Punnen, A. P. (2007). Tsp software. In Gutin, G. and Punnen, A. P., editors, *The Traveling Salesman Problem and Its Variations*, pages 737–749. Springer US, Boston, MA.

Longbottom, R. (2014). Whetstone benchmark history and results. `http://www.roylongbottom.org.uk/whetstone.htm`. [Online; accessed 15-February-2018].

Maniezzo, V., Stützle, T., and Voß, S. (2010). *Matheuristics, Hybridizing Metaheuristics and Mathematical Programming, Annals of Information Systems*. 10. Springer US.

McDonald, J. J. (1972). Vehicle scheduling — a case study. *Journal of the Operational Research Society*, 23(4):433–444.

Mester, D. and Bräysy, O. (2005). Active guided evolution strategies for large-scale vehicle routing problems with time windows. *Computers & Operations Research*, 32(6):1593–1614.

Miller, L. R. (1970). *Heuristic algorithms for the generalized vehicle dispatch problem*. PhD thesis, Department of Mathematics and Statistics, University of Missouri–Rolla.

Millman, K. J. and Aivazis, M. (2011). Python for scientists and engineers. *Computing in Science & Engineering*, 13(2):9–12.

Mole, R. (1979). A survey of local delivery vehicle routing methodology. *Journal of the Operational Research Society*, 30(3):245–252.

Mole, R. and Jameson, S. (1976). A sequential route-building algorithm employing a generalised savings criterion. *Journal of the Operational Research Society*, 27(2):503–511.

Nelson, M. D., Nygard, K. E., Griffin, J. H., and Shreve, W. E. (1985). Implementation techniques for the vehicle routing problem. *Computers & Operations Research*, 12(3):273–283.

Newman, N. (1999). The origins and future of open source software. Netaction White Paper.

Newton, R. M. and Thomas, W. H. (1969). Design of school bus routes by computer. *Socio-Economic Planning Sciences*, 3(1):75–85.

Newton, R. M. and Thomas, W. H. (1974). Bus routing in a multi-school system. *Computers & Operations Research*, 1(2):213–222.

OscaR team (2012). OscaR: Scala in OR. Available from `https://bitbucket.org/oscarlib/oscar`.

Paessens, H. (1988). The savings algorithm for the vehicle routing problem. *European Journal of Operational Research*, 34(3):336–344.

Pecin, D., Pessoa, A., Poggi, M., and Uchoa, E. (2017). Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation*, 9(1):61–100.

Perez, F., Granger, B. E., and Hunter, J. D. (2011). Python: an ecosystem for scientific computing. *Computing in Science & Engineering*, 13(2):13–21.

Pierce, J. F. (1969). Direct search algorithms for truck-dispatching problems. *Transportation Research*, 3(1):1–42.

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

Prins, C. (2002). Efficient heuristics for the heterogeneous fleet multitrip VRP with application to a large-scale real case. *Journal of Mathematical Modelling and Algorithms*, 1(2):135–150.

Prins, C. (2004). A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, 31(12):1985–2002.

Prins, C., Lacomme, P., and Prodhon, C. (2014). Order-first split-second methods for vehicle routing problems: A review. *Transportation Research Part C: Emerging Technologies*, 40:179–200.

Ralphs, T. K. and Güzelsoy, M. (2005). The SYMPHONY callable library for mixed integer programming. In *The next wave in computing, optimization, and decision technologies*, pages 61–76. Springer.

Rardin, R. L. and Uzsoy, R. (2001). Experimental evaluation of heuristic optimization algorithms: A tutorial. *Journal of Heuristics*, 7(3):261–304.

Rasku, J., Kärkkäinen, T., and Musliu, N. (2016). Feature extractors for describing vehicle routing problem instances. In *OASIcs, SCOR'16*, volume 50. Dagstuhl Publishing.

Rasku, J., Musliu, N., and Kärkkäinen, T. (2019a). Feature and algorithm selection for capacitated vehicle routing problems. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2019)*, pages 373–378. Ciaco - i6doc.com.

Rasku, J., Musliu, N., and Kärkkäinen, T. (2019b). On automatic algorithm configuration of vehicle routing problem solvers. *Journal on Vehicle Routing Algorithms*.

Rego, C. (1998). A subpath ejection method for the vehicle routing problem. *Management Science*, 44(10):1447–1459.

Reimann, M., Doerner, K., and Hartl, R. F. (2004). D-Ants: Savings based ants divide and conquer the vehicle routing problem. *Computers & Operations Research*, 31(4):563–591.

Reinelt, G. (1991). TSPLIB — a traveling salesman problem library. *ORSA Hournal on Computing*, 3(4):376–384.

Renaud, J., Boctor, F. F., and Laporte, G. (1996). An improved petal heuristic for the vehicle routeing problem. *Journal of the Operational Research Society*, 47(2):329–336.

Robbins, J. A. and Turner, W. C. (1979). CAWLIP - Clarke and Wright–Lin Interchange Program for vehicle routing problems. *Computers & Industrial Engineering*, 3(1):89–100.

Rochat, Y. and Taillard, É. D. (1995). Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, 1(1):147–167.

Russell, R. A. (1977). An effective heuristic for the m-tour traveling salesman problem with some side conditions. *Operations Research*, 25(3):517–524.

Ryan, D. M., Hjorring, C., and Glover, F. (1993). Extensions of the petal method for vehicle routeing. *Journal of the Operational Research Society*, 44(3):289–296.

Savelsbergh, M. W. (1992). The vehicle routing problem with time windows: Minimizing route duration. *ORSA Journal on Computing*, 4(2):146–154.

Schneider, C. A., Rasband, W. S., and Eliceiri, K. W. (2012). NIH Image to ImageJ: 25 years of image analysis. *Nature Methods*, 9(7):671.

Schröder, S. and GraphHopper team (2018). jsprit open source toolkit for solving routing problems. `https://github.com/graphhopper/jsprit`. Accessed: 2018-04-10.

Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H., and Dueck, G. (2000). Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, 159(2):139–171.

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

Sörensen, K., Arnold, F., and Palhazi Cuervo, D. (2019). A critical analysis of the "improved clarke and wright savings algorithm". *International Transactions in Operational Research*.

Steinhaus, M. (2015). *The application of the self organizing map to the vehicle routing problem*. PhD thesis, University of Rhode Island.

Stewart, W. R. and Golden, B. L. (1984). A lagrangean relaxation heuristic for vehicle routing. *European Journal of Operational Research*, 15(1):84–88.

Stützle, T. (2002). ACOTSP: A software package of various ant colony optimization algorithms applied to the symmetric traveling salesman problem.

Subramanyam, R. and Krishnan, M. S. (2003). Empirical analysis of CK metrics for object-oriented design complexity: Implications for software defects. *IEEE Transactions on Software Engineering*, 29(4):297–310.

Taillard, É. D. (1993). Parallel iterative search methods for vehicle routing problems. *Networks*, 23(8):661–673.

Tange, O. (2011). GNU parallel - the command-line power tool. *;login: The USENIX Magazine*, 36(1):42–47.

Thompson, P. M. and Psaraftis, H. N. (1993). Cyclic transfer algorithm for multi-vehicle routing and scheduling problems. *Operations Research*, 41(5):935–946.

Tillman, F. A. and Cochran, H. (1968). A heuristic approach for solving delivery problem. *Journal of Industrial Engineering*, 19(7):354.

Toth, P. and Vigo, D. (2002a). Models, relaxations and exact approaches for the capacitated vehicle routing problem. *Discrete Applied Mathematics*, 123(1-3):487–512.

Toth, P. and Vigo, D. (2002b). *The vehicle routing problem*. SIAM.

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

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

Toth, P. and Vigo, D. (2014). *Vehicle Routing: Problems, Methods, and Applications, Second Edition*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

Turner, W. C., Ghare, P. M., and Foulds, L. R. (1974). Transportation routing problem — a survey. *AIIE Transactions*, 6(4):288–301.

Tyagi, M. S. (1968). A practical method for the truck dispatching problem. *Journal of the Operations Research Society of Japan*, 10:76–92.

Uchoa, E., Pecin, D., Pessoa, A., Poggi, M., Vidal, T., and Subramanian, A. (2017). New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845–858.

Van Breedam, A. (1994). *An Analysis of the Behavior of Heuristics for the Vehicle Routing Problem for a Selectrion of Problems with Vehicle-related, Customer-related, and Time-related Constraints*. PhD thesis, Faculty of Applied Economics, University of Antwerp, Belgium - RUCA.

Van Breedam, A. (2002). A parametric analysis of heuristics for the vehicle routing problem with side-constraints. *European Journal of Operational Research*, 137(2):348–370.

Vidal, T., Crainic, T. G., Gendreau, M., and Prins, C. (2013). Heuristics for multi-attribute vehicle routing problems: A survey and synthesis. *European Journal of Operational Research*, 231(1):1–21.

Wark, P. and Holt, J. (1994). A repeated matching heuristic for the vehicle routeing problem. *Journal of the Operational Research Society*, 45(10):1156–1167.

Watson-Gandy, C. and Foulds, L. R. (1981). Vehicle scheduling problem: a survey. *New Zealand Operational Research*, 9(2):73–92.

Webb, M. (1964). A study in transport routing. *Glass Technology*, 5:178–181.

Webb, M. (1972). Relative performance of some sequential methods of planning multiple delivery journeys. *Journal of the Operational Research Society*, 23(3):361–372.

Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82.

Wren, A. and Carr, J. (1971). *Computers in transport planning and operation*. Ian Allan.

Wren, A. and Holliday, A. (1972). Computer scheduling of vehicles from one or more depots to a number of delivery points. *Journal of the Operational Research Society*, 23(3):333–344.

Yellow, P. (1970). A computational modification to the savings method of vehicle scheduling. *Journal of the Operational Research Society*, 21:281–283.

Zachariadis, E. E. and Kiranoudis, C. T. (2010). A strategy for reducing the computational complexity of local search-based methods for the vehicle routing problem. *Computers & Operations Research*, 37(12):2089–2105.

PV

FEATURE AND ALGORITHM SELECTION FOR CAPACITATED VEHICLE ROUTING PROBLEMS

by

Jussi Rasku, Nysret Musliu, Tommi Kärkkäinen 2019

In Proceedings of European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2019)

ESANN 2019 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2019, i6doc.com publ., ISBN 978-287-587-065-0.
Available from http://www.i6doc.com/en/.

Feature and Algorithm Selection for Capacitated Vehicle Routing Problems

Jussi Rasku[1], Nysret Musliu[2], and Tommi Kärkkäinen[1]

1- Faculty of Information Technology
University of Jyvaskyla, FI-40014 Jyvaskyla - Finland

2- CD-Lab Artis, Institute of Logic and Computation,
TU Wien, A-1040 Vienna - Austria

Abstract. Many exact, heuristic, and metaheuristic algorithms have been proposed to effectively produce high quality solutions to vehicle routing problems. However, it remains an open question which algorithm is the most appropriate for solving a given problem instance, mostly because the different strengths and weaknesses of algorithms are still not well understood. We propose an extensive feature set for describing capacitated vehicle routing problem instances and illustrate how it can be used in algorithm selection, and how different feature selection approaches can be used to recognize the most relevant features for this task.

1 Introduction

Vehicle routing problem is one of the most intensively studied problems in operations research because of its many applications. Recently we proposed using an autoencoder based approach [6] to recognize interesting Capaciated Vehicle Routing Problem (CVRP) features [7]. These features were proposed earlier in [14], where we demonstrated that the features can be used in algorithm configuration and unsupervised learning. In this study we focus on feature selection and gaining a deeper understanding on feature relevance in meta-optimization. More specifically, we study the predictive accuracy of subsets of the proposed features in a task of algorithm selection of classical CVRP heuristics.

Automatically selecting the most suitable algorithm for solving Traveling Salesman Problems (TSPs) has been studied for example in [16, 5, 10] and for Vehicle Routing Problems (VRPs) in [8, 11, 21, 18]. However, recognizing the most relevant features has received little attention. The questions by Smith-Miles and van Hemert [16] are relevant here: Which of the features prove useful when predicting algorithm performance and solution quality?

According to Rice's framework [15, 17] the requirements for successfully applying algorithm selection on a given problem are: (i) that there are large and diverse problem instance sets available, (ii) there are several competitive algorithms for solving those problem instances, (iii) there is a way to measure the algorithm performance or accuracy, and (iv) access to features that are suitable to describe the problem instances can be recognized. Addressing (i) and (iii) is trivial as the quality of a VRP heuristic algorithm is usually measured with the optimality gap and there are many well-known problem sets for CVRP [22]. For the other requirements we have proposed feature extractors for CVRP in

ESANN 2019 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2019, i6doc.com publ., ISBN 978-287-587-065-0.
Available from http://www.i6doc.com/en/.

[14] and have implemented 15 classical heuristics in [13]. To recognize the most relevant features, we propose two algorithm selection scenarios and use them to show how an ensemble of VRP algorithms can yield high quality solutions.

2 VRP Feature Extraction and Selection

In this study, we consider the classical capaciated vehicle routing problems. Through the years, several algorithms have been proposed to solve them effectively [20]. The modern metaheuristics rely on stochasticity to produce high quality results in a reasonable time, but the stochasticity can make experimental study of heuristic algorithms tricky [12]. Furthermore, the computational experiment conventions have been inconsistent and limited, which makes building algorithm selection scenarios from the results published by different authors impossible. To sidestep these issues, we used our open-source library of classical CVRP algorithms [13]. The classical heuristics are ideal for experimentally verifying the suitability of our proposed feature set because of their deterministic behavior and ability to solve also the larger problem instances.

Problem instances come from the CVRPLIB[1] problem sets A, B, E, F, M, P, X, CMT, RT, Golden, and Li, together with additional CVRP instance sets Gaskell, V, and van Breedam. Together, these form a heterogeneous set of 454 problem instances. For descriptions of these sets we refer to [22] and [13]. To characterize the instances, we further extended our comprehensive set of CVRP features proposed in [14]: The measurements of the number of checked and accepted local search moves, as proposed earlier for TSP in [10], are now included. Additionally, the ratio between integer and non-integer values and their distribution from an exact solver probing are recorded similarly to [5]. The problem instances with the maximum route duration constraint in this study warranted additional features describing the tightness of this constraint. This was calculated using a greedy TSP algorithm and the DBSCAN clusters. In total, our feature extractor set[2] produces 433 values for each instance.

As one can see, we followed the recommendation of [3] and erred on the side of being inclusive instead of risking to omit useful information in the feature construction stage. Unfortunately, this means that the proposed feature set is quite extensive and the issues due to the curse of dimensionality must be addressed, especially since we only have relatively few problem instances. The canonical way is to use principal component analysis (PCA), but also many *feature selection* (FS) methods can be used. These allow recognizing a relevant subset F_r of the set of features [3], which has an additional advantage of being able to omit the computation of any unnecessary features.

[1] `http://vrp.atd-lab.inf.puc-rio.br/index.php/en/`
[2] `http://users.jyu.fi/\%7Ejuherask/selection/FEtable.pdf`

ESANN 2019 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2019, i6doc.com publ., ISBN 978-287-587-065-0. Available from http://www.i6doc.com/en/.

3 Experimental Procedure

Infinite feature values were replaced with a value 10 times the largest real valued measurement and the features were normalized by scaling the values to the [0, 1] range. Furthermore, similarly to Pihera and Musliu [10], we did our experiments also with discretized feature data. As a discretization algorithm we used the MDLP multi-interval algorithm of Fayyad and Irani [1].

Using the algorithm performance and accuracy data from [13] we set the classification label to be the best algorithm for each problem instance. Here, the total solution cost was the primary comparison criteria and the ties were resolved by solution time. Hence, the class label of each problem instance is determined by the best algorithm. Predicting this class among the 14 alternatives (see Fig. 3 and [13] for a list) is our first scenario. Foreseeing that this is a difficult task, we also prepared an easier scenario where the class is predicted among the three most successful algorithms: GAP, PTL, and PS|G2P.

In addition to PCA we applied three different FS methods: minimal-redundancy-maximal-relevance criterion (mRMR) [9], Correlation-based Feature Selection (CFS) [4], and feature boosting with extremely randomized trees [2] (ERTB). Then, four classifiers: 3-nearest neighbor (3NN); multilayer perceptron Softmax classifier (MPL) with hidden layer of $|F_r|$ neurons, sigmoid activation functions, and quasi-Newton backpropagation; random forest with 100 trees and max. depth of 10 (RF); and C-support vector machine (SVM) with $C = 1.0$ and $\gamma = 1/|F_r|$ were trained and evaluated. We used leave-one-out cross validation with and without MDLP discretization, and GNU parallel [19] and Scikit-learn on a 72 core F-series Azure VM to compute the results.

4 Results and Conclusions

It turned out that many of the proposed features do not have any MDLP [1] cut candidates that satisfy the minimum description length criterion. For these, all values belong to the same bin and they no longer contribute to the algorithm selection task. With those features removed, we are left with 163 features on the 14 algorithm scenario and 190 on the three algorithm scenario.

According to the feature importance analysis (see Fig. 4), there is a sharp change in feature importance after 10 features. These 10 features together with the ones recognized by other feature selection methods are presented in Fig. 1 together with the earlier results from the autoencoder (AE) based method [7]. Unsurprisingly, the final scoring reveals that the local search probing (LSP) features are highly relevant in the heuristic selection task, but also features related to exact solving attempt (BCP), constraints (DC), and nearest neighbor digraph (NN) are important.

For the scenario of selecting between the 14 heuristics the baseline accuracy (majority class heuristic) is 21.6%. The best measured accuracy was 48.5% with the RF classifier and 100 features from the mRMR feature selection. This was also the upper limit for the number of features in our experiments, and it

ESANN 2019 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2019, i6doc.com publ., ISBN 978-287-587-065-0.
Available from http://www.i6doc.com/en/.

FS \ Rank	1	2	3	4	5	6	7	8	9	10
CFS	BCP2 (rt.)	LSP2 (avg)	NN3 (avg)	DC12 (LB)	LSP4 (SD)	NN11 (min)	ND2 (CV)	DC3 (dist.)	LSP7 (2q.m.)	DC6 (rt.)
mRMR	BCP2 (rt.)	LSP1 (avg)	NN3 (avg)	LSP7 (2q.m.)	DC6 (rt.)	LSP1 (CV)	DC3 (dist.)	NN11 (min)	ND2 (avg)	LSP2 (avg)
ERTB	LSP1 (CV)	LSP2 (avg)	LSP1 (avg)	LSP12 (CV)	NN3 (CV)	ND4 (min)	LSP10 (SD)	LSP2 (SD)	DC3 (dist.)	DC6 (rt.)
AE [7]	LSP12 (SD)	LSP7 (1q.c.)	LSP7 (1q.w.)	LSP4 (kurt)	LSP2 (CV)	LSP12 (CV)	LSP1 (SD)	ND1 (avg)	ND4 (avg)	LSP3 (skew)
Score	BCP2 (rt.) 20 pts.	LSP2 (avg) 19 pts.	LSP1 (avg) 17 pts.	NN3 (avg) 16 pts.	LSP12 (SD/CV) 16 pts.	LSP1 (CV/SD) 14 pts.	DC3 (dist.) 9 pts.	LSP7 (1q.c.) 9 pts.	LSP7 (2q.m.) 9 pts.	DC6 (rt.) 8 pts.

Fig. 1: Top 10 discretized features on the three algorithm scenario.

is possible that RF could have benefited from additional features. The most problematic decision boundaries seem to be between GAP, PLT, and PS|G2P heuristics (see Fig. 3). Please note that in this scenario the best algorithm can be determined with a very small margin. If we accept predictions where the heuristic is among the three best for each instance the accuracy rises to 75.3%.

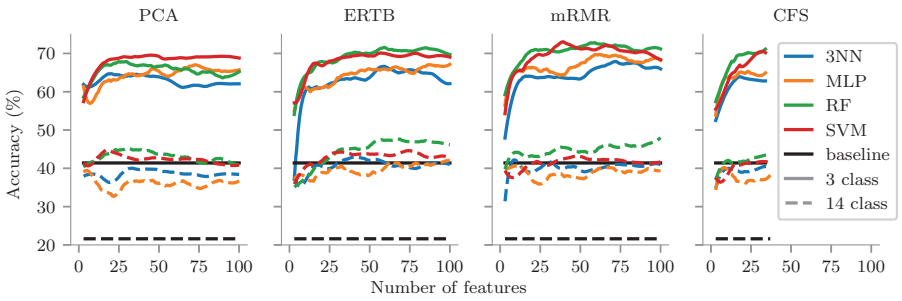


Fig. 2: Savitzky-Golay smoothed classifier accuracy on the discretized data.

For the three algorithm scenario, the baseline is 41.4% and the best accuracy is 74.0% with 75 features chosen using ERTB (see Fig. 4). The accuracy level and the improved the selection accuracy when using discretization are similar to the 3-class experiments in [10] where the accuracy ranged from 64% to 69%. Steinhaus [18] managed to choose the best performing algorithm with 84 % accuracy between three alternatives. However, in our study the algorithms were chosen according their proven performance, which makes our three class scenario more challenging than the one of Steinhaus.

ESANN 2019 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2019, i6doc.com publ., ISBN 978-287-587-065-0.
Available from http://www.i6doc.com/en/.

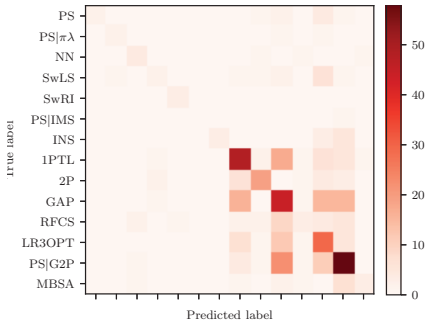


Fig. 3: Confusion matrix for the 14 classical CVRP heuristics.

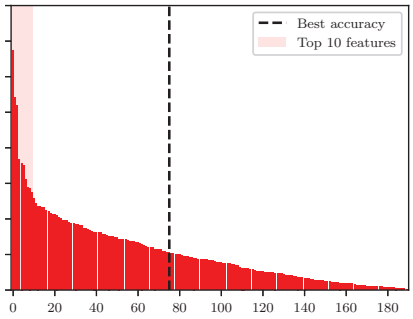


Fig. 4: Feature importances on discretized values in three class scenario.

Regarding the validity of our results, please note that we did not use a separate training set for the feature selection or discretization due to the limited number of samples. We acknowledge that this may induce some positive bias to the selection accuracy. In an extended study it would be advisable to use nested cross-validation, although it would probably necessitate the use of problem generator with the related pitfalls [12, p. 271-273]. Furthermore, while our feature extraction framework is already quite comprehensive, it would be possible to further extend it to describe other VRP variants, and, e.g., to include additional metrics typically used in TSP and VRP solution space analysis. Here, algorithm selection of automatically configured modern metaheuristics would be the most natural, albeit computationally intensive, direction to extend our study to. Aforementioned extensions to the experiments would enable a more in depth analysis of the discriminatory power of the most important features.

Taken together, feature and algorithm selection allowed us to recognize the most interesting and relevant features among the extensive set of 433 features proposed in this study. Additionally, we have shown that the feature set has a good discriminating power and it can be used to leverage an ensemble of vehicle routing heuristics with good results.

References

[1] Fayyad, U. M. and Irani, K. B. (1993). Multi-interval discretization of continuous valued attributes for classification learning. In *In Proceedings of the 13th International Joint Conference on Artificial Intelligence*, page 1022–1029. Morgan-Kaufmann.

[2] Geurts, P., Ernst, D., and Wehenkel, L. (2006). Extremely randomized trees. *Machine learning*, 63(1):3–42.

[3] Guyon, I. and Elisseeff, A. (2006). An introduction to feature extraction. In *Feature extraction*. Springer.

[4] Hall, M. A. (2000). Correlation-based feature selection for discrete and numeric class machine learning. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML '00)*, pages 359–366. Morgan-Kaufmann.

ESANN 2019 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2019, i6doc.com publ., ISBN 978-287-587-065-0. Available from http://www.i6doc.com/en/.

[5] Hutter, F., Xu, L., Hoos, H. H., and Leyton-Brown, K. (2014). Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111.

[6] Kärkkäinen, T. (2015). Assessment of feature saliency of MLP using analytic sensitivity. In *23rd European Symposium on Artificial Neural Networks (ESANN 2015)*, pages 273–278.

[7] Kärkkäinen, T. and Rasku, J. (2019). Application of a knowledge discovery process to study instances of capacitated vehicle routing problems. In Diez, P., Neittaanmäki, P., Periaux, J., Tuovinen, T., and Jordi, P.-P., editors, *Computation and Big Data for Transport - Digital innovations in surface and air transport systems*. Springer. (To appear in).

[8] Nygard, K. E., Juell, P., and Kadaba, N. (1990). Neural networks for selective vehicle routing heuristics. *ORSA Journal on Computing*, 2(4):353–364.

[9] Peng, H., Long, F., and Ding, C. (2005). Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on pattern analysis and machine intelligence*, 27(8):1226–1238.

[10] Pihera, J. and Musliu, N. (2014). Application of machine learning to algorithm selection for TSP. In *Tools with Artificial Intelligence (ICTAI), IEEE 26th International Conference on*, pages 47–54. IEEE.

[11] Potvin, J.-Y., Lapalme, G., and Rousseau, J.-M. (1990). Integration of AI and OR techniques for computer-aided algorithmic design in the vehicle routing domain. *Journal of the Operational Research Society*, 41(6).

[12] Rardin, R. L. and Uzsoy, R. (2001). Experimental evaluation of heuristic optimization algorithms: A tutorial. *Journal of Heuristics*, 7(3):261–304.

[13] Rasku, J. (2019). Meta-survey and implementations of classic capacitated vehicle routing heuristics with reproductions of earlier results. Manuscript, University of Jyväskylä, Finland. (to appear).

[14] Rasku, J., Kärkkäinen, T., and Musliu, N. (2016). Feature Extractors for Describing Vehicle Routing Problem Instances. In *5th Student Conference on Operational Research (SCOR 2016)*, volume 50 of *OpenAccess Series in Informatics (OASIcs)*, pages 7:1–7:13, Dagstuhl, Germany.

[15] Rice, J. (1976). The algorithm selection problem. *Advances in Computers*, 15:65–118.

[16] Smith-Miles, K. and van Hemert, J. (2011). Discovering the suitability of optimisation algorithms by learning from evolved instances. *Annals of Mathematics and Artificial Intelligence*, 61(2):87–104.

[17] Smith-Miles, K. A. (2009). Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys (CSUR)*, 41(1):6.

[18] Steinhaus, M. (2015). *The Application of the Self Organizing Map to the Vehicle Routing Problem*. PhD thesis, University of Rhode Island, US.

[19] Tange, O. (2011). GNU parallel - the command-line power tool. *;login: The USENIX Magazine*, 36(1):42–47.

[20] Toth, P. and Vigo, D. (2014). *Vehicle Routing: Problems, Methods, and Applications*. SIAM, PA, USA.

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

[22] Uchoa, E., Pecin, D., Pessoa, A., Poggi, M., Vidal, T., and Subramanian, A. (2017). New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845–858.

PVI

ON AUTOMATIC ALGORITHM CONFIGURATION OF VEHICLE ROUTING PROBLEM SOLVERS

by

Jussi Rasku, Nysret Musliu, Tommi Kärkkäinen 2019

Journal on Vehicle Routing Algorithms

Journal on Vehicle Routing Algorithms
https://doi.org/10.1007/s41604-019-00010-9

REGULAR PAPER

On automatic algorithm configuration of vehicle routing problem solvers

Jussi Rasku[1] · **Nysret Musliu**[2] · **Tommi Kärkkäinen**[1]

Received: 2 February 2018 / Accepted: 30 January 2019

Abstract
Many of the algorithms for solving vehicle routing problems expose parameters that strongly influence the quality of obtained solutions and the performance of the algorithm. Finding good values for these parameters is a tedious task that requires experimentation and experience. Therefore, methods that automate the process of algorithm configuration have received growing attention. In this paper, we present a comprehensive study to critically evaluate and compare the capabilities and suitability of seven state-of-the-art methods in configuring vehicle routing metaheuristics. The configuration target is the solution quality of eight metaheuristics solving two vehicle routing problem variants. We show that the automatic algorithm configuration methods find good parameters for the vehicle route optimization metaheuristics and clearly improve the solutions obtained over default parameters. Our comparison shows that despite some observable differences in configured performance there is no single configuration method that always outperforms the others. However, largest gains in performance can be made by carefully selecting the right configurator. The findings of this paper may give insights on how to effectively choose and extend automatic parameter configuration methods and how to use them to improve vehicle routing solver performance.

Keywords Vehicle routing problem · Automatic algorithm configuration · Metaheuristics · Meta-optimization

1 Introduction

The vehicle routing problem (VRP) is a practical, relevant, and challenging problem that has been extensively studied by the artificial intelligence (AI) and operations research (OR) communities. One of the main trends in solving VRPs is the shift toward more generic and robust route optimization algorithms [56]. However, optimization models and algorithms are still typically hand-tuned by experts on a case-by-case basis [14, 56]. The need for an expert in this process creates a barrier for the widespread use of the latest scientific advances to solve real-life optimization problems. Therefore, to build more flexible academic and commercial solvers for routing problems, the hand-tuning of the algorithms should be automated. One step toward this goal is to automate the search of the right optimization parameters [14, 31]. This opportunity has been recognized, e.g., by Hutter et al. [30]: "automated algorithm configuration methods ...will play an increasingly prominent role in the development of high-performance algorithms and their applications."

Automatic algorithm configuration [31] (or *parameter tuning* [15]) means off-line modification of an algorithm's parameters. Recently, researchers have proposed several automatic configuration methods, which have proven successful in different domains such as evolutionary computation [55], Boolean satisfiability [1, 30], and mixed-integer programming [25, 34]. In the field of vehicle routing research, Pellegrini and Birattari [48] compared the performance of different metaheuristics with and without automatic algorithm configuration and concluded that, in every instance, the automatically configured version of the solution algorithm yielded better results than the corresponding non-configured one. Furthermore, automatic algorithm

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s41604-019-00010-9) contains supplementary material, which is available to authorized users.

✉ Jussi Rasku
jussi.rasku@jyu.fi

[1] Faculty of Information Technology, University of Jyvaskyla, P.O. Box 35, 40014 Jyväskylä, Finland

[2] Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling, Institute of Logic and Computation, DBAI, TU Wien, 1040 Vienna, Austria

Published online: 22 February 2019

configuration enabled a fair comparison, which makes it a recommended practice for algorithm developers [16].

Besides our preliminary work presented in [52], there is no comprehensive comparative study on automatic algorithm configuration of vehicle routing solvers. Consequently, this study addresses this knowledge gap by investigating the performance of recent automatic configuration methods in the domain of routing algorithms. In particular, our aim is to answer the following questions:

1. Are existing automatic algorithm configuration methods suitable for configuring routing algorithms?
2. How do these configurators compare, and are there methods that should be preferred when configuring routing algorithms?
3. How does the performance of configurators vary with different metaheuristics, vehicle routing variants, and problem instances?
4. How robust are the methods in configuring routing algorithms?

To address these questions, we compare the performance of seven state-of-the-art automatic algorithm configuration methods on metaheuristics for two different variants of the vehicle routing problem. This extends our previous study [52] by adding new configuration targets, improving experimental setup, and including a thorough analysis of the configuration method performance and resulting parameter configuration values. In our experiments we concentrate on optimizing solution quality instead of algorithm runtime on relatively small problem instance sets. Our results confirm that with these conditions the algorithm performance can be clearly improved by using automatic configuration. Also, while some configuration methods perform better, and are more robust in some algorithm configuration tasks, no single method invariably outperforms all the others.

The paper is structured as follows: Sects. 2 and 3 introduce the vehicle routing and the algorithm configuration problems, and describe the automatic algorithm configuration methods used in this paper. Section 4 contains a literature review on algorithm configuration in routing. Section 5 describes the experimental design used to test the configurators followed by Sect. 6 where the numerical results and analysis are presented and discussed. Finally, Sect. 7 concludes the study and proposes topics for future research.

2 The vehicle routing problem

In the classical vehicle routing problem (VRP) the goal is to find optimal *routes* for *vehicles* leaving from a *depot* to serve a specified number of *customers*. Each customer must be visited exactly once by exactly one vehicle. Each vehicle must leave from the depot and return there after serving all customers on its route. Typical objectives are to minimize the number of vehicles and the total length of the routes. Thus, VRP is a generalization of the well-known travelling salesman problem (TSP).

Multiple extensions and variants of VRP have been proposed in the literature. Many of these add new constraints, such as vehicle capacity, maximal route length, and time windows, or introduce new features, such as stochasticity, split deliveries, or multiple depots. For an introduction to different variants and extensions to VRP, refer to [32]. Problems where several constraints and complex objectives are combined to tackle real-world cases are called *'rich'* VRPs [12].

In this paper, we focus on two variants: the capacitated vehicle routing problem (CVRP) and the vehicle routing problem with stochastic demands (VRPSD). In CVRP, each customer has a demand that needs to be fulfilled and each identical vehicle has a capacity that cannot be exceeded. The objective is to find feasible routes so that the number of vehicles and the total distance of routes are minimized. Also the vehicles in VRPSD have limited capacity, but in this variant the exact demands of the customers are not known until they are served. However, the distributions of the demands are known and should be considered in the optimization of the routes [8].

Algorithms for solving the VRP can be divided into two families: exact and heuristic. The aggregated results from Uchoa et al. [59] suggest that exact algorithms cannot consistently solve CVRP instances with more than two hundred customers, and, therefore, different (meta)heuristics have been proposed to solve larger problems. Examples of such methods include simulated annealing (SA), tabu search (TS), evolutionary algorithms (EA), ant colony optimization (ACO), and iterated local search (ILS). For surveys of the topic, refer to Laporte [35] and Mester and Bräysy [42].

Recently, the trend has been toward developing adaptive and cooperative hybrid algorithms [4, 33, 49, 56], but as Hutter et al. [31], Battiti and Brunato [6], and Sevaux et al. [54] have noted, even these tend to have many parameters that need to be fixed. Therefore, these new approaches further emphasize the need for automatic algorithm configuration.

3 The algorithm configuration problem

Many advanced search algorithms have *free parameters* that can be set by the user. The parameters are usually used to balance the algorithm elements and make trade-offs between diversification, intensification, co-operation, and other aspects [61]. These parameters must be configured in order for the method to perform well, which is a nontrivial task. In fact, Smit and Eiben [55] point out that finding the right

values for the parameters "is a complex optimization task with a nonlinear objective function, interacting variables, multiple local optima, and noise." With stochastic local search (SLS, see [24]) algorithms for VRP, this noise comes from the random problem instance selection and stochasticity of the algorithm that is being configured.

One of the main challenges of automatic algorithm configuration according to Eiben et al. [15] comes from the complex interactions between the parameters. Sometimes the parameters can be configured individually, but the result may be suboptimal, whereas trying all different combinations is often impossible due to the sheer number of possible combinations.

Next, we define the algorithm configuration problem and describe approaches that have been proposed to solve it.

3.1 Introducing the problem

Hutter et al. [30] defines the goal of automatic algorithm configuration to be finding a set of parameter values, *a parameter configuration*, for a given *target algorithm* so that the algorithm achieves the best possible performance, or *utility*, on the given input data set. Formal definitions of the problem are presented by Birattari et al. [9] and by Hutter et al. [30].

Depending on when the algorithm parameters are changed, *automatic algorithm configuration* and *parameter control* can be distinguished from each other [15]. Automatic algorithm configuration is the off-line task of finding good values for the parameters before the actual deployment of the algorithm into production. In contrast, parameter control reactively changes the values of the parameters while the algorithm is running.

Algorithm parameters can be *numerical*, *ordinal*, or *categorical*. Numerical parameters have a value that is an integer or a real number. Ordinal and categorical parameters have a finite set of values that the parameter may take, but categorical parameters cannot be ordered in a meaningful way.

3.2 Automatic algorithm configuration methods

The performance of different *configuration methods* (or *configurators*) has been studied earlier, for example, for mixed-integer programming solvers [26], evolutionary algorithms [45, 55], and SAT solvers [1, 30, 37]. Actually, Kadioglu et al. [34] states that there has been a renaissance in the field of automatic algorithm configuration during the first decade of the 21st century. For a recent review of these methods see Hoos [23]. Eiben and Smit [16] presents a similar survey for the evolutionary algorithm tuning community. In addition to exploring the concepts such as robustness and performance measures, they propose a useful taxonomy for the configuration methods.

Recently, the focus has been in overcoming the challenges posed by heterogeneous and large problem instances. Prime examples of this research are recent studies from Styles and Hoos [57] and Mascia et al. [41], where new techniques for reducing computational effort are proposed. These alone are not always sufficient, as finding good parameter configurations still often requires considerable computational resources. Combining automatic configuration with parallel and cloud computing demonstrates how increased availability of computational resources can allow performing the configuration tasks within reasonable time [18, 28].

In this study, we focus on seven state-of-the-art algorithm configuration methods: CMA-ES [21, 64], GGA [1], Iterated F-Race [3], ParamILS [30], REVAC [46], SMAC [27], and URS [64]. The primary criterion to include a method into this study was previously documented use of the automatic algorithm configuration method on VRP or TSP targets. The secondary criterion was the availability of an implementation, as not all recently introduced automatic configuration methods are publicly available. Short descriptions for each of the selected methods are given below.

CMA-ES is a continuous optimization method that was proposed by Hansen [21]. The method is based on the ideas of self-adaptive evolution strategies. It works by sampling new vectors from a multivariate Gaussian distribution, whose covariance matrix is cumulatively adapted using the search evolution path to form rotationally invariant scatter estimates. CMA-ES is known to be reasonably robust and is therefore suitable for automatic algorithm configuration [55]. We extended CMA-ES with a basic discretization scheme to make it support ordinal and categorical parameters, as they were not supported natively. Recently, Vidal et al. [60] used CMA-ES to configure a hybrid VRP solver with eight numerical parameters.

GGA (Gender-Based Genetic Algorithm) is a robust population-based automatic algorithm configuration method proposed by Ansótegui et al. [1]. The method divides the population into two genders, where the selection pressure is only on the other gender. If the dependencies between the configured parameters are specified, they are taken into account in recombination phase. In addition, GGA uses the aging and death of individuals, and random mutations in the new offspring. The parameters of GGA include truncation percentage X for breeding selection, tree branch inheritance probability B, mutation rate M along with mutation variance S, and maximum age A. GGA also requires the initial population size P and number of generations G to be set. Ansótegui et al. [1] did not report the number of optimized parameters being configured in their experiments, but according to [30] the number of parameters for these targets ranges from 4 to

26 with varying composition of categorical and numerical parameters.

F-Race [9] races a finite set of candidate parameter configurations against each other. The method draws inspiration from Maron and Moore [39] where racing was used to solve a similar problem. At each step of F-Race, candidates are evaluated by running the target algorithm on a single problem instance from the training set. A Friedman test is then used to eliminate those configurations that are significantly worse than the best one. The race is terminated when a maximum number of configurations have been sampled, when the predefined computational budget is used, or when the Friedman test indicates that a dominating best configuration is found.

I/F-Race (Iterated F-Race) is an iterated extension of the F-Race proposed by Balaprakash et al. [3]. In I/F-Race, a relatively small set of new candidates is sampled during each iteration. After each race iteration some or all of the surviving candidates are promoted as elite. Each candidate in the new iteration is sampled from a distribution centered on a randomly selected elite candidate. The standard deviations for this distribution are reduced on each iteration [37]. I/F-Race is parametrized by the number of iterations I, the computation budget for each iteration eb_I, the number of candidates for each iteration N_I, and the stopping condition parameter $N_{\min}$. The additional stopping parameter allows a race iteration to be terminated when only $N_{\min}$ candidates are remaining [9, 37], which will help ensure sufficient exploration in the parameter configuration space [10]. The experiments described by Birattari et al. [10] contained at most 12 configured parameters.

ParamILS [30] uses iterated local search (ILS), which has proven to be a good heuristic for solving a variety of discrete optimization problems [38]. It uses an one-exchange neighborhood (one change to one parameter at a time) to search the space of all possible algorithm parameter value combinations. The ParamILS algorithm starts by sampling R random parameter configurations from which it selects the one performing best on the target algorithm. Then it performs a local search where it moves toward a local optimum. To avoid getting stuck, ParamILS employs random perturbations and restart strategies. The ILS approach allows ParamILS to configure any algorithm, even those with many parameters. However, ParamILS is able to handle only ordinal and categorical parameters and requires discretization of continuous parameters.

REVAC (Relevance Estimation and Value Calibration) by Nannen and Eiben [46] is a population-based estimation-of-distribution algorithm. REVAC starts from an assumption of a uniform distribution over the range of each free parameter. It samples new individuals from the constantly updated parameter distributions and aims, through transformation operations with multi-parent crossover (where N best individuals are selected) and an interval shrinking operation governed by a parameter H, to narrow down on the most promising range of each parameter. After the initial population of size M has been evaluated, only one new individual is sampled at each iteration. After the method has finished, relevance estimates can be used to recognize which parameters are essential to the performance of the target algorithm. Categorical parameters are not supported. EA targets configured by REVAC seem to typically have around six parameters [55].

SMAC [27] is the latest configurator from a series of sequential model-based optimization (SMBO) methods [5, 25, 29]. SMBO is an iterative framework for methods that alternate between fitting a regression model, and using that model to predict performance of new candidates. However, SMAC is the first one to extend this paradigm to general algorithm configuration problems. Thus, while Bartz-Beielstein et al. [5] were one of the first to use these black box continuous optimization methods in algorithm configuration, Hutter et al. [27] further extended the applicability of SMBO by adding support for multiple instances, categorical and conditional parameters, and an option to model the parameter configuration response surface more accurately. More precisely, a random forest with instance features is used to create a surrogate model for the algorithm's performance, which is then used in local search of promising configurations. SMAC and ParamILS were used and tested in scenarios with nearly 80 free parameters by Hutter et al. [28].

URS (Uniform Random Sampling) [64] is used in this study as a reference parameter configurator. During an iteration, a candidate is sampled uniformly from the set of all possible parameter configurations and evaluated on all instances in the training set, while keeping track of the best encountered configuration. The method sets a baseline for the more sophisticated configuration methods presented above.

The features of the seven automatic algorithm configuration methods are summarized in Table 1. The first group of columns from the left shows which target algorithm parameter types are supported by the configurator. The second group shows the algorithmic building blocks that the configurators employ to allocate search efforts effectively. Here, effective allocation is one that concentrates the target algorithm evaluations mostly on the promising parameter configurations. The features also ensure that exploration and exploitation are balanced and the stochasticity of the search and target algorithm properly addressed [64]. Sampling is a strategy that all configurators share, but otherwise these methods

Table 1 Features of the automatic algorithm configuration methods used in this study

Method	Parameter types				Algorithmic concepts				Effort reduction techniques						Number of features
	Categorical	Ordinal	Numerical	Continuous	Sampling	Populations	Statistical model	Local search	Restarts	Capping	Racing	Blocking	Sharpening	Seed configs.	Cond. params.	
CMA-ES			✓	✓	✓		✓		✓					✓		6
GGA	✓	✓	✓	✓	✓	✓						✓	✓	✓	✓	10
I/F-Race	✓	✓	✓	✓	✓		✓		✓		✓	✓	✓	✓	✓	12
ParamILS	✓	✓			✓			✓	✓	✓	✓	✓	✓	✓	✓	11
REVAC			✓	✓	✓	✓	✓									5
SMAC	✓	✓	✓	✓	✓		✓	✓			✓	✓	✓	✓	✓	12
URS	✓	✓	✓	✓	✓											5

use different approaches to solve the automatic algorithm configuration problem.

The third group lists effort reduction techniques that are used to save parameter configuration evaluations by changing how candidate configurations are tested: *Capping* [30] terminates the evaluation as soon as it becomes clear that the candidate configuration cannot produce a good parameter configuration. This is convenient when the objective is to minimize the runtime of a target algorithm, but capping is not applicable to solution quality-based configuration that we are doing in this paper. In *racing* [9] good and bad parameter configurations are recognized early by increasing the number of instances and random seeds to evaluate on each step of the race. This technique is closely related to *blocking* [40], where the parameter configuration candidates are evaluated on the same instances and seeds called a block. These techniques control the noise from the variance in the configuration objective between instances and seeds. *Sharpening* [55] controls the number of available problem instances per iteration, and *seed configurations* allow the use of the default or other user-provided parameter configurations at initialization.

The concepts racing, blocking, and sharpening can be combined like in the *intensify* approach of ParamILS and SMAC [27, 30]. There, the history of evaluations on the best parameter configuration is stored and after a new evaluation is added, new configurations are compared against the history on the same problem instances and seeds. New configurations are rejected or declared as the new best-known configuration early, that is, after there is enough evidence.

Conditional parameters, also known as *parameter hierarchies*, were introduced in ParamILS [31]. They allow the user to specify that algorithm parameters are active only with activation of some other parameter, and, thus, available for automatic configuration. This prevents the configurator from changing parameter values when they have no effect. An in-depth survey of techniques and concepts related to automatic algorithm configuration is given by Hoos [23].

The rightmost column of Table 1 shows the total number of features for each configuration method. Out of the listed methods, CMA-ES, URS and REVAC, rely on a smaller number of features compared to others as they do not use the more sophisticated search effort reduction techniques. We are aware that generic continuous optimization methods such as CMA-ES and URS can be augmented with effort reduction mechanisms. For example, in [64] they were used to identify good parameter configurations with minimal evaluation effort, similarly to racing in I/F-Race. However, CMA-ES has also been used in algorithm configuration without such extensions (see, e.g., [60]), and, additionally, our research was better served with distinctly different approaches to automatic algorithm configuration than variations on the I/F-Race pattern. Those interested in extending

continuous optimization methods such as CMA-ES with effort reduction techniques are referred to [64].

4 Automatic algorithm configuration in routing

In this section, we give a short survey on configuring routing algorithms. Because the number of articles on automatic algorithm configuration of VRP algorithms is relatively small, we also survey the relevant studies for the traveling salesman problem (TSP).

Coy et al. [14] recognized the importance of configuring VRP metaheuristics already in 2001 and proposed a procedure to find a set of good parameter values for a target VRP algorithm. Their procedure is based on a statistical design of experiments that requires expert knowledge on each step of the process. Similar to the more recent automatic algorithm configuration methods, their procedure contains local, inexact steepest descent search on the response surface and uses an average of the locally optimal parameter configurations as the final result. The authors concluded that their method managed to improve the default settings of their VRP algorithms, and that the procedure outperformed random parameter sampling.

Pellegrini [47] used F-Race to configure two heuristic algorithm variants solving a specific rich VRP variant, a VRP with with multiple time windows and heterogeneous fleet. Later, Pellegrini and Birattari [48] showed the benefits of automatically configuring VRP metaheuristics. They configured the IRIDIA VRPSD solvers with F-Race and noted that the configured algorithms were able to clearly outperform the out-of-the-box implementations with default parameters. Becker et al. [7] used racing to configure the parameters of a commercial VRP solver on a heterogeneous training set of 47 real-world routing problem instances.

Balaprakash et al. [3] used automatic algorithm configuration on three different routing variants, including VRPSD, to show the advantages of the iterated F-Race over the standard F-Race. Garrido et al. [17] proposed a hyperheuristic where REVAC was used to choose the low-level heuristics solving CVRPs. More recently, Vidal et al. [60] used CMA-ES to automatically configure his record breaking hybrid genetic algorithm (GA) for multi-depot and periodic vehicle routing problems. By using a meta-GA to configure a hybrid GA, Wink et al. [62] were able to reduce the optimality gap on CVRP benchmark instances from Augerat et al. [2] by 91 % (a 0.55 percentage point improvement) compared to an extensively hand-tuned hybrid GA. They were also able to find a new best-known solution for a 200-customer instance in another problem set by using the same automatic configuration approach.

Even though there are studies of automatic algorithm configuration of routing solvers, we were able to find only three comparative studies on automatic algorithm configuration of TSP solvers. Montero et al. [44] compared F-Race, REVAC, and ParamILS to recognize unused operators in solving the TSP with an evolutionary algorithm. In a second study, Montero et al. [45] focused on comparing the performance of the three previously mentioned configurators. They concluded that all three methods have comparable configuration performance and that they are able to improve the performance of metaheuristics targeting single problem instances. Yuan et al. [64] compared CMA-ES, URS, and three other methods in configuring the ACO algorithm for the TSP, and Styles and Hoos [57] introduced two racing protocols that allow different levels of difficulty of problem instances in training and validation sets. To solve the TSP instances they used an implementation of the Lin–Kernighan algorithm (LKH). They concluded that for various sizes of configuration problems, especially for those with many numerical parameters, CMA-ES appears to be a robust algorithm.

In addition to our workshop paper [52] reporting some preliminary results, we are not aware of comparative studies on automatic algorithm configuration methods configuring vehicle routing solvers. VRP solvers have been configured in many studies, but the lack of comparative experiments with different automatic algorithm configurators makes it hard to determine which method one should use when dealing with different VRP metaheuristics. Also, from the existing literature, it is hard to infer how much the solution quality is expected to improve when a VRP metaheuristic is configured with automatic algorithm configuration.

5 Comparison of methods for configuring VRP solvers

Next, we will describe our computational comparison for the automatic algorithm configuration methods. We will explain the experiments that we carried out, and the costs and benefits of adding a layer of meta-optimization on top of a VRP solver. As noted in the study by Hepdogan et al. [22], the configurator for heuristic algorithms should be fast, efficient, and outperform random parameter value selection. Thus, the additional complexity caused by the automatic algorithm configuration must be empirically justified. We will also present the VRP solvers used as the target algorithms.

5.1 Solvers and benchmark problems

VRPH is a heuristic solver library for the CVRP developed by Groër et al. [20]. The library uses the Clarke-Wright construction heuristic and a selection of well-known local search operators: one-point-move (1*ptm*), two-point-move

Table 2 Free parameters of the VRPH and VRPSD solvers

VRPH	Name	Type	Default	Range	VRPSD	Name	Type	Default	Range
Shared	1*ptm*	B	1	{0, 1}	Shared	p	B	0	{0, 1}
	2*ptm*	B	1	{0, 1}		t	B	0	{0, 1}
	two	B	1	{0, 1}	ACO	m	I	7	[1, 100]
	oro	B	0	{0, 1}		τ	R	0.5	[0.0, 1.0]
	tho	B	0	{0, 1}		ψ	R	0.3	[0.0, 1.0]
	3*ptm*	B	0	{0, 1}		ρ	R	0.1	[0.0, 1.0]
EJ	m	I	10	[0, 45]		q	R	$1e7$	[10.0, $1e7$]
	t	I	1000	[0, $1e4$]		α	R	1.0	[0.0, 5.0]
	s	B	0	{0, 1}	EA	p	I	10	[1, $1e3$]
RTR	D	I	30	[1, 100]		mr	R	0.5	[0.0, 1.0]
	δ	R	0.01	[0.0, 0.1]		amr	B	0	{0, 1}
	K	I	5	[0, 100]	ILS	x	R	10.0	[0.0, $1e3$]
	N	I	4	[0, 75]	SA	μ	R	0.01	[0.0, 0.1]
	P	I	2	[1, 10]		α	R	0.98	[0.0, 1.0]
	p	B	1	{0, 1}		ψ	I	1	[1, 100]
	a	B	1	{0, 1}		ρ	I	20	[1, 100]
	t	I	0	[0, 50]	TS	*ttf*	R	0.8	[0.0, 1.0]
SA	T	R	2.0	[0.0, 10.0]		p_t	R	0.8	[0.0, 1.0]
	n	I	200	[0, $1e3$]		p_o	R	0.3	[0.0, 1.0]
	i	I	2	[0, 10]					
	α	R	0.99	[0.8, 1.0]					
	N	I	10	[0, 100]					

The following parameter type key is used: 'B' for Boolean switch (was treated as numerical, or as categorical if the option was available), I for integer values (numerical), R for real values (numerical, continuous)

(2*ptm*), three-point-move (3*ptm*), two-opt (*two*), three-opt (*tho*), and Or-opt (*oro*). These operators can be enabled and disabled using six switches common to all solvers (listed as shared in Table 2). VRPH implements also the cross-exchange operator, but we disabled it because of its tendency to produce infeasible routes.

Other solver parameters do not have an effect on the behavior of the local search operators. Use of the library's local search operators is orchestrated by three metaheuristics: Record-to-Record travel (RTR, 6 + 8 free parameters, where the first 6 are the shared parameters between all VRPH metaheuristics and the other 8 parameters are specific to the RTR metaheuristic), Simulated Annealing (SA, 6 + 5), and neighborhood ejection (EJ, 6 + 3). We omit the descriptions of the heuristics, metaheuristics, and solver parameters and refer the reader to Groër et al. [20] and Table 2.

The other set of solvers we used in our experiments was the IRIDIA VRPSD metaheuristics presented by Bianchi et al. [8]. For local search, the IRIDIA VRPSD solvers rely on only one operator, Or-opt. For the metaheuristic, one can choose between ant colony optimization (ACO, 2 + 6 parameters), evolutionary algorithm (EA, 2 + 3), iterated local search (ILS, 2 + 1), simulated annealing (SA, 2 + 4), and tabu search (TS, 2 + 3). The two shared parameters, p and t, are related to determining the local search move cost approximation method. For a thorough explanation of the solver parameters refer to Table 2 and Bianchi et al. [8].

The size of the training set is an important variable when doing automatic algorithm configuration. If the training set is excessively large, evaluating every parameter set on all instances, as it is done in URS and REVAC, becomes infeasible. Even the more sophisticated configuration methods require a significant subset of a large heterogeneous training set to get a reliable estimate on the parameter configuration utility. Conversely, if the training set is small, there is a danger that it is not a representative sample, and even if the resulting parameter configuration can be used to solve the training set effectively it may have been over-tuned and its performance does not generalize [1]. For our configuration tasks, we decided to use a training and validation set size of 14 instances, which is consistent with the experiences of Becker et al. [7] from configuring real-world routing problems.

We acknowledge that the chosen number of instances is atypically small for automatic algorithm configuration tasks. The reasons leading to small number of training instances was threefold: 1. Out of the compared configuration methods, only the advanced ones support sharpening and blocking. To avoid major modifications and extensions to the less

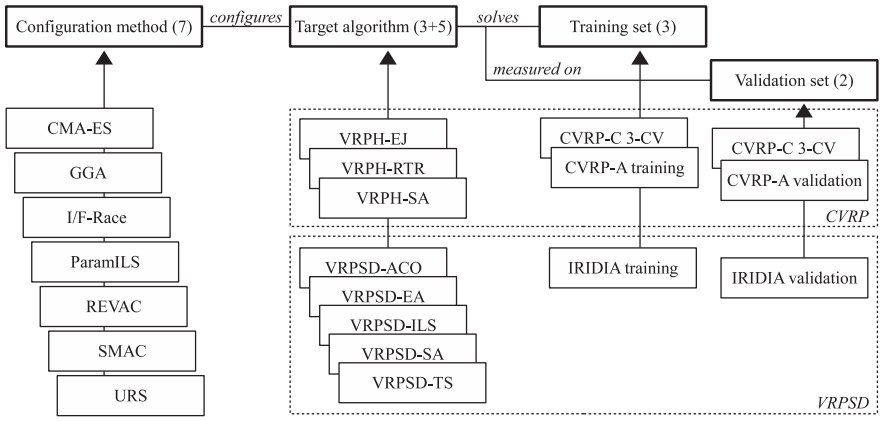


Fig. 1 The experiment setup where the total number of configurators, configuration targets, and VRP problem instance sets are given in the parenthesis

sophisticated configurators, we simply evaluate the entire training set for each parameter configuration candidate. A large training instance set would make this approach infeasible. 2. We wanted to keep the size constant over all targets, and 14 was the size of the smallest problem set used in our experiments. 3. Finally, we would like to point out that a promising practical application of automatic configuration in vehicle routing is the automatic fine tuning of algorithms used in real-world routing [11, 53]. Especially in industry one might not be able to access a large number of specific routing problems because of time and human resource limitations. By using a restricted problem set size we tried to ensure that this study stays relevant to this audience.

For the VRPH solving CVRPs, we used the classic benchmark set CMT with 14 problem instances originating from Christofides et al. [13], which has problems with sizes ranging from 50 to 200 customers. Paired with the three metaheuristics, this creates configuration targets VRPH-EJ-C, VRPH-RTR-C, and VRPH-SA-C. We used a threefold cross-validation with stratified sampling by problem size for this benchmark set because dividing this set into separate training and validation sets would have produced prohibitively small problem sets.

In order to examine the effect different problem sets can have on configuration performance, and how well the performance gains generalize to similar problems, we used the A and B CVRP sets from Augerat et al. [2]. These sets have 27 and 23 instances with sizes from 31 to 79 customers. The problem sizes and demand distributions are similar, but the customers in set A are uniformly distributed and in B clustered. To fix one variable, the size of the training set, we decided to use a subset of the original instance set in our experiments. We used a stratified sampling of 14 instances from set A and set B, to construct disjoint training and validation sets. This forms the next three configuration targets: VRPH-EJ-A, VRPH-RTR-A, and VRPH-SA-A.

Finally, to test the IRIDIA VRPSD solvers, we used training and validation subsets, again with a stratified sampling of 14 instances each, from the IRIDIA problem set of 120 randomly generated instances with 50 to 200 customers [8]. Supporting material[1] for [8] includes the algorithms and description of the problem instances. The configuration targets for VRPSD are: VRPSD-ACO, VRPSD-EA, VRPSD-ILS, VRPSD-SA, and VRPSD-TS.

The experimental setup is illustrated in Fig. 1. To summarize, we selected seven automatic configuration methods, three target algorithms solving the CVRP, and five solving the VRPSD. For each of the eight target algorithms solving a set of VRP benchmarks, the configurators try to find a set of parameters that maximize the quality of the solutions produced. This means there are interchangeable objects in the three levels: a problem set, a solver with the metaheuristic and local search operators, and a configurator that optimizes solver performance. In addition, two of these levels have free parameters: the solver has parameters being configured and the configurator has its own parameters that must be set manually by the experimenter. Furthermore, the selection of the problem instances to the training and validation sets may cause variability in the configuration performance.

5.2 Experimental design

The VRP solvers used in this study were considered to be black boxes from the configurators' point of view. Only the free parameters and their ranges were known prior to starting the configuration task.

When using heuristic algorithms, reaching the optimum in a reasonable time is not guaranteed. Thus, we cannot use the total running time of the target algorithm to compare parameter configuration efficiency, even if it is a more common target for automatic algorithm configuration (see, e.g., [30]). Instead of solver time, we decided to optimize for solution quality and set a 10 CPU second cutoff for all

[1] http://iridia.ulb.ac.be/supp/IridiaSupp2004-001/index.html.

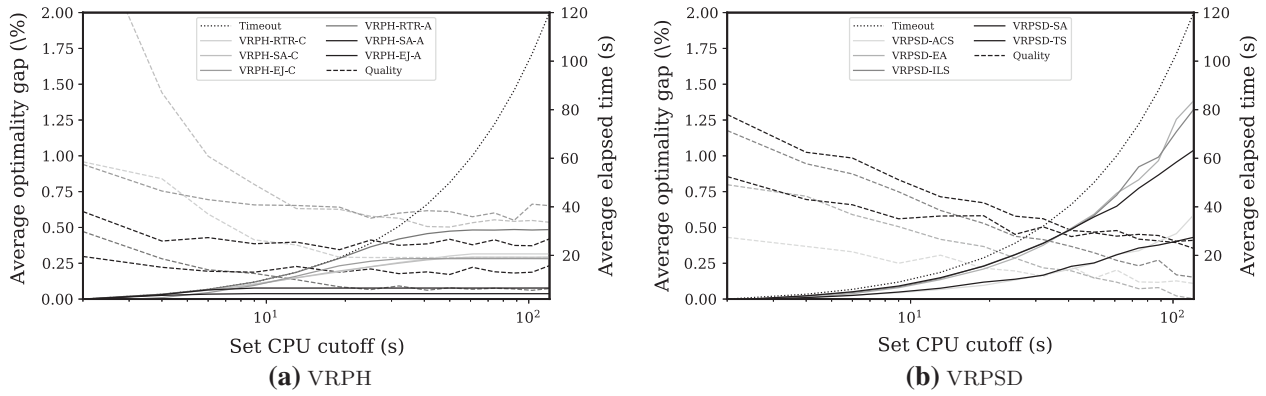


Fig. 2 Effect of a cutoff to the elapsed algorithm runtime and resulting solution quality on an automatically configured solver

invocations of VRP solvers. If an algorithm had not stopped after 10 s, it was terminated and with the quality of the current best solution.

Choosing 10 s as the CPU cutoff was not arbitrary. The experimental design presented here already creates a large number of combinations to test, and the stochasticity in the target algorithms and in the configurators themselves requires multiple configuration trials for statistical reliability. Thus, the algorithm runtime had to be reasonably small. Figure 2 illustrates the effect cutoff has on actual elapsed time of the algorithm and the resulting solution quality. VRPH solvers are able to utilize the more generous computational resources only with two targets and the additional time gives diminishing returns after 10 s. Also, while considering the effect of this decision, please note the scaling of the x-axis. The curves have a negative exponential multiplier, and thus, the quality improvement is logarithmic (not linear) when the CPU time is increased for VRPSD targets. Furthermore, the selected cutoff is in line with [19], where the Augerat et al. [2] instances are solved within 0.3 % of optimal solution on average in 3.5 s. Similarly, Groër [19] reports that the larger CMT [13] instances are solved close to optimality on average in 12.94 s (RTR) or 21.9 s (EJ) on a 2.3 GHz AMD processor.

From Fig. 2 we also see that VRPSD can utilize the more generous computational resources. In their experiments Pellegrini and Birattari [48] used a slightly more generous CPU timeout of 30 s for these targets, although with significantly slower AMD Opteron 244 processor than the Intel Xeon E7 used in this study. Yuan et al. [64] used a 5 s cutoff for ACO solving medium sized TSP instances, which was then configured using, e.g., CMA-ES. These further validate the decision of using a relatively strict cutoff in such configuration scenarios. Furthermore, because our comparison already had many changing variables (configurator, target metaheuristic and its parameters, problem sets and instances, and local search operator selection), we decided to fix the cutoff for all targets.

We acknowledge that the runtime of a routing algorithm to solve large real-world problem with thousands of customers may be measured in hours, especially in cases with complex constraints such as dynamic travel times, compartment compatibilities, or other extensions including separate pickups and deliveries or a heterogeneous fleet [7]. Despite this, modern metaheuristics are usually able to find proper solutions for all but the largest benchmark problems in a few seconds.

As the utility metric we use an aggregated solution objective function value over the training instance set. Aggregation is a sum over the estimated objective function values for the resulting VRP solutions in the instance set. Thus, the configuration task is to minimize:

$$\underset{\theta}{\text{minimize}} \quad \hat{c} = \sum_{i \in I_t} \hat{a}(i, \theta) \tag{1}$$

Here, $\hat{c}$ is the utility metric estimator, I_t is the training problem set, $\hat{a}$ is an estimator for the utility function for algorithm a a, which in turn solves problem instance $i \in I_t$ guided by parameter configuration θ. Because the metaheuristic algorithm a is stochastic, we also define $\hat{a}$ to be an estimator of the algorithm solution quality. In practice, the estimate is formed through repeated evaluation of the algorithm on the same problem instance and parameter configuration, but with different random seed.

Even if we did not solve rich problems in this study, the number of evaluations that can be allocated into finding a reasonably good parameter configuration remains an important factor. Especially since we would like to keep this study relevant to the operations research practitioners who are solving large-scale real-world vehicle routing instances with complex constraints who could benefit from

automatic algorithm configuration. In their case, the number of solver invocations cannot be too large.

In our experiments each configuration task was given an evaluation budget that defines the number of solver invocations allowed during a configuration task. One call of the routing solver with one parameter configuration and one problem instance is counted as one evaluation. To compare the effect of different budgets, every configurator—solver combination was run with evaluation budgets of 100, 500, and 1000. In addition, VRPSD-ACO was configured with an evaluation budget of 5000 to see the effect of a larger budget.

Whenever the option was available, configurators were set to expect non-deterministic behavior from the target algorithm. Thus, allocating the budget for new parameter configurations and problem instances and controlling the stochasticity through additional evaluations was left for the configurator.

GGA, I/F-Race, ParamILS, and SMAC use effort reduction techniques that can save evaluations, for example, by evaluating only a subset of the training instances on each iteration, whereas CMA-ES, REVAC, and URS evaluated all problem instances in the training set on each iteration. All the tested target parameters could be represented with an integer or real number with a suitable range and an optional discretization step (in ParamILS).

All seven automatic algorithm configuration methods contain a set of parameters related to the basic technique and its actual implementation. The default parameters provided by the original authors were used in the experiments whenever possible. A full listing of the configurator parameters can be found in the online supplementary material for this paper. Also, the target algorithm defaults were provided as a seed configuration for all configurators excluding REVAC and URS, which did not support it.

CMA-ES is claimed to be quasi-parameter-free [21], so we did not change the initial parameters of the Python implementation.[2] For optimization, all continuous parameters were normalized between 0.0 and 1.0 with an initial standard deviation of $\sigma_0 = 0.5$. The restart mechanism of this CMA-ES implementation was not used, because it is not applicable to fixed and relatively small evaluation budgets. Also, instead of relying entirely on the self-adaptive parameters, on tasks with an evaluation budget of 100, we used a population size of 7 to help CMA-ES stay within the specified budget.

For GGA, we used the implementation of [1] with the default values (10, 90, 10, 3, 10) for (X, B, M, A, S). Ansótegui et al. used several different population and generation ratios. We decided to use the $P/G = 2/1$ ratio to avoid extinction of the population. For evaluations with the budget of 5000, we used the $P/G = 4/3$ ratio that Ansótegui et al. [1] used to configure SAT solvers. Using this ratio we set the population size and number of generations carefully on a budget-to-budget basis, because in order to use the evaluation budget effectively, the evolutionary process must converge at the right time. Because GGA did not respect the specified budget for evaluations, setting G and P was the only way to get it to spend approximately the right number of target algorithm evaluations. Also, GGA required each instance to be paired with a fixed random seed.

F-Race is implemented for the statistical software environment R. The Iterated F-Race automatic algorithm configuration method `irace`[3] from López-Ibánez et al. [37] uses it to implement the iterated variant of the racing method. We used defaults, but for an evaluation budget of 100, the parameter eb_1, which governs the computation budget for each iteration step, was set to 60 to make I/F-Race more closely respect the evaluation budget.

ParamILS [30] and SMAC [27] are available online.[4] For ParamILS, we used linear discretization of 10 steps for each of the continuous free parameters. Selecting the most suitable discretization for each parameter can be seen as an additional level of configuration, and therefore, it was omitted from this study. To take the stochastic nature of the target algorithms into consideration, we used the ParamILS built-in FocusedILS approach to limit the time spent on evaluating each parameter configuration [30]. SMAC was used with default parameters. Its ability to use problem instance characteristics to improve the predictive power of the surrogate model for the target algorithm was not used.

For REVAC, we used the implementation from Montero et al. [44, 45]. To allow it to use the evaluation budget effectively, the control parameters M, N, and H were set using the ratios recommended in the literature: $N = M/2$ and $H = N/10$ with a minimum value of 2 for H. For evaluation budgets of 100, 500, and 1000, M was given values 5, 10, and 20, respectively.

Similarly to [64], each configuration task had 10 trials for VRPH-A and VRPSD targets. In threefold cross-validation of the VRPH-C targets, the cross-validation was repeated five times. The folds were different between repetitions but the same between the target algorithms and budgets. This blocking guarantees that the configuration tasks are comparable between methods. After configuring the algorithms, the resulting parameter configurations were evaluated by

[2] Version 0.9.93.4r2658, http://www.lri.fr/~hansen/cmaesintro.html.

[3] Version 0.9, http://iridia.ulb.ac.be/irace/.

[4] Versions 2.3.5 (ParamILS) and 2.0.2 (SMAC), http://www.cs.ubc.ca/labs/beta/Projects/ParamILS/http://www.cs.ubc.ca/labs/beta/Projects/SMAC/.

Table 3 Median automatic configuration results for the VRPH CMT targets with threefold cross-validation

EB	CMA-ES	GGA	I/F-Race	ParamILS	REVAC	SMAC	URS
VRPH-EJ-C, defaults: 0.96 (0.12)							
100	*0.99(0.08)*$^-$	*0.86(0.09)*$^+$	***0.72(0.09)***$^+$	*0.93(0.10)*$^+$	1.00(0.22)	0.81(0.10)	**0.77(0.12)**
500	*0.83(0.12)*$^+$	*0.79(0.10)*$^-$	0.71(0.06)	0.76(0.10)	0.68(0.09)	0.70(0.09)	0.73(0.10)
1000	0.78(0.09)	*0.81(0.09)*$^-$	0.66(0.06)	0.71(0.09)	0.73(0.09)	0.69(0.09)	0.75(0.09)
VRPH-RTR-C, defaults: 1.42 (0.06)							
100	1.24(0.14)	*0.90(0.14)*$^-$	*1.04(0.15)*$^+$	1.00(0.11)	1.22(0.25)	0.94(0.06)	**0.81(0.14)**
500	0.82(0.15)	*0.84(0.03)*$^-$	0.75(0.05)	0.91(0.17)	1.06(0.14)	0.78(0.12)	0.83(0.10)
1000	0.76(0.08)	*0.85(0.06)*$^-$	**0.63(0.09)**	0.67(0.06)	0.74(0.03)	0.79(0.06)	0.78(0.06)
VRPH-SA-C, defaults: 0.80 (0.05)							
100	1.70(0.52)	0.88(0.18)	***0.73(0.04)***$^+$	0.89(0.09)	1.68(0.36)	0.77(0.03)	1.39(0.26)
500	1.09(0.21)	*0.89(0.11)*$^-$	*0.81(0.08)*$^-$	0.84(0.08)	1.29(0.10)	**0.78(0.04)**	1.04(0.18)
1000	1.03(0.16)	*0.89(0.10)*$^-$	0.79(0.08)	0.75(0.09)	1.15(0.12)	0.77(0.03)	0.97(0.13)

Results are given as percentage from the aggregated best-known solution (relative optimality gap). Statistically better results of the single best, or pair of best solvers (in cases where no single configurator dominated), are in bold typeface. Evaluation budget (EB) violations of more than 5% are italicized, with $^+$ indicating exceeding and $^-$ falling short of the budget

running them on all problem instances 10 times and calculating the aggregated objective cost for each repetition.

All configuration tasks were run on a computing server with 64 Intel(R) Xeon(R) CPU E7 2.67 GHz cores and 1 TB of RAM. The server was running the OpenSUSE 12.3 operating system.

6 Numerical results and analysis

The experiment data contain results of 2695 configuration runs.[5] Together with the verification evaluations these took around 250 CPU days to compute. Considering all results, automatic algorithm configuration methods were able to find improved configurations over defaults in 84.1% of the configuration trials. This is a promising result considering that the smallest used budget of 100 evaluations is a very tight restriction for automatic algorithm configuration. Also, the default parameters of the VRPH solvers are expected to be tailored for typical scientific benchmark instances such as those we used. The suitability of the defaults is even more prominent in the case of VRPSD, where the solvers and the benchmarks instances come from the same source.

[5] $(((3 \times 3) \times (3 \times 5)) + ((3 + 5) \times 3 + 1) \times 10) \times 7$ The 3 VRPH-C targets with 3 different budgets were configured using threefold cross-validation repeated 5 times. The 3 VRPH-A targets and 5 VRPSD targets, each with 3 different budgets, plus (1) VRPSD ACS with a budget of 5000, with 10 trials each. All the previous experiments were done for all the 7 automatic configuration methods.

6.1 Performance of the configurators

A median aggregated solution quality and median absolute deviation were calculated for each configuration task. The median was taken over a set of 10 evaluations on validation set for each of the 10 resulting parameter configurations (that is, over 100 aggregated quality values).

The median was used, because we were mostly interested in measuring the typical performance of a configurator. Meanwhile, the median absolute deviation gives an estimate for the robustness of the configurators. The aggregated solution quality for each configuration task is given as a deviation from the sum of best-known solutions for instances in the validation set (relative optimality gap). The VRPSD benchmarks had no recorded best-known solutions, so we used the best observed solution for each problem instance as the best-known solution. Please note that the result data, with a full set of figures and tables with other descriptive statistics, can be found in the online supplementary material.

The results in Tables 3, 4, and 5 are grouped by the target algorithm. The -C and -A suffixes are used to differentiate between the CMT and Augerat et al. [2] benchmarks for the VRPH targets. Each row shows results for a single configuration task consisting of a triplet: target algorithm, evaluation budget, and problem instance set. When comparing the results we note that out of the tested configurators only ParamILS and SMAC strictly, and URS and REVAC closely, respected the evaluation budget. Other methods frequently ignored the input parameter for the evaluation budget and exceeded or fell short of the budget. Results deviating from the given budget by more than 5% are marked with italics. A nonparametric Mann–Whitney U-test ($p < 0.05$) was used with the Bonferroni adjustment

Table 4 Median automatic configuration results for the VRPH Augerat et al. [2] targets on the validation set B

EB	CMA-ES	GGA	I/F-Race	ParamILS	REVAC	SMAC	URS
VRPH-EJ-A, defaults: 0.73 (0.03)							
100	0.42(0.06)	*0.43(0.08)*$^{+}$	*0.50(0.14)*$^{+}$	0.44(0.08)	0.42(0.07)	**0.41(0.06)**	0.38(0.03)
500	*0.40(0.06)*$^{+}$	*0.37(0.02)*$^{-}$	0.37(0.04)	0.42(0.09)	0.42(0.06)	0.37(0.02)	0.38(0.04)
1000	*0.38(0.03)*$^{+}$	0.40(0.05)	**0.37(0.04)**	0.37(0.02)	0.42(0.05)	**0.35(0.02)**	0.37(0.03)
VRPH-RTR-A, defaults: 1.40 (0.05)							
100	0.38(0.07)	*0.36(0.17)*$^{+}$	*0.38(0.20)*$^{+}$	0.44(0.08)	0.50(0.16)	0.62(0.21)	0.37(0.22)
500	*0.35(0.06)*$^{-}$	0.34(0.06)	0.42(0.23)	0.45(0.16)	0.32(0.08)	0.66(0.26)	0.37(0.09)
1000	0.34(0.06)	*0.31(0.09)*$^{-}$	0.34(0.11)	0.39(0.22)	0.38(0.11)	0.63(0.27)	0.34(0.06)
VRPH-SA-A, defaults: 0.90 (0.01)							
100	0.90(0.19)	*0.65(0.12)*$^{+}$	***0.61(0.16)***$^{+}$	0.70(0.24)	0.98(0.10)	**0.65(0.12)**	0.73(0.17)
500	*0.62(0.25)*$^{+}$	*0.48(0.17)*$^{-}$	**0.39(0.11)**	0.61(0.12)	0.66(0.10)	**0.38(0.24)**	0.58(0.20)
1000	*0.41(0.22)*$^{+}$	*0.38(0.24)*$^{-}$	0.33(0.20)	0.38(0.22)	0.53(0.17)	0.34(0.15)	0.34(0.23)

Table 5 Median automatic configuration results for the VRPSD IRIDIA targets on the validation set

EB	CMA-ES	GGA	I/F-Race	ParamILS	REVAC	SMAC	URS
VRPSD-ACO, defaults: 0.63 (0.04)							
100	**0.39(0.07)**	*0.43(0.07)*$^{+}$	*0.41(0.04)*$^{+}$	0.39(0.04)	0.43(0.06)	**0.37(0.02)**	0.39(0.04)
500	***0.31(0.05)***$^{+}$	0.38(0.03)	0.36(0.05)	0.36(0.04)	0.41(0.05)	**0.30(0.08)**	0.35(0.05)
1000	***0.28(0.06)***$^{+}$	*0.37(0.03)*$^{-}$	0.37(0.03)	0.33(0.07)	0.37(0.03)	**0.27(0.07)**	0.35(0.06)
5000	0.30(0.09)	0.32(0.06)	0.27(0.06)	0.26(0.06)	0.40(0.02)	**0.16(0.06)**	0.31(0.05)
VRPSD-EA, defaults: 0.77 (0.03)							
100	0.72(0.10)	*0.68(0.08)*$^{+}$	*0.57(0.04)*$^{+}$	0.59(0.07)	0.67(0.06)	**0.53(0.05)**	0.58(0.05)
500	*0.62(0.06)*$^{+}$	*0.57(0.07)*$^{-}$	**0.48(0.04)**	0.57(0.06)	0.58(0.04)	0.51(0.04)	0.49(0.05)
1000	0.56(0.07)	*0.56(0.06)*$^{-}$	0.48(0.04)	0.55(0.06)	0.57(0.04)	0.49(0.04)	0.49(0.05)
VRPSD-ILS, defaults: 0.78 (0.04)							
100	0.71(0.06)	*0.75(0.06)*$^{+}$	*0.74(0.07)*$^{+}$	0.76(0.03)	0.78(0.05)	0.72(0.03)	*0.78(0.03)*$^{+}$
500	*0.73(0.03)*$^{+}$	*0.72(0.08)*$^{-}$	0.76(0.05)	0.76(0.03)	0.77(0.13)	0.71(0.07)	0.78(0.03)
1000	*0.73(0.03)*$^{+}$	***0.71(0.04)***$^{-}$	0.74(0.08)	0.76(0.03)	0.77(0.13)	0.77(0.03)	0.78(0.03)
VRPSD-SA, defaults: 0.79 (0.04)							
100	0.83(0.05)	***0.77(0.07)***$^{+}$	*0.88(0.06)*$^{+}$	0.88(0.08)	1.18(0.23)	0.86(0.05)	*0.87(0.06)*$^{+}$
500	0.84(0.03)	0.78(0.06)	0.87(0.06)	0.85(0.06)	0.88(0.12)	0.88(0.04)	0.86(0.06)
1000	0.84(0.03)	*0.77(0.06)*$^{-}$	0.82(0.03)	0.85(0.06)	0.88(0.11)	0.85(0.02)	0.86(0.06)
VRPSD-TS, defaults: 1.86 (0.13)							
100	**0.75(0.08)**	*1.80(0.05)*$^{+}$	*1.75(0.07)*$^{+}$	**0.72(0.14)**	1.77(0.07)	1.73(0.10)	1.78(0.04)
500	***0.60(0.11)***$^{+}$	*1.74(0.09)*$^{+}$	1.74(0.11)	**0.61(0.12)**	1.73(0.09)	1.74(0.07)	1.70(0.04)
1000	**0.59(0.10)**	*1.75(0.09)*$^{-}$	1.80(0.08)	**0.59(0.10)**	1.73(0.09)	1.83(0.10)	1.70(0.04)

to test whether the differences to defaults and other configurators were statistically significant. Whenever a single dominating method for an algorithm target was not found, existence of a dominating pair of methods was checked. Statistically significantly better automatic algorithm configuration methods (or pairs) for each configuration task are marked in bold typeface.

If we consider only the best configuration found for each configuration task, and average over all targets, automatic algorithm configuration was able to reduce the optimality gap by 0.72. This is a 69.7% improvement compared to using default parameters. According to our results, this is the improvement that can be expected when a suitable configurator is used. On average, the optimality gap was reduced by 0.27 (a 25.2% improvement over defaults). The greatest single improvement was seen on VRPSD-TS, where ParamILS was able to reduce the optimality gap on by 1.51, allowing an 81.2% improvement over defaults.

Before focusing on the differences between configuration methods, we compare the performance of more sophisticated methods against the reference configurator that was

Table 6 Configurator performance on the metaheuristics, which are split into three difficulty classes (D_c, 1 being easiest and 3 hardest)

Target	$\#P_{(B/I/R)}$	d.s.	D_c	CMA-ES	GGA	I/F-Race	ParamILS	REVAC	SMAC	URS	Rank on defaults	EB's		
												100	500	1000
VRPH-EJ-C	9 (7/2/0)	3	3			1				1	2	**1**	**1**	**2**
VRPH-RTR-C	14 (9/4/1)	3	3			1				1	3	3	**2**	**1**
VRPH-SA-C	11 (6/3/2)	1	1			1			1		**1**	**2**	**3**	**3**
VRPH-EJ-A	9 (7/2/0)	3	3			1			2	1	**1**	**2**	**2**	**3**
VRPH-RTR-A	14 (9/4/1)	3	3								3	**1**	**1**	**1**
VRPH-SA-A	11 (6/3/2)	2	2			2			2		2	**3**	**3**	**2**
VRPSD-ACO	8 (2/1/5)	3	1	3					4		**1**	**1**	**1**	**1**
VRPSD-EA	5 (3/1/1)	3	3			1			1		2	**2**	**2**	**2**
VRPSD-ILS	3 (2/0/1)	3	1		1						4	3	4	4
VRPSD-SA	6 (2/2/2)	1	1		1						3	5	5	5
VRPSD-TS	5 (2/0/3)	2	2	3			3				5	4	**3**	**3**
	Total wins			6	2	7	3	0	**10**	3				

Based on the results, the suitability of the default parameters (d.s.) is estimated. Here 1 stands for good default parameters. Middle columns keep score for the statistically significantly best configurators for each target. The #*P* column indicates the number of parameters for each metaheuristic, and $\#P_B$, $\#P_I$, and $\#P_R$ their division into **B**oolean, **I**nteger, and **R**eal valued parameters. Rightmost columns show the ranking between VRP solvers with the default and the automatically configured parameters. Also, the table illustrates how the ranking changes when the solvers are configured. Results with bold ranks are of better or equal utility in comparison to the best solver for that instance set with default parameters

the uniform random sampling (URS). Contrary to expectations, the configurators are able to produce statistically significantly better results over URS only in 35.8% of the pairwise Mann–Whitney U tests ($p < 0.05$). In contrast, the observed performance was worse than URS in 35.3% of the pairwise comparisons. However, as can be seen from the main result tables (Tables 3, 4, and 5), the results in contrast to URS are not evenly distributed. Additionally, the two Augerat et al. [2] instance sets were included to see how well the performance gains of automatic algorithm configuration generalize to similar problems. Therefore, it was expected a see that random strategy (URS) works well. According to Coy et al. [14], such behavior can be caused by a large heterogeneity among the problem instances in a problem set, but it seems this applies also to heterogeneity between training and validation sets. Also I/F-Race, and to some extent SMAC, show a good generalization ability from a problem set to another on these targets.

Another noteworthy observation is that REVAC seems to struggle with all algorithm targets and it is able to beat URS only in 6.1% of the pairwise parameter configuration comparisons. As a whole, our results indicate that performance of REVAC on routing targets is worse than that of SMAC and I/F-Race. This is in contrast to results from Montero et al. [44], where they reported only small differences between F-Race, ParamILS, and REVAC in automatically configuring an EA for the TSP. If we leave out the Augerat et al. [2] targets and REVAC from the pairwise comparisons against URS, configurators are better than URS in 50.4% and worse in 26.4% of the tests. In addition, as the evaluation budget is increased, the advantages of more sophisticated configuration methods become more apparent (see, e.g., ACO in Table 5).

Table 6 shows that SMAC, I/F-Race, and CMA-ES are the methods that most frequently tend to find good parameter configurations for the VRP metaheuristics in this study. However, please note that CMA-ES, I/F-Race and GGA have the tendency to exceed the specified evaluation budget. Of the statistically significant results, only I/F-Race for the VRPH-SA-C and VRPH-SA-A targets with a budget of 100 exceeded the given budget by more than 15% (by 25 % to be exact) and this may give them some unfounded advantage. Still, considering the competitive performance of I/F-Race on those targets with budgets of 500 and 1000 this should not induce significant bias into our analysis.

I/F-Race, together with SMAC, and in some cases URS, seem to be the configuration methods to choose when faced with a highly limited computational budget. These methods are able to quickly produce relatively high-quality parameter configurations. However, no single method clearly dominates the others. The summary of winning configurators in Table 6 illustrates that different automatic algorithm configuration methods are successful with different targets, although if a method manages to find good parameter configurations for a target with a specific evaluation budget, it seems to be able do this with other budgets as well.

Regarding the parameter types and composition, our results support the observation made by Yuan et al. [64] that CMA-ES is suitable for configuration tasks with a high number of continuous parameters. We also note that I/F-Race

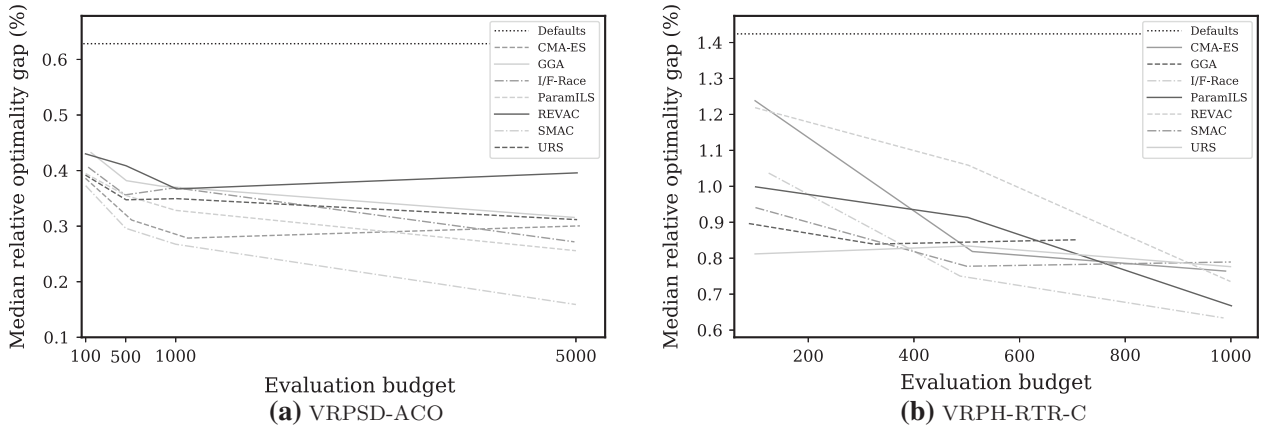


Fig. 3 Comparison of different configurators on two selected targets

seems to perform well in tasks that contain many Boolean parameters.

Random sampling (URS) works surprisingly well on VRPH-EJ, VRPH-RTR, and VRPSD-EA. The ruggedness of the configuration target fitness landscape probably interferes with the exploitation schemes of the more advanced automatic configuration methods. URS is, by definition, very explorative and is therefore capable of effectively exploring large areas of the parameter configuration search space. Hutter et al. [27] utilizes this in another configurator called ROAR, which can be described roughly as URS with configuration effort reduction techniques. However, as we can see from the result of configuring VRPSD-TS, sampling is not a strategy without disadvantages.

SMAC dominates in configuring VRPSD-ACO with a budget of 5000 evaluations (Fig. 3a). We also observe a possible case of over-tuning in the results of REVAC and CMA-ES. The effect is smaller with CMA-ES so there is a possibility that CMA-ES cannot effectively use the larger budget and prematurely converges to a local optimum.

Overall, despite the relatively small training set size, there is reasonably little over-tuning as can be seen from Fig. 4. In a case of over-tuning the figure would show good performance on training set, but poor performance on validation set. That is, the data point would be clearly above the dashed line designating unequal performance between the sets. The largest over-tuning effect is seen on the left side of the figure where SMAC automatically configures VRPH-RTR-A (Fig. 4a). In fact, VRPH-RTR-A and VRPH-SA-A targets show relatively large difference in training and validation set solution quality. This is due to the inherent differences of the training and validation sets with the Augerat et al. [2] targets. This is not surprising as this benchmark set was included to test how well the performance gains of automatic algorithm configuration transfer to solving similar problem

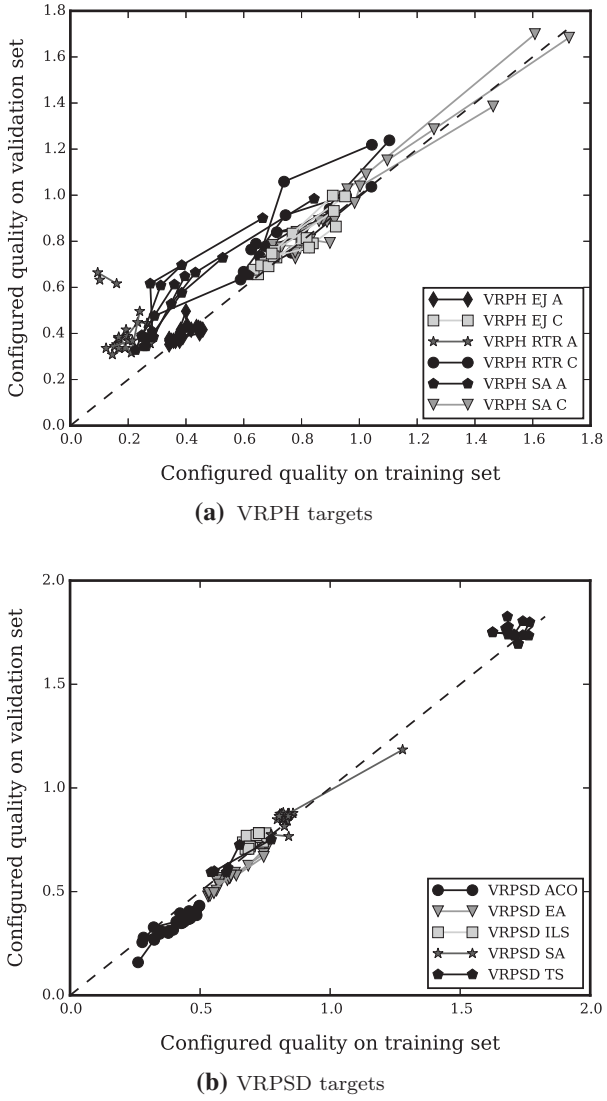


Fig. 4 Solution quality on validation versus training set

instances. Furthermore, if the data is examined per configuration method, none of the configurators shows a clear tendency to over-tune.

To examine robustness, we study the median absolute deviations (MAD) in Tables 3, 4, and 5. Out of the tested configuration methods, SMAC is the most robust. That is, is able to consistently produce good parameter configurations. It closely followed by I/F-Race as they both have the lowest average MAD and still provide good automatic configuration performance. However, the differences over all experiments are small, and even SMAC fails to always produce good parameter configurations for configuring some targets such as VRPH-RTR-A and VRPSD-TS where in turn CMA-ES excels.

6.2 Configuration target difficulty

By comparing the configuration performance of URS against other methods in Tables 3, 4, and 5, we recognize three difficulty classes in the tested VRP algorithm targets (see Table 6). The first class consists of targets VRPH-SA-C, VRPSD-ACO, VRPSD-ILS, and VRPSD-SA, which seem to have relatively smooth parameter configuration landscapes where sophisticated intensification and search techniques work well. Pellegrini and Birattari [48] reported similar results that showed that ACO, ILS, and SA are metaheuristics that respond favorably to automatic configuring and that F-Race outperforms random sampling on these targets. Note that the default parameters for the targets VRPH-SA-C and VRPSD-SA seem to be already very good because only 25.6% of the parameter configurations produced by the configurators show improved performance over them. GGA seems to be the best method to automatically configure VRPSD-SA, although it, likewise, struggles to find better configurations than the defaults. For the other targets in this class, 95.3% of the produced configurations are better than the defaults. As we can see from Fig. 3, the results also seem to be getting better as we increase the evaluation budget.

The second class of automatic algorithm configuration problems contains VRPSD-TS and VRPH-SA-A. Configuration performance on these targets shows large variation. For VRPSD-TS, only CMA-ES and ParamILS are able to find parameter configurations that clearly outperform the defaults, whereas other methods are able to only slightly improve the solution quality. VRPH-SA-A shows similar behavior with high variance. For this algorithm, all of the configurators, except REVAC, were repeatedly able to produce a parameter configuration that allowed solving all of the 14 instances in the Augerat et al. [2] validation set to optimality. One such configuration is given later in Table 7.

In the third difficulty class, we have the targets VRPH-EJ, VRPH-RTR, and VRPSD-EA. Based on our experiments, these seem to be hard to configure effectively and even the more sophisticated automatic algorithm configuration methods struggle to challenge the uniform random sampling on these targets. As can be seen from Table 6 these targets share the feature of having many binary parameters. If we examine the boxplot of Fig. 5, the multimodal nature of these configuration targets can be seen as clustering of outliers around a local optimum of the configuration search space. However, even for these targets, the configurators were able to improve the solution quality over the default parameter configuration with a success rate of 93.8%. Additionally, improvements were often found even with an evaluation budget as small as 100.

Our experiments clearly indicate that the nature of the configured target or, more specifically, the solver algorithms and the problem instance set, has great impact to the configurability, configuration method selection, and generic performance of the solver. In Table 6, the solver performance is compared among the metaheuristics solving the same problem set. In solving the CVRP, we can see that VRPH-RTR clearly benefits from using automatic algorithm configuration. For the CMT instances, VRP-SA-C produces the best-quality solutions with default parameters, but after configuration has been performed, it is beaten by VRPH-RTR-C and VRPH-EJ-C. With Augerat et al. [2] instances on small configuration budgets VRPH-EJ-A and VRPH-RTR-A are performance-wise very similar. With an evaluation budget of 1000, GGA is able to find very good parameters for VRPH-RTR-A, which outperforms the other two solvers on the Augerat instance set. Note that the large median absolute deviation in VRPH-SA-A results indicates that there is a lot of variation between the configured parameter configurations or their evaluations, which means that this good performance is inconsistent. The reason behind this may be in the optimal cooling schedule band for SA is known to be narrow [43]. For the IRIDIA instances, automatic configuration changes the ranking between the solvers only slightly. VRPSD-ACO is the winner in solving given VRPSD instances with VRPSD-EA being a close second, not surprisingly given the state-of-the-art performance of the evolutionary approach in the literature [50, 60].

6.3 Automatically configured parameters

The parameter configurations with the best median solution quality can be found in Table 7. However, it is likely that the parameter values are highly instance and solver implementation specific, which limits our ability to make general recommendations. Also, please remember that a 10 s cutoff was used in our experiments, and this should be considered when generalizing the parameter values. Still, the best found parameter configurations offer a basis for our discussion on algorithm nature and solution space structure in the

Table 7 The best median solution quality Q for a single parameter configuration

			Local search operators	Optimized parameters							
	Q_C	Q_A	1*pm*/2*pm*/*two*/*oro*/*tho*/3*pm*	m	t	s					
VRPH EJ default	0.96	0.73	1 / 1 / 1 / 0 / 0 / 0	10	1000	0					
VRPH EJ C (SMAC)	0.54		1 / 1 / 1 / 0 / 1 / 1	17	7465	1					
VRPH EJ A (SMAC)		0.34	1 / 1 / 1 / 0 / 0 / 1	19	5316	1					
	Q_C	Q_A	1*pm*/2*pm*/*two*/*oro*/*tho*/3*pm*	D	δ	K	N	P	p/a	t	
VRPH RTR default	1.42	1.40	1 / 1 / 1 / 0 / 0 / 0	30	0.01	5	4	1	1/1	0	
VRPH RTR C (GGA)	0.40		1 / 1 / 1 / 1 / 0 / 1	18	0.01	51	11	4	0/0	39	
VRPH RTR A (GGA)		0.01	1 / 1 / 1 / 1 / 0 / 0	98	0.04	43	30	6	1/0	6	
	Q_C	Q_A	1*pm*/2*pm*/*two*/*oro*/*tho*/3*pm*	T	n	i	α		N		
VRPH SA default	0.80	0.90	1 / 1 / 1 / 0 / 0 / 0	2.00	200	2	0.99		10		
VRPH SA C (GGA)	0.63		1 / 1 / 1 / 0 / 0 / 0	2.00	200	5	0.99		10		
VRPH SA A (SMAC)		0.01	1 / 1 / 1 / 1 / 1 / 1	8.79	498	5	0.99		23		
		Obj.f. est.		Optimized parameters							
	Q	p	t	m	au	ψ		ρ	q	α	
VRPSD ACO default	0.63	0	0	7	0.50	0.30		0.10	1.0e7	1.00	
VRPSD ACO (GGA)	0.15	0	0	1	0.53	0.85		0.41	4.2e6	3.12	
	Q	p	t	p	mr	amr					
VRPSD EA default	0.77	0	0	0	0.20	0					
VRPSD EA (GGA)	0.42	1	1	1	0.63	1					
	Q	p	t	x							
VRPSD ILS default	0.78	0	0	10.00							
VRPSD ILS (SMAC)	0.70	0	0	29.80							
	Q	p	t	μ	α	ψ		ρ			
VRPSD SA default	0.79	0	0	0.01	0.98	1		20			
VRPSD SA (GGA)	0.77	0	0	0.08	0.18	1		20			
	Q	p	t	ttf	p_t	p_o					
VRPSD TS default	1.86	0	0	0.80	0.80	0.30					
VRPSD TS (CMA-ES)	0.51	1	0	1.00	1.00	1.00					

conclusions. Table 7 also allows comparison of parameter values and resulting solution quality between the default configuration and the configured one. Out of the compared configurators SMAC and GGA seem to be most successful in finding very good parameter configurations. If this evidence is combined with observations from boxplots such as the one presented in Fig. 5, the overall impression is that SMAC has more consistent performance, while GGA is occasionally able find better configurations.

Analysis of the configured parameter configurations reveals that in VRPSD-TS, where we observe strikingly different performance between two groups of configurators, the good utility is achieved when at least one of the values for the parameters ttf, p_t, and p_o is at the minimum or maximum. This can also be seen from Table 7. Statistically, it is improbable for a uniform sampling to produce exactly the parameter endpoint value of an interval. Therefore, methods that uniformly sample from within the given range are unable to find these good parameter configurations for VRPSD-TS, whereas configurators that use a robust statistical model or local search are well-suited to the task. The effect was not considered by Balaprakash et al. [3] when they introduced the iterative sampling extension to F-Race, and, to our knowledge, this effect has not previously been

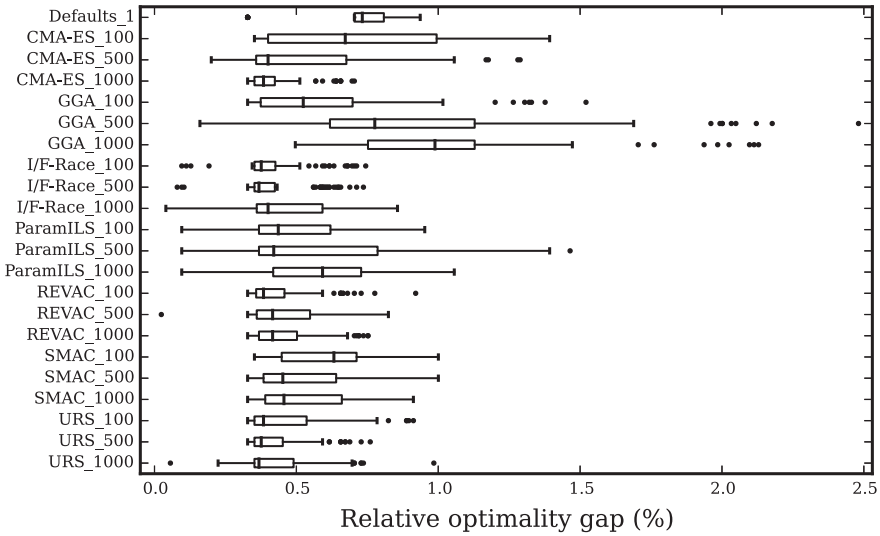


Fig. 5 The distribution of the parameter configuration utility for VRPH-EJ-A target

reported in automatic algorithm configuration literature. The original F-Race would probably find the values of these good parameter configurations, given parameter range end points are chosen as design points in the factorial design. However, VRPSD-TS is a special target in this regard, and F-Race based on factorial design candidate configuration generation would probably show far worse automatic configuration performance on different kind of targets. This is especially true in our configuration scenarios because a full factorial design requires a rather large configuration budget. Also, note that Pellegrini and Birattari [48] did not use the iterative variant of F-Race, and, thus, this behavior did not manifest in their results.

If we now turn to the resulting parameter configurations of the VRPH targets, we can examine how the probability of a local search heuristic to be selected changes with the metaheuristic and the instance set (see Fig. 6). Out of the tested targets, VRPH-SA-C seems to somewhat differ from the rest in its composition of local search operators. With this algorithm, configurations that avoid the more computationally intensive Or-opt, three-opt, and three-point-move operations yield higher utility (routes with a lower cost). Also, in VRPH-SA-C the selection of the operator plays a major role in the resulting solution quality as the local search operator composition of the top 10% parameter configurations differs clearly from the worst 90%. A similar effect can be observed in VRPH-EJ-A, where the use of two-opt operators is preferred over other operations.

It seems that definite connections exist among the composition of local search operators, performance of a metaheuristic, and the instances to be solved. Automatic algorithm configuration makes it possible to find suitable local search operator composition to optimize the performance of a routing solver. This verifies the observation made by Garrido et al. [17] that careful selection of local search operators for a set of instances is a relatively stable way of improving the overall performance of a solver. However, in this study we refrain from examining the differences in local search operator selection between the configurators further.

7 Conclusions and future research

In this paper, we have presented a comprehensive empirical evaluation of seven well-known automatic algorithm configuration methods in the task of configuring eight metaheuristic algorithms solving two vehicle routing problem (VRP) variants. The tested configurators were CMA-ES, GGA, I/F-Race, ParamILS, REVAC, SMAC, and URS. The VRPH library, which is used to solve capacitated vehicle routing problems, offers three solvers with EJ, SA, and RTR metaheuristics. The IRIDIA solvers for the VRPSD uses ACS, EA, ILS, SA, and TS metaheuristics. The solvers had from 3 to 14 free parameters. Each configurator was given a task to find a parameter configuration producing high quality solutions for each algorithm used to solve a relatively small benchmark set of VRP instances. Runtime of the solvers was limited to 10 s.

The results show that, in general, the configuration methods were able to find parameter configurations that produced better solutions than the solver default, even when restricted to as little as 100 solver invocations. This is consistent with previous research where it has been shown repeatedly that automatic algorithm configuration can remarkably improve the performance of stochastic search algorithms over the default parameters. Despite this prior assumption, the low computational cost of achieving performance improvement can be considered surprising. Using just a plain random uniform sampling strategy with a highly limited computational budget would often produce a clearly better performing

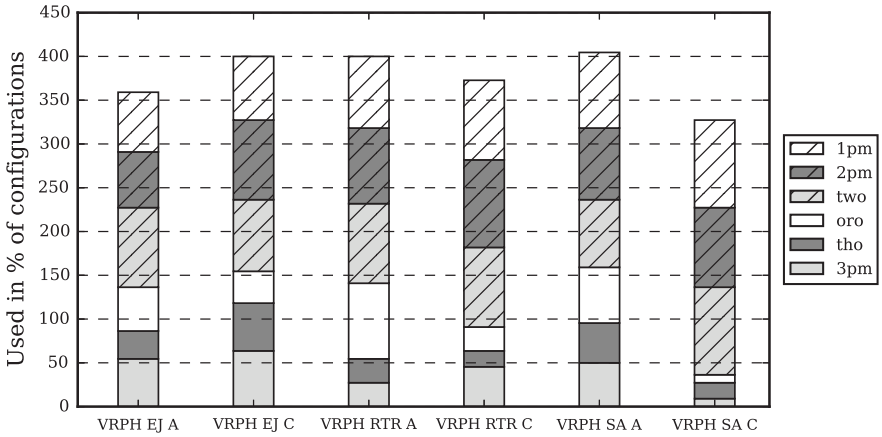


Fig. 6 Local search operator composition of the 10% best VRPH parameter configurations for each VRPH target

parameter configuration than the defaults. Occasionally, random sampling even beat the more advanced configurators by a clear margin as sophistication does not dominate in cases where the solution space has little structure to exploit.

To answer the question of configuration method suitability, our analysis suggests that there is no single best automatic algorithm configuration method for the tested VRP metaheuristics. However, the statistically significant evidence in this study verified that CMA-ES is a good choice when dealing with targets that have many continuous parameters, and that I/F-Race is well-suited for algorithm configuration targets that have many on-off switches for enabling and disabling solver features. Our experimentation also revealed that GGA and REVAC require a lot of trial-and-error and expertise to find parameter values that enable them to use the evaluation budget effectively. This creates an additional level of parameters to tweak on top of the original problem, which makes it hard to effectively apply these configuration methods.

We argue that robustness, and being parameter-free, are desirable properties for an automatic configuration method. Based on our survey, out of the tested configurators CMA-ES, I/F-Race, ParamILS, SMAC, and URS fulfill these requirements. If good performance and robustness is required, and a relatively generous evaluation budget is available, we would recommend SMAC and I/F-Race. Based on our experiments they are both capable of reliably producing good quality parameter configurations. Also GGA is in some situations a competitive choice, but in our experiments it was not as robust as SMAC and I/F-Race. While we could not give a definite recommendation on which single configurator one should use to configure VRP metaheuristics, the results together with the provided survey should help VRP researchers and practitioners to select a method that is probably a good fit. Also, confirming that these results apply with other metaheuristics, time limits, and problem instance sizes will warrant additional computational experiments of configuring VRP algorithms.

The way a target algorithm responded to configuration efforts varied between configurators, evaluation budgets, and even between problem instance sets. We acknowledge that the time limits do affect the results of the comparison of metaheuristics or configuration methods. In our experiments, we varied the evaluation budget and tested the configurator performance on three problem instance sets, but used the same time limit in all experiments. Considering this, the results of our experiments conclude that there does not exist a single, best configuration method for different algorithms. However, we were able to distinguish differing configurator behavior with the different evaluation budget constraints. The results suggest that SMAC and I/F-Race would be most appropriate configurators for larger instances with longer execution times. Our study also showed that in our set of configuration problems some configurators are more robust than others.

Our recommended strategy to address the inconclusive nature of the results is to have several state-of-the-art automatic algorithm configuration methods at the user's disposal. Experimenting with different configurators helps one to see when a good fit is found, as the solver usually responds quickly to automatic configuration attempts even with a small evaluation budget. Furthermore, our findings have important implications for future practice. Contributed evidence to the usefulness of automatic algorithm configuration of VRP metaheuristics strongly suggests that routing algorithm developers should start using an automatic algorithm configuration method in their experiments. This is important in particular when making algorithm performance comparisons, as configuring the parameters of a set of algorithms allows one to avoid confirmation bias, that is, the performance of the algorithm is not determined by the suitability of its default parameters or the amount of manual fine-tuning it receives.

Regarding generalization of the results, we see from Table 1 that the included configurators address a large set of different features and aspects of automatic algorithm

configuration. The same holds true for the various features represented in benchmark problems and their solvers as depicted in Table 2: we hypothesize that the experimental results hold true also for different mixtures of the same solution method constituents. The use of these approaches is common in designing heuristics for combinatorial optimization problems [54].

Additional and extensive comparison between these configurators with different experiment parameters, e.g., with larger, more difficult, or 'rich' [12] problem instances or with a longer metaheuristic CPU runtime, would be required to reliably estimate how well our observations generalize. The recent advances in the field of automatic algorithm configuration addressing the issues with long running algorithms are relevant here [18, 28, 41, 57]. The experiments could also be extended with configuration targets that have more binary and categorical parameters.

A typical use for a routing solver is to solve sets of slightly different problem instances repeatedly. Automatic configuration in such a scenario can be considered as modeling the interactions of the triplet: instance, parameter configuration, and solution quality. Further work is required to establish the feasibility of utilizing these previously discovered interactions in future solving tasks. This research avenue is also recognized, e.g., in [61]. The reasonable next step could be to explore the feature extraction of VRP instances, solutions, and routes, and then investigate the suitability of instance-specific algorithm configuration methods. These methods use instance features to make utility predictions for the parameter configuration candidates. SMAC can be used as instance-specific method, but there are other methods, such as ISAC from Kadioglu et al. [34]. Also, of particular interest from practical operations research and vehicle routing viewpoint would be extending our investigations to algorithm selection. Especially applicability and implications of using algorithm selectors, such as Hydra [63] or AutoFolio from Lindauer et al. [36], should be explored.

It is also important to acknowledge the drawbacks of automatic algorithm configuration. The configuration methods rarely provides useful information on why a certain parameter configuration was selected. Here, domain knowledge and understanding of the target algorithm is required to understand the general implications of the resulting parameter configuration. This is especially important in academic research, where understanding why an optimization strategy works is of paramount importance. Therefore, we see implementing in features like parameter sensitivity analysis and visualization of the parameter configuration search space as important development and research aims of configuration method community.

Regarding validity of the study, we would like discuss four things: configuration budget, CPU cutoff, problem set size, configuration objective, and generalization of the results to other VRP variants. The possible limitations of the study are due to the extensive computational requirements required with each additional variability dimension introduced to the comparison. Also, the research questions and the specifics in solving vehicle routing problems made it possible, or in some cases necessary, to fix some aspects of the experimental setup.

In this study an evaluation budget was used to limit the computational resources available during automatic algorithm configuration. However, some configuration methods failed to adhere to this budget. To control the effect this issue might have on validity, we have addressed deviations from the budget in our analysis.

We also acknowledge that future comparisons should study the effect of a more generous configuration budget on configuring VRP metaheuristics. In our study we tested a single target with a budget of 5000 evaluations, which does not allow analyzing the variation between configuration targets in the scenario of a larger budget.

The large number of experiments, and a decision to keep as many of the variables constant as possible in the experimental setup lead us to limit the CPU time of the solvers to 10 s. We experimentally verified that the VRPH solver performance stabilizes by the 10 s mark. However, it is likely that this creates a bias to prefer parameter configurations specifying a more explorative search strategy, especially on larger instances of the CMT problem set and when solving VRPSD instances. Similarly. the choice to use a problem set size of 14 was done to keep it constant over all three configuration target groups, and to include automatic algorithm configuration methods that lack training set subset evaluation mechanisms. We acknowledge that when using a small training set, there is a danger of overfitting. However, in our experiments only few selected configuration targets show weak signs of such behavior, and these do not affect our overall results and recommendations.

VRP is a challenging, well-known, and well-studied combinatorial optimization problem that generalizes several other problems. Therefore, it can serve as an interesting benchmark for evaluating the robustness of automated algorithm configuration methods and tools. In this study we focused only on CVRP and VRPSD, but there are many other variants with different constraints, objectives and features. Also, even different problem instances of a single variant can have differing characteristics (see, e.g., [51]). Thus, we would like to see automatic algorithm configuration method comparisons on the VRPTW (VRP with time windows), PDP (pickup and delivery problem), large-scale CVRP, and rich VRP benchmarks, which could show different facets of configuring VRP solvers and perhaps provide further support for our results.

Acknowledgements Open access funding provided by University of Jyväskylä (JYU). The financial support by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development for Nyset Musliu is gratefully acknowledged.

Compliance with ethical standards

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Ansótegui, C., Sellmann, M., Tierney, K.: A gender-based genetic algorithm for the automatic configuration of algorithms. In: Gent, I. (ed.) Principles and Practice of Constraint Programming - CP'09. Lecture Notes in Computer Science, vol. 5732, pp. 142–157. Springer, Berlin (2009)
2. Augerat, P., Belenguer, J., Benavent, E., Corberán, A., Naddef, D., Rinaldi, G.: Computational results with a branch and cut code for the capacitated vehicle routing problem. Technical Report 949-M, Universite Joseph Fourier, Grenoble, France (1995)
3. Balaprakash, P., Birattari, M., Stützle, T.: Improvement strategies for the F-Race algorithm: sampling design and iterative refinement. Technical Report TR/IRIDIA/2007-011, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium (2007)
4. Barbucha, D.: Experimental study of the population parameters settings in cooperative multi-agent system solving instances of the VRP. In: Nguyen, N.T. (ed.) Transactions on Computational Collective Intelligence IX. Lecture Notes in Computer Science, vol. 7770, pp. 1–28. Springer, Berlin (2013)
5. Bartz-Beielstein, T., Lasarczyk, C., Preuß, M.: Sequential parameter optimization. In: IEEE Congress on Evolutionary Computation—CEC'05, vol. 1, pp. 773–780 (2005)
6. Battiti, R., Brunato, M.: Reactive search optimization: learning while optimizing. In: Gendreau, M., Potvin, J.Y. (eds.) Handbook of Metaheuristics, International Series in Operations Research & Management Science, vol. 146, 2nd edn, pp. 543–571. Springer, New York (2010)
7. Becker, S., Gottlieb, J., Stützle, T.: Applications of racing algorithms: an industrial perspective. In: Proceedings of the 7th International Conference on Artificial Evolution—EA'05, pp. 271–283. Springer, Berlin (2006)
8. Bianchi, L., Birattari, M., Chiarandini, M., Manfrin, M., Mastrolilli, M., Paquete, L., Rossi-Doria, O., Schiavinotto, T.: Hybrid metaheuristics for the vehicle routing problem with stochastic demands. J. Math. Model. Algorithms **5**(1), 91–110 (2005)
9. Birattari, M., Stützle, T., Paquete, L., Varrentrapp, K.: A racing algorithm for configuring metaheuristics. In: Proceedings of the Genetic and Evolutionary Computation Conference—GECCO'02, pp. 11–18. Morgan Kaufmann, San Francisco, CA (2002)
10. Birattari, M., Yuan, Z., Balaprakash, P., Stützle, T.: F-Race and iterated F-Race: an overview. In: Bartz-Beielstein, T., Chiarandini, M., Paquete, L., Preuss, M. (eds.) Experimental Methods for the Analysis of Optimization Algorithms, pp. 311–336. Springer, Berlin (2010)
11. Bräysy, O., Hasle, G.: Vehicle Routing: Problems, Methods, and Applications, chap. 12 Software tools and emerging technologies for vehicle routing and intermodal transportation, pp. 351–380. In: [58] (2014)
12. Caceres-Cruz, J., Arias, P., Guimarans, D., Riera, D., Juan, A.A.: Rich vehicle routing problem: survey. ACM Comput. Surv. (CSUR) **47**(2), 32 (2015)
13. Christofides, N., Mingozzi, A., Toth, P.: The vehicle routing problem. Revue Française d'Informatique et de Recherche Opérationnelle **10**(2), 55–70 (1976)
14. Coy, S.P., Golden, B.L., Runger, G.C., Wasil, E.A.: Using experimental design to find effective parameter settings for heuristics. J. Heuristics **7**, 77–97 (2001)
15. Eiben, A., Hinterding, R., Michalewicz, Z.: Parameter control in evolutionary algorithms. IEEE Trans. Evolut. Comput. **3**(2), 124–141 (1999)
16. Eiben, A., Smit, S.: Parameter tuning for configuring and analyzing evolutionary algorithms. Swarm Evol. Comput. **1**(1), 19–31 (2011)
17. Garrido, P., Castro, C., Monfroy, E.: Towards a flexible and adaptable hyperheuristic approach for VRPs. In: Arabnia, H.R., de la Fuente, D., Olivas, J.A. (eds.) Proceedings of the 2009 International Conference on Artificial Intelligence—ICAI'09, pp. 311–317. CSREA Press, USA (2009)
18. Geschwender, D., Hutter, F., Kotthoff, L., Malitsky, Y., Hoos, H.H., Leyton-Brown, K.: Algorithm configuration in the cloud: a feasibility study. In: Pardalos, M.P., Resende, G.M., Vogiatzis, C., Walteros, L.J. (eds.) Learning and Intelligent Optimization: 8th International Conference, Lion 8, Gainesville, FL, USA, February 16–21, 2014. Revised Selected Papers, pp. 41–46. Springer (2014)
19. Groër, C.: Parallel and serial algorithms for vehicle routing problems. Dissertation, University of Maryland (2008)
20. Groër, C., Golden, B., Wasil, E.: A library of local search heuristics for the vehicle routing problem. Math. Program. Comput. **2**(2), 79–101 (2010)
21. Hansen, N.: The CMA evolution strategy: a comparing review. In: Lozano, J., Larranaga, P., Inza, I., Bengoetxea, E. (eds.) Towards a New Evolutionary Computation. Advances on Estimation of Distribution Algorithms, pp. 75–102. Springer, Berlin (2006)
22. Hepdogan, S., Moraga, R., DePuy, G., Whitehouse, G.: Nonparametric comparison of two dynamic parameter setting methods in a meta-heuristic approach. J. Syst. Cybern. Inf. **5**(5), 46–52 (2008)
23. Hoos, H.H.: Automated algorithm configuration and parameter tuning. In: Hamadi, Y., Monfroy, E., Saubion, F. (eds.) Autonomous Search, pp. 37–71. Springer, Berlin (2012)
24. Hoos, H.H., Stützle, T.: Stochastic Local Search: Foundations and Applications. Elsevier, Amsterdam (2004)
25. Hutter, F., Bartz-Beielstein, T., Hoos, H., Leyton-Brown, K., Murphy, K.: Sequential model-based parameter optimization: an experimental investigation of automated and interactive approaches. In: Bartz-Beielstein, T., Chiarandini, M., Paquete, L., Preuss, M. (eds.) Experimental Methods for the Analysis of Optimization Algorithms, pp. 363–414. Springer, Berlin (2010a)
26. Hutter, F., Hoos, H., Leyton-Brown, K.: Automated configuration of mixed integer programming solvers. In: Lodi, A., Milano, M., Toth, P. (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Lecture Notes in Computer Science, vol. 6140, pp. 186–202. Springer, Berlin (2010b)
27. Hutter, F., Hoos, H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Coello, C.A.C. (ed.) Learning and Intelligent Optimization. Lecture Notes in Computer Science, vol. 6683, pp. 507–523. Springer, Berlin (2011)

28. Hutter, F., Hoos, H., Leyton-Brown, K.: Parallel algorithm configuration. In: Hamadi, Y., Schoenauer, M. (eds.) Learning and Intelligent Optimization. Lecture Notes in Computer Science, vol. 7219, pp. 55–70. Springer, Berlin (2012)
29. Hutter, F., Hoos, H., Leyton-Brown, K., Murphy, K.: Time-bounded sequential parameter optimization. In: Blum, C., Battiti, R. (eds.) Learning and Intelligent Optimization. Lecture Notes in Computer Science, vol. 6073, pp. 281–298. Springer, Berlin Heidelberg (2010c)
30. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: An automatic algorithm configuration framework. J. Artif. Intell. Res. (JAIR) **36**, 267–306 (2009)
31. Hutter, F., Hoos, H.H., Stützle, T.: Automatic algorithm configuration based on local search. In: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, pp. 1152–1157. AAAI Press, Menlo Park, CA (2007)
32. Irnich, S., Toth, P., Vigo, D.: Vehicle Routing: Problems, Methods, and Applications, chap. 1 the family of vehicle routing problems, pp. 1–33. In: [58] (2014)
33. Jourdan, L., Basseur, M., Talbi, E.G.: Hybridizing exact methods and metaheuristics: a taxonomy. Eur. J. Oper. Res. **199**(3), 620–629 (2009)
34. Kadioglu, S., Malitsky, Y., Sellmann, M., Tierney, K.: ISAC—instance-specific algorithm configuration. In: Coelho, H., Studer, R., Wooldridge, M. (eds.) 19th European Conference on Artificial Intelligence—ECAI 2010, Frontiers in Artificial Intelligence and Applications, vol. 215, pp. 751–756. IOS Press, Amsterdam, Netherlands (2010)
35. Laporte, G.: What you should know about the vehicle routing problem. Nav. Res. Log. **54**(8), 811–819 (2007)
36. Lindauer, M., Hoos, H., Hutter, F., Schaub, T.: Autofolio: An automatically configured algorithm selector. J. Artif. Intell. Res. **53**, 745–778 (2015)
37. López-Ibánez, M., Dubois-Lacoste, J., Stützle, T., Birattari, M.: The irace package, iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles (2011)
38. Lourenço, H.R., Martin, O.C., Stützle, T.: Iterated local search: framework and applications. In: Gendreau, M., Potvin, J.Y. (eds.) Handbook of Metaheuristics, International Series in Operations Research & Management Science, Chap. 12, vol. 146, 2nd edn, pp. 363–397. Springer, New York (2010)
39. Maron, O., Moore, A.W.: Hoeffding races: Accelerating model selection search for classification and function approximation. In: Cowan, J., Tesauro, G., Alspector, J. (eds.) Advances in Neural Information Processing Systems, vol. 6, pp. 59–66. Morgan Kaufmann, San Francisco (1994)
40. Maron, O., Moore, A.W.: The racing algorithm: model selection for lazy learners. Artif. Intell. Rev. **11**(1–5), 193–225 (1997)
41. Mascia, F., Birattari, M., Stützle, T.: Tuning algorithms for tackling large instances: an experimental protocol. In: Nicosia, G., Pardalos, P. (eds.) Learning and Intelligent Optimization. Lecture Notes in Computer Science, vol. 7997, pp. 410–422. Springer, Berlin (2013)
42. Mester, D., Bräysy, O.: Active-guided evolution strategies for large-scale capacitated vehicle routing problems. Comput. Oper. Res. **34**(10), 2964–2975 (2007)
43. Miki, M., Hiroyasu, T., Jitta, T.: Adaptive simulated annealing for maximum temperature. In: 2003 IEEE International Conference on Systems, Man and Cybernetics—SMC 2003, vol. 1, pp. 20–25 (2003)
44. Montero, E., Riff, M., Neveu, B.: New requirements for off-line parameter calibration algorithms. In: 2010 IEEE Congress on Evolutionary Computation—CEC'10, pp. 1–8 (2010)
45. Montero, E., Riff, M.C., Neveu, B.: An evaluation of off-line calibration techniques for evolutionary algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference—GECCO'10, pp. 299–300. ACM, New York (2010)
46. Nannen, V., Eiben, A.E.: Efficient relevance estimation and value calibration of evolutionary algorithm parameters. In: 2007 IEEE Congress on Evolutionary Computation—CEC'07, pp. 103–110 (2007)
47. Pellegrini, P.: Application of two nearest neighbor approaches to a rich vehicle routing problem. Technical Report TR/IRIDIA/2005-015, IRIDIA, Université Libre de Bruxelles (2005)
48. Pellegrini, P., Birattari, M.: Implementation effort and performance. In: Stützle, T., Birattari, M., Hoos, H.H. (eds.) Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics. Lecture Notes in Computer Science, vol. 4638, pp. 31–45. Springer, Berlin (2007)
49. Penna, P.H.V., Subramanian, A., Ochi, L.S., Vidal, T., Prins, C.: A hybrid heuristic for a broad class of vehicle routing problems with heterogeneous fleet. Ann. Oper. Res. **273**(1–2), 5–74 (2017)
50. Prins, C.: A simple and effective evolutionary algorithm for the vehicle routing problem. Comput. Oper. Res. **31**(12), 1985–2002 (2004)
51. Rasku, J., Kärkkäinen, T., Musliu, N.: Feature extractors for describing vehicle routing problem instances. In: SCOR, OASICS, vol. 50, pp. 7:1–7:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2016)
52. Rasku, J., Musliu, N., Kärkkäinen, T.: Automating the parameter selection in VRP: an off-line parameter tuning tool comparison. In: Fitzgibbon, W., Kuznetsov, A.Y., Neittaanmäki, P., Pironneau, O. (eds.) Modeling, Simulation and Optimization for Science and Technology, Proceedings of Optimization and PDEs with Applications Workshop, June 18–19, 2012, University of Jyväskylä, Finland, Computational Methods in Applied Sciences, vol. 34, pp. 191–209. Springer (2014)
53. Rasku, J., Puranen, T., Kalmbach, A., Kärkkäinen, T.: Automatic customization framework for efficient vehicle routing system deployment. In: Diez, P., Neittaanmäki, P., Periaux, J., Tuovinen, T., Bräysy, O. (eds.) Computational Methods and Models for Transport: New Challenges for the Greening of Transport Systems, pp. 105–120. Springer, New York (2018)
54. Sevaux, M., Sörensen, K., Pillay, N.: Adaptive and multilevel metaheuristics. In: Martí, R., Panos, P., Resende, M. (eds.) Handbook of Heuristics, pp. 1–19. Springer, Cham (2018)
55. Smit, S.K., Eiben, A.E.: Comparing parameter tuning methods for evolutionary algorithms. In: 2009 IEEE Congress on Evolutionary Computation—CEC'09, pp. 399–406 (2009)
56. Sörensen, K., Sevaux, M., Schittekat, P.: Multiple neighbourhood search in commercial VRP packages: evolving towards self-adaptive methods. Stud. Comp. Intell. **136**, 239–253 (2008)
57. Styles, J., Hoos, H.: Using racing to automatically configure algorithms for scaling performance. In: Nicosia, G., Pardalos, P. (eds.) Learning and Intelligent Optimization. Lecture Notes in Computer Science, vol. 7997, pp. 382–388. Springer, Berlin (2013)
58. Toth, P., Vigo, D.: Vehicle Routing: Problems, Methods, and Applications. SIAM, Philadelphia (2014)
59. Uchoa, E., Pecin, D., Pessoa, A., Poggi, M., Vidal, T., Subramanian, A.: New benchmark instances for the capacitated vehicle routing problem. Eur. J. Oper. Res. **257**(3), 845–858 (2017)
60. Vidal, T., Crainic, T.G., Gendreau, M., Lahrichi, N., Rei, W.: A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. Oper. Res. **60**(3), 611–624 (2012)
61. Vidal, T., Crainic, T.G., Gendreau, M., Prins, C.: Heuristics for multi-attribute vehicle routing problems: a survey and synthesis. Eur. J. Oper. Res. **231**(1), 1–21 (2013)

62. Wink, S., Back, T., Emmerich, M.: A meta-genetic algorithm for solving the capacitated vehicle routing problem. In: IEEE Congress on Evolutionary Computation—CEC'12, pp. 1–8 (2012)
63. Xu, L., Hoos, H.H., Leyton-Brown, K.: Hydra: Automatically configuring algorithms for portfolio-based selection. In: Twenty-Fourth Conference of the Association for the Advancement of Artificial Intelligence (AAAI-10), pp. 210–216 (2010)
64. Yuan, Z., de Oca, M.A.M., Birattari, M., Stützle, T.: Continuous optimization algorithms for tuning real and integer parameters of swarm intelligence algorithms. Swarm Intell. **6**(1), 49–75 (2012)